
pysubgroup Documentation

Release 0.0.2

Florian Lemmerich

Apr 24, 2020

CONTENTS:

1	Pysubgroup	3
1.1	Subgroup Discovery	3
1.2	Prerequisites and Installation	4
1.3	How to use:	4
2	How pysubgroup works	5
3	Selectors	11
3.1	Basic Selectors	11
3.2	Negations	12
3.3	Conjunctions	12
3.4	Disjunctions	12
3.5	Implementing your own	13
4	Targets and Quality Functions	15
4.1	Frequent Itemset Targets	15
4.2	Binary Targets	15
4.3	Nominal Targets	15
4.4	Numeric Targets	16
4.5	Custom Quality Function	16
5	GP-Growth	17
5.1	Basic usage	17
5.2	Create a custom target	18
6	pysubgroup	21
6.1	pysubgroup package	21
7	Indices and tables	33
	Python Module Index	35
	Index	37

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.

As of 2018, this library is still in a prototype phase. It has, however, been already successfully employed in active application projects.

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.

As of 2018, this library is still in a prototype phase. It has, however, been already successfully employed in active application projects.

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 successrate 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:

1.2 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
```

1.3 How to use:

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

```
[1]: import pysubgroup as ps

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

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

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

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

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

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

```
[5]: result.to_dataframe()

[5]:   quality      description
0  0.132150      Sex=='female'
1  0.101331  Parch==0 AND Sex=='female'
2  0.079142  Sex=='female' AND SibSp: [0:1[
3  0.077663  Cabin.isnull() AND Sex=='female'
4  0.071746  Embarked=='S' AND Sex=='female'
```


HOW PYSUBGROUP WORKS

just ignore the following code block for non technical people, but have a look at the example dataFrame (table) that we created

```
[1]: import pysubgroup as ps
import pandas as pd
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', 300)
products = ['toast', 'bread', 'bread', 'toast', 'bread', 'bread', 'toast', 'bread',
↳ 'bread', 'pizza']
amount = [100,1000,100,1000,10000,540,750,860,350,400]
was_fraud = [1,0,0,0,1,1,1,0,0,0]
day_of_month = [15,20,30,17,12,7,11,14,20,27]
is_gold = [1,0,1,0,0,0,1,1,1,0]
df = pd.DataFrame.from_dict({'product': products, 'was_fraud': was_fraud, 'amount':,
↳ amount, 'day_of_month' : day_of_month, 'is_gold':is_gold})
df
```

```
[1]:
```

	product	was_fraud	amount	day_of_month	is_gold
0	toast	1	100	15	1
1	bread	0	1000	20	0
2	bread	0	100	30	1
3	toast	0	1000	17	0
4	bread	1	10000	12	0
5	bread	1	540	7	0
6	toast	1	750	11	1
7	bread	0	860	14	1
8	bread	0	350	20	1
9	pizza	0	400	27	0

```
[2]: target = ps.BinaryTarget('was_fraud', 1) # we are looking for fraud=1
search_space = ps.create_selectors(df, ignore='was_fraud') # define what we are,
↳ looking for
search_space
```

```
[2]: [product=='toast',
product=='bread',
product=='pizza',
amount<350,
amount: [350:540[,
amount: [540:860[,
amount: [860:1000[,
amount>=1000,
day_of_month<12,
day_of_month: [12:15[,
```

(continues on next page)

(continued from previous page)

```

day_of_month: [15:20[,
day_of_month: [20:27[,
day_of_month>=27,
is_gold==0,
is_gold==1]

```

As you can see, pysubgroup created selectors for us, treating nominal columns (product, is_gold) different from numeric columns (where it uses intervals selectors) you can also create your own selectors see below

```

[3]: my_selector = ps.IntervalSelector('amount', 500, 25000)
print(my_selector)
search_space.append(my_selector) # add my selector to searchspace

amount: [500:25000[

```

Now that we have defined where we want to search we write all that information into a task object:

```

[4]: quality_function = ps.StandardQF(0) # Looks for subgroups with highest true positives,
    ↪ratio
min_quality = 0.2 # Minimum required quality = min true positive ratio
task = ps.SubgroupDiscoveryTask(df, target, search_space, quality_function, result_
    ↪set_size=10, depth=3, min_quality=min_quality)

```

now that we have that task object we can run the algorithm (Here Depth first search)

```

[5]: result = ps.SimpleDFS().execute(task) # Run the algorithm
result.to_dataframe(include_info=True) # Show the output

```

	quality		description	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		
0	0.6	amount: [500:25000[AND amount: [540:860[AND product=='toast'		1	
↪	10	1	4	9	0.1
↪		0.9	0.25	0.75	1.0
↪	0.333333		0.4	2.5	
1	0.6	amount: [500:25000[AND day_of_month<12 AND product=='toast'		1	
↪	10	1	4	9	0.1
↪		0.9	0.25	0.75	1.0
↪	0.333333		0.4	2.5	
2	0.6	amount: [540:860[AND day_of_month<12 AND product=='toast'		1	
↪	10	1	4	9	0.1
↪		0.9	0.25	0.75	1.0
↪	0.333333		0.4	2.5	
3	0.6	amount<350 AND product=='toast'		1	
↪	10	1	4	9	0.1
↪		0.9	0.25	0.75	1.0
↪	0.333333		0.4	2.5	
4	0.6	amount: [540:860[AND product=='toast'		1	
↪	10	1	4	9	0.1
↪		0.9	0.25	0.75	1.0
↪	0.333333		0.4	2.5	
5	0.6	amount<350 AND day_of_month: [15:20[AND product=='toast'		1	
↪	10	1	4	9	0.1
↪		0.9	0.25	0.75	1.0
↪	0.333333		0.4	2.5	
6	0.6	amount: [540:860[AND is_gold==1 AND product=='toast'		1	
↪	10	1	4	9	0.1
↪		0.9	0.25	0.75	1.0
↪	0.333333		0.4	2.5	

(continues on next page)

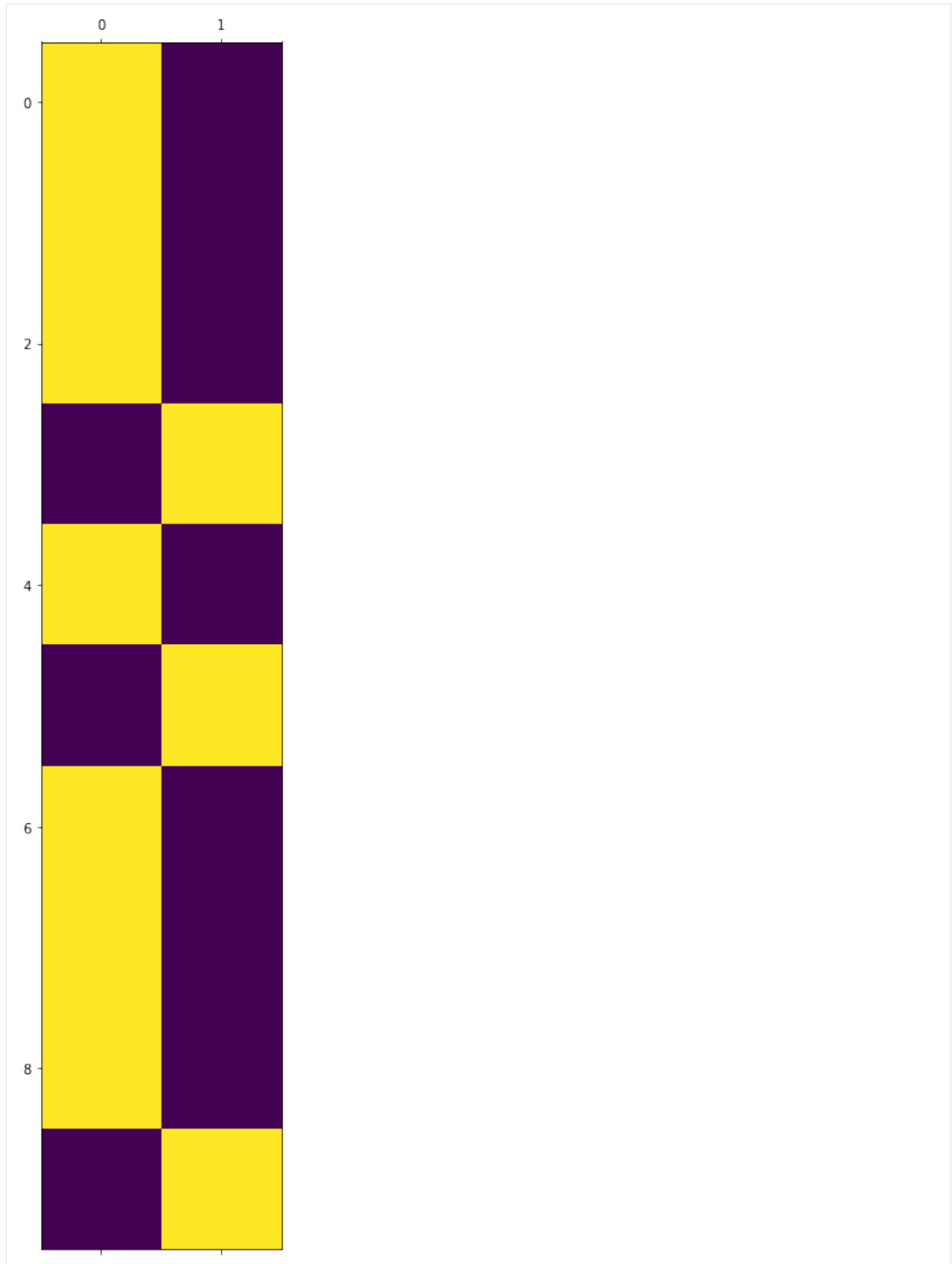
(continued from previous page)

7	0.6	day_of_month<12 AND is_gold==1 AND product=='toast'					1	↳
↳	10	1	4	9	0.1		↳	↳
↳		0.9	0.25	0.75	1.0		↳	↳
↳	0.333333		0.4	2.5				↳
8	0.6	day_of_month<12 AND product=='toast'					1	↳
↳	10	1	4	9	0.1		↳	↳
↳		0.9	0.25	0.75	1.0		↳	↳
↳	0.333333		0.4	2.5				↳
9	0.6	amount<350 AND is_gold==1 AND product=='toast'					1	↳
↳	10	1	4	9	0.1		↳	↳
↳		0.9	0.25	0.75	1.0		↳	↳
↳	0.333333		0.4	2.5				↳

```
[6]: # Visualize resultset, see there is quite some redundancy
import matplotlib.pyplot as plt
plt.matshow(result.supportSetVisualization())
```

Discarding 8 entities that are not covered

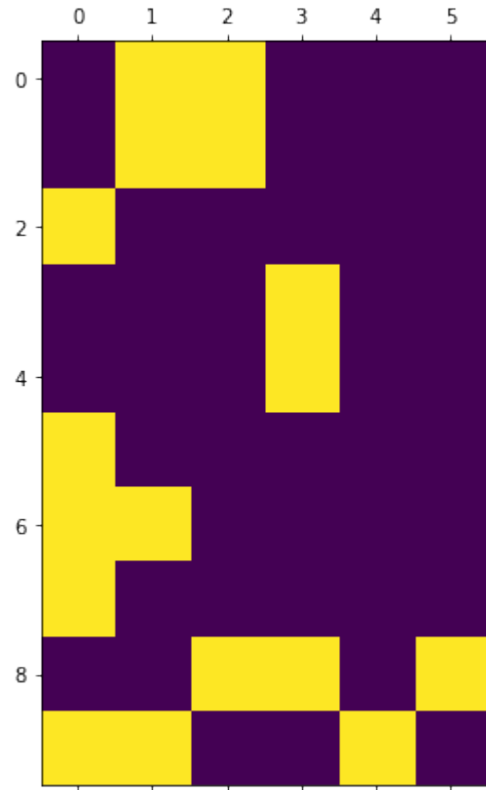
```
[6]: <matplotlib.image.AxesImage at 0x2c8420d7b48>
```



```
[7]: # we can already avoid redundancy a little but its quite slow!
quality_function = ps.GeneralizationAwareQF(ps.StandardQF(0)) # Looks for subgroups_
↳with highest true positives ratio, trying to avoid redundancy
task = ps.SubgroupDiscoveryTask(df, target, search_space, quality_function, result_
↳set_size=10, depth=3, min_quality=min_quality)
result = ps.SimpleDFS().execute(task)
plt.matshow(result.supportSetVisualization())
```

Discarding 4 entities that are not covered

```
[7]: <matplotlib.image.AxesImage at 0x2c84212da48>
```



```
[8]: result.to_dataframe(include_info=True) # Show the output, avoiding redundancy
```

```
[8]:      quality      description  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
0  0.600000      amount: [540:860[      2      10      0.8
↳ 2      4      8      0.2      0.250000
↳ 0.50      0.50      1.000000      0.250000
↳ 0.4  2.500000
1  0.600000      day_of_month<12      2      10      0.8
↳ 2      4      8      0.2      0.250000
↳ 0.50      0.50      1.000000      0.250000
↳ 0.4  2.500000
2  0.500000  amount<350 AND day_of_month: [15:20[      1      10      0.9
↳ 1      4      9      0.1      0.333333
↳ 0.25      0.75      1.000000      0.333333
↳ 0.4  2.500000
```

(continues on next page)

(continued from previous page)

3	0.500000	amount>=1000 AND day_of_month: [12:15[1	10	
→ 1		4 9	0.1		0.9
→	0.25	0.75 1.000000		0.333333	
→	0.4	2.500000			
4	0.500000	day_of_month: [12:15[AND is_gold==0	1	10	
→ 1		4 9	0.1		0.9
→	0.25	0.75 1.000000		0.333333	
→	0.4	2.500000			
5	0.500000	day_of_month: [15:20[AND is_gold==1	1	10	
→ 1		4 9	0.1		0.9
→	0.25	0.75 1.000000		0.333333	
→	0.4	2.500000			
6	0.333333	is_gold==1 AND product=='toast'	2	10	
→ 2		4 8	0.2		0.8
→	0.50	0.50 1.000000		0.250000	
→	0.4	2.500000			
7	0.333333	amount<350 AND product=='toast'	1	10	
→ 1		4 9	0.1		0.9
→	0.25	0.75 1.000000		0.333333	
→	0.4	2.500000			
8	0.266667	is_gold==0 AND product=='bread'	3	10	
→ 2		4 7	0.3		0.7
→	0.50	0.50 0.666667		0.285714	
→	0.4	1.666667			
9	0.266667	product=='toast'	3	10	
→ 2		4 7	0.3		0.7
→	0.50	0.50 0.666667		0.285714	
→	0.4	1.666667			

[]:

SELECTORS

Selectors are objects that if applied to a dataset yield a set of instances. If an instance is returned from a selector we say that the selector 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 `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 selector yourself.

3.1 Basic Selectors

The `pysubgroup` package provides two basic selectors: The `EqualitySelector` and the `IntervalSelector`. Let's 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 code selects all instances from the database, which are in the age range 18 (included) to 40 (excluded).

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

```
[False True True]
```

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

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

3.2 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].

3.3 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.

3.4 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.

3.5 Implementing your own

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

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

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

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

```
[ True False  True]
```

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

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.

4.1 Frequent Itemset Targets

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

4.2 Binary Targets

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

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

The *StandardQF* also supports an optimistic estimate.

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

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

4.3 Nominal Targets

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

4.4 Numeric Targets

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

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

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

4.5 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
```

GP-GROWTH

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

Contents

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

5.1 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.tests.DataSets import get_titanic_data
data = get_titanic_data()

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

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

5.2 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.

5.2.1 Valuation Basis

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

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

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

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

5.2.2 Required Methods

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

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

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

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

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

    @property
    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
```

5.2.3 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
```


PYSUBGROUP

6.1 pysubgroup package

6.1.1 Submodules

6.1.2 pysubgroup.algorithms module

Created on 29.04.2016

@author: lemmerfn

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

```
    execute (task)
```

```
    get_next_level (promising_candidates)
```

```
    get_next_level_candidates (task, result, next_level_candidates)
```

```
    get_next_level_candidates_vectorized (task, result, next_level_candidates)
```

```
    get_next_level_numba (promising_candidates)
```

```
    reprune_lower_levels (promising_candidates, depth)
```

```
class BeamSearch (beam_width=20, beam_width_adaptive=False)
```

Implements the BeamSearch algorithm. Its a basic implementation

```
    execute (task)
```

```
class BestFirstSearch
```

```
    execute (task)
```

```
class DFS (apply_representation)
```

Implementation of a depth-first-search with look-ahead using a provided datastructure.

```
    execute (task)
```

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

```
class DFSNumeric
```

```
    execute (task)
```

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

```

    tpl
        alias of size_mean_parameters
class GeneralisingBFS

    execute (task)
class SimpleDFS

    execute (task, use_optimistic_estimates=True)
    search_internal (task, prefix, modification_set, result, use_optimistic_estimates)
class SimpleSearch (show_progress=True)

    execute (task)
class SubgroupDiscoveryTask (data, target, search_space, qf, result_set_size=10, depth=3,
                             min_quality=0, weighting_attribute=None)
    Capsulates all parameters required to perform standard subgroup discovery

```

6.1.3 pysubgroup.boolean_expressions module

```

class BooleanExpressionBase

    abstract append_and (to_append)
    abstract append_or (to_append)
class Conjunction (selectors)

    append_and (to_append)
    append_or (to_append)
    covers (instance)
    property depth
    pop_and ()
    pop_or ()
class DNF (selectors=None)

    append_and (to_append)
    append_or (to_append)
    pop_and ()
class Disjunction (selectors)

    append_and (to_append)
    append_or (to_append)
    covers (instance)

```

6.1.4 pysubgroup.fi_target module

Created on 29.09.2017

@author: lemmerfn

class AreaQF

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

is_applicable (*subgroup*)

supports_weights ()

class CountQF

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

is_applicable (*subgroup*)

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

supports_weights ()

class FITarget

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

get_attributes ()

get_base_statistics (*data*, *subgroup*, *weighting_attribute=None*)

class SimpleCountQF

calculate_constant_statistics (*task*)

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

gp_get_null_vector ()

gp_get_params (*_cover_arr*, *v*)

gp_get_stats (_)

gp_merge (*l*, *r*)

property **gp_requires_cover_arr**

gp_to_str (*stats*)

tpl

alias of CountQF_parameters

6.1.5 pysubgroup.gp_growth module

```
class GpGrowth (mode='b_u')

    calculate_quality_function_for_patterns (patterns, selectors_sorted, arrs)
    check_constraints (node)
    create_copy_of_path (nodes, new_nodes, stats)
    create_copy_of_tree_top_down (root, nodes=None, parent=None)
    create_new_tree_from_nodes (nodes)
    execute (task)
    get_nodes_upwards (node)
    get_prefixes_top_down (alpha, max_length)
    get_stats_for_class (cls_nodes)
    get_top_down_tree_for_class (cls_nodes, cls)
    insert_into_tree (root, nodes, new_stats, classes, max_depth)
        Creates a tree of a maximum depth = depth
    merge_trees_top_down (nodes, mutable_root, other_root)
    nodes_to_cls_nodes (nodes)
    normal_insert (root, nodes, new_stats, classes)
    prepare_selectors (search_space)
    recurse (cls_nodes, prefix, is_single_path=False)
    recurse_top_down (cls_nodes, root, depth_in=0)
    remove_infrequent_class (nodes, cls_nodes, stats_dict)
    remove_infrequent_nodes (new_nodes)
    to_file (task, path)
```

6.1.6 pysubgroup.measures module

Created on 28.04.2016

@author: lemmerfn

```
class AbstractInterestingnessMeasure

    ensure_statistics (subgroup, statistics_or_data)
    abstract is_applicable (subgroup)
    abstract supports_weights ()

class BoundedInterestingnessMeasure

class CombinedInterestingnessMeasure (measures, weights=None)

    evaluate_from_dataset (data, subgroup, weighting_attribute=None)
```

```

evaluate_from_statistics (instances_dataset, positives_dataset, instances_subgroup, positives_subgroup)

is_applicable (subgroup)

optimistic_estimate_from_dataset (data, subgroup, weighting_attribute=None)

optimistic_estimate_from_statistics (instances_dataset, positives_dataset, instances_subgroup, positives_subgroup)

supports_weights ()

class CountCallsInterestingMeasure (qf)

    calculate_statistics (sg, data=None)

    is_applicable (subgroup)

    supports_weights ()

class GeneralizationAwareQF (qf)

    calculate_constant_statistics (task)

    calculate_statistics (subgroup, data=None)

    evaluate (subgroup, statistics_or_data=None)

    class ga_tuple (subgroup_quality, generalisation_quality)
        Create new instance of ga_tuple(subgroup_quality, generalisation_quality)

        property generalisation_quality
            Alias for field number 1

        property subgroup_quality
            Alias for field number 0

    get_qual_and_previous_qual (subgroup, data)

    is_applicable (subgroup)

    supports_weights ()

class GeneralizationAwareQF_stats (qf)

    calculate_constant_statistics (task)

    calculate_statistics (subgroup, data=None)

    evaluate (subgroup, statistics_or_data=None)

    ga_tuple
        alias of ga_stats_tuple

    get_max (*args)

    get_stats_and_previous_stats (subgroup, data)

    is_applicable (subgroup)

    supports_weights ()

maximum_statistic_filter (result_set, statistic, maximum)

minimum_quality_filter (result_set, minimum)

```

```

minimum_statistic_filter (result_set, statistic, minimum, data)
overlap_filter (result_set, data, similarity_level=0.9)
overlaps_list (sg, list_of_sgs, data, similarity_level=0.9)
unique_attributes (result_set, data)

```

6.1.7 pysubgroup.model_target module

```

class EMM_Likelihood (model)

    calculate_constant_statistics (task)
    calculate_statistics (subgroup, data=None)
    evaluate (subgroup, statistics=None)
    get_tuple (sg_size, params, cover_arr)
    gp_get_params (cover_arr, v)
    is_applicable (_)
    supports_weights (_)
    tpl
        alias of EMM_Likelihood

class PolyRegression_ModelClass (x_name='x', y_name='y', degree=1)

    calculate_constant_statistics (task)
    fit (subgroup, data=None)
    gp_get_null_vector (_)
    gp_get_params (v)
    gp_get_stats (row_index)
    static gp_merge (u, v)
    likelihood (stats, sg)
    loglikelihood (stats, sg)

class beta_tuple (beta, size)
    Create new instance of beta_tuple(beta, size)

    property beta
        Alias for field number 0

    property size
        Alias for field number 1

```

6.1.8 pysubgroup.nominal_target module

6.1.9 pysubgroup.numeric_target module

Created on 29.09.2017

@author: lemmerfn

```

class GStandardQFNumeric(a, invert=False)

    evaluate_from_dataset(data, subgroup, weighting_attribute=None)
    is_applicable(subgroup)
    supports_weights()

class NumericTarget(target_variable)

    calculate_statistics(subgroup, data)
    get_attributes()
    get_base_statistics(data, subgroup, weighting_attribute=None)

class StandardQFNumeric(a, invert=False, estimator='sum')

    class Average_Estimator(qf)

        calculate_constant_statistics(task)
        get_data(task)
        get_estimate(subgroup, sg_size, sg_mean, cover_arr, _)

    class Ordering_Estimator(qf)

        calculate_constant_statistics(task)
        get_data(task)
        get_estimate(subgroup, sg_size, sg_mean, cover_arr, target_values_sg)
        get_estimate_numpy(values_sg, a, mean_dataset)

    class Summation_Estimator(qf)

        calculate_constant_statistics(task)
        get_data(task)
        get_estimate(subgroup, sg_size, sg_mean, cover_arr, _)

    calculate_constant_statistics(task)
    calculate_statistics(subgroup, data=None)
    evaluate(subgroup, statistics=None)
    is_applicable(subgroup)
    optimistic_estimate(subgroup, statistics=None)

```

```
static standard_qf_numeric(a, _, mean_dataset, instances_subgroup, mean_sg)
supports_weights()
tpl
    alias of StandardQFNumeric_parameters
get_max_generalization_mean(data, subgroup, weighting_attribute=None)
```

6.1.10 pysubgroup.refinement_operator module

```
class RefinementOperator
class StaticGeneralizationOperator(selectors)

    refinements(sG)
class StaticSpecializationOperator(selectors)

    refinements(subgroup)
```

6.1.11 pysubgroup.representations module

```
class BitSetRepresentation(df, selectors_to_patch)

    Conjunction
        alias of BitSet_Conjunction
    Disjunction
        alias of BitSet_Disjunction
    patch_classes()
    patch_selector(sel)
class BitSet_Conjunction(*args, **kwargs)

    append_and(to_append)
    compute_representation()
    n_instances = 0
    property size
class BitSet_Disjunction(*args, **kwargs)

    append_or(to_append)
    compute_representation()
    property size
class NumpySetRepresentation(df, selectors_to_patch)

    Conjunction
        alias of NumpySet_Conjunction
```



```
    patch_classes()
    patch_selector(sel)
class NumpySet_Conjunction(*args, **kwargs)

    all_set = None
    append_and(to_append)
    compute_representation()
    property size
class RepresentationBase(new_conjunction, selectors_to_patch)

    patch_all_selectors()
    patch_classes()
    patch_selector(sel)
    undo_patch_classes()
class SetRepresentation(df, selectors_to_patch)

    Conjunction
        alias of Set_Conjunction
    patch_classes()
    patch_selector(sel)
class Set_Conjunction(*args, **kwargs)

    all_set = {}
    append_and(to_append)
    compute_representation()
    property size
```

6.1.12 pysubgroup.subgroup module

Created on 28.04.2016

@author: lemmerfn

```
class EqualitySelector(attribute_name, attribute_value, selector_name=None)

    property attribute_name
    property attribute_value
    classmethod compute_descriptions(attribute_name, attribute_value, selector_name)
    covers(data)
    set_descriptions(attribute_name, attribute_value, selector_name=None)
```

```
class IntervalSelector (attribute_name, lower_bound, upper_bound, selector_name=None)

    property attribute_name
    classmethod compute_descriptions (attribute_name, lower_bound, upper_bound, selector_name=None)
    classmethod compute_string (attribute_name, lower_bound, upper_bound, rounding_digits)
    covers (data_instance)
    property lower_bound
    set_descriptions (attribute_name, lower_bound, upper_bound, selector_name=None)
    property upper_bound

class NegatedSelector (selector)

    property attribute_name
    covers (data_instance)
    set_descriptions (selector)

class SelectorBase

    abstract set_descriptions (*args, **kwargs)

class Subgroup (target, subgroup_description)

    calculate_statistics (data, weighting_attribute=None)
    covers (instance)
    get_base_statistics (data, weighting_attribute=None)
create_nominal_selectors (data, ignore=None)
create_nominal_selectors_for_attribute (data, attribute_name, dtypes=None)
create_numeric_selector_for_attribute (data, attr_name, nbins=5, intervals_only=True, weighting_attribute=None)
create_numeric_selectors (data, nbins=5, intervals_only=True, weighting_attribute=None, ignore=None)
create_selectors (data, nbins=5, intervals_only=True, ignore=None)
remove_target_attributes (selectors, target)
```

6.1.13 pysubgroup.utils module

Created on 02.05.2016

@author: lemmerfn

```
class SubgroupDiscoveryResult (results, task)

    supportSetVisualization (in_order=True, drop_empty=True)
    to_dataframe (include_info=False)
```

```

    to_descriptions()

    to_subgroups()

add_if_required(result, sg, quality, task, check_for_duplicates=False)

as_df(data, result, statistics_to_show=('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'),
    autoround=False, weighting_attribute=None, include_target=False)

conditional_invert(val, invert)

count_bits(bitset_as_int)

effective_sample_size(weights)

equal_frequency_discretization(data, attribute_name, nbins=5, weighting_attribute=None)

find_set_bits(bitset_as_int)

float_formatter(x, digits=2)

intersect_of_ordered_list(list_1, list_2)

is_categorical_attribute(data, attribute_name)

is_numerical_attribute(data, attribute_name)

minimum_required_quality(result, task)

overlap(sg, another_sg, data)

perc_formatter(x)

powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)

print_result_set(data, result, statistics_to_show, weighting_attribute=None, print_header=True, include_target=False)

remove_selectors_with_attributes(selector_list, attribute_list)

result_as_table(data, result, statistics_to_show, weighting_attribute=None, print_header=True, include_target=False)

results_as_df(data, result, statistics_to_show=('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'), autoround=False, weighting_attribute=None, include_target=False)

results_df_autoround(df)

to_bits(list_of_ints)

to_latex(data, result, statistics_to_show)

```

6.1.14 pysubgroup.visualization module

compare_distributions_numeric (*sgs, data, bins*)

plot_distribution_numeric (*sg, data, bins*)

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

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

plot_sgbars (*result_df, _, ylabel='target share', title='Discovered Subgroups', dynamic_widths=False, _suffix=""*)

similarity_dendrogram (*result, data*)

similarity_sgs (*sgd_results, data, color=True*)

6.1.15 Module contents

INDICES AND TABLES

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

PYTHON MODULE INDEX

p

- [pysubgroup](#), 32
- [pysubgroup.algorithms](#), 21
- [pysubgroup.boolean_expressions](#), 22
- [pysubgroup.fi_target](#), 23
- [pysubgroup.gp_growth](#), 24
- [pysubgroup.measures](#), 24
- [pysubgroup.model_target](#), 26
- [pysubgroup.numeric_target](#), 27
- [pysubgroup.refinement_operator](#), 28
- [pysubgroup.representations](#), 28
- [pysubgroup.subgroup](#), 29
- [pysubgroup.utils](#), 30
- [pysubgroup.visualization](#), 32

A

AbstractInterestingnessMeasure (class in *py-subgroup.measures*), 24
 add_if_required() (in module *pysubgroup.utils*), 31
 all_set (NumpySet_Conjunction attribute), 29
 all_set (Set_Conjunction attribute), 29
 append_and() (BitSet_Conjunction method), 28
 append_and() (BooleanExpressionBase method), 22
 append_and() (Conjunction method), 22
 append_and() (Disjunction method), 22
 append_and() (DNF method), 22
 append_and() (NumpySet_Conjunction method), 29
 append_and() (Set_Conjunction method), 29
 append_or() (BitSet_Disjunction method), 28
 append_or() (BooleanExpressionBase method), 22
 append_or() (Conjunction method), 22
 append_or() (Disjunction method), 22
 append_or() (DNF method), 22
 Apriori (class in *pysubgroup.algorithms*), 21
 AreaQF (class in *pysubgroup.fi_target*), 23
 as_df() (in module *pysubgroup.utils*), 31
 attribute_name() (EqualitySelector property), 29
 attribute_name() (IntervalSelector property), 30
 attribute_name() (NegatedSelector property), 30
 attribute_value() (EqualitySelector property), 29

B

BeamSearch (class in *pysubgroup.algorithms*), 21
 BestFirstSearch (class in *pysubgroup.algorithms*), 21
 beta() (beta_tuple property), 26
 beta_tuple (class in *pysubgroup.model_target*), 26
 BitSet_Conjunction (class in *pysub-group.representations*), 28
 BitSet_Disjunction (class in *pysub-group.representations*), 28
 BitSetRepresentation (class in *pysub-group.representations*), 28
 BooleanExpressionBase (class in *pysub-group.boolean_expressions*), 22

BoundedInterestingnessMeasure (class in *py-subgroup.measures*), 24

C

calculate_constant_statistics() (EMM_Likelihood method), 26
 calculate_constant_statistics() (GeneralizationAwareQF method), 25
 calculate_constant_statistics() (GeneralizationAwareQF_stats method), 25
 calculate_constant_statistics() (PolyRegression_ModelClass method), 26
 calculate_constant_statistics() (SimpleCountQF method), 23
 calculate_constant_statistics() (StandardQFNumeric method), 27
 calculate_constant_statistics() (StandardQFNumeric.Average_Estimator method), 27
 calculate_constant_statistics() (StandardQFNumeric.Ordering_Estimator method), 27
 calculate_constant_statistics() (StandardQFNumeric.Summation_Estimator method), 27
 calculate_quality_function_for_patterns() (GpGrowth method), 24
 calculate_statistics() (CountCallsInterestingness_Measure method), 25
 calculate_statistics() (EMM_Likelihood method), 26
 calculate_statistics() (FITarget method), 23
 calculate_statistics() (GeneralizationAwareQF method), 25
 calculate_statistics() (GeneralizationAwareQF_stats method), 25
 calculate_statistics() (NumericTarget method), 27
 calculate_statistics() (SimpleCountQF method), 23
 calculate_statistics() (StandardQFNumeric method), 27

`calculate_statistics()` (*Subgroup method*), 30
`check_constraints()` (*GpGrowth method*), 24
`CombinedInterestingnessMeasure` (*class in pysubgroup.measures*), 24
`compare_distributions_numeric()` (*in module pysubgroup.visualization*), 32
`compute_descriptions()` (*EqualitySelector class method*), 29
`compute_descriptions()` (*IntervalSelector class method*), 30
`compute_representation()` (*BitSet_Conjunction method*), 28
`compute_representation()` (*BitSet_Disjunction method*), 28
`compute_representation()` (*NumpySet_Conjunction method*), 29
`compute_representation()` (*Set_Conjunction method*), 29
`compute_string()` (*IntervalSelector class method*), 30
`conditional_invert()` (*in module pysubgroup.utils*), 31
`Conjunction` (*BitSetRepresentation attribute*), 28
`Conjunction` (*class in pysubgroup.boolean_expressions*), 22
`Conjunction` (*NumpySetRepresentation attribute*), 28
`Conjunction` (*SetRepresentation attribute*), 29
`count_bits()` (*in module pysubgroup.utils*), 31
`CountCallsInterestingMeasure` (*class in pysubgroup.measures*), 25
`CountQF` (*class in pysubgroup.fi_target*), 23
`covers()` (*Conjunction method*), 22
`covers()` (*Disjunction method*), 22
`covers()` (*EqualitySelector method*), 29
`covers()` (*IntervalSelector method*), 30
`covers()` (*NegatedSelector method*), 30
`covers()` (*Subgroup method*), 30
`create_copy_of_path()` (*GpGrowth method*), 24
`create_copy_of_tree_top_down()` (*GpGrowth method*), 24
`create_new_tree_from_nodes()` (*GpGrowth method*), 24
`create_nominal_selectors()` (*in module pysubgroup.subgroup*), 30
`create_nominal_selectors_for_attribute()` (*in module pysubgroup.subgroup*), 30
`create_numeric_selector_for_attribute()` (*in module pysubgroup.subgroup*), 30
`create_numeric_selectors()` (*in module pysubgroup.subgroup*), 30
`create_selectors()` (*in module pysubgroup.subgroup*), 30

D

`depth()` (*Conjunction property*), 22
`DFS` (*class in pysubgroup.algorithms*), 21
`DFSNumeric` (*class in pysubgroup.algorithms*), 21
`Disjunction` (*BitSetRepresentation attribute*), 28
`Disjunction` (*class in pysubgroup.boolean_expressions*), 22
`DNF` (*class in pysubgroup.boolean_expressions*), 22

E

`effective_sample_size()` (*in module pysubgroup.utils*), 31
`EMM_Likelihood` (*class in pysubgroup.model_target*), 26
`ensure_statistics()` (*AbstractInterestingnessMeasure method*), 24
`equal_frequency_discretization()` (*in module pysubgroup.utils*), 31
`EqualitySelector` (*class in pysubgroup.subgroup*), 29
`evaluate()` (*AreaQF method*), 23
`evaluate()` (*CountQF method*), 23
`evaluate()` (*EMM_Likelihood method*), 26
`evaluate()` (*GeneralizationAwareQF method*), 25
`evaluate()` (*GeneralizationAwareQF_stats method*), 25
`evaluate()` (*StandardQFNumeric method*), 27
`evaluate_from_dataset()` (*CombinedInterestingnessMeasure method*), 24
`evaluate_from_dataset()` (*GASandardQFNumeric method*), 27
`evaluate_from_statistics()` (*CombinedInterestingnessMeasure method*), 24
`execute()` (*Apriori method*), 21
`execute()` (*BeamSearch method*), 21
`execute()` (*BestFirstSearch method*), 21
`execute()` (*DFS method*), 21
`execute()` (*DFSNumeric method*), 21
`execute()` (*GeneralisingBFS method*), 22
`execute()` (*GpGrowth method*), 24
`execute()` (*SimpleDFS method*), 22
`execute()` (*SimpleSearch method*), 22

F

`find_set_bits()` (*in module pysubgroup.utils*), 31
`fit()` (*PolyRegression_ModelClass method*), 26
`FITarget` (*class in pysubgroup.fi_target*), 23
`float_formatter()` (*in module pysubgroup.utils*), 31

G

`ga_tuple` (*GeneralizationAwareQF_stats attribute*), 25
`GASandardQFNumeric` (*class in pysubgroup.numeric_target*), 27

`generalisation_quality()` (*GeneralizationAwareQF.ga_tuple property*), 25
`GeneralisingBFS` (class in *pysubgroup.algorithms*), 22
`GeneralizationAwareQF` (class in *pysubgroup.measures*), 25
`GeneralizationAwareQF.ga_tuple` (class in *pysubgroup.measures*), 25
`GeneralizationAwareQF_stats` (class in *pysubgroup.measures*), 25
`get_attributes()` (*FITarget method*), 23
`get_attributes()` (*NumericTarget method*), 27
`get_base_statistics()` (*FITarget method*), 23
`get_base_statistics()` (*NumericTarget method*), 27
`get_base_statistics()` (*Subgroup method*), 30
`get_data()` (*StandardQFNumeric.Average_Estimator method*), 27
`get_data()` (*StandardQFNumeric.Ordering_Estimator method*), 27
`get_data()` (*StandardQFNumeric.Summation_Estimator method*), 27
`get_estimate()` (*StandardQFNumeric.Average_Estimator method*), 27
`get_estimate()` (*StandardQFNumeric.Ordering_Estimator method*), 27
`get_estimate()` (*StandardQFNumeric.Summation_Estimator method*), 27
`get_estimate_numpy()` (*StandardQFNumeric.Ordering_Estimator method*), 27
`get_max()` (*GeneralizationAwareQF_stats method*), 25
`get_max_generalization_mean()` (in module *pysubgroup.numeric_target*), 28
`get_next_level()` (*Apriori method*), 21
`get_next_level_candidates()` (*Apriori method*), 21
`get_next_level_candidates_vectorized()` (*Apriori method*), 21
`get_next_level_numba()` (*Apriori method*), 21
`get_nodes_upwards()` (*GpGrowth method*), 24
`get_prefixes_top_down()` (*GpGrowth method*), 24
`get_qual_and_previous_qual()` (*GeneralizationAwareQF method*), 25
`get_stats_and_previous_stats()` (*GeneralizationAwareQF_stats method*), 25
`get_stats_for_class()` (*GpGrowth method*), 24
`get_top_down_tree_for_class()` (*GpGrowth method*), 24
`get_tuple()` (*EMM_Likelihood method*), 26
`gp_get_null_vector()` (*PolyRegression_ModelClass method*), 26
`gp_get_null_vector()` (*SimpleCountQF method*), 23
`gp_get_params()` (*EMM_Likelihood method*), 26
`gp_get_params()` (*PolyRegression_ModelClass method*), 26
`gp_get_params()` (*SimpleCountQF method*), 23
`gp_get_stats()` (*PolyRegression_ModelClass method*), 26
`gp_get_stats()` (*SimpleCountQF method*), 23
`gp_merge()` (*PolyRegression_ModelClass static method*), 26
`gp_merge()` (*SimpleCountQF method*), 23
`gp_requires_cover_arr()` (*SimpleCountQF property*), 23
`gp_to_str()` (*SimpleCountQF method*), 23
`GpGrowth` (class in *pysubgroup.gp_growth*), 24
I
`insert_into_tree()` (*GpGrowth method*), 24
`intersect_of_ordered_list()` (in module *pysubgroup.utils*), 31
`IntervalSelector` (class in *pysubgroup.subgroup*), 29
`is_applicable()` (*AbstractInterestingnessMeasure method*), 24
`is_applicable()` (*AreaQF method*), 23
`is_applicable()` (*CombinedInterestingnessMeasure method*), 25
`is_applicable()` (*CountCallsInterestingnessMeasure method*), 25
`is_applicable()` (*CountQF method*), 23
`is_applicable()` (*EMM_Likelihood method*), 26
`is_applicable()` (*GASstandardQFNumeric method*), 27
`is_applicable()` (*GeneralizationAwareQF method*), 25
`is_applicable()` (*GeneralizationAwareQF_stats method*), 25
`is_applicable()` (*StandardQFNumeric method*), 27
`is_categorical_attribute()` (in module *pysubgroup.utils*), 31
`is_numerical_attribute()` (in module *pysubgroup.utils*), 31
L
`likelihood()` (*PolyRegression_ModelClass method*), 26
`loglikelihood()` (*PolyRegression_ModelClass method*), 26
`lower_bound()` (*IntervalSelector property*), 30
M
`maximum_statistic_filter()` (in module *pysubgroup.measures*), 25
`merge_trees_top_down()` (*GpGrowth method*), 24

`minimum_quality_filter()` (in module `pysubgroup.measures`), 25
`minimum_required_quality()` (in module `pysubgroup.utils`), 31
`minimum_statistic_filter()` (in module `pysubgroup.measures`), 25

N

`n_instances` (*BitSet_Conjunction* attribute), 28
`NegatedSelector` (class in `pysubgroup.subgroup`), 30
`nodes_to_cls_nodes()` (*GpGrowth* method), 24
`normal_insert()` (*GpGrowth* method), 24
`NumericTarget` (class in `pysubgroup.numeric_target`), 27
`NumpySet_Conjunction` (class in `pysubgroup.representations`), 29
`NumpySetRepresentation` (class in `pysubgroup.representations`), 28

O

`optimistic_estimate()` (*CountQF* method), 23
`optimistic_estimate()` (*StandardQFNumeric* method), 27
`optimistic_estimate_from_dataset()` (*CombinedInterestingnessMeasure* method), 25
`optimistic_estimate_from_statistics()` (*CombinedInterestingnessMeasure* method), 25
`overlap()` (in module `pysubgroup.utils`), 31
`overlap_filter()` (in module `pysubgroup.measures`), 26
`overlaps_list()` (in module `pysubgroup.measures`), 26

P

`patch_all_selectors()` (*RepresentationBase* method), 29
`patch_classes()` (*BitSetRepresentation* method), 28
`patch_classes()` (*NumpySetRepresentation* method), 28
`patch_classes()` (*RepresentationBase* method), 29
`patch_classes()` (*SetRepresentation* method), 29
`patch_selector()` (*BitSetRepresentation* method), 28
`patch_selector()` (*NumpySetRepresentation* method), 29
`patch_selector()` (*RepresentationBase* method), 29
`patch_selector()` (*SetRepresentation* method), 29
`perc_formatter()` (in module `pysubgroup.utils`), 31
`plot_distribution_numeric()` (in module `pysubgroup.visualization`), 32
`plot_npspace()` (in module `pysubgroup.visualization`), 32

`plot_roc()` (in module `pysubgroup.visualization`), 32
`plot_sgbars()` (in module `pysubgroup.visualization`), 32
`PolyRegression_ModelClass` (class in `pysubgroup.model_target`), 26
`pop_and()` (*Conjunction* method), 22
`pop_and()` (*DNF* method), 22
`pop_or()` (*Conjunction* method), 22
`powerset()` (in module `pysubgroup.utils`), 31
`prepare_selectors()` (*GpGrowth* method), 24
`print_result_set()` (in module `pysubgroup.utils`), 31
`pysubgroup` (module), 32
`pysubgroup.algorithms` (module), 21
`pysubgroup.boolean_expressions` (module), 22
`pysubgroup.fi_target` (module), 23
`pysubgroup.gp_growth` (module), 24
`pysubgroup.measures` (module), 24
`pysubgroup.model_target` (module), 26
`pysubgroup.numeric_target` (module), 27
`pysubgroup.refinement_operator` (module), 28
`pysubgroup.representations` (module), 28
`pysubgroup.subgroup` (module), 29
`pysubgroup.utils` (module), 30
`pysubgroup.visualization` (module), 32

R

`recurse()` (*GpGrowth* method), 24
`recurse_top_down()` (*GpGrowth* method), 24
`RefinementOperator` (class in `pysubgroup.refinement_operator`), 28
`refinements()` (*StaticGeneralizationOperator* method), 28
`refinements()` (*StaticSpecializationOperator* method), 28
`remove_infrequent_class()` (*GpGrowth* method), 24
`remove_infrequent_nodes()` (*GpGrowth* method), 24
`remove_selectors_with_attributes()` (in module `pysubgroup.utils`), 31
`remove_target_attributes()` (in module `pysubgroup.subgroup`), 30
`RepresentationBase` (class in `pysubgroup.representations`), 29
`reprune_lower_levels()` (*Apriori* method), 21
`result_as_table()` (in module `pysubgroup.utils`), 31
`results_as_df()` (in module `pysubgroup.utils`), 31
`results_df_around()` (in module `pysubgroup.utils`), 31

S

`search_internal()` (*DFS method*), 21
`search_internal()` (*DFSNumeric method*), 21
`search_internal()` (*SimpleDFS method*), 22
`SelectorBase` (*class in pysubgroup.subgroup*), 30
`Set_Conjunction` (*class in pysubgroup.representations*), 29
`set_descriptions()` (*EqualitySelector method*), 29
`set_descriptions()` (*IntervalSelector method*), 30
`set_descriptions()` (*NegatedSelector method*), 30
`set_descriptions()` (*SelectorBase method*), 30
`SetRepresentation` (*class in pysubgroup.representations*), 29
`similarity_dendrogram()` (*in module pysubgroup.visualization*), 32
`similarity_sgs()` (*in module pysubgroup.visualization*), 32
`SimpleCountQF` (*class in pysubgroup.fi_target*), 23
`SimpleDFS` (*class in pysubgroup.algorithms*), 22
`SimpleSearch` (*class in pysubgroup.algorithms*), 22
`size()` (*beta_tuple property*), 26
`size()` (*BitSet_Conjunction property*), 28
`size()` (*BitSet_Disjunction property*), 28
`size()` (*NumpySet_Conjunction property*), 29
`size()` (*Set_Conjunction property*), 29
`standard_qf_numeric()` (*StandardQFNumeric static method*), 27
`StandardQFNumeric` (*class in pysubgroup.numeric_target*), 27
`StandardQFNumeric.Average_Estimator` (*class in pysubgroup.numeric_target*), 27
`StandardQFNumeric.Ordering_Estimator` (*class in pysubgroup.numeric_target*), 27
`StandardQFNumeric.Summation_Estimator` (*class in pysubgroup.numeric_target*), 27
`StaticGeneralizationOperator` (*class in pysubgroup.refinement_operator*), 28
`StaticSpecializationOperator` (*class in pysubgroup.refinement_operator*), 28
`Subgroup` (*class in pysubgroup.subgroup*), 30
`subgroup_quality()` (*GeneralizationAwareQF.ga_tuple property*), 25
`SubgroupDiscoveryResult` (*class in pysubgroup.utils*), 30
`SubgroupDiscoveryTask` (*class in pysubgroup.algorithms*), 22
`supports_weights()` (*AbstractInterestingnessMeasure method*), 24
`supports_weights()` (*AreaQF method*), 23
`supports_weights()` (*CombinedInterestingness-Measure method*), 25
`supports_weights()` (*CountCallsInterestingMeasure method*), 25
`supports_weights()` (*CountQF method*), 23

`supports_weights()` (*EMM_Likelihood method*), 26
`supports_weights()` (*GAStandardQFNumeric method*), 27
`supports_weights()` (*GeneralizationAwareQF method*), 25
`supports_weights()` (*GeneralizationAwareQF_stats method*), 25
`supports_weights()` (*StandardQFNumeric method*), 28
`supportSetVisualization()` (*SubgroupDiscoveryResult method*), 30

T

`to_bits()` (*in module pysubgroup.utils*), 31
`to_dataframe()` (*SubgroupDiscoveryResult method*), 30
`to_descriptions()` (*SubgroupDiscoveryResult method*), 30
`to_file()` (*GpGrowth method*), 24
`to_latex()` (*in module pysubgroup.utils*), 31
`to_subgroups()` (*SubgroupDiscoveryResult method*), 31
`tpl` (*DFSNumeric attribute*), 21
`tpl` (*EMM_Likelihood attribute*), 26
`tpl` (*SimpleCountQF attribute*), 23
`tpl` (*StandardQFNumeric attribute*), 28

U

`undo_patch_classes()` (*RepresentationBase method*), 29
`unique_attributes()` (*in module pysubgroup.measures*), 26
`upper_bound()` (*IntervalSelector property*), 30