senstivity_analysis

February 6, 2021

0.1 Background about GR system simulation wrapper

Our GR system (`gr_ipc.jar`) is a goal recognition tool which can return a CSV file contains all the inference results of a certain domain with a specific parameter settings. The inference results are divided into 5 groups based on observation percentage (10%, 30%, 50%, 70% & 100%).

For example, in `blocks-world` 10% group, there are multiple problems (testing cases) to test how well our GR system can perform in `blocks-world` domain if we only observe 10% actions. Hence, our GR system will also return multiple results for this 10% group. Having the multiple results, we can calculate average precision for 10% group (`p_10`), average recall for 10% group (`r_10`) and average accuracy for 10% group (`a_10`).

Similarly, we also calculate `p_30`, `r_30`, `a_30`, `p_50`, `r_50`, `a_50`, `p_70`, `r_70`, `a_70`, `p_100`, `r_100`, `a_100` and `p_avg`, `r_avg`, `a_avg` (notice `_avg` means the average over all 10%, 30%, 50%, 70% and 100% groups). For a certain domain with a certain parameter settings, we can calculate all the statistics above.

Then we use the EMA-workbench to define a model to wrap up our GR system. The EMA-workbench provides simulation functions to randomly select parameters within pre-defined ranges and then run the GR system for many times (Simulation). It will collect performance statistics (`p_10`, `r_10`, `a_10`, `p_30`, `r_30`, `a_30`, `p_50`, `r_50`, `a_50`, `p_70`, `r_70`, `a_70`, `p_100`, `r_100`, `a_100`, `p_avg`, `r_avg`, `a_avg`) corresponding to each set of randomly selected parameters. The simulated performance statistics are stored in a `.tar.gz` files, it will be analysed by this notebook in following sections.

At this stage, the model (including uncertainties, levers, constants and outcomes) is defined in another python module `model.py`. This module is imported by another python program `experiment_simulator.py`.

Data: There are two directories `./topk/` and `./diverse/` to store plans generated by top-k and diverse planner correspondingly.

In either directory, the simulator will run 1000 scenarios on each domain using latin hypercude sampling, the experiment results are saved in `1000_<domain>.tar.gz`. And it will run 10500 scenarios (base number = 1050) on each domain using SOBOL sampling, the results are saved in `1050_<domain>.tar.gz`

In this notebook, we assume all experimental simulations are done and we have obtained all the results saved in `.tar.gz` files. So, we just load the stored data and do sensitivity analysis.

0.2 Requirements

Install the following requirements if they haven't been installed. Be careful with python version, we are using python3. Also be careful with `pip3`, some machine is just `pip`. If you found that some library can not be imported, just put `!` in the front and use `pip3` to install within the jupyter notebook environment.

`! pip3 install <package_name>`

Or

`! pip install <package_name>`

[2]:
```
# install packages within jupyter notebook
#!pip3 install numpy
#!pip3 install pandas
#!pip3 install seaborn
#!pip3 install matplotlib
#!pip3 install ema_workbench
#!pip3 install SALib
```

[3]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from ema_workbench import load_results
from ema_workbench.analysis import feature_scoring
from ema_workbench.analysis import pairs_plotting
from ema_workbench.analysis import prim
```

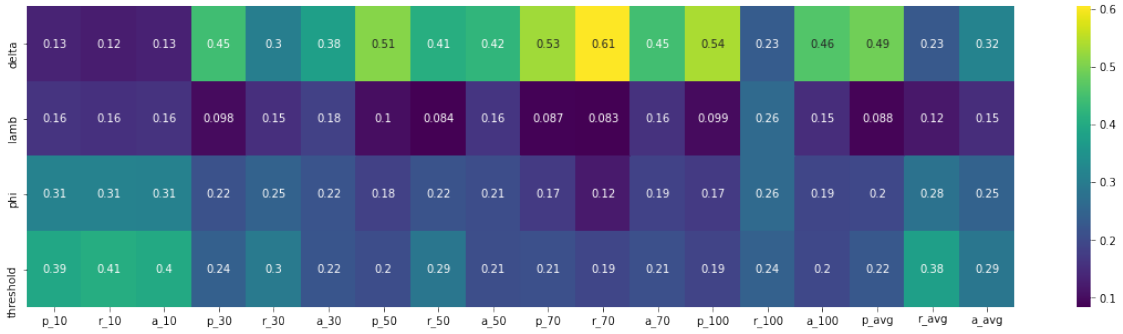
0.3 Show feature scores and plots

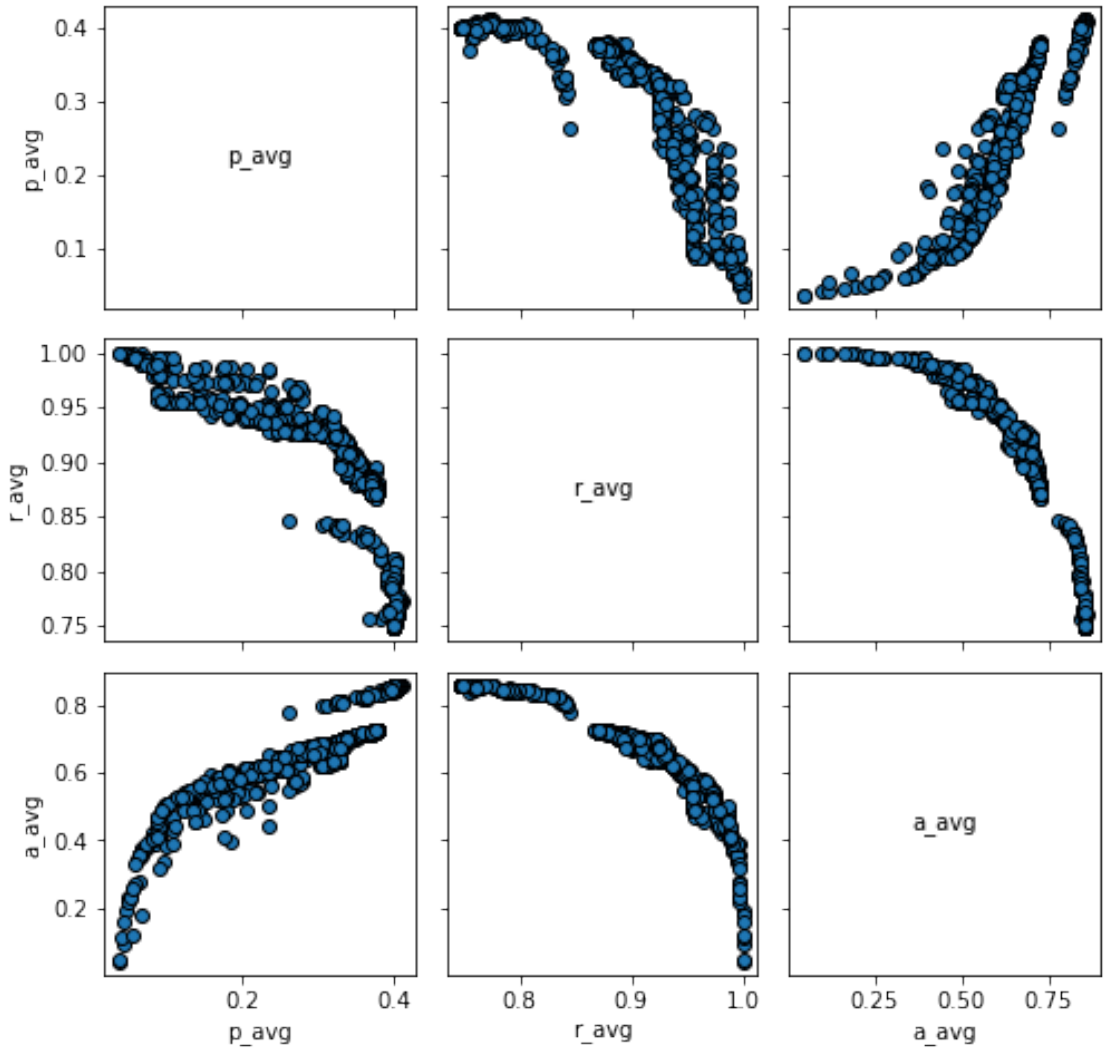
[4]:
```
# for feature score and plots
def show_feature_score(results):
    experiments, outcomes = results
    x = experiments
    y = outcomes

    fs = feature_scoring.get_feature_scores_all(x, y)
    plt.figure(figsize = (20,5))
    sns.heatmap(fs, cmap='viridis', annot=True)
    plt.show()

def show_points(results, label):
    experiments, outcomes = results

    # partial dict
    if label == "avg":
```

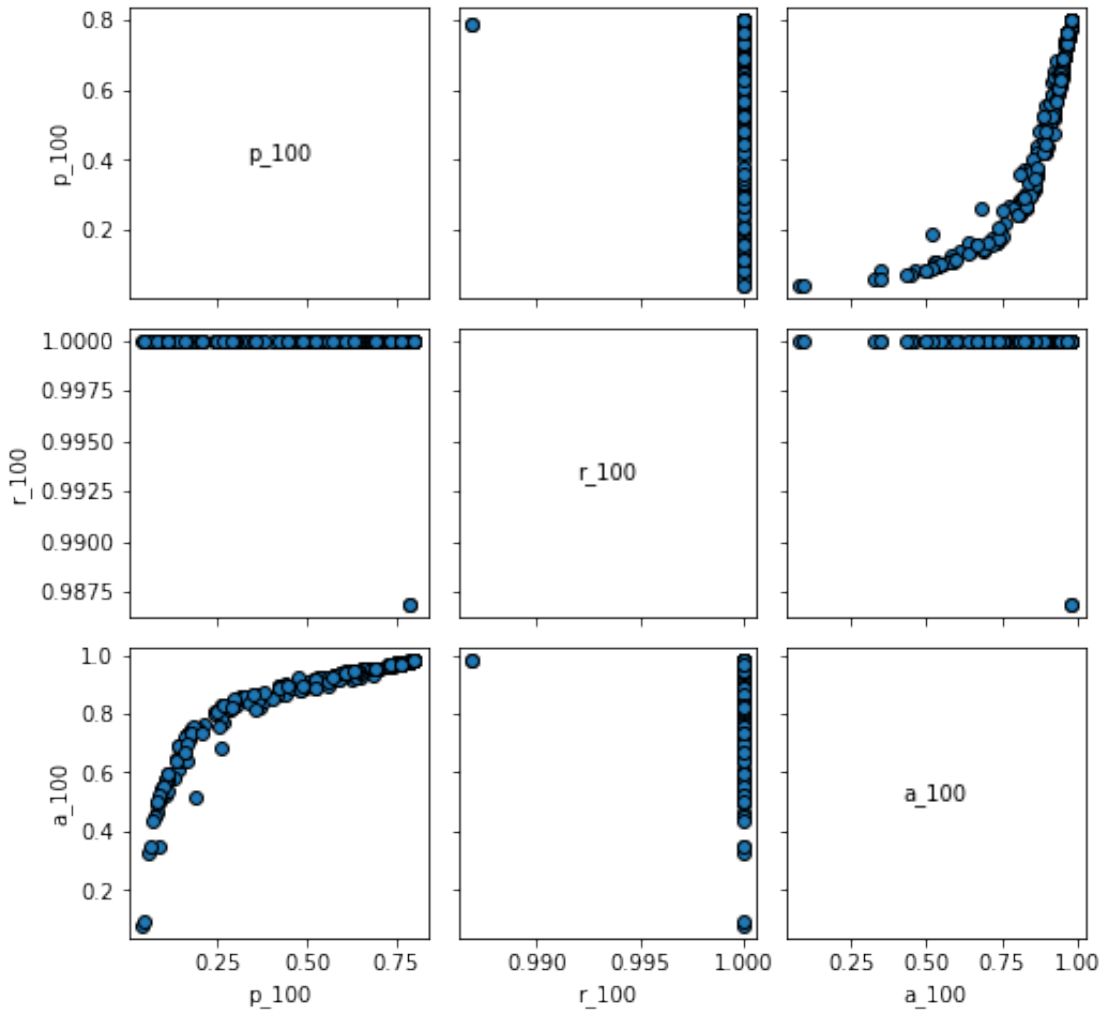
```
        partial_outcomes = dict((key,value) for key, value in outcomes.items()␣
↪if key in ["p_avg", "r_avg", "a_avg"])
    if label == "100":
        partial_outcomes = dict((key,value) for key, value in outcomes.items()␣
↪if key in ["p_100", "r_100", "a_100"])
    if label == "70":
        partial_outcomes = dict((key,value) for key, value in outcomes.items()␣
↪if key in ["p_70", "r_70", "a_70"])
    if label == "50":
        partial_outcomes = dict((key,value) for key, value in outcomes.items()␣
↪if key in ["p_50", "r_50", "a_50"])
    if label == "30":
        partial_outcomes = dict((key,value) for key, value in outcomes.items()␣
↪if key in ["p_30", "r_30", "a_30"])
    if label == "10":
        partial_outcomes = dict((key,value) for key, value in outcomes.items()␣
↪if key in ["p_10", "r_10", "a_10"])

    fig, axes = pairs_plotting.pairs_scatter(experiments, partial_outcomes,␣
↪group_by='policy',
                                             legend=False)
    fig.set_size_inches(8,8)
    plt.show()

def display_scores_and_plots(results):
    show_feature_score(results)
    show_points(results, "avg")
    show_points(results, "100")
    show_points(results, "70")
    show_points(results, "50")
    show_points(results, "30")
    show_points(results, "10")
```

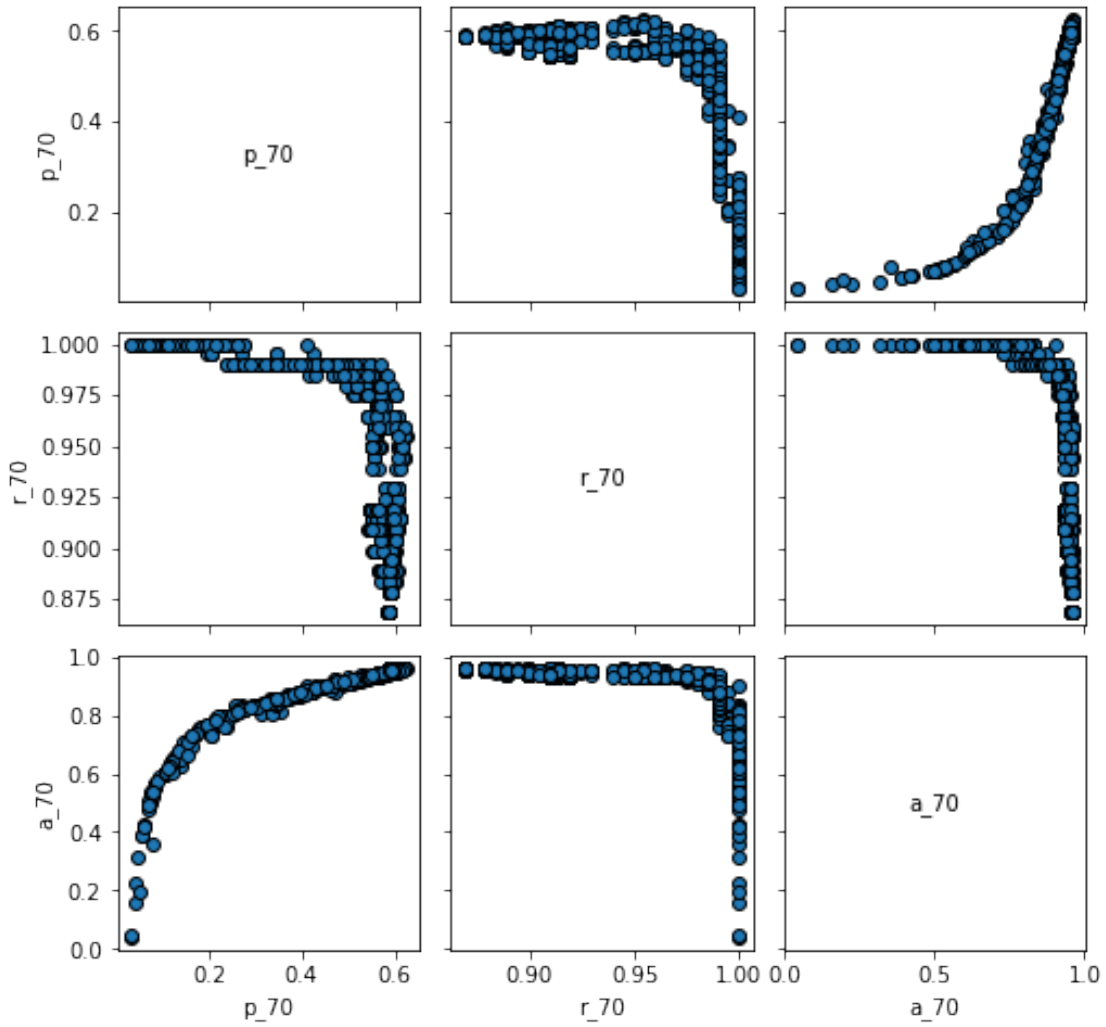
[3]:
```
# specify a domain: (select one of below)
domain_name = "blocks-world"
#domain_name = "campus"
#domain_name = "depots"
#domain_name = "driverlog"
#domain_name = "dwr"
#domain_name = "easy-ipc-grid"
#domain_name = "ferry"
#domain_name = "intrusion-detection"
#domain_name = "kitchen"
#domain_name = "logistics"
#domain_name = "miconic"
```

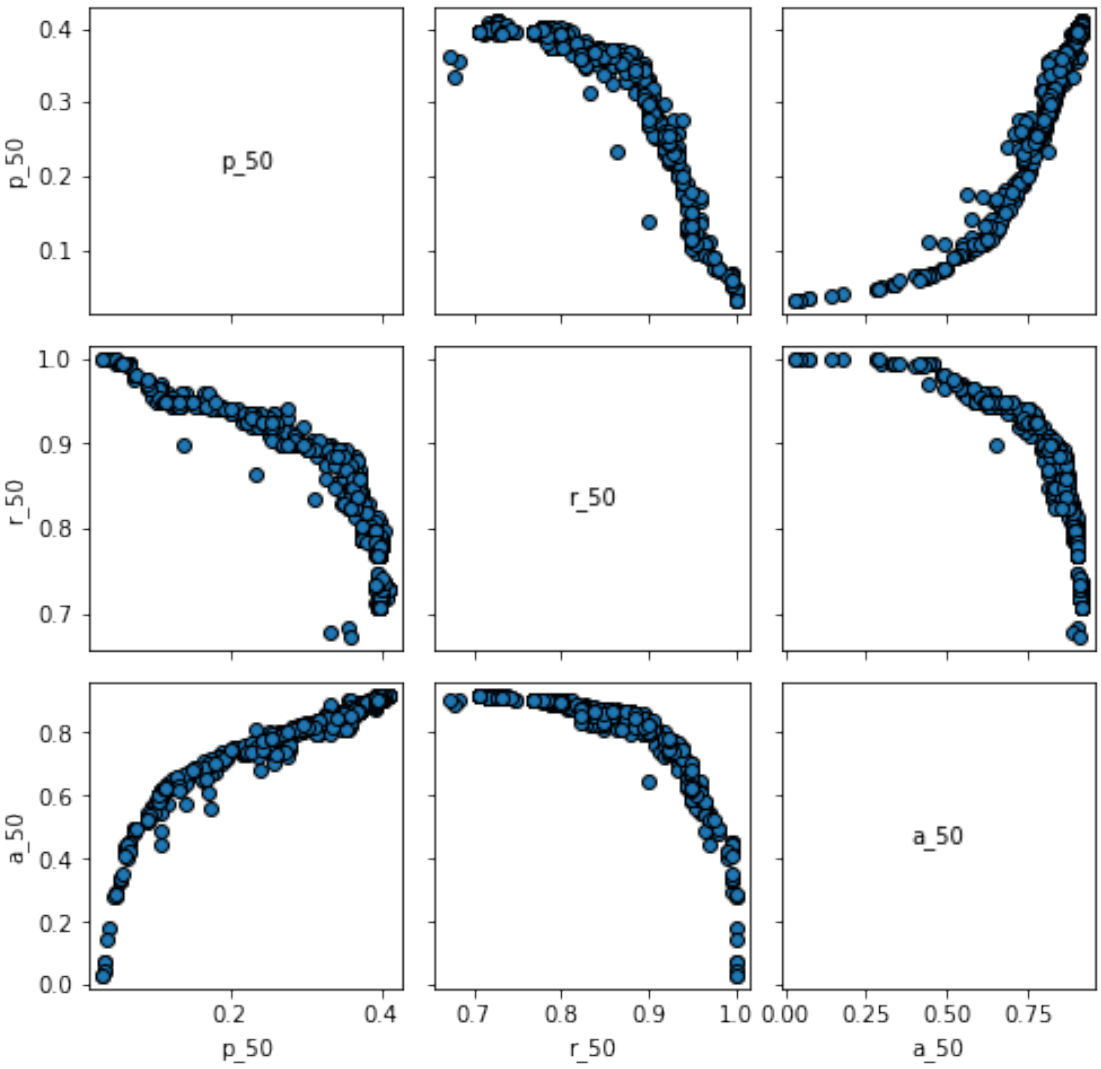
```
#domain_name = "rovers"
#domain_name = "satellite"
#domain_name = "sokoban"
#domain_name = "zeno-travel"

# load data
results = load_results('./topk/1000_' + domain_name + '.tar.gz')

# show feature score and plots
display_scores_and_plots(results)
```

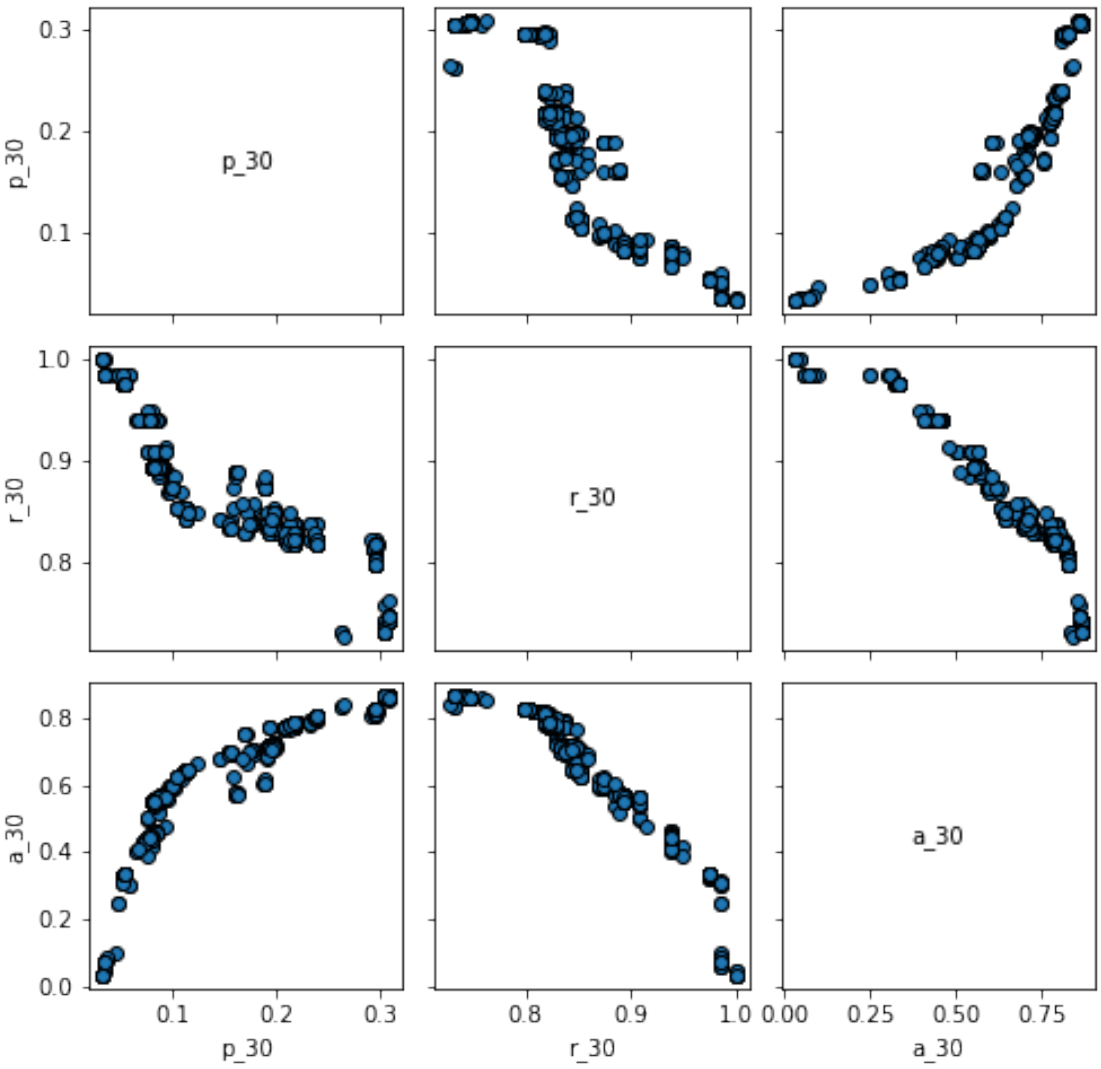


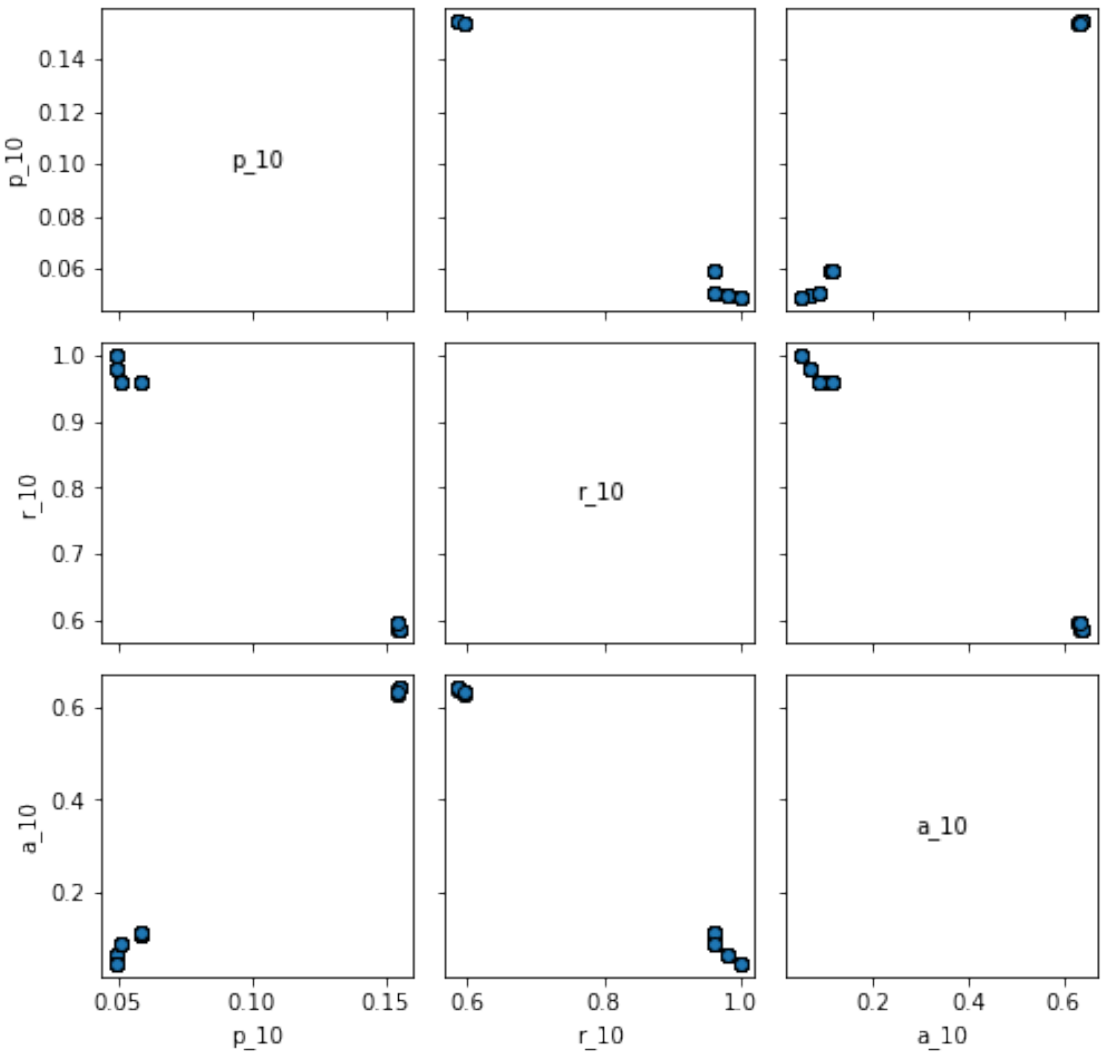

p_avg
r_avg
a_avg
0.4
0.3
0.2
0.1
1.00
0.95
0.90
0.85
0.80
0.75
0.8
0.6
0.4
0.2
0.2
0.4
0.8
0.9
1.0
0.25
0.50
0.75


p_100
r_100
a_100
0.8
0.6
0.4
0.2
1.0000
0.9975
0.9950
0.9925
0.9900
0.9875
1.0
0.25
0.50
0.75
0.990
0.995
1.000
1.00


p_70
r_70
a_70
0.6
0.4
0.2
1.000
0.975
0.950
0.925
0.900
0.875
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.90
0.95
1.00
0.0
0.5
1.0


p_50
r_50
a_50
0.4
0.3
0.2
0.1
1.0
0.9
0.8
0.7
0.6
0.4
0.2
0.0
0.2
0.4
0.7
0.8
0.9
1.0
0.00
0.25
0.50
0.75


p_30
r_30
a_30
0.3
0.2
0.1
1.0
0.9
0.8
0.6
0.4
0.0
0.25
0.50
0.75



0.4 PRIM Algorithm

Once we get simultion results stored in `.tar.gz` file. We can apply PRIM algorithm to show what ranges of parameters lead to better performance. Here, we used `a_avg` as a metric to filter out the experiments with top 10% average accuracy (`a_avg`). There are **two steps** for finding the parameter ranges.

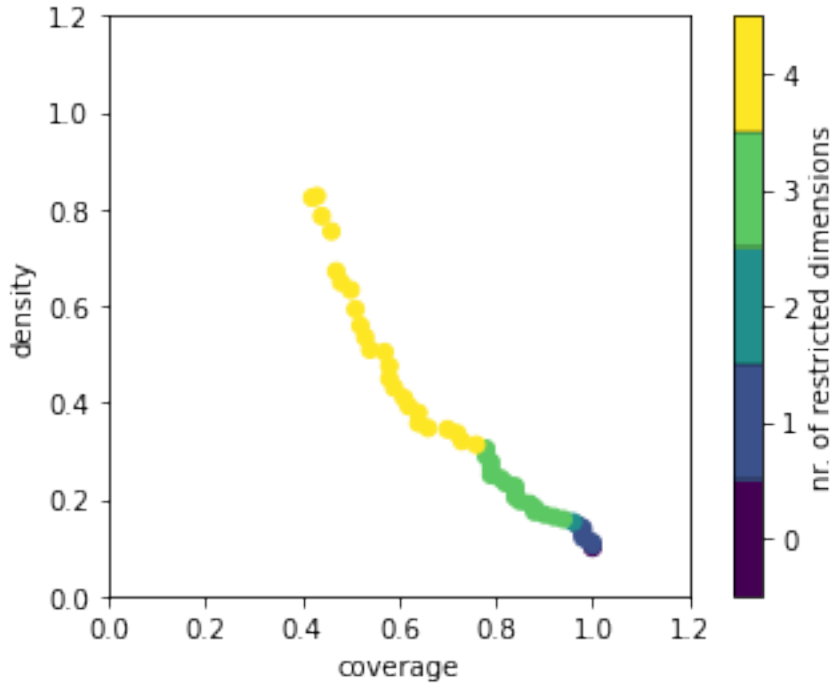
[5]:
```
# load functions for PRIM algorithm
def get_top(results, percentage, metric):
    experiments, outcomes = results
    experiments[metric] = outcomes[metric]
    num = int(len(experiments.index) * percentage)
    df = experiments.sort_values(by=[metric], ascending=False).head(num)
```

```
    '''
    # show the dataframe
    print(df)
    range_list = list(df["threshold"])
    range_list.sort()
    # print(range_list)
    print("threshold range: ", range_list[0], range_list[-1])

    range_list = list(df["lamb"])
    range_list.sort()
    print("lamb range: ", range_list[0], range_list[-1])

    range_list = list(df["delta"])
    range_list.sort()
    print("delta range: ", range_list[0], range_list[-1])

    range_list = list(df["phi"])
    range_list.sort()
    print("phi range: ", range_list[0], range_list[-1])
    '''
    return df.tail(1)["a_avg"]

def find_box(results, metric, lower_limit):
    experiments1, outcomes = results
    x = experiments1
    y = outcomes[metric] >= lower_limit

    prim_alg = prim.Prim(x, y, threshold=0.1)    #### need to select a proper␣
 ↪threshold here
    box1 = prim_alg.find_box()

    box1.show_tradeoff()
    plt.show()
    # lim = prim_alg.limits()

    return box1

def show_ranges(box_num, box1):
    box1.inspect(box_num)
    box1.inspect(box_num, style='graph')
    plt.show()

    box1.show_pairs_scatter(box_num)
    plt.show()
```

Step 1: show the trade-off graph The trade off graph as below. It contains many dots from the dark blue to yellow. We can inspect each dot by giving a `box_num` in **step two**. The right-hand

side bar indicates number of parameters (phi, theta, lambda, threshold). The graph shows the trade-off between coverage and denstiy. So, when we set a `box_num`, we also need to consider the trade-off. The `box_num` starts from 0, which means the highest coverage but lowest density. The upper bound of `box_num` is different from domain to domain, usually around 50-60. If you set a `box_num` too large, it will give you errors, then you need to adjust it to some smaller number.

[6]:
```
domain_name = "blocks-world"

# select the best 10% rows (experiments) with highest average accuracy
results = load_results('./topk/1000_' + domain_name + '.tar.gz')
a_avg = get_top(results, 0.1, "a_avg")

#results of PRIM algorithm (need to reload the data)
results = load_results('./topk/1000_' + domain_name + '.tar.gz')
box1 = find_box(results, 'a_avg', float(a_avg))
```



Step 2: set a proper box_num (need to consider the trade off between density and coverage) In this step, we need to find a proper index of `box_num`. Our we have two functions for finding the `box_num`.

1. significance oriented: `box_num` starts from 0 (highest coverage), then iterate to the last index `box_num` = n. For each iteration, if the qp_values for all parameters are less then the significance threshold (0.05 as default), we will stop the loop and return that `box_num` index, parameter ranges and qp_values. (this method usually give us very large index)
2. trade-off oriented: in this method, we don't want to be very aggressive to get large index

(high density, but low coverage), we want coverage around 0.8 or we want to find the median of index.

Nocite: you can check more detailed information to understand the logic, for example, `box1.qp` can access the qp_values of all boxes; `getattr(box1,'box_lims')` can get all parameter ranges for all domains.

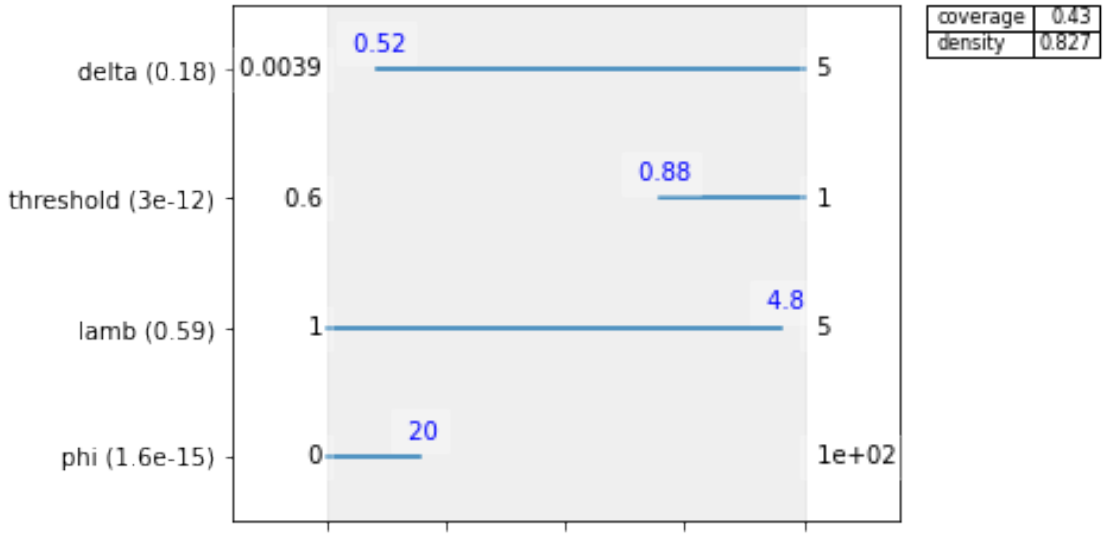
[7]:
```
# find ranges:
def fine_valid_range(box1, significance = 0.05):
    param_range = getattr(box1,'box_lims')
    box_length = len(param_range)
    store = None

    for i in range(1, box_length):
        qp_dict = box1.qp[i]
        valid_params = []
        for key in qp_dict.keys():
            valid = False
            if qp_dict[key][0] != -1 and qp_dict[key][0] < significance:
                valid = True

            if qp_dict[key][1] != -1 and qp_dict[key][1] < significance:
                valid = True

            if valid:
                valid_params.append(key)

        store = [i, param_range[i], qp_dict]
        if len(valid_params) == 4:
            break

    return store

valid_ranges = fine_valid_range(box1)
valid_ranges
```

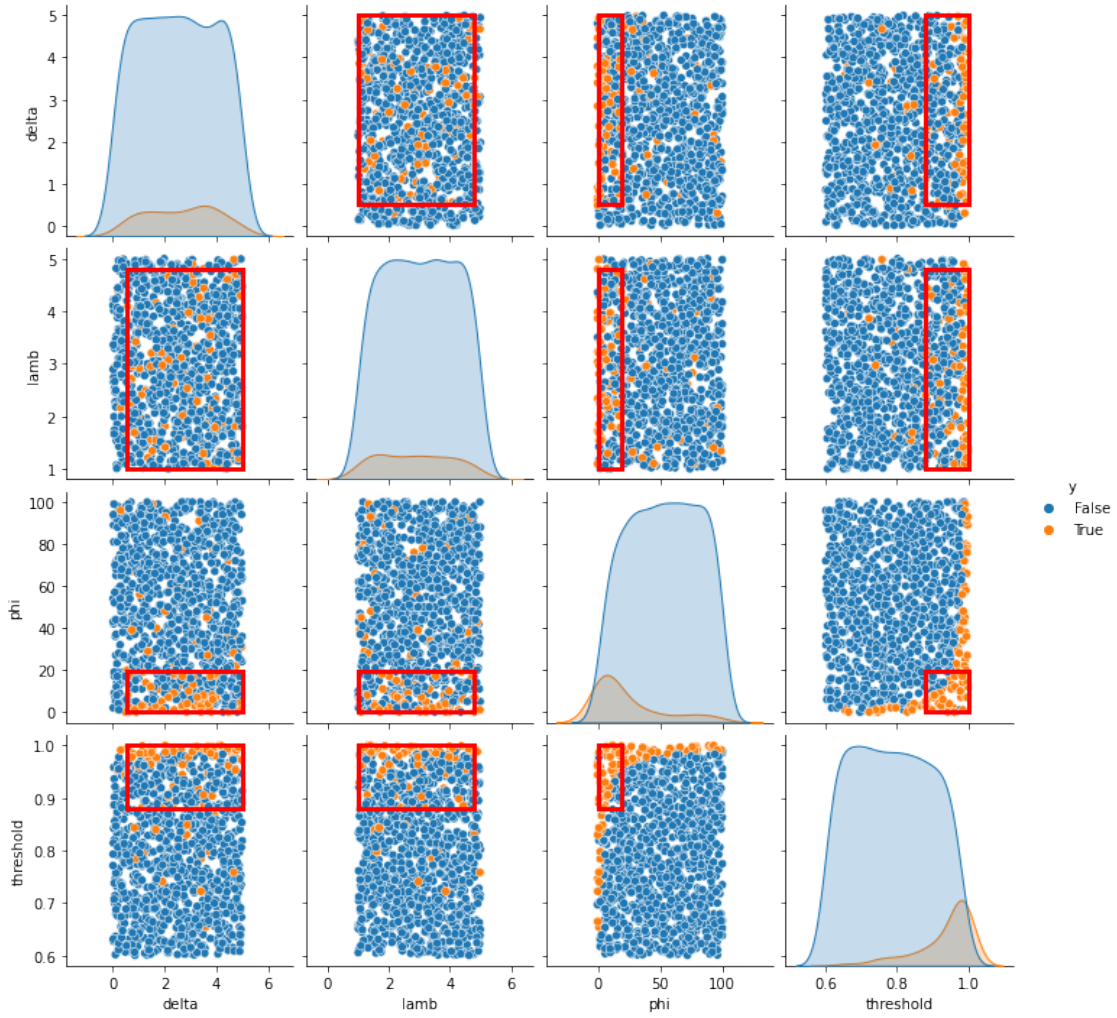
[7]:
```
[50,
       delta      lamb   phi  threshold
 0  0.517886  1.003900   0.0   0.879567
 1  4.997435  4.810672  19.5   0.999859,
 {'delta': [0.17834844930969904, -1.0],
  'lamb': [-1.0, 0.5878050850484972],
  'phi': [-1.0, 1.6188507007121986e-15],
  'threshold': [3.0375970058123697e-12, -1.0]}]
```

[9]:
```
# inspect the box according the proper index we found.
box_num = valid_ranges[0]
show_ranges(box_num, box1)
```

```
coverage        0.43
density     0.826923
id                50
mass           0.052
mean        0.826923
res_dim            4
Name: 50, dtype: object
```

box 50	min	max	qp values
phi	0.000000	19.500000	[-1.0, 1.6188507007121986e-15]
lamb	1.003900	4.810672	[-1.0, 0.5878050850484972]
threshold	0.879567	0.999859	[3.0375970058123697e-12, -1.0]
delta	0.517886	4.997435	[0.17834844930969904, -1.0]



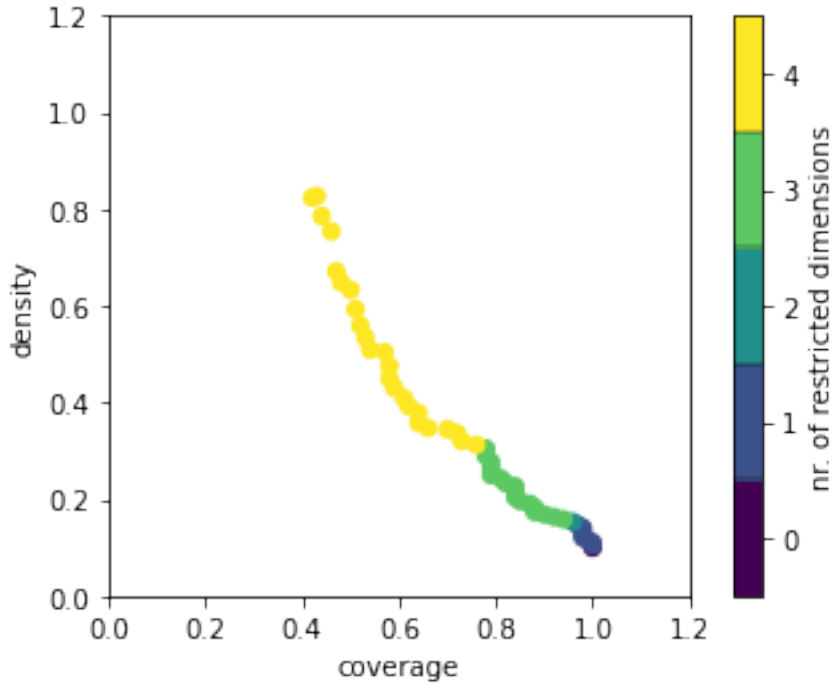


0.5 Show the parameter range overlaps for all domains

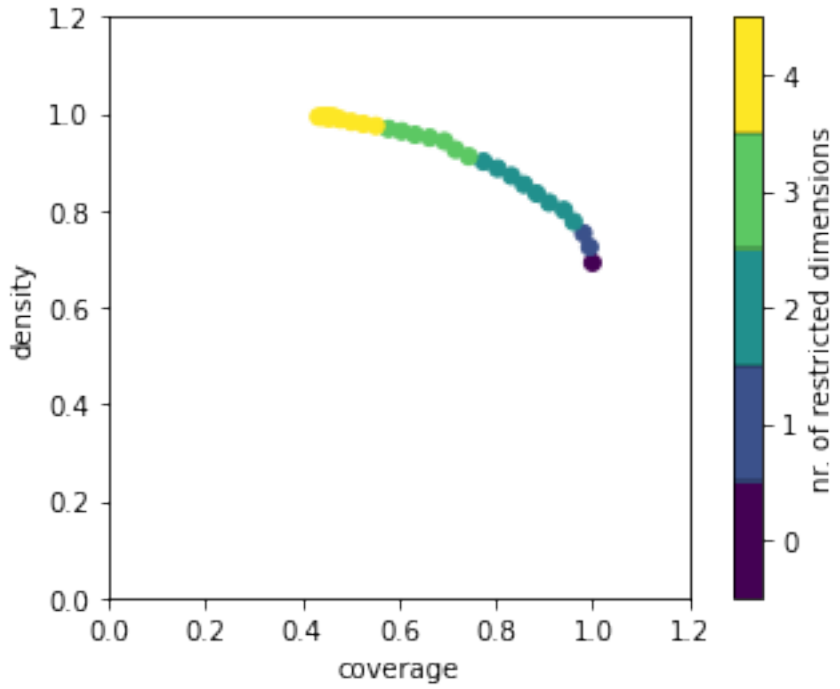
[10]:
```
# Run the PRIM algorithm for all domain, store the selected ranges in␣
↪param_ranges_list

domain_list = ["blocks-world", "campus", "depots", "driverlog", "dwr",␣
↪"easy-ipc-grid", "ferry",
               "intrusion-detection", "kitchen", "logistics", "miconic",␣
↪"rovers", "satellite",
               "sokoban", "zeno-travel"]

param_ranges_list = []

for i in range(15):
    domain_name = domain_list[i]
```

```
    print(domain_name)
    #PRIM
    results = load_results('./topk/1000_' + domain_name + '.tar.gz')
    a_avg = get_top(results, 0.1, "a_avg")

    results = load_results('./topk/1000_' + domain_name + '.tar.gz')
    box1 = find_box(results, 'a_avg', float(a_avg))

    param_ranges_list.append(fine_valid_range(box1))
```

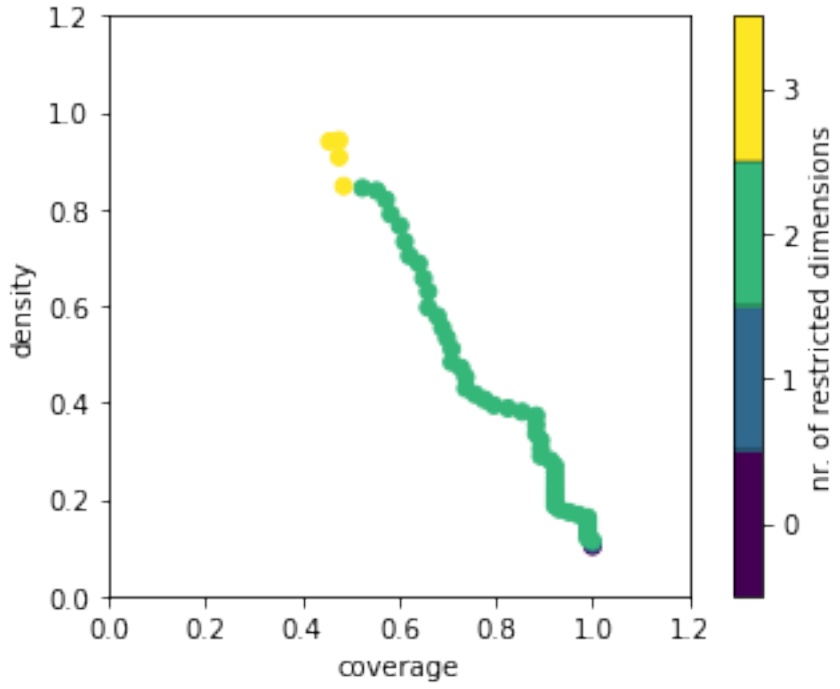
```
blocks-world
```



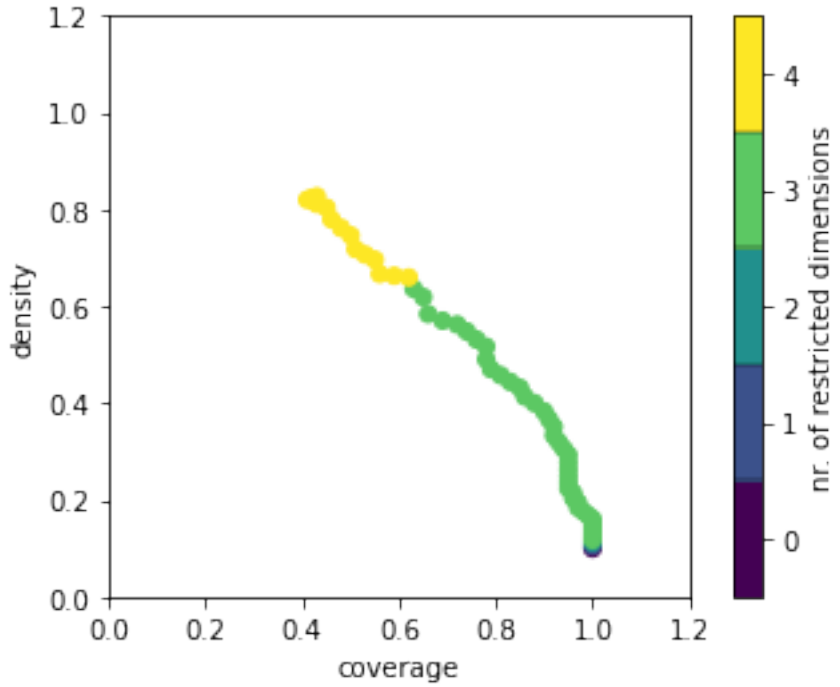

```
campus
```


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4

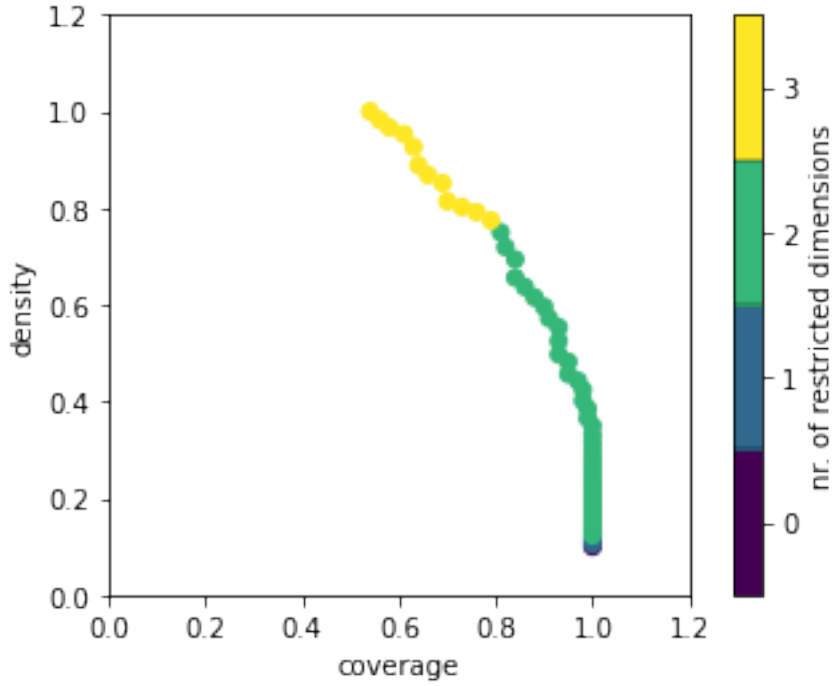
depots


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3

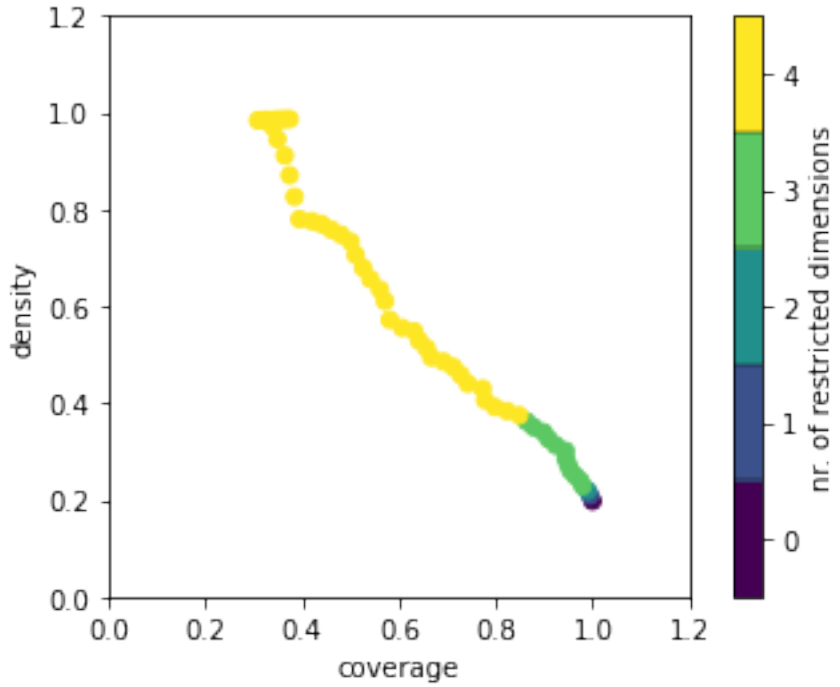
driverlog



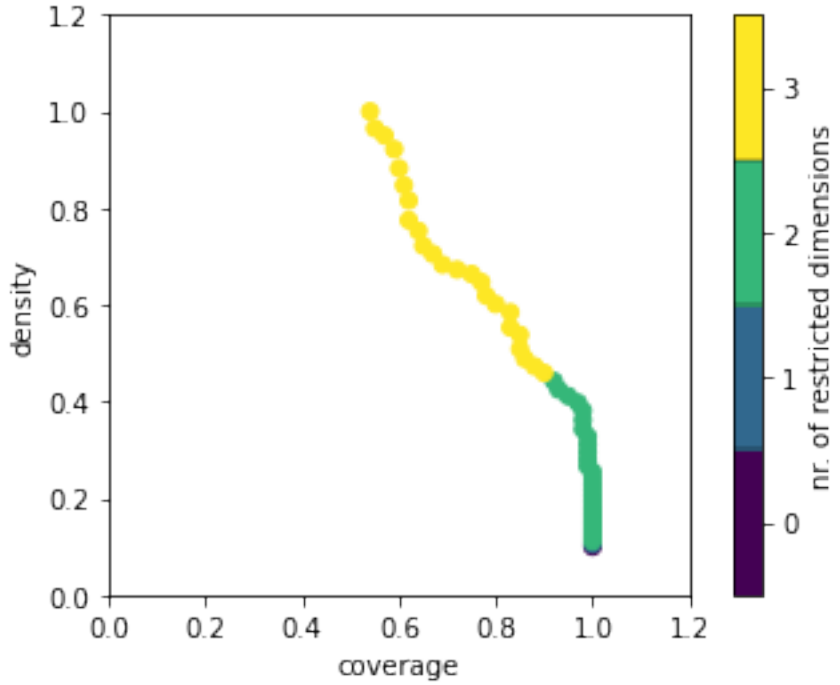
dwr



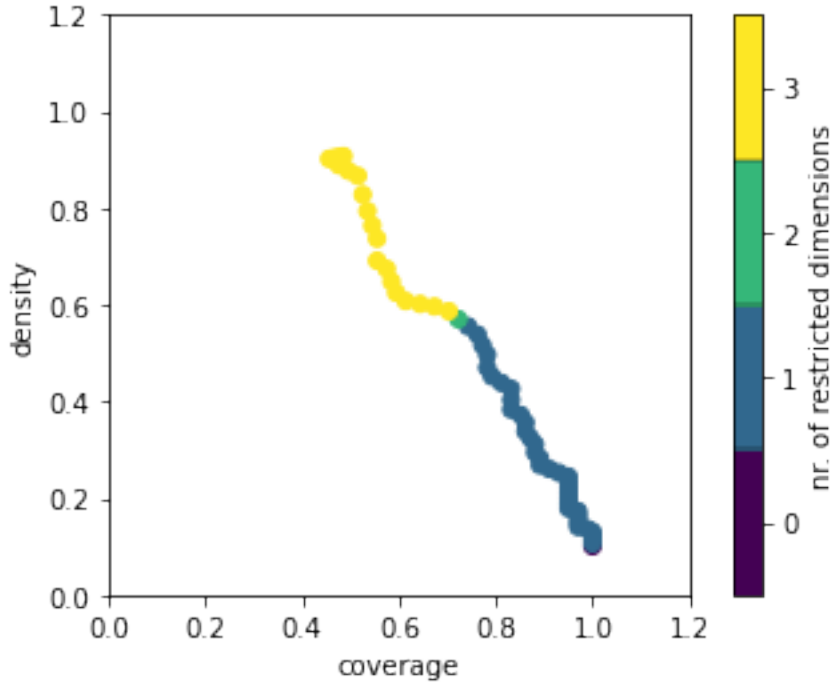
easy-ipc-grid



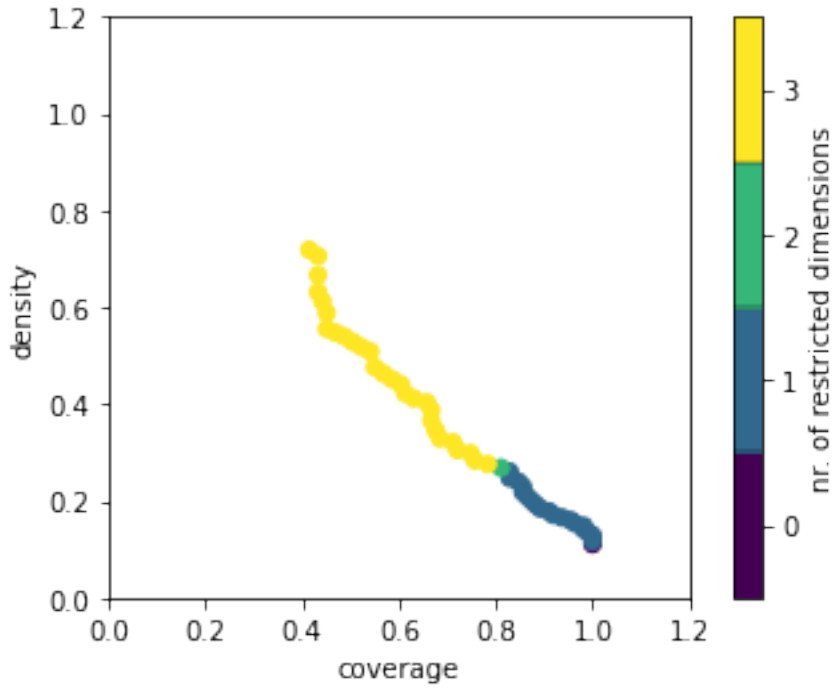
ferry



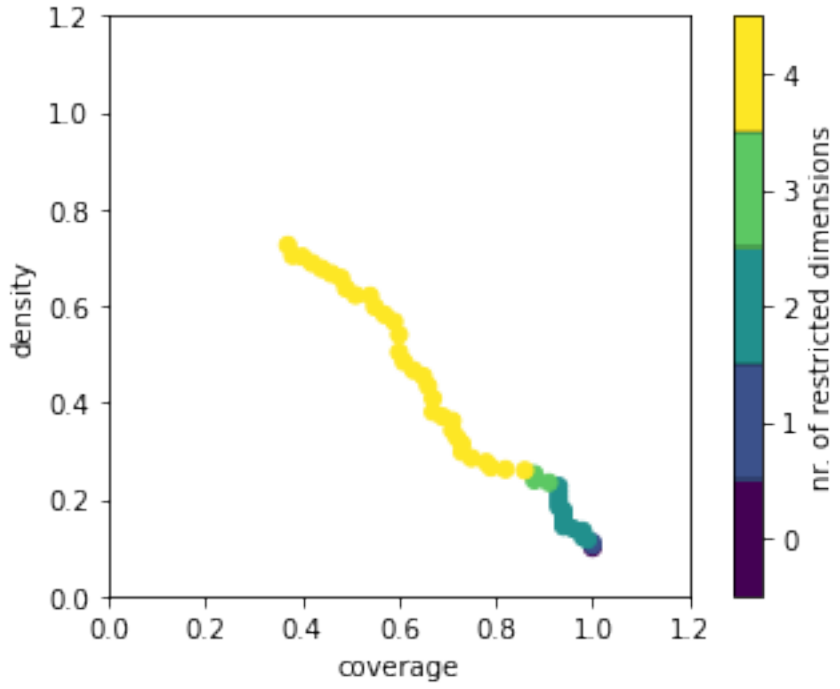
intrusion-detection



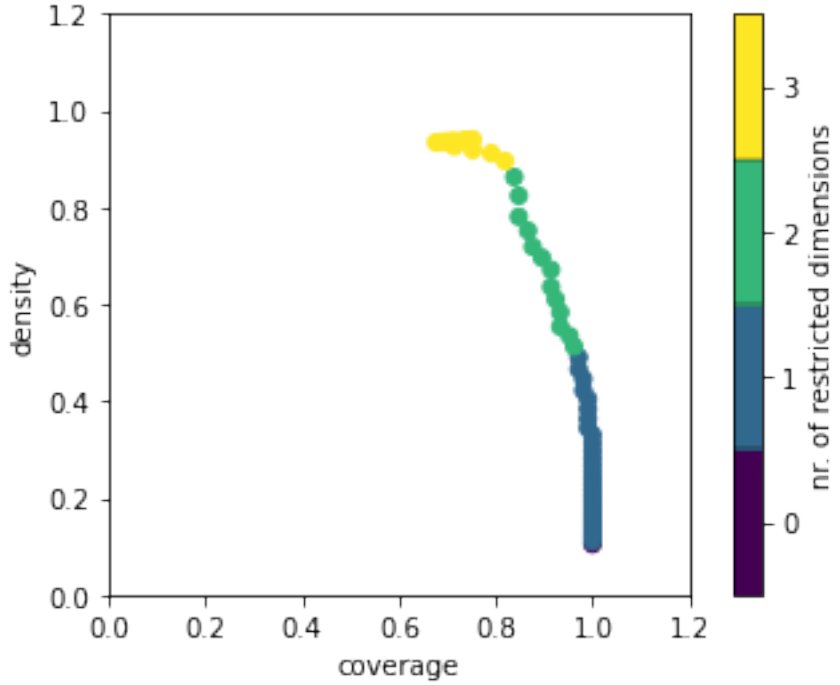
kitchen


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3

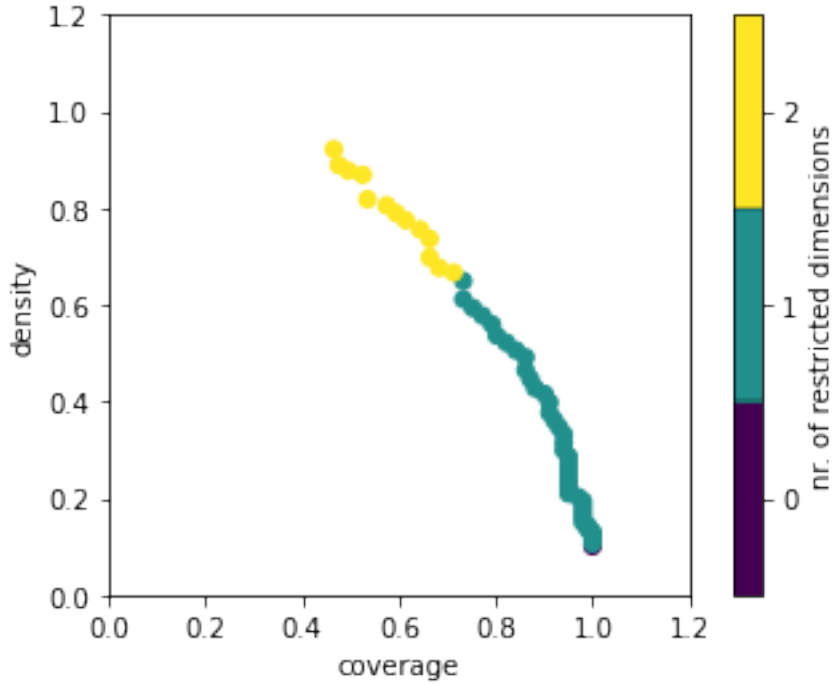
logistics


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4

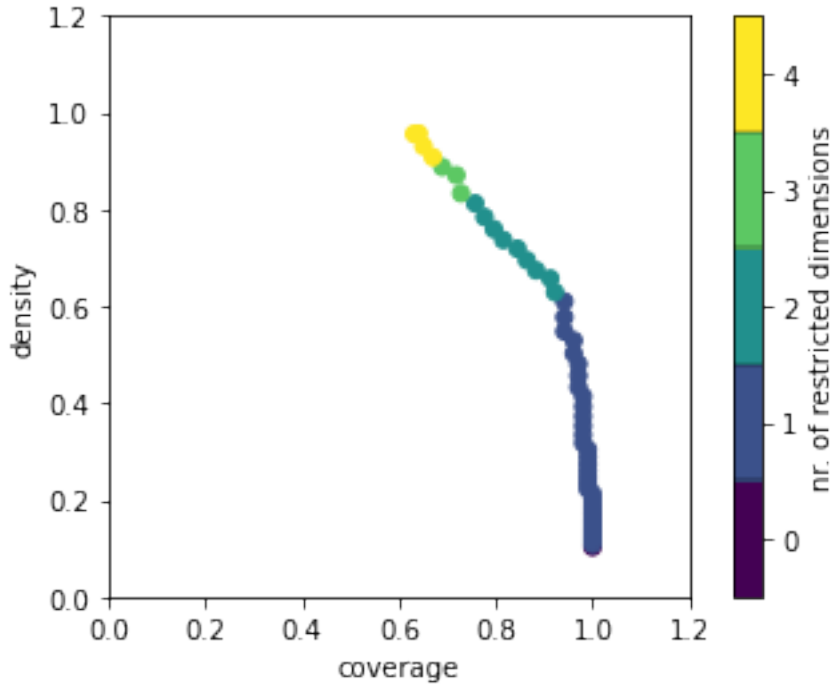
miconic



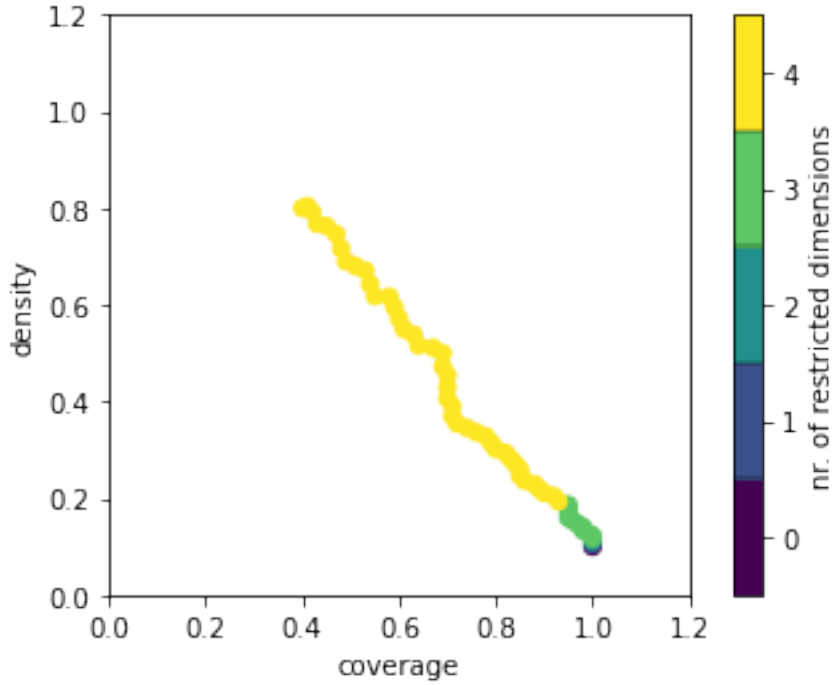
rovers



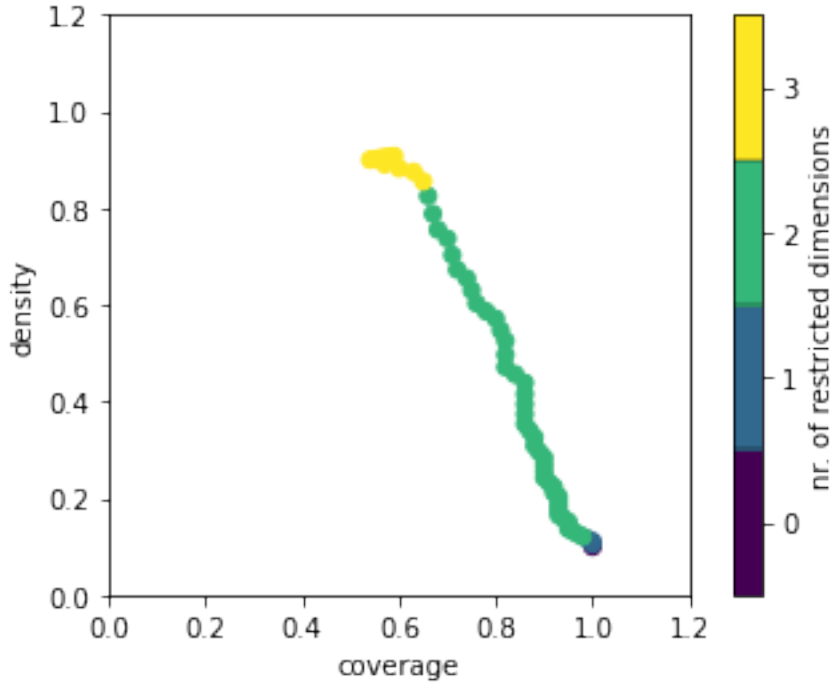
satellite



sokoban



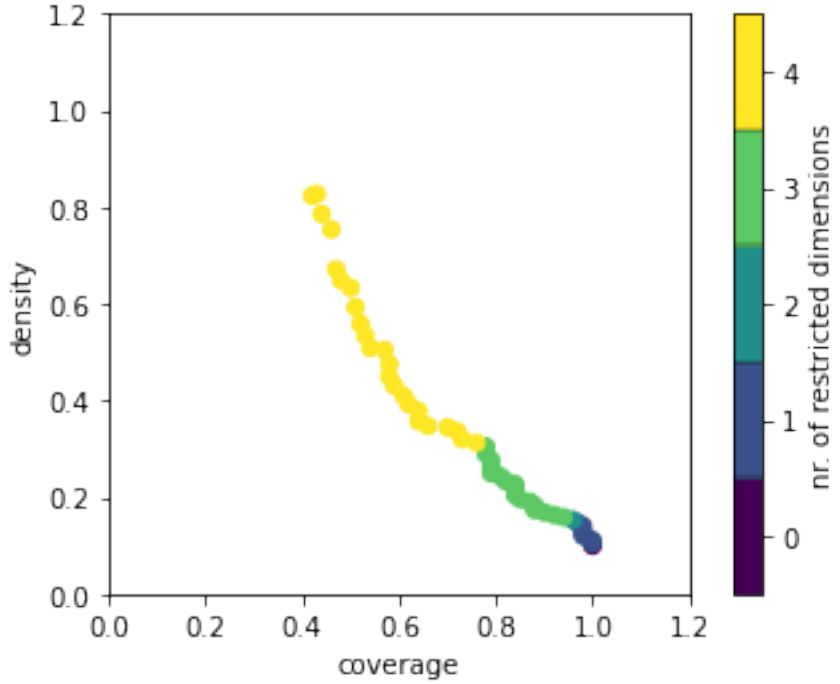
zeno-travel



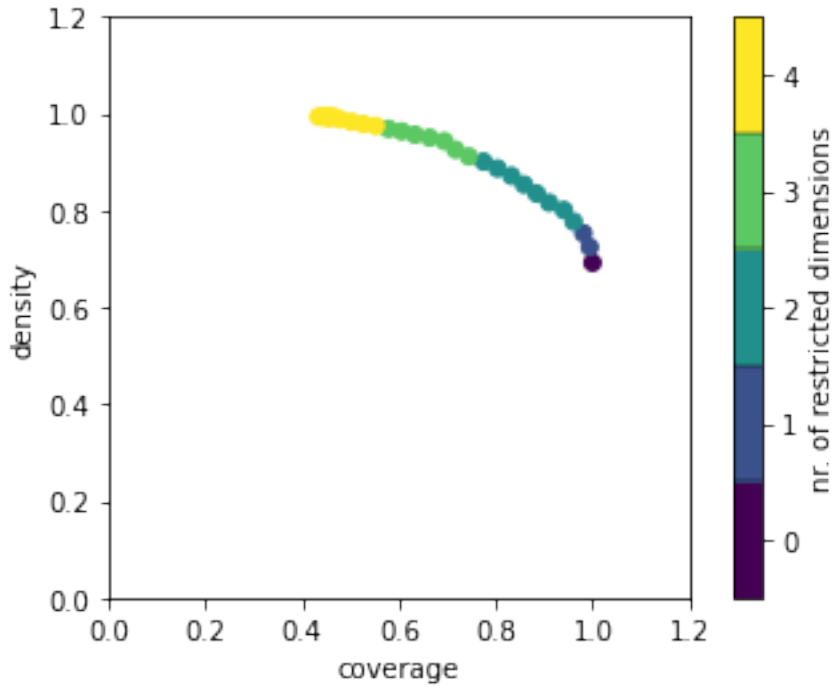
[10]:
```
# Run the PRIM algorithm for all domain, store the selected ranges in␣
↪param_ranges_list

domain_list = ["blocks-world", "campus", "depots", "driverlog", "dwr",␣
↪"easy-ipc-grid", "ferry",
               "intrusion-detection", "kitchen", "logistics", "miconic",␣
↪"rovers", "satellite",
               "sokoban", "zeno-travel"]

param_ranges_list = []

for i in range(15):
    domain_name = domain_list[i]

    print(domain_name)
    #PRIM
    results = load_results('./topk/1000_' + domain_name + '.tar.gz')
    a_avg = get_top(results, 0.1, "a_avg")

    results = load_results('./topk/1000_' + domain_name + '.tar.gz')
```

```
    box1 = find_box(results, 'a_avg', float(a_avg))

    param_ranges_list.append(fine_valid_range(box1))
```

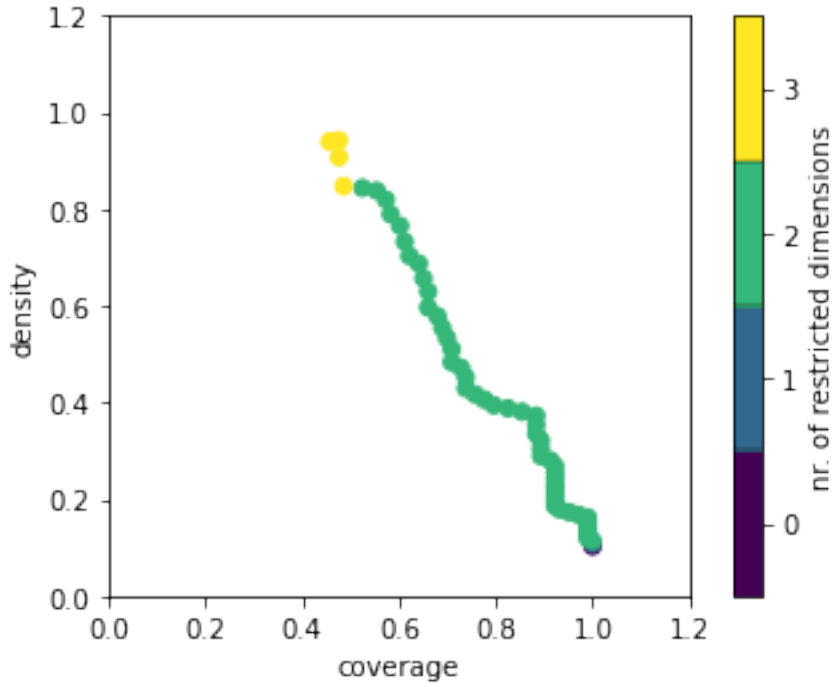
blocks-world



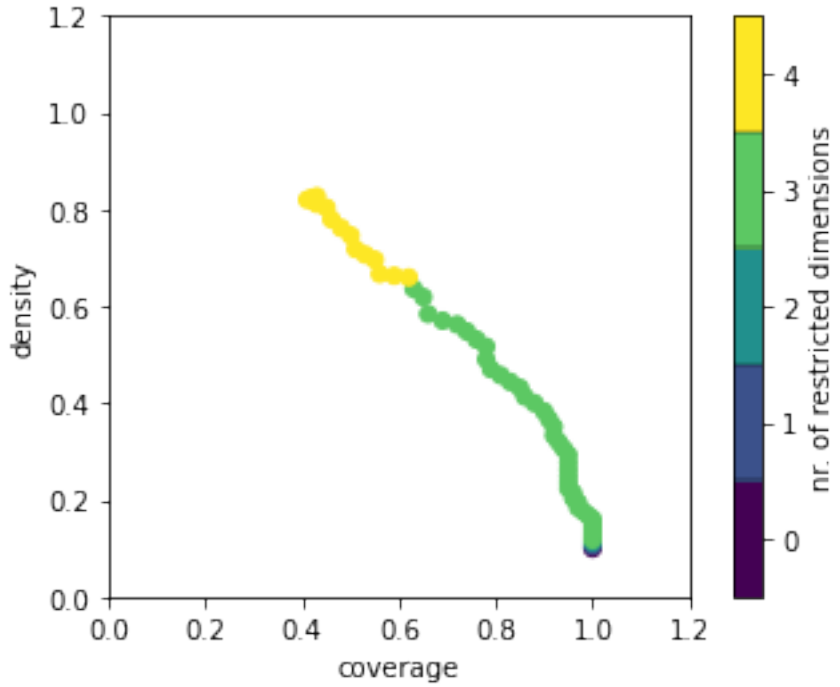
campus


1.2
1.0
0.8
0.6
0.4
0.2
0.0
density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
coverage
4
3
2
1
0
nr. of restricted dimensions

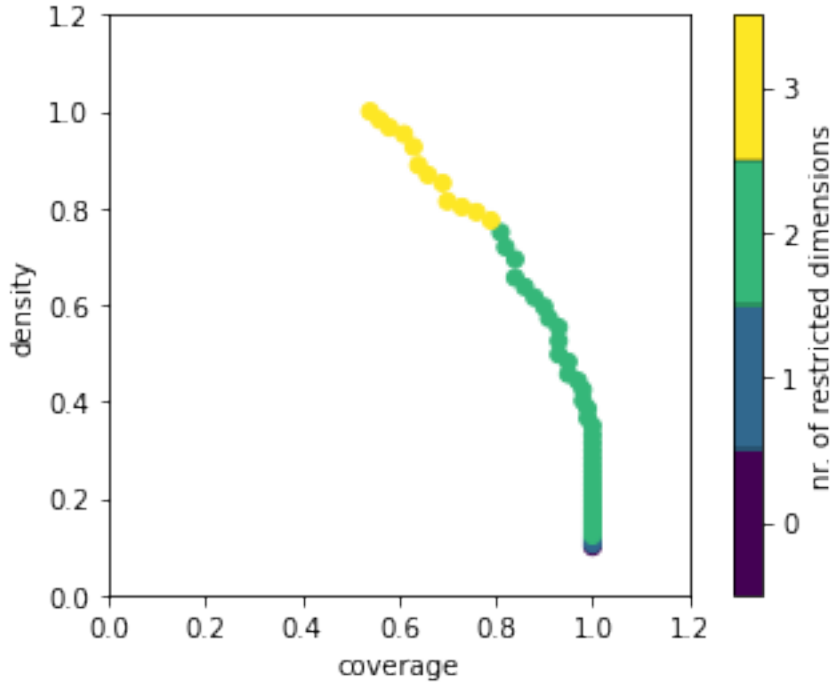
depots


1.2
1.0
0.8
0.6
0.4
0.2
0.0
density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
coverage
3
2
1
0
nr. of restricted dimensions

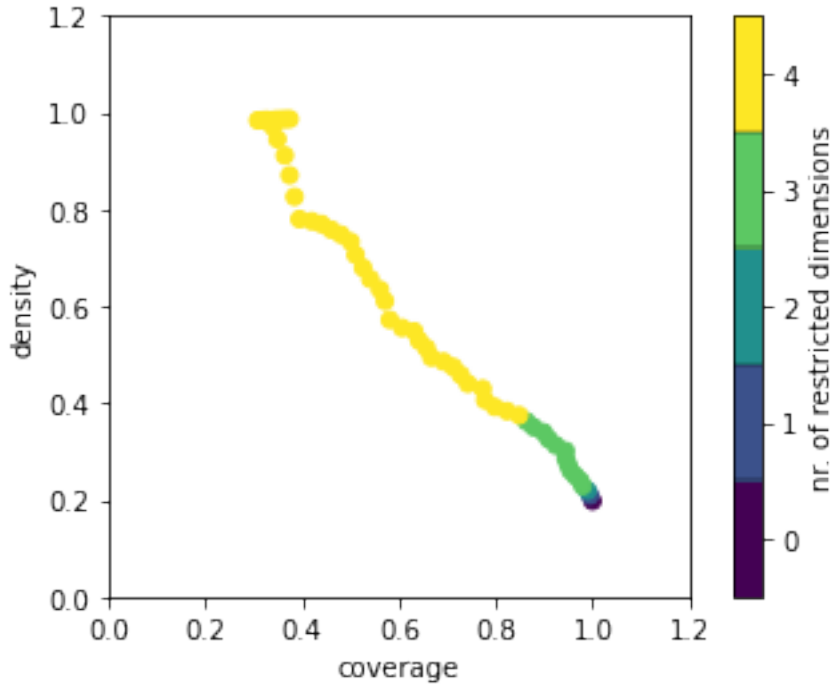
driverlog



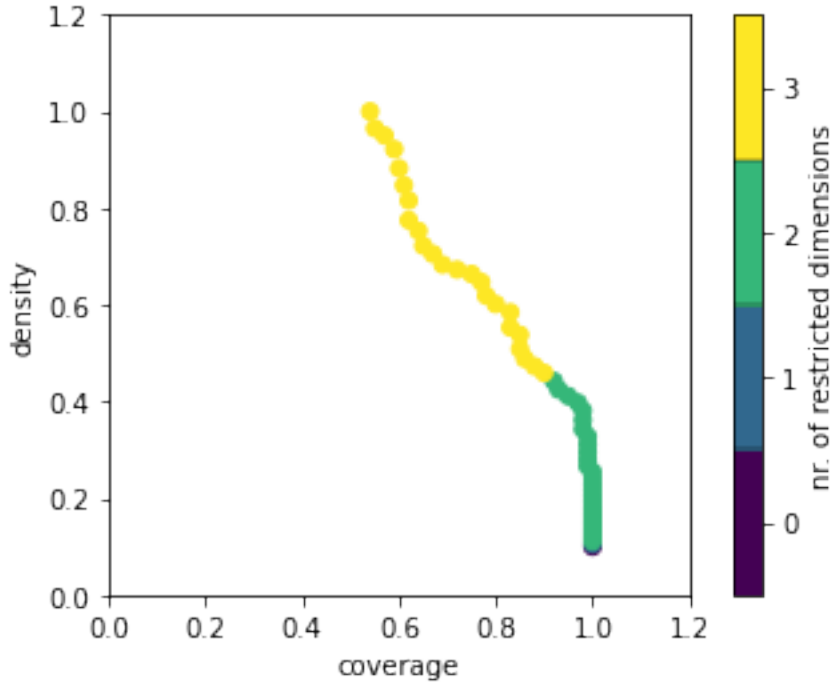
dwr



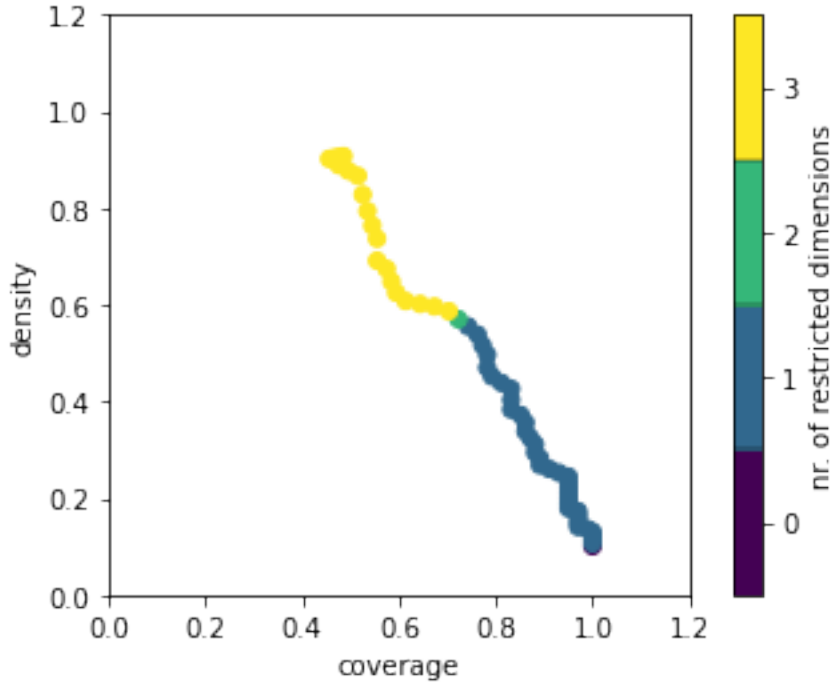
easy-ipc-grid



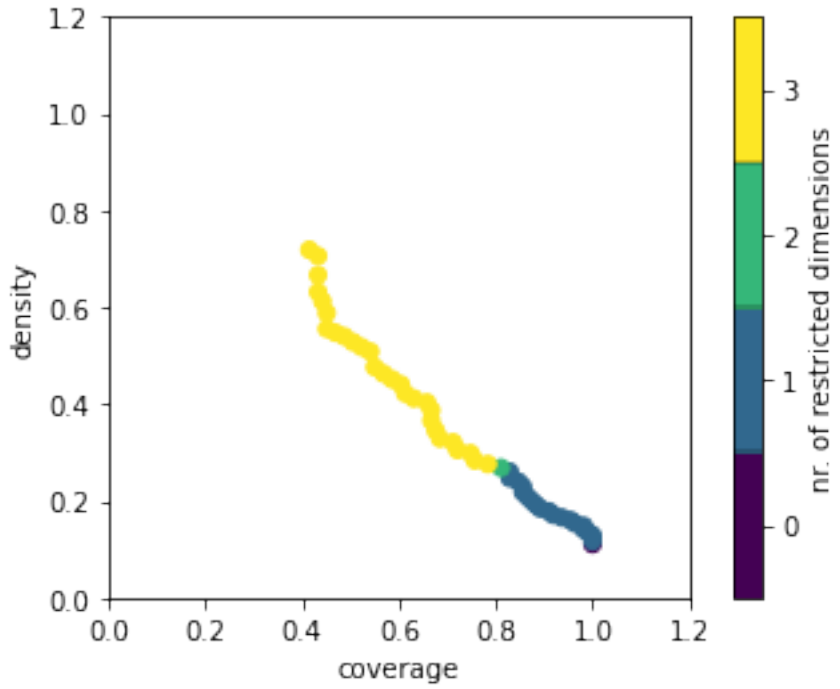
ferry



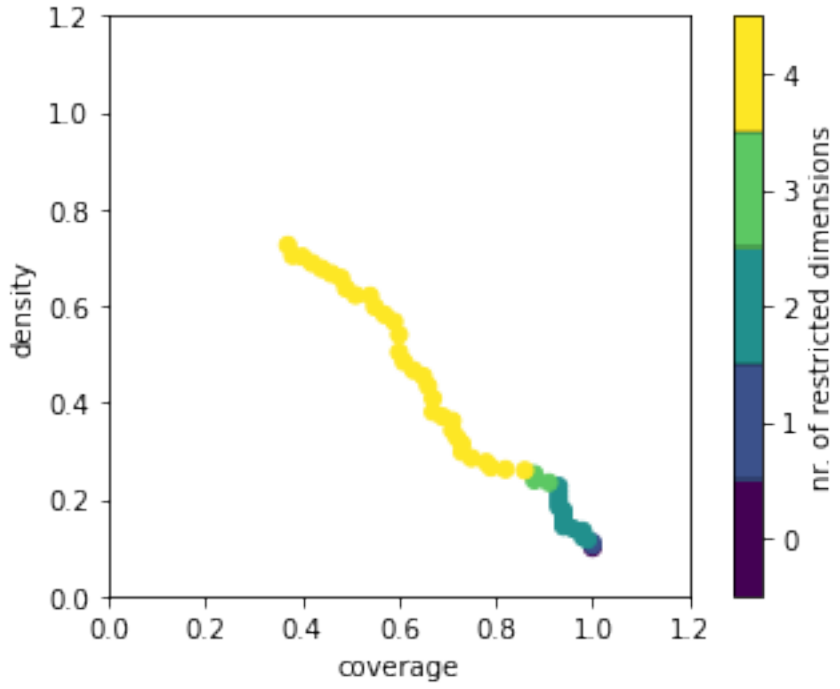
intrusion-detection



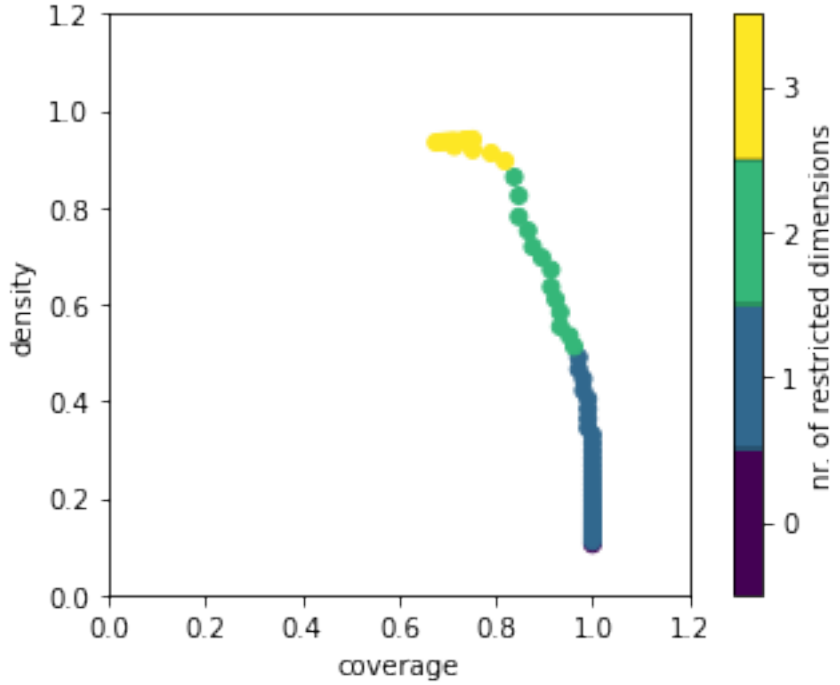
kitchen


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3

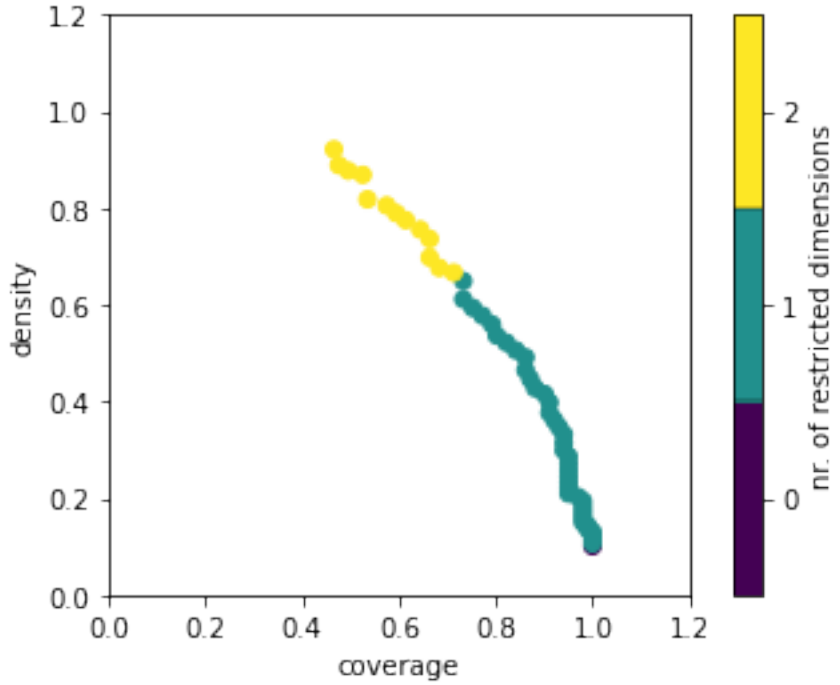
logistics


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4

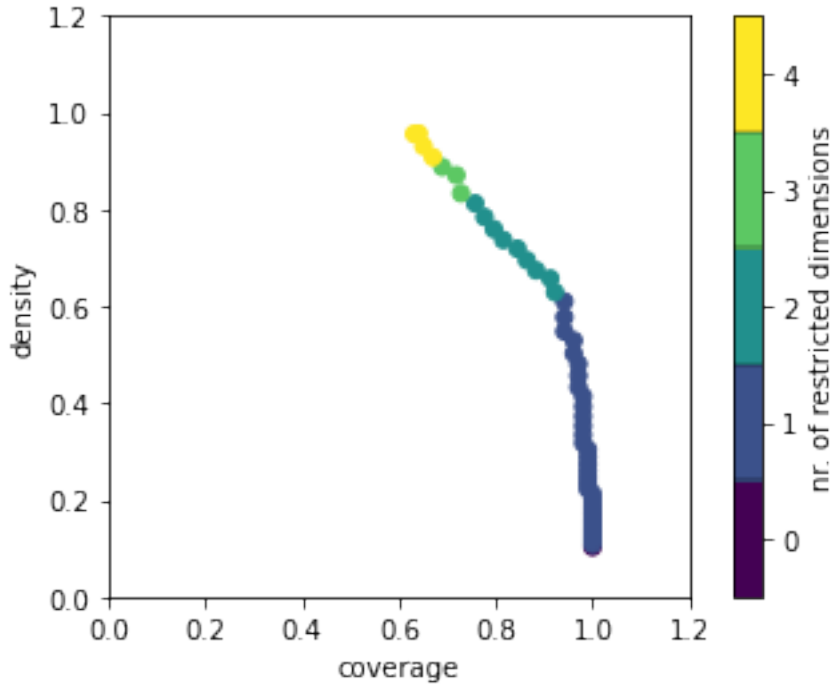
miconic



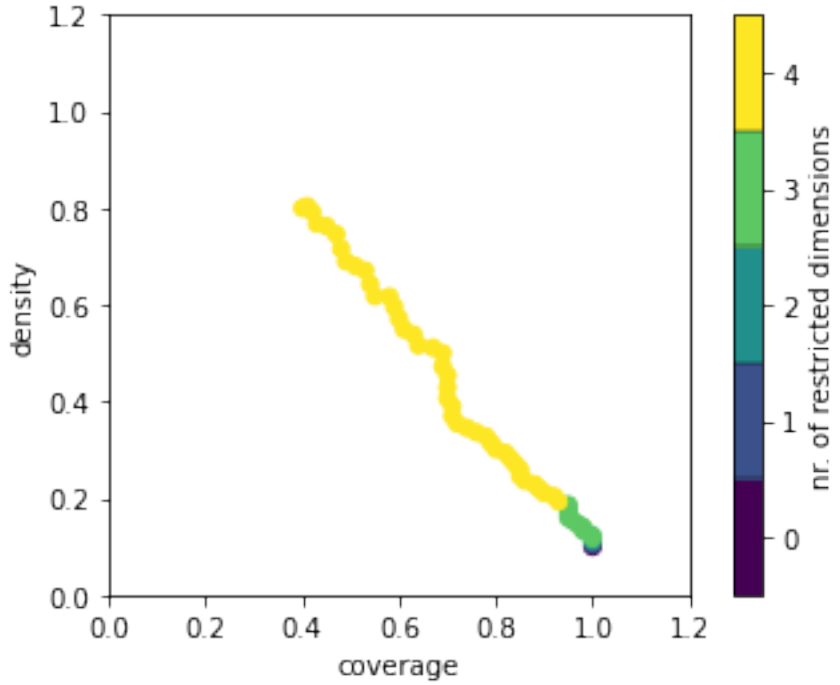
rovers



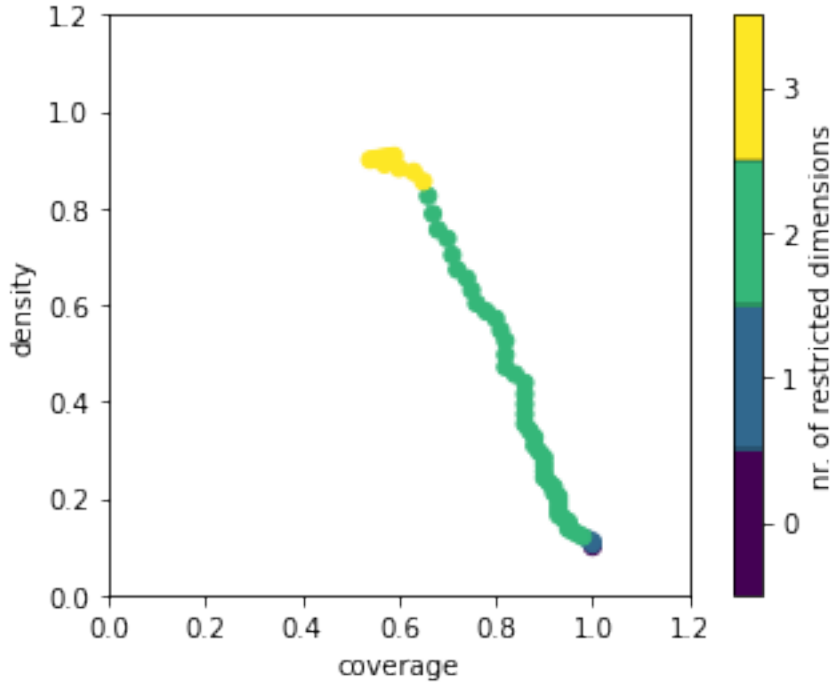
satellite



sokoban

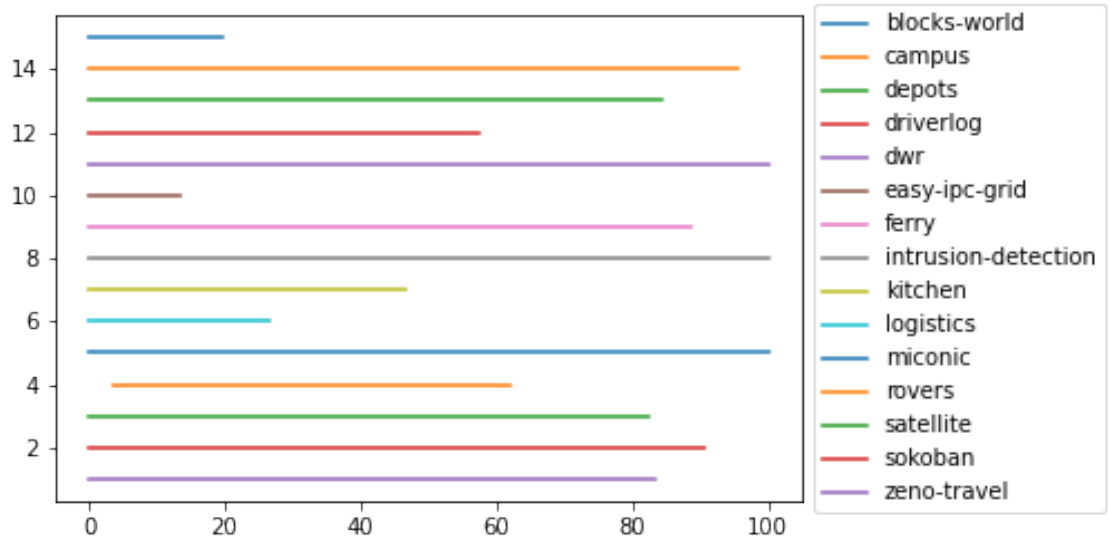


zeno-travel



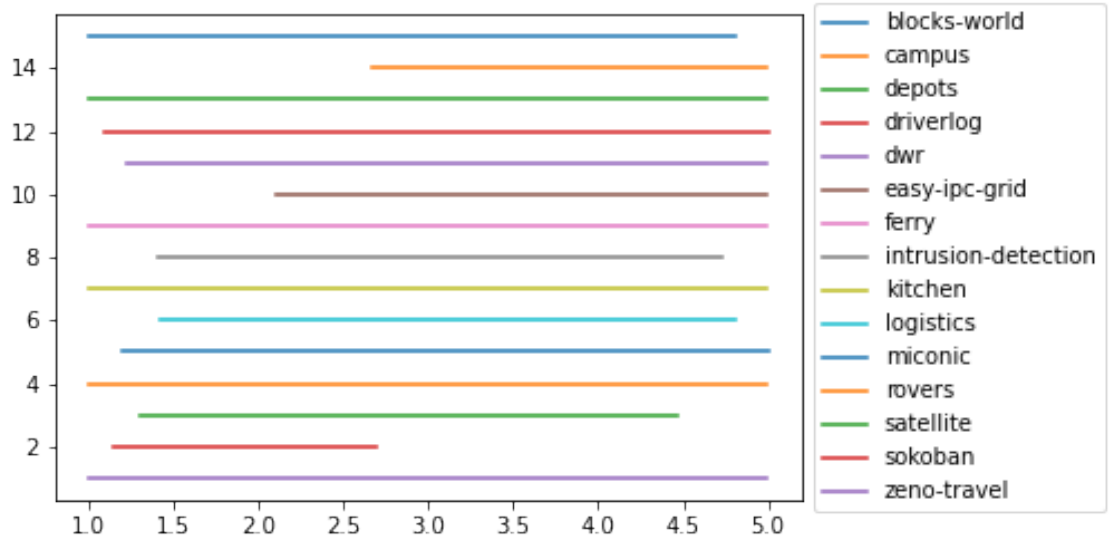
[33]:
```
def show_ranges_overlap(param_ranges_list, param):
    print(param)
    for i in range(15):
        item = param_ranges_list[i]
        x_values = [item[1][param][0], item[1][param][1]]
        y_values = [15 - i, 15 - i]

        plt.plot(x_values, y_values)
        plt.legend(["blocks-world", "campus", "depots", "driverlog", "dwr",␣
↪"easy-ipc-grid", "ferry",
                "intrusion-detection", "kitchen", "logistics", "miconic",␣
↪"rovers", "satellite",
                "sokoban", "zeno-travel"], loc='center left', bbox_to_anchor=(1,␣
↪0.5))

show_ranges_overlap(param_ranges_list, "phi")
```

phi



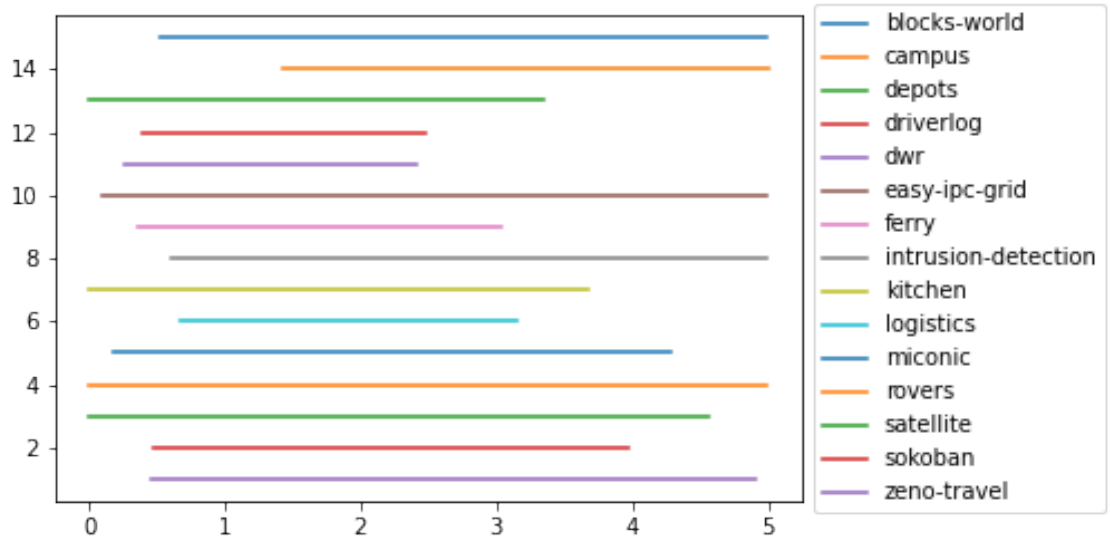
[34]:
```
show_ranges_overlap(param_ranges_list, "lamb")
```

```
lamb
```



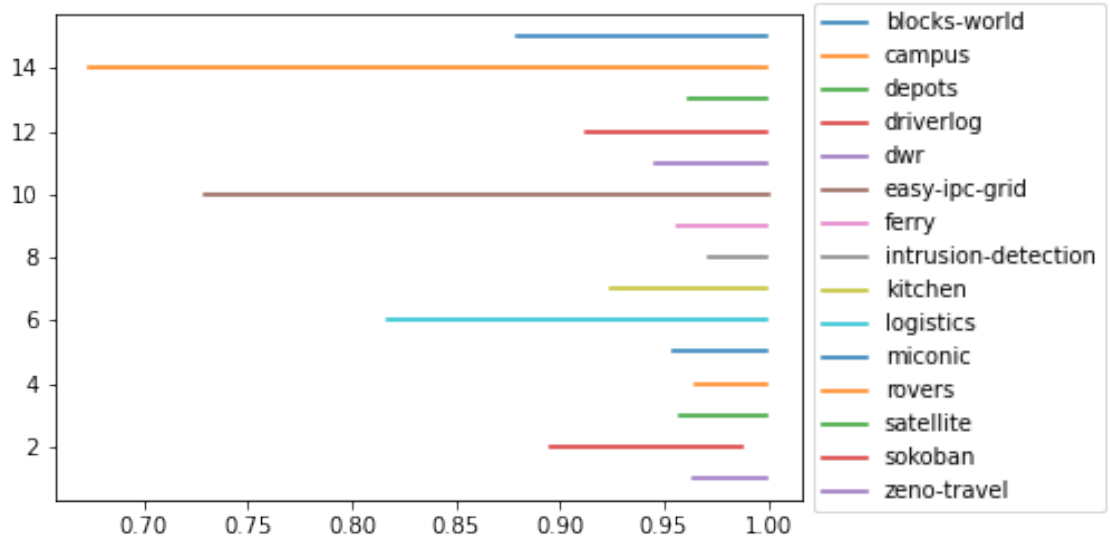
[35]:
```
show_ranges_overlap(param_ranges_list, "delta")
```

```
delta
```



[36]:
```
show_ranges_overlap(param_ranges_list, "threshold")
```

```
threshold
```



[ ]:
```

```

0.6 SOBOL Sampling

The SOBOL analysis indicates if a parameter has significant impact on performance or not, details see here.

Firstly, we need to define a model as following code block. The four parameters (phi, delta, lambda, threshold) are defined as uncertainties, domain is defined as a constant, the outcomes are performance statistics (p_10, r_10, a_10, p_30, r_30, a_30, p_50, r_50, a_50, p_70, r_70, a_70, p_100, r_100, a_100, p_avg, r_avg, a_avg). The details of how to define a model can be found here

[8]:
```
from model import gr_system
import os
import sys
import subprocess
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from ema_workbench import (RealParameter, IntegerParameter,␣
 ↪CategoricalParameter,
                           ScalarOutcome, Constant,
                           Model, ema_logging, perform_experiments)

domain = "a_domain_name"
model = Model('grsystem', function=gr_system)

# set uncertainties
model.uncertainties = [IntegerParameter("phi", 0,100),
                RealParameter("delta", 0, 5),
                RealParameter("lamb", 1, 5),
                RealParameter("threshold", 0.6, 1.0)]

# set domain to a constant
model.constants = [Constant("domain", domain)]

# specify outcomes
model.outcomes = [ScalarOutcome('p_10'),
                  ScalarOutcome('r_10'),
                  ScalarOutcome('a_10'),
                  ScalarOutcome('p_30'),
                  ScalarOutcome('r_30'),
                  ScalarOutcome('a_30'),
                  ScalarOutcome('p_50'),
                  ScalarOutcome('r_50'),
                  ScalarOutcome('a_50'),
                  ScalarOutcome('p_70'),
                  ScalarOutcome('r_70'),
                  ScalarOutcome('a_70'),
                  ScalarOutcome('p_100'),
                  ScalarOutcome('r_100'),
```

```
                    ScalarOutcome('a_100'),
                    ScalarOutcome('p_avg'),
                    ScalarOutcome('r_avg'),
                    ScalarOutcome('a_avg')]
```

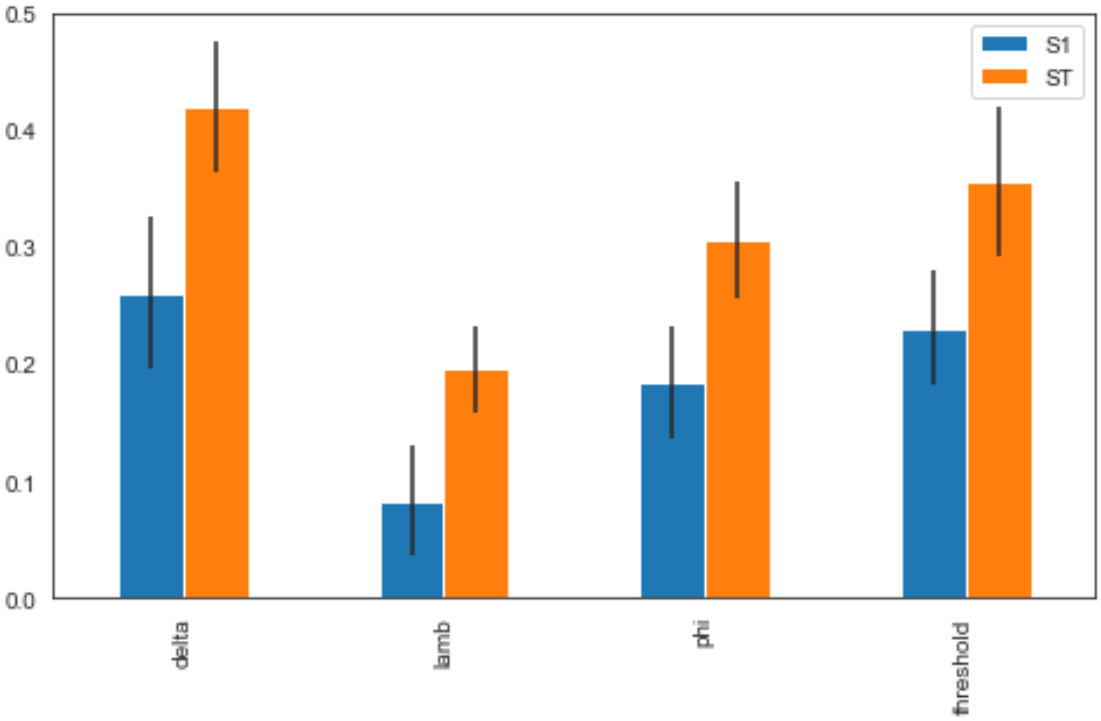
Load the SOBOL analysis helper functions Here we show the first order effect and total effect. And displayed SOBOL analysis results coresponding to all performance statistics (p_10, r_10, a_10, p_30, r_30, a_30, p_50, r_50, a_50, p_70, r_70, a_70, p_100, r_100, a_100, p_avg, r_avg, a_avg). Generally, we are more interested in the significance of impacts on `a_avg` (avg: accuracy).

[9]:
```
# the first order and total:

from SALib.analyze import sobol
from ema_workbench.em_framework.salib_samplers import get_SALib_problem

def show_effects(results, outcomes_field):
    experiments, outcomes = results
    problem = get_SALib_problem(model.uncertainties)
    Si = sobol.analyze(problem, outcomes[outcomes_field],
                       calc_second_order=True, print_to_console=False)

    scores_filtered = {k:Si[k] for k in ['ST','ST_conf','S1','S1_conf']}
    Si_df = pd.DataFrame(scores_filtered, index=problem['names'])

    sns.set_style('white')
    fig, ax = plt.subplots(1)

    indices = Si_df[['S1','ST']]
    err = Si_df[['S1_conf','ST_conf']]

    indices.plot.bar(yerr=err.values.T,ax=ax)
    fig.set_size_inches(8,6)
    fig.subplots_adjust(bottom=0.3)
    plt.show()

def display_sobol_analysis_results(results):
    print("avg: accuracy")
    show_effects(results, 'a_avg')
    print("avg: recall")
    show_effects(results, 'r_avg')
    print("avg: precision")
    show_effects(results, 'p_avg')
    print("100: accuracy")
    show_effects(results, 'a_100')
    print("100: recall")
    show_effects(results, 'r_100')
```

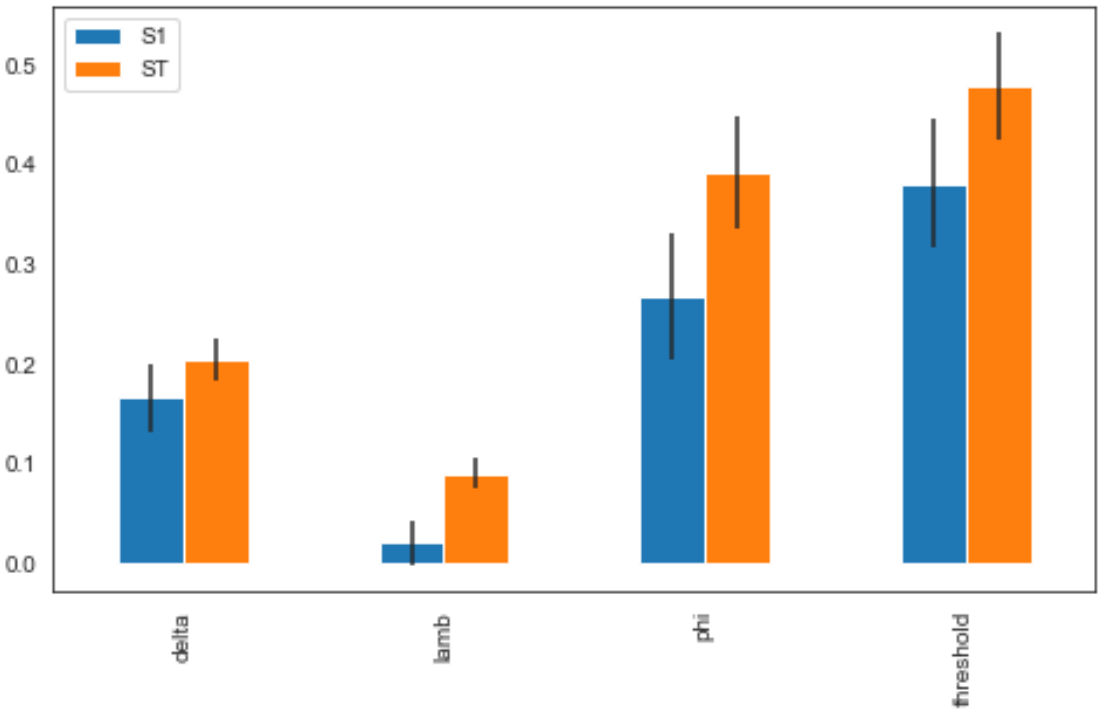
```
    print("100: precision")
    show_effects(results, 'p_100')
    print("70: accuracy")
    show_effects(results, 'a_70')
    print("70: recall")
    show_effects(results, 'r_70')
    print("70: precision")
    show_effects(results, 'p_70')
    print("50: accuracy")
    show_effects(results, 'a_50')
    print("50: recall")
    show_effects(results, 'r_50')
    print("50: precision")
    show_effects(results, 'p_50')
    print("30: accuracy")
    show_effects(results, 'a_30')
    print("30: recall")
    show_effects(results, 'r_30')
    print("30: precision")
    show_effects(results, 'p_30')
    print("10: accuracy")
    show_effects(results, 'a_10')
    print("10: recall")
    show_effects(results, 'r_10')
    print("10: precision")
    show_effects(results, 'p_10')


# this function only show "a_avg"
def display_sobol_analysis_results_a_avg(results):
    print("avg: accuracy")
    show_effects(results, 'a_avg')
```

[10]:
```
# load data
domain_name = "blocks-world"
sobol_results = load_results('./topk/1050_' + domain_name + '.tar.gz')

display_sobol_analysis_results(sobol_results)
```

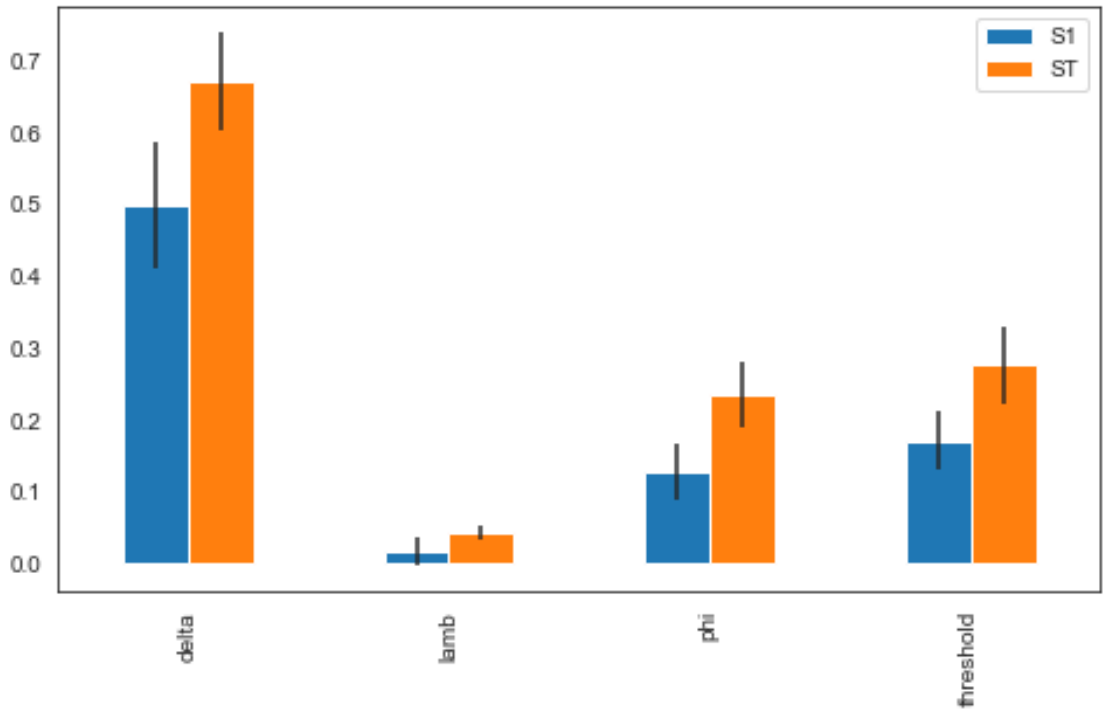
```
avg: accuracy
```


0.5
0.4
0.3
0.2
0.1
0.0
S1
ST
delta
lamb
phi
threshold

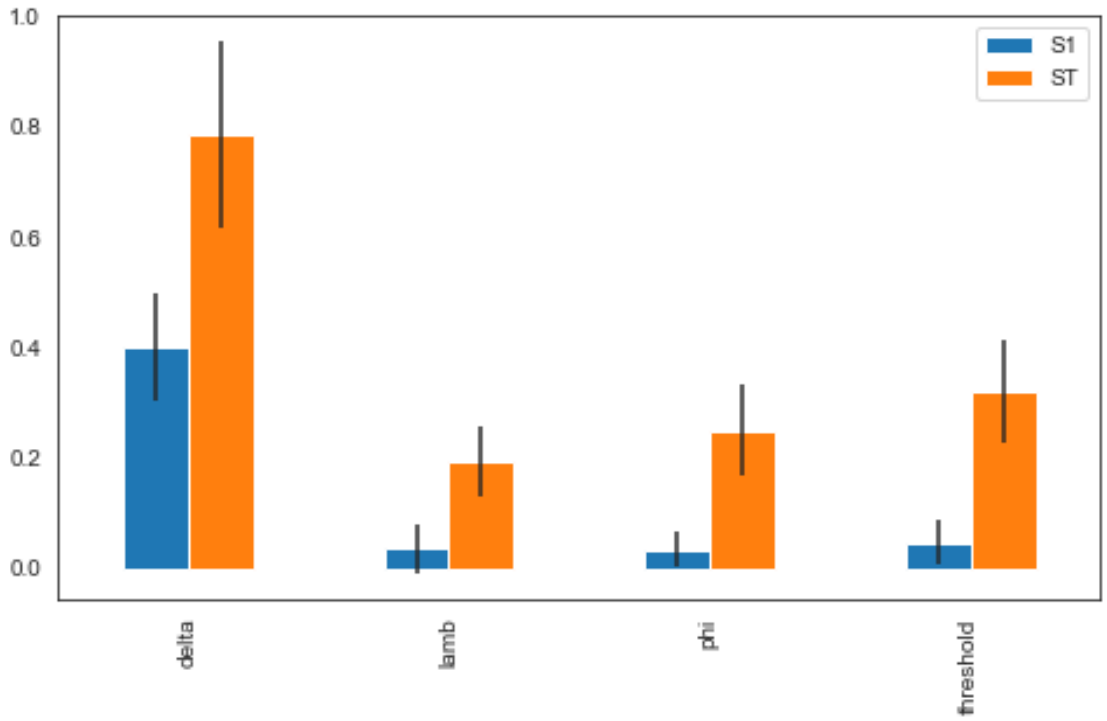
avg: recall


S1
ST
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

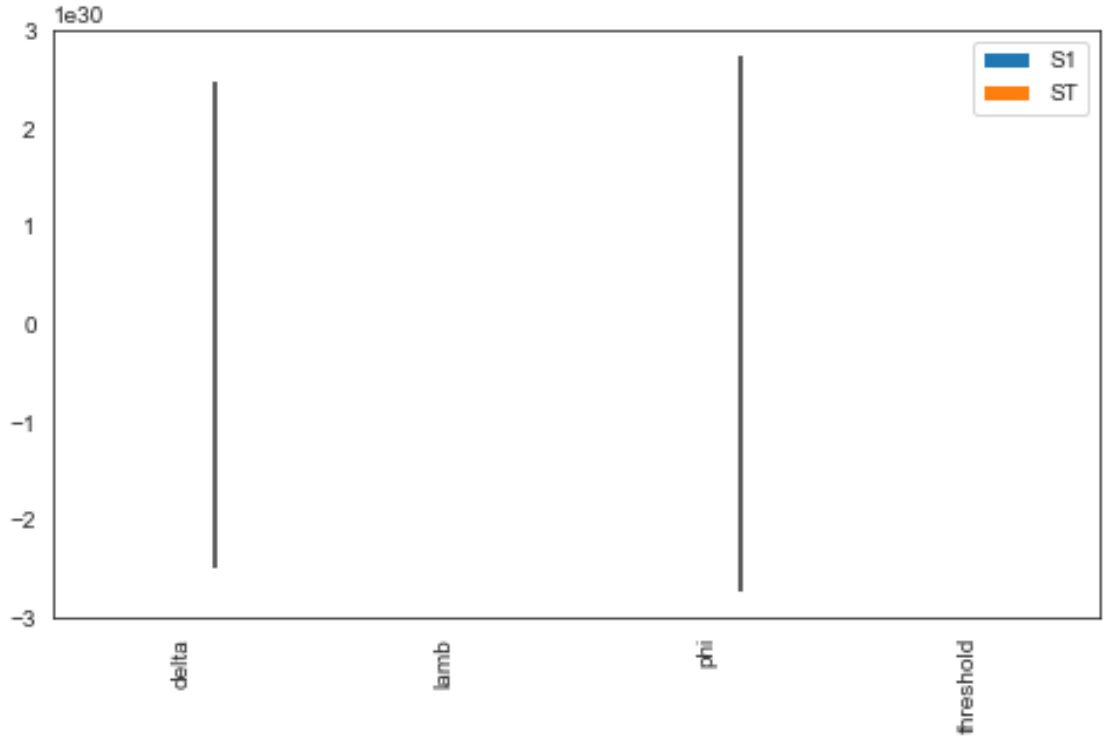
avg: precision



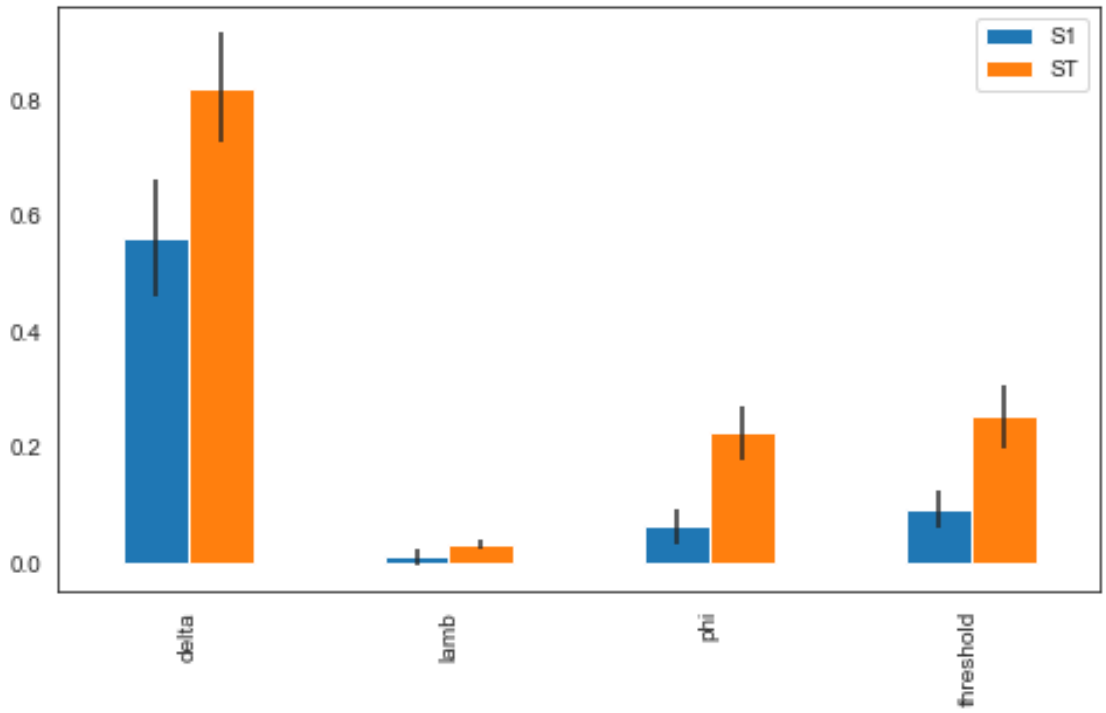
100: accuracy


1.0
0.8
0.6
0.4
0.2
0.0
S1
ST
delta
lamb
phi
threshold

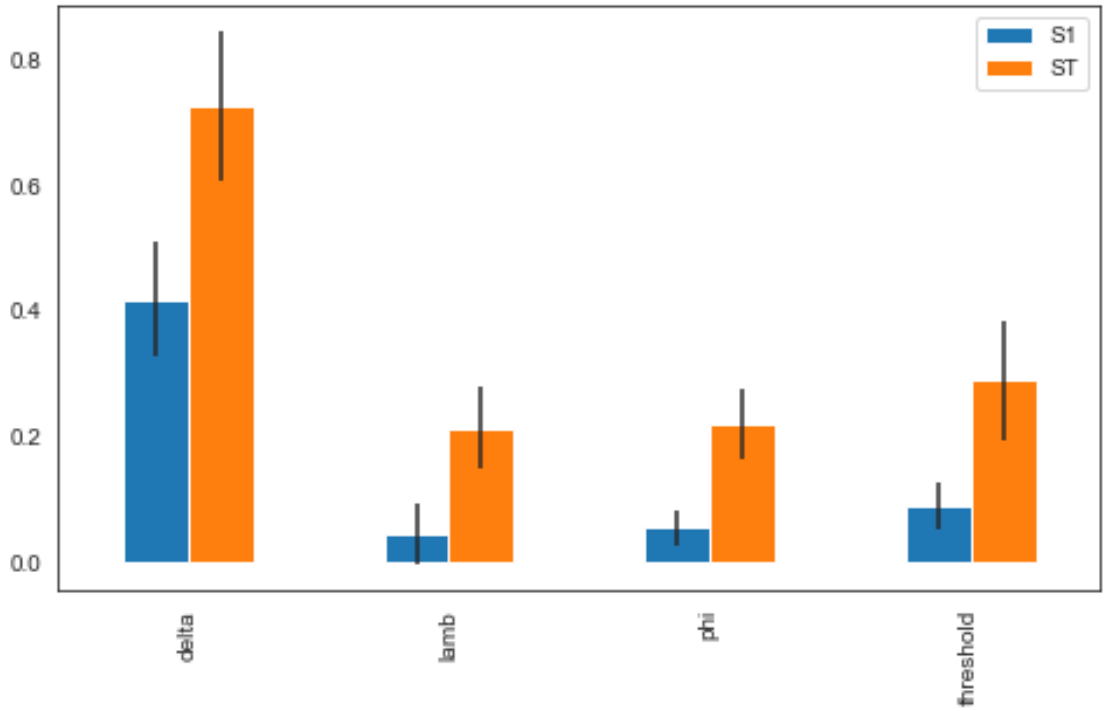
100: recall


1e30
3
2
1
0
−1
−2
−3
S1
ST
delta
lamb
phi
threshold

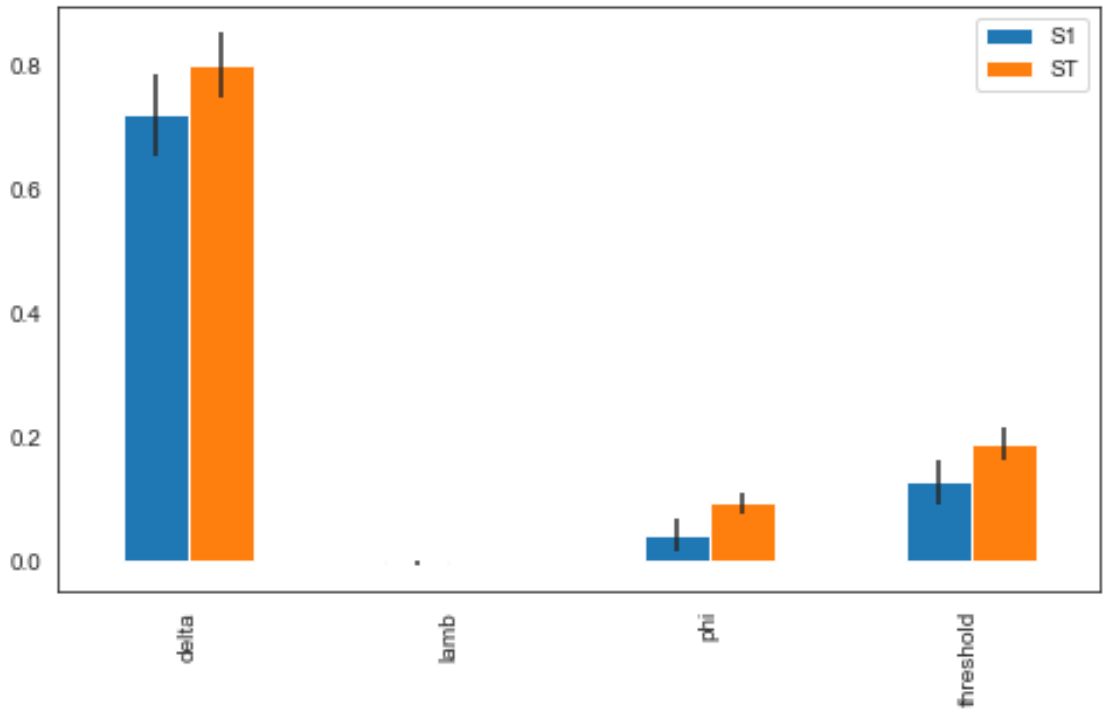
100: precision



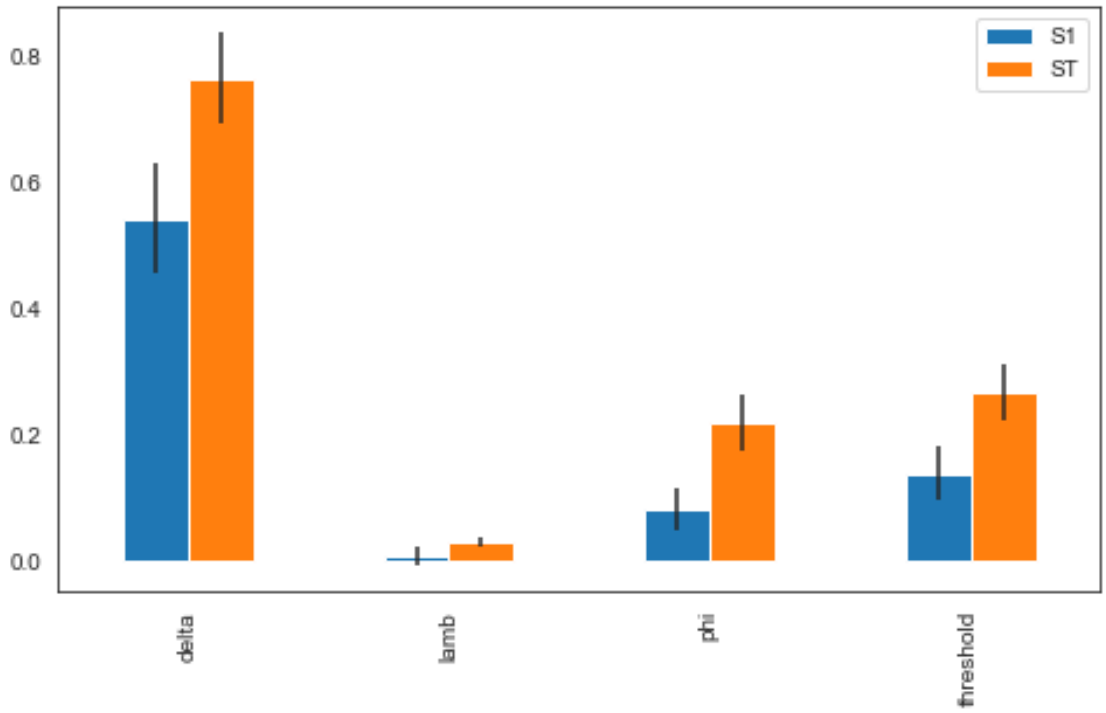
70: accuracy


S1
ST
0.8
0.6
0.4
0.2
0.0
delta
lamb
phi
threshold

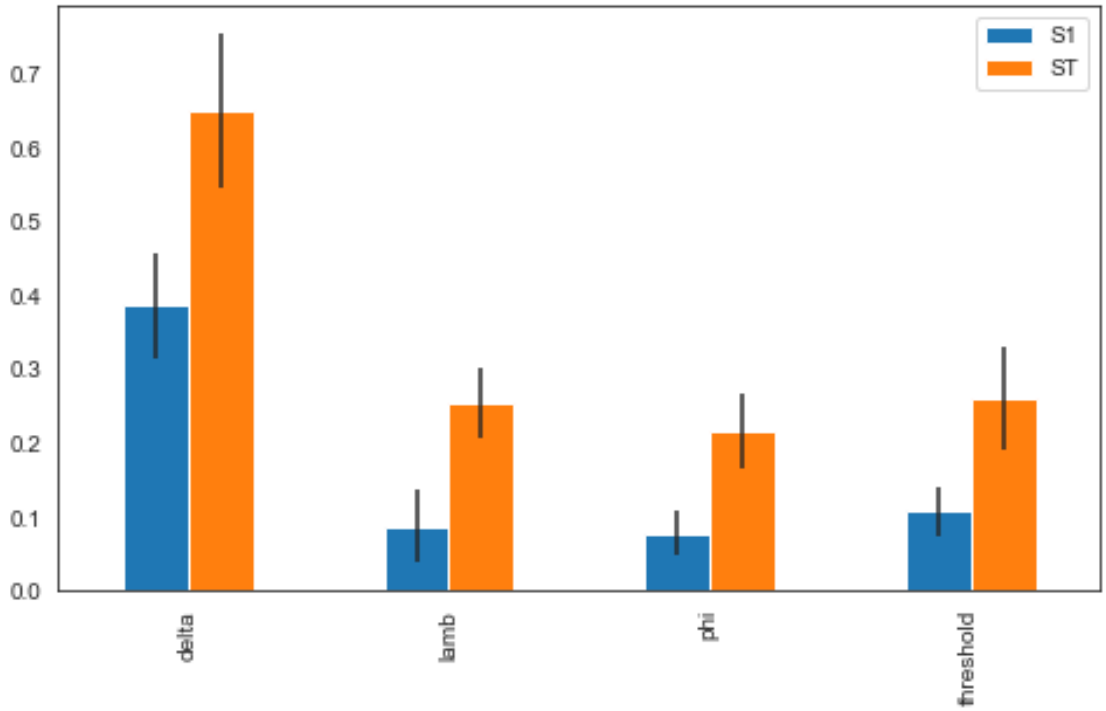
70: recall


S1
ST
0.8
0.6
0.4
0.2
0.0
delta
lamb
phi
threshold

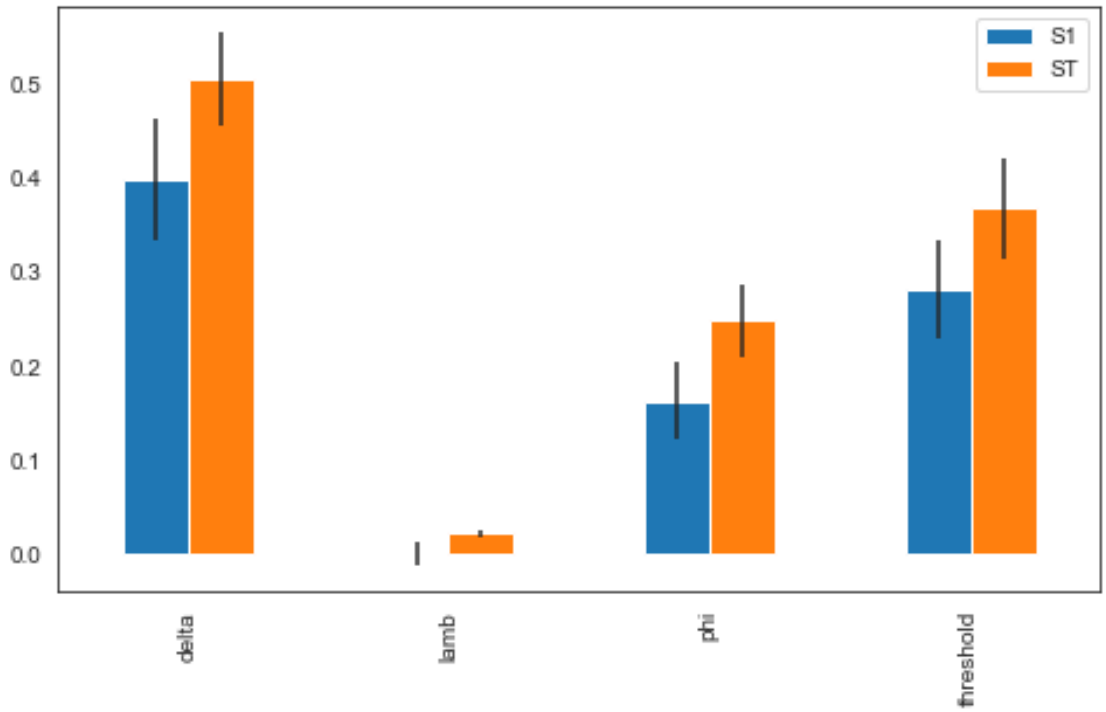
70: precision



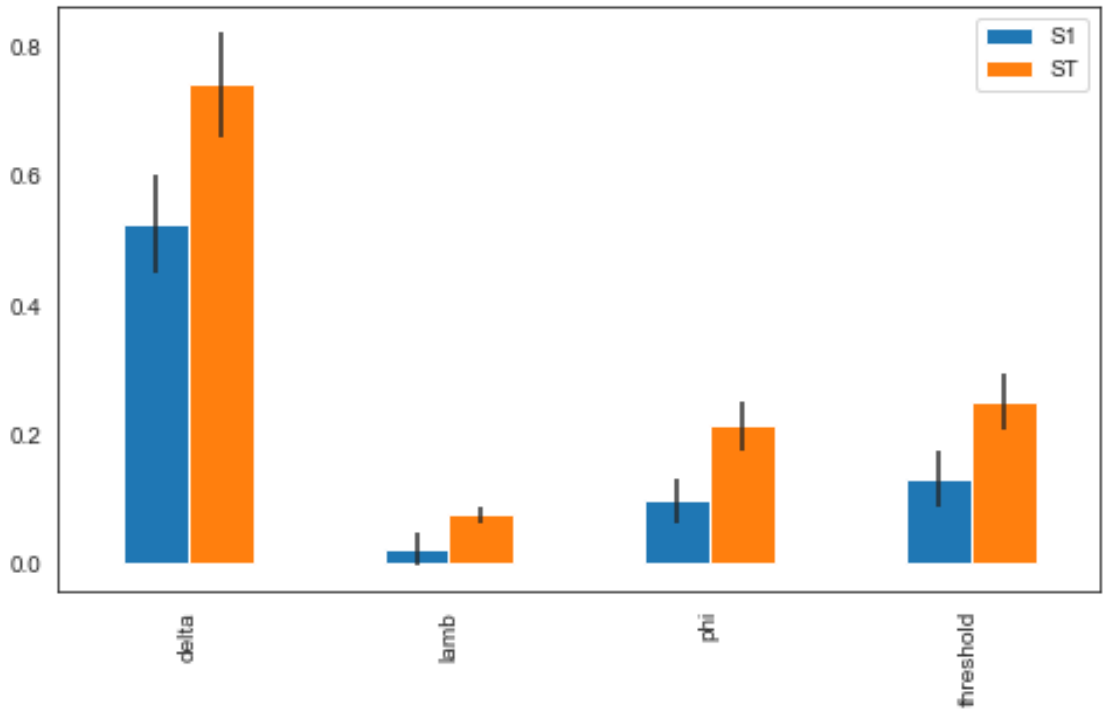
50: accuracy


S1
ST
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

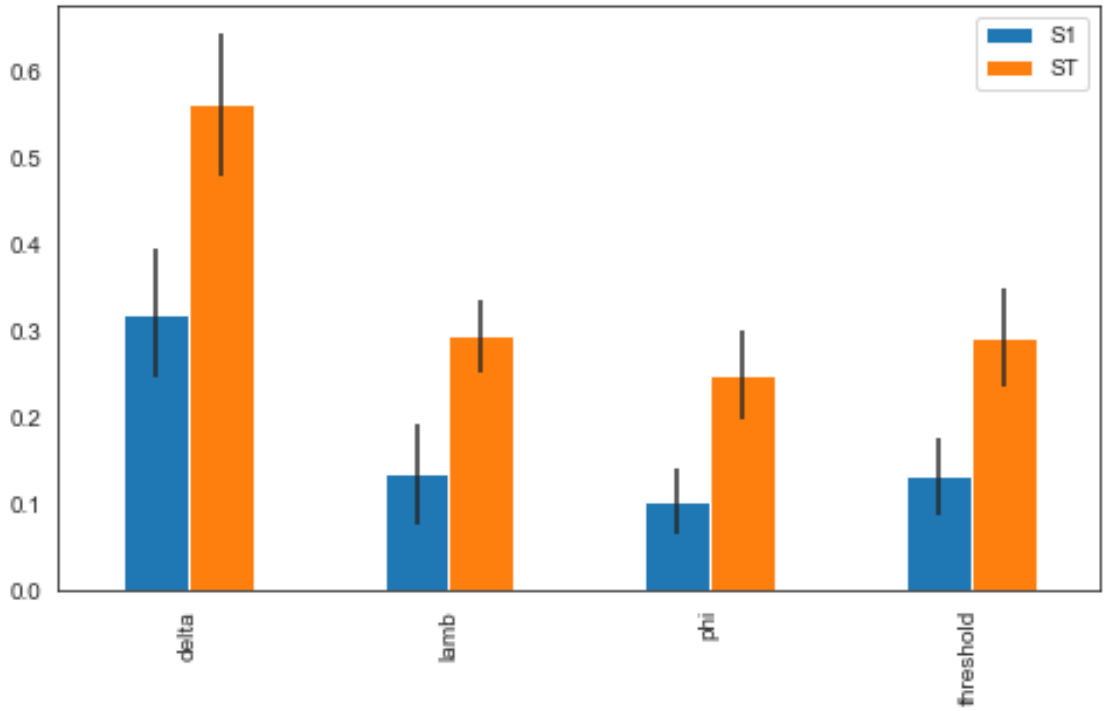
50: recall


S1
ST
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

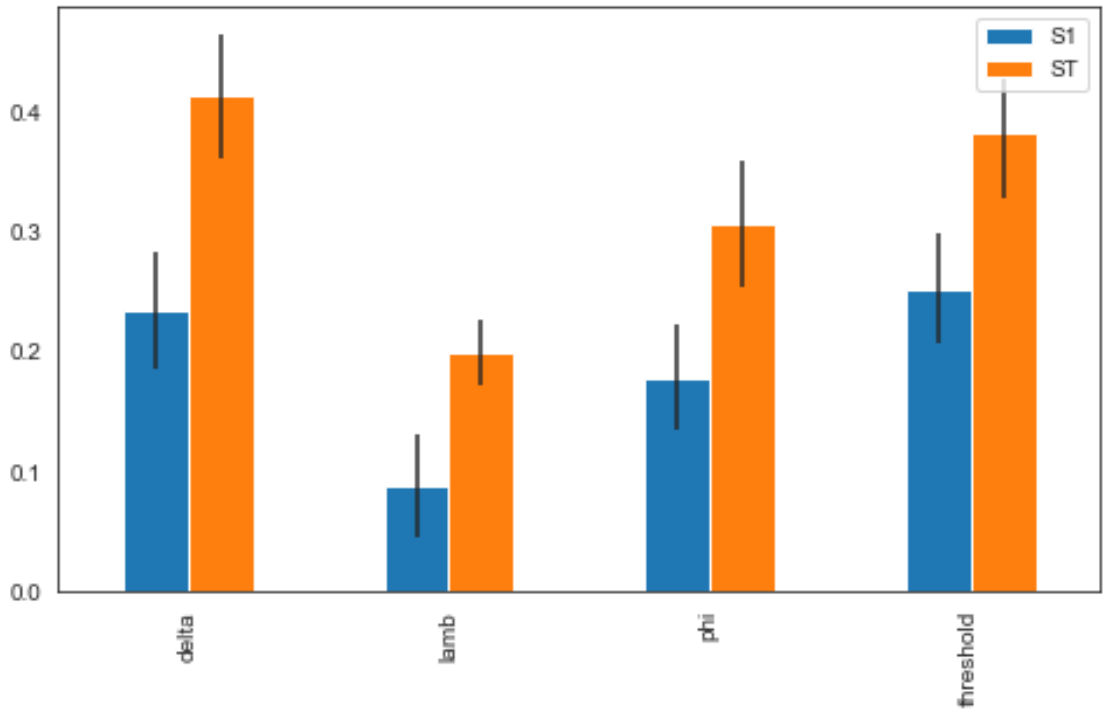
50: precision



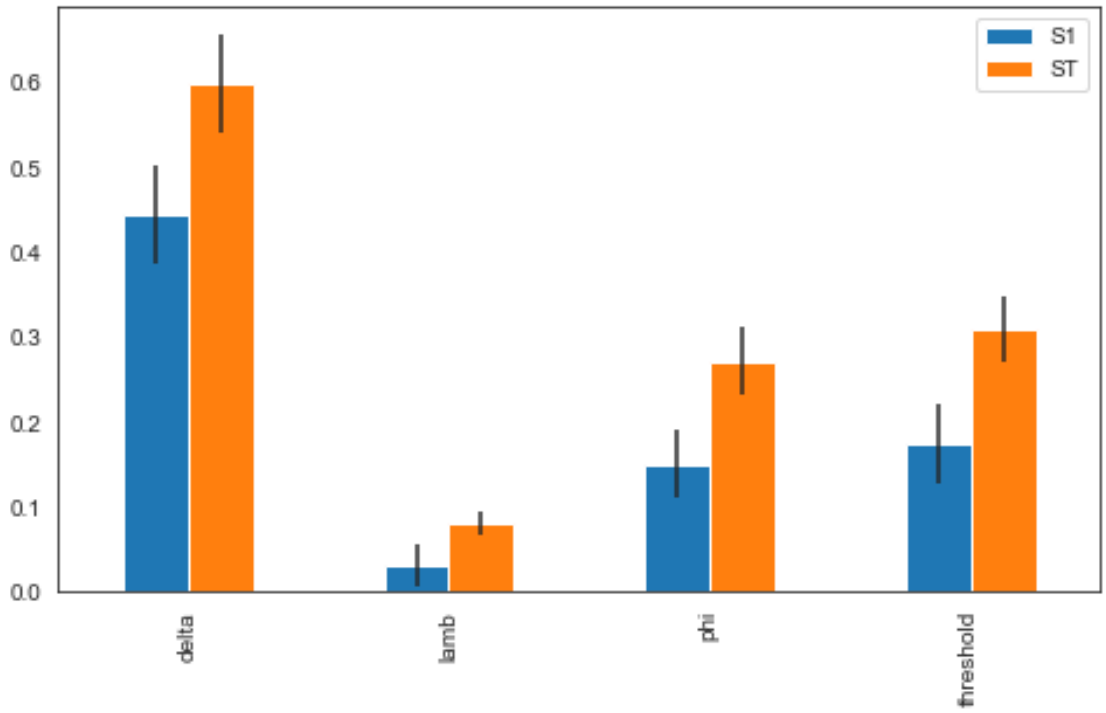
30: accuracy


S1
ST
0.6
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

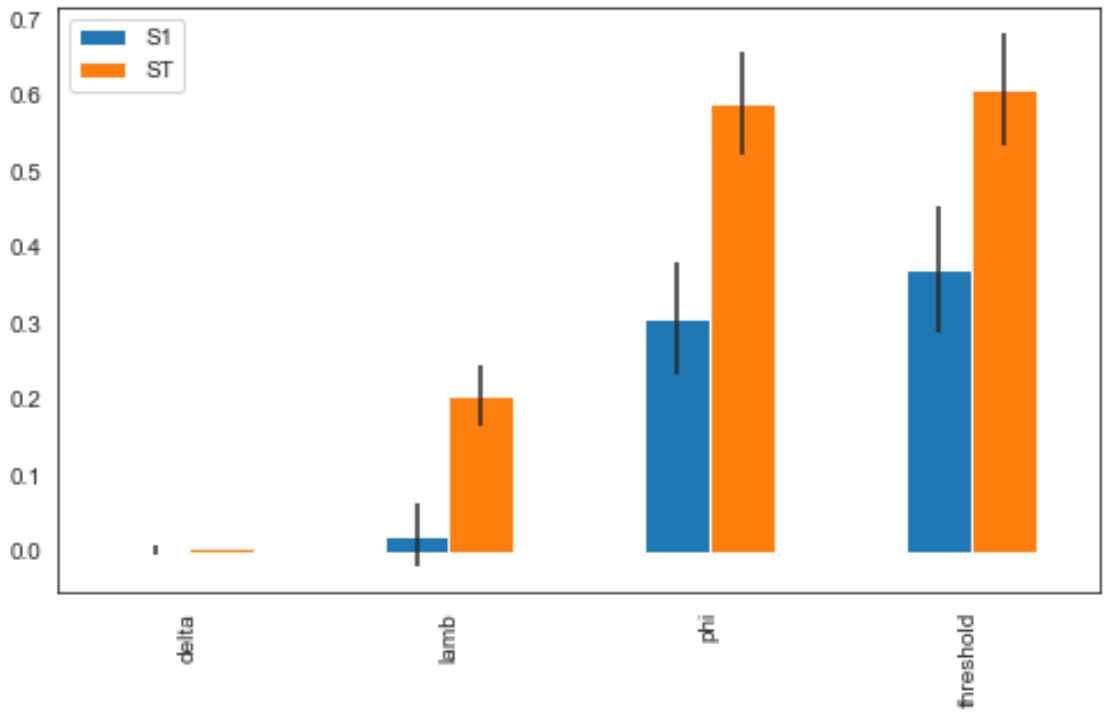
30: recall


S1
ST
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

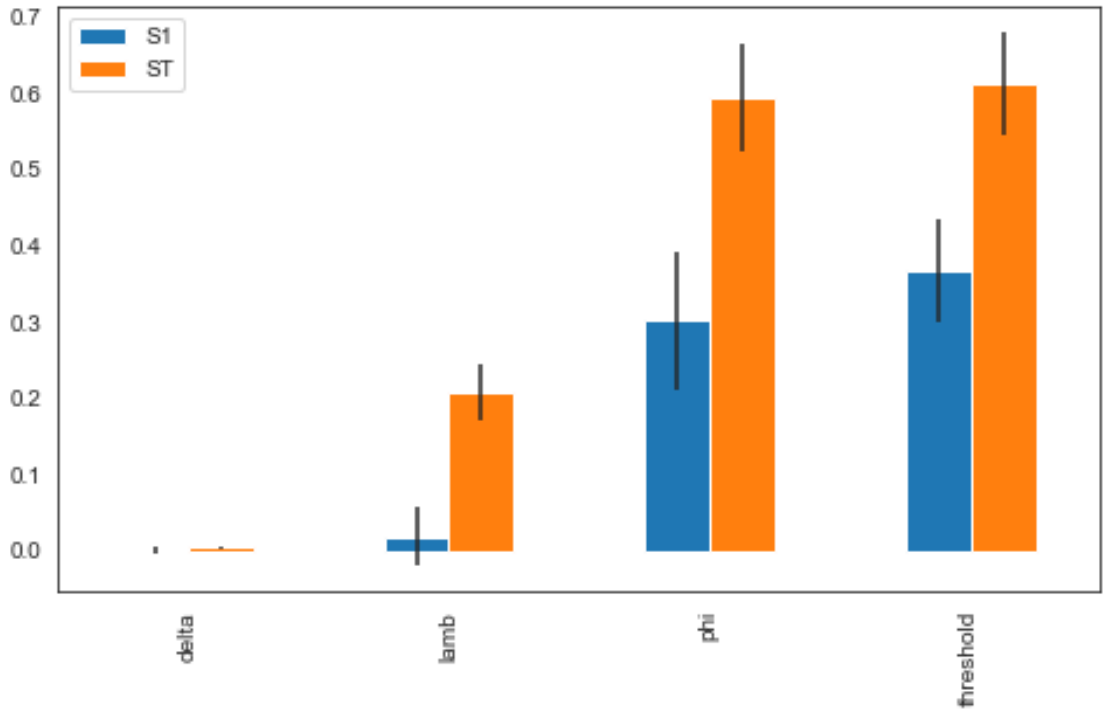
30: precision



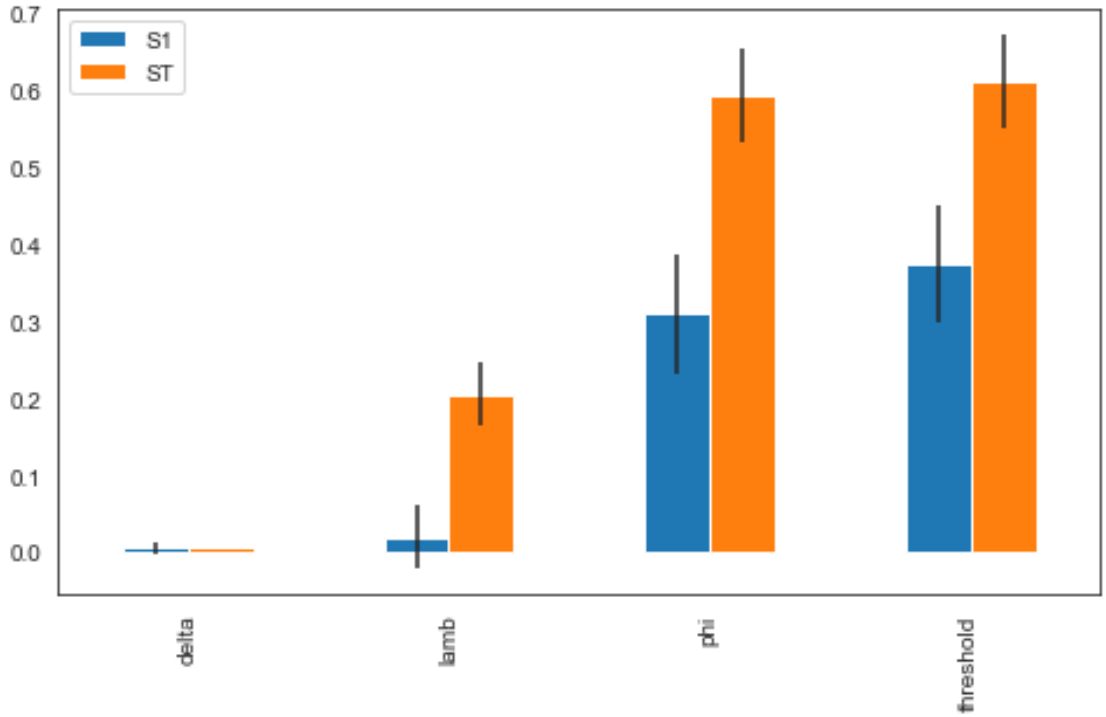
10: accuracy


S1
ST
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

10: recall


S1
ST
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

```
10: precision
```



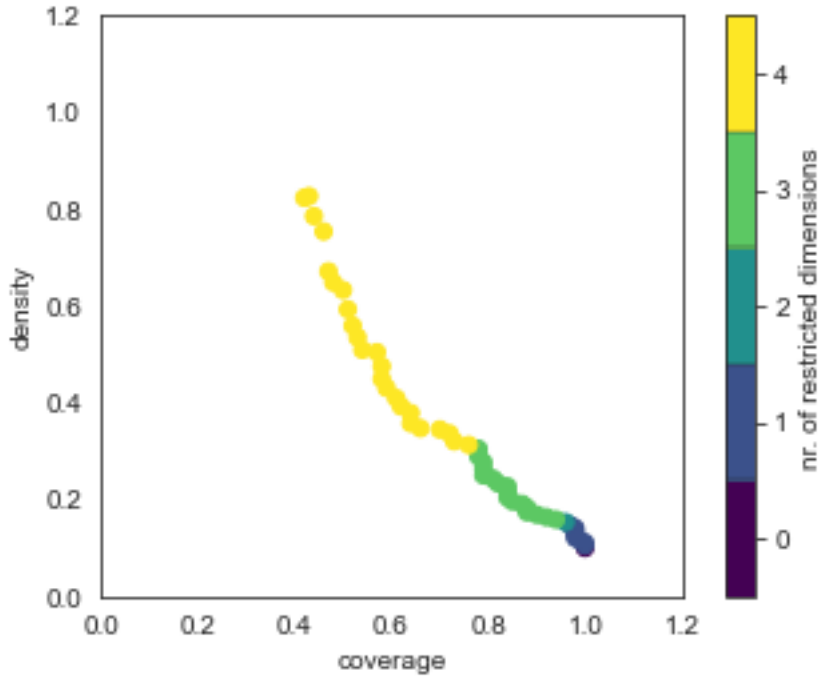
0.7 Show results for all domains

The above code blocks explain and show the details of sensitivity analysis using `blocks-world` domain as an example. The following codes show a SOBOL analysis results on `a_avg` for 15 domains (including blocks-world). And also show the parameter ranges using PRIM algorithm.

[11]:
```
# There are 15 domains

domain_list = ["blocks-world", "campus", "depots", "driverlog", "dwr",␣
 ↪"easy-ipc-grid", "ferry",
               "intrusion-detection", "kitchen", "logistics", "miconic",␣
 ↪"rovers", "satellite",
               "sokoban", "zeno-travel"]

# tweak and select a proper box_num for each domain
box_num_list = [50, 25, 55, 55, 50, 50, 50, 50, 30, 50, 50, 50, 50, 50, 50]

for i in range(15):
    domain_name = domain_list[i]
    box_num = box_num_list[i]
```

```
    print(domain_name)
    #PRIM
    results = load_results('./topk/1000_' + domain_name + '.tar.gz')
    a_avg = get_top(results, 0.1, "a_avg")

    results = load_results('./topk/1000_' + domain_name + '.tar.gz')
    box1 = find_box(results, 'a_avg', float(a_avg))
    show_ranges(box_num, box1)

    # SOBOL
    sobol_results = load_results('./topk/1050_' + domain_name + '.tar.gz')
    display_sobol_analysis_results_a_avg(sobol_results)
```

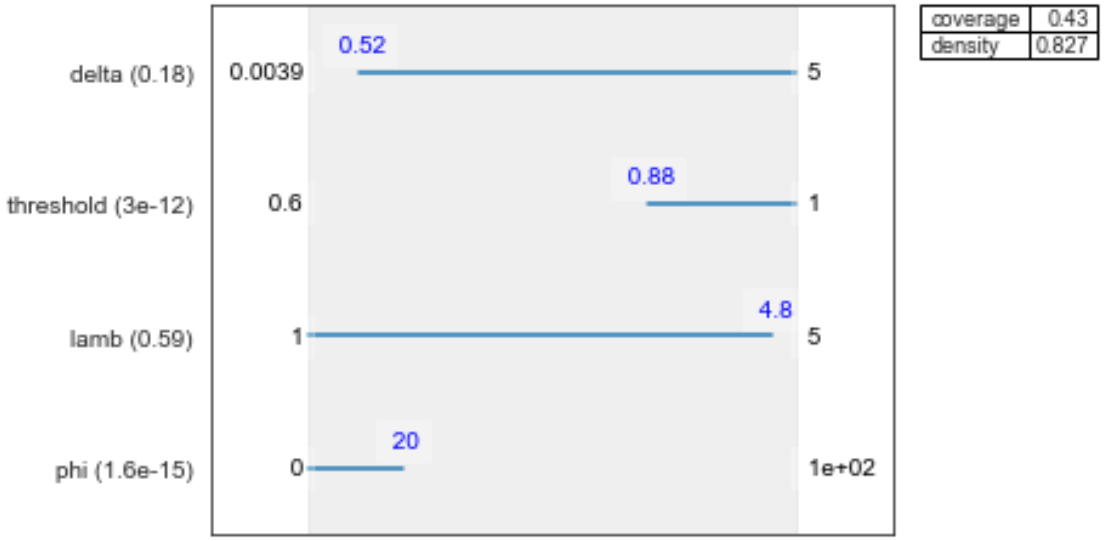
```
blocks-world
```

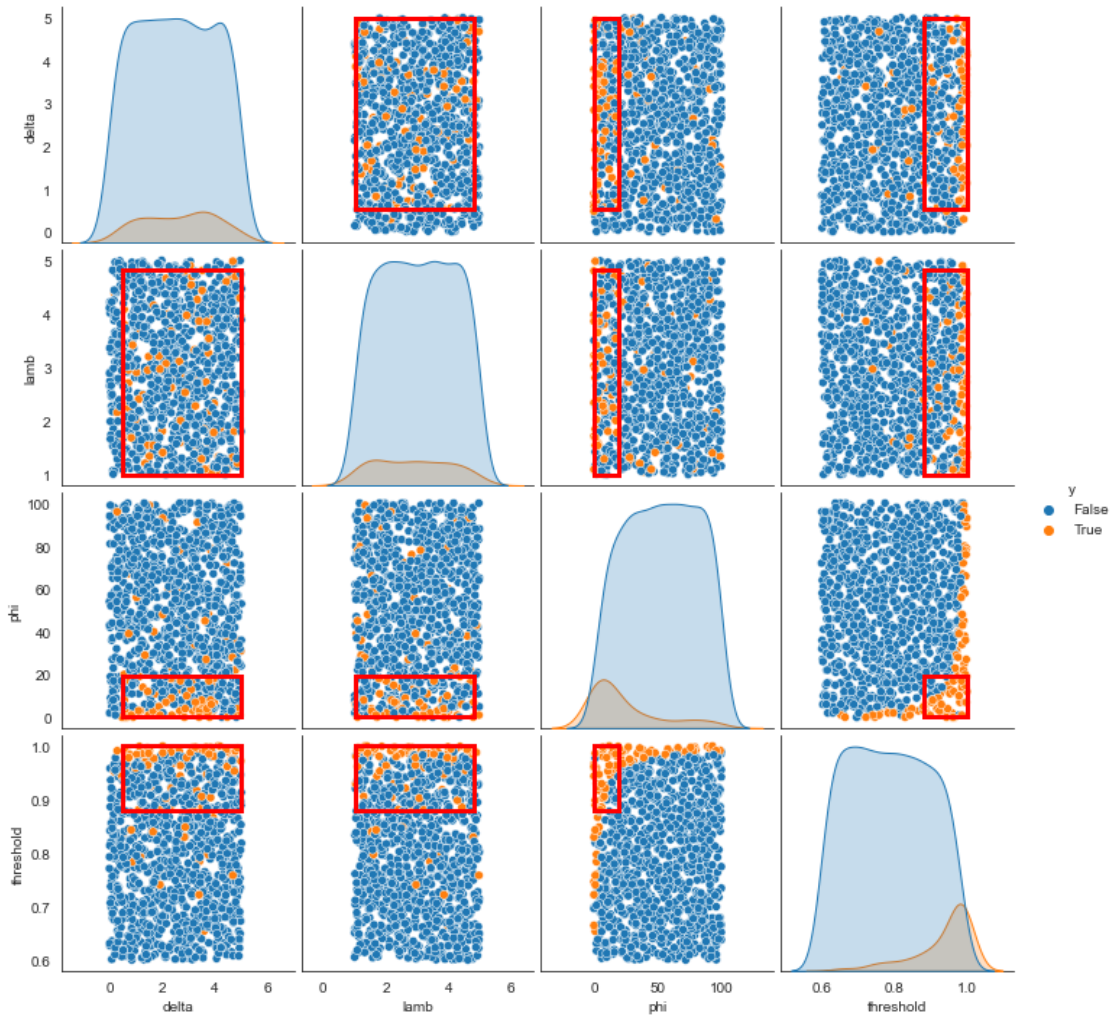



```
coverage        0.43
density     0.826923
id                50
mass           0.052
mean        0.826923
res_dim            4
Name: 50, dtype: object
```

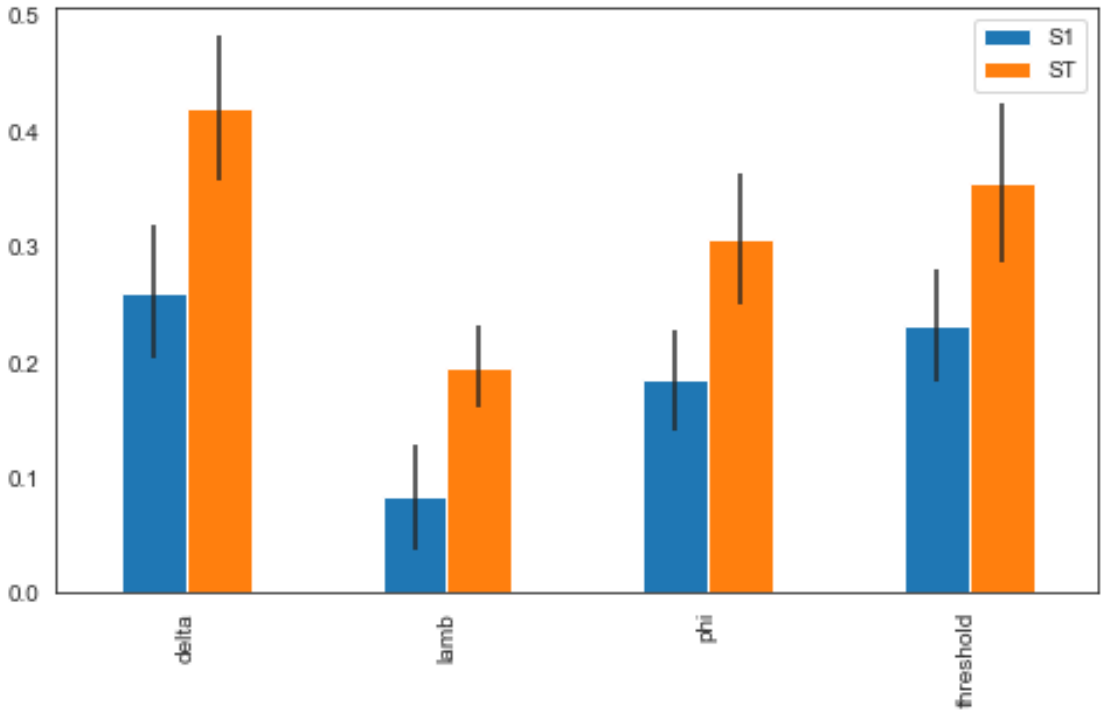
```
             box 50
```

	min	max	qp values
phi	0.000000	19.500000	[-1.0, 1.6188507007121986e-15]
lamb	1.003900	4.810672	[-1.0, 0.5878050850484972]
threshold	0.879567	0.999859	[3.0375970058123697e-12, -1.0]
delta	0.517886	4.997435	[0.17834844930969904, -1.0]

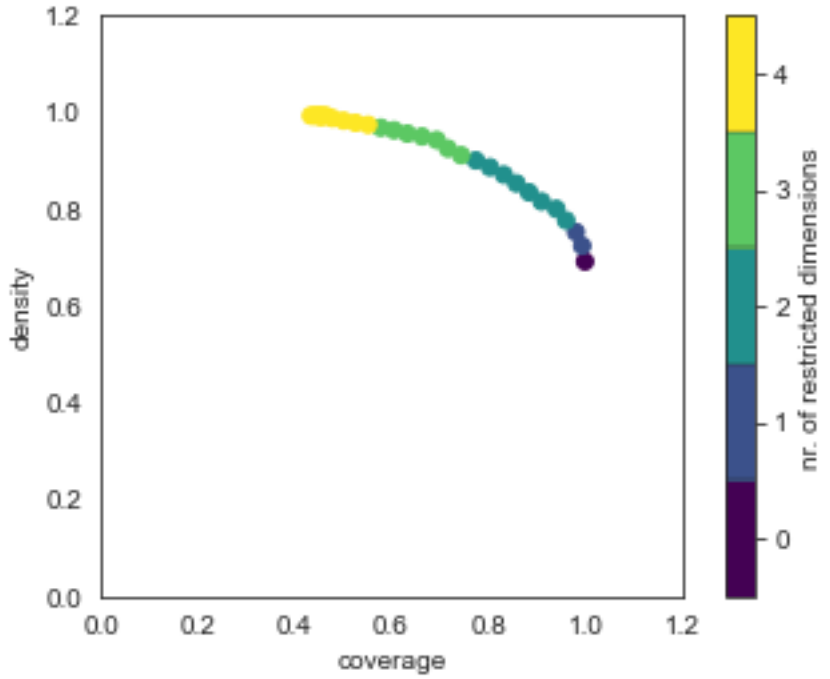




avg: accuracy

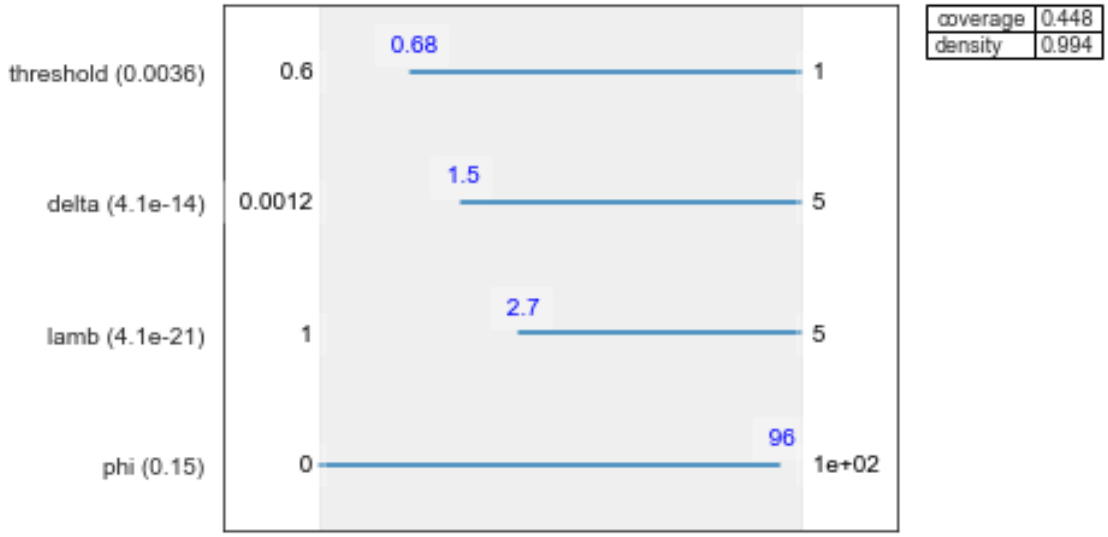

0.5
0.4
0.3
0.2
0.1
0.0
S1
ST
delta
lamb
phi
threshold

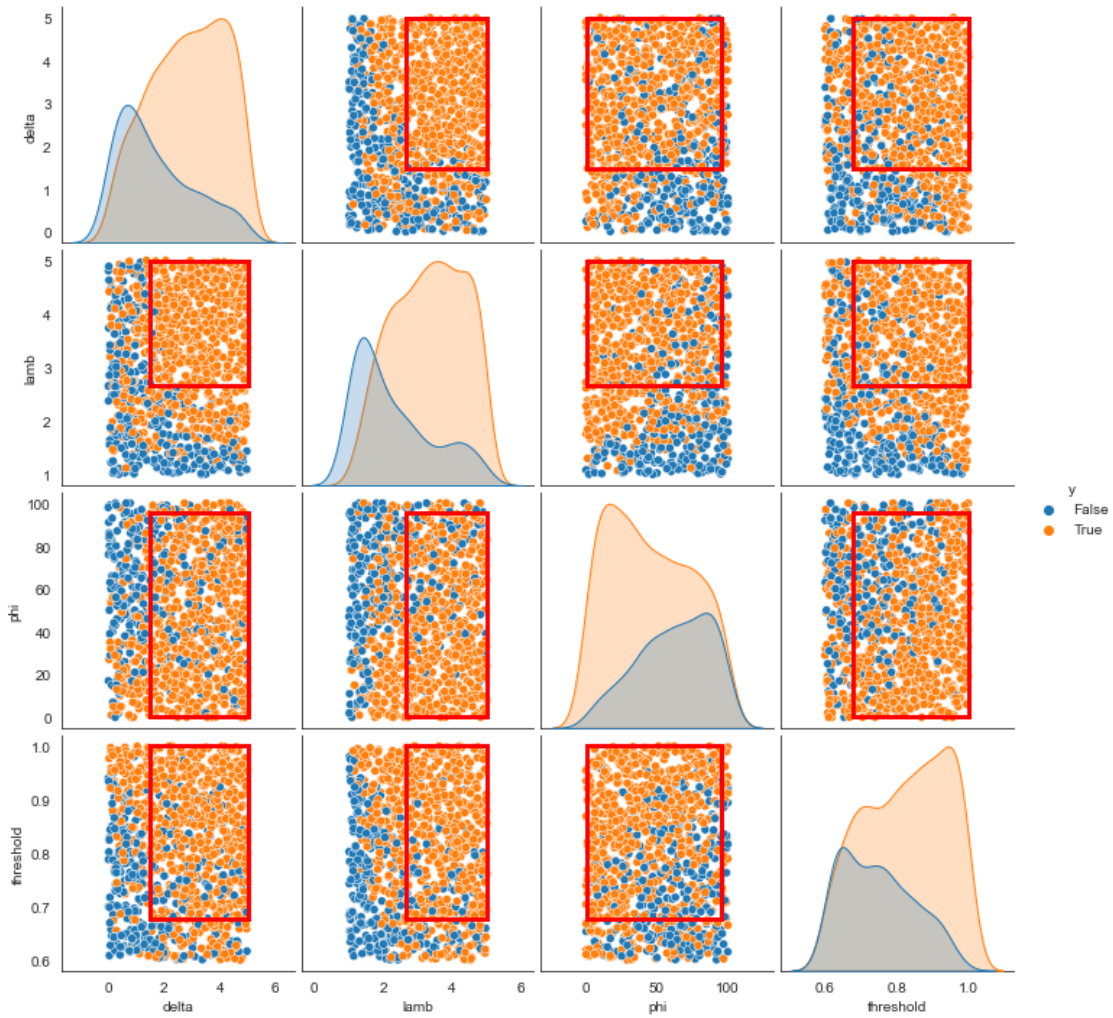
campus


1.2
1.0
0.8
0.6
0.4
0.2
0.0
density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
coverage
4
3
2
1
0
nr. of restricted dimensions

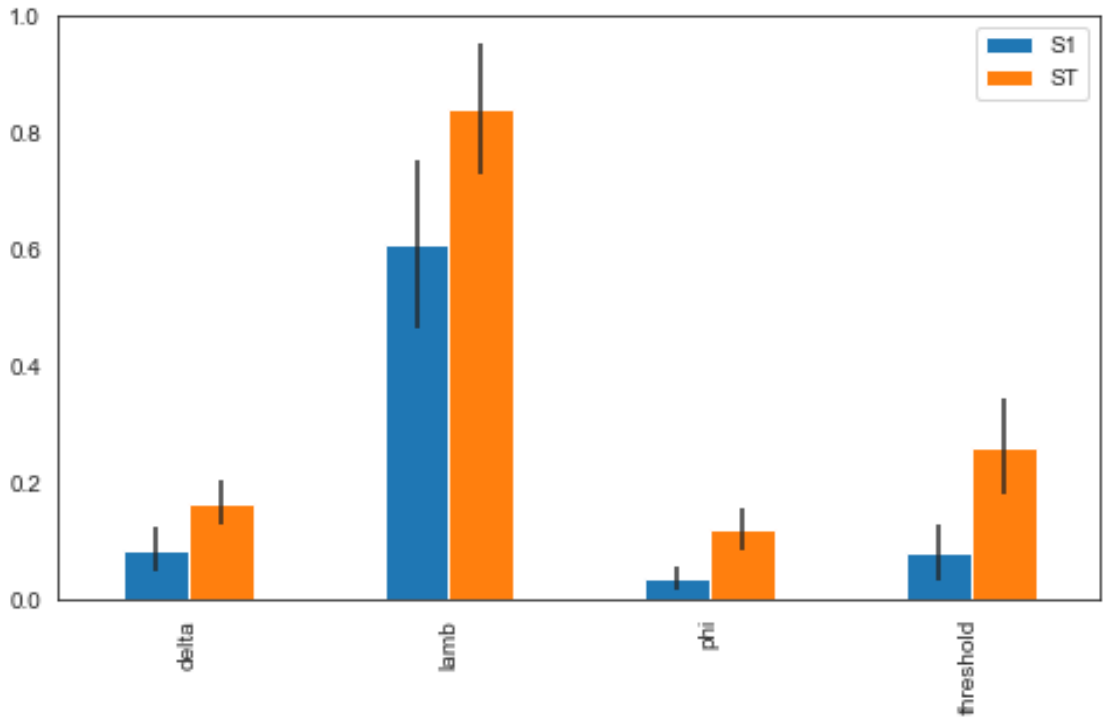
```
coverage    0.447977
density      0.99359
id                25
mass           0.312
mean         0.99359
res_dim            4
Name: 25, dtype: object
```

	box 25		
	min	max	qp values
phi	0.000000	95.500000	[-1.0, 0.15036965222897253]
lamb	2.667136	4.998111	[4.095205492809191e-21, -1.0]
delta	1.472077	4.999669	[4.126765009434943e-14, -1.0]
threshold	0.677180	0.999732	[0.003555642481970172, -1.0]

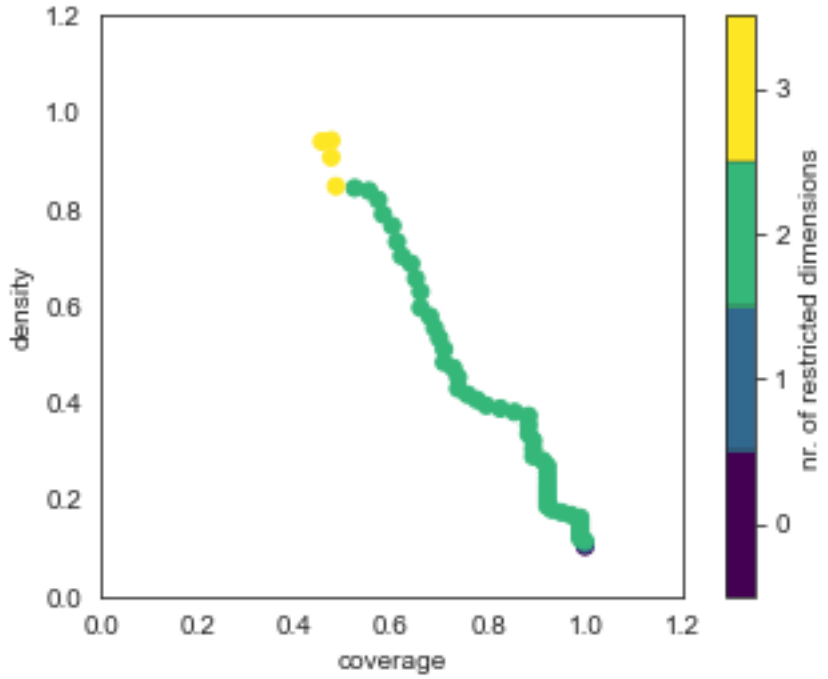




avg: accuracy

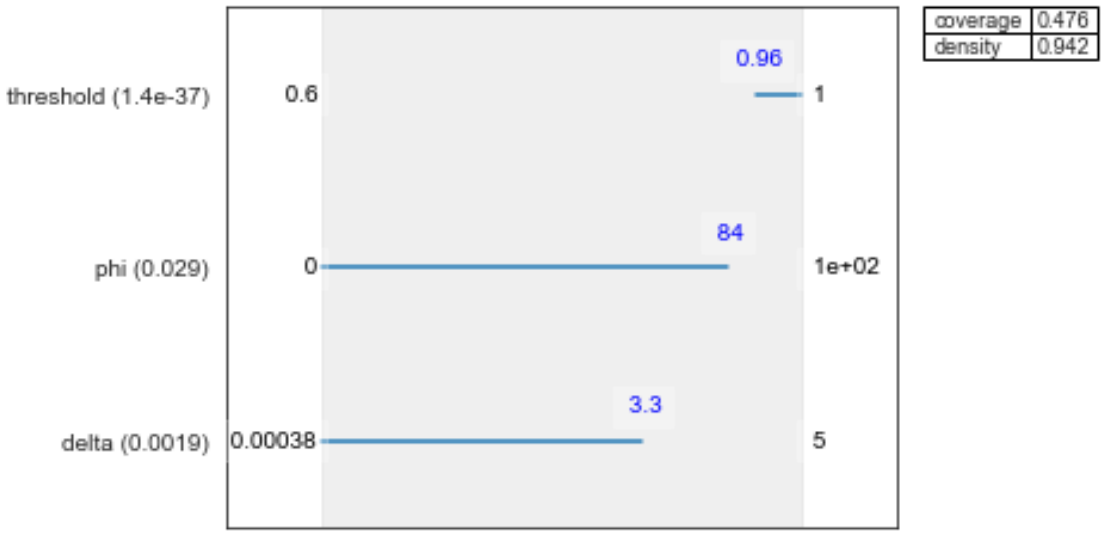

1.0
0.8
0.6
0.4
0.2
0.0
S1
ST
delta
lamb
phi
threshold

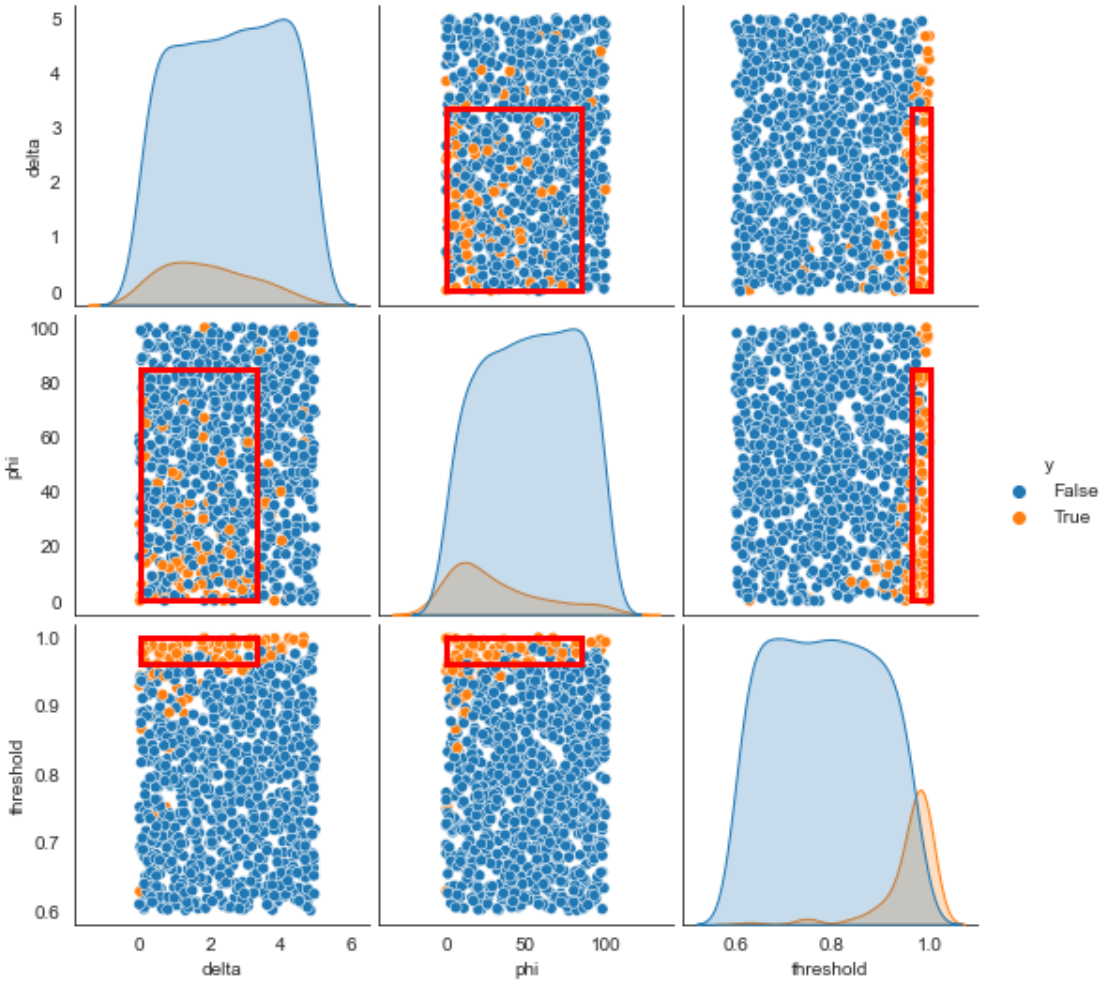
depots


1.2
1.0
0.8
0.6
0.4
0.2
0.0
density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
coverage
3
2
1
0
nr. of restricted dimensions

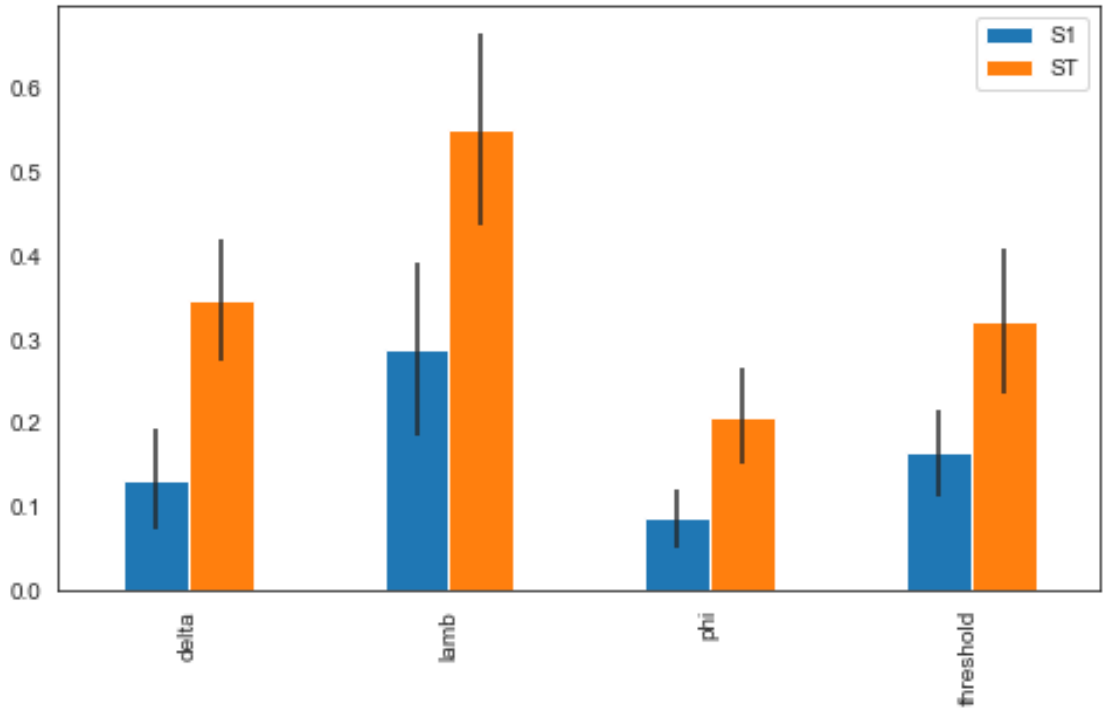
```
coverage    0.475728
density     0.942308
id                55
mass           0.052
mean        0.942308
res_dim            3
Name: 55, dtype: object
```

box 55	min	max	qp values
delta	0.000384	3.345904	[-1.0, 0.0019232542124863446]
phi	0.000000	84.500000	[-1.0, 0.028600541025970174]
threshold	0.962366	0.999900	[1.398681815586161e-37, -1.0]

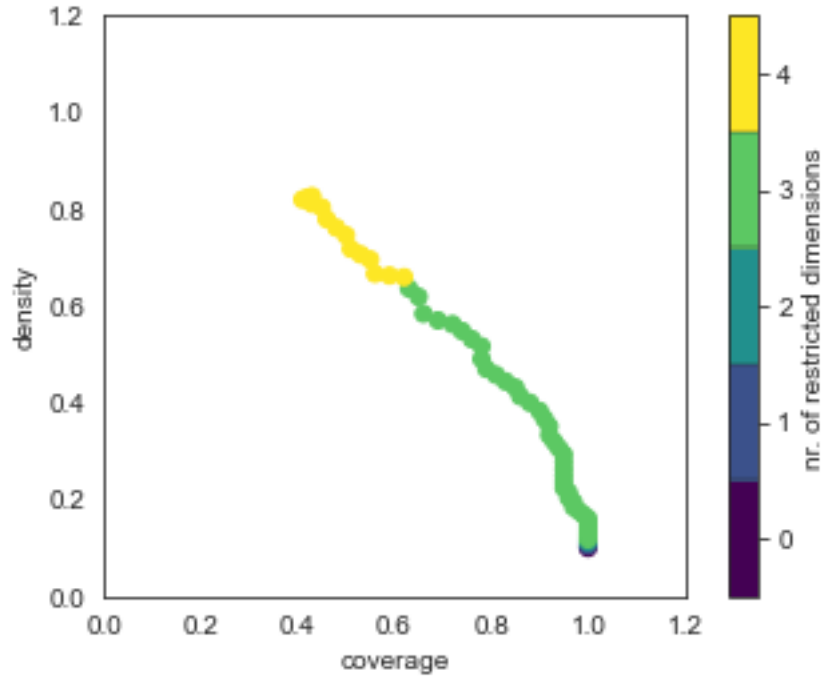





```
avg: accuracy
```

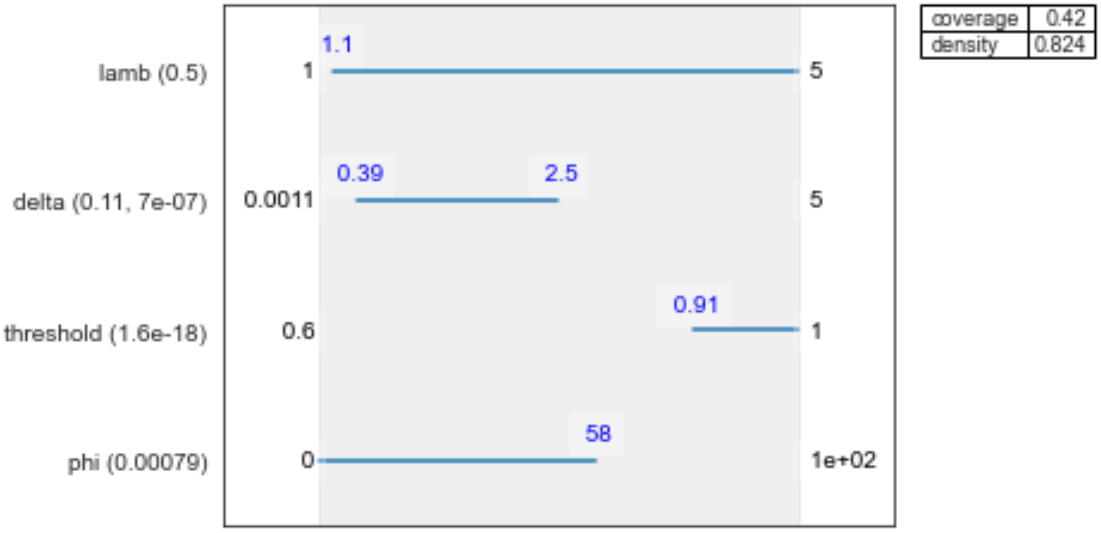

S1
ST
0.0
0.1
0.2
0.3
0.4
0.5
0.6
delta
lamb
phi
threshold

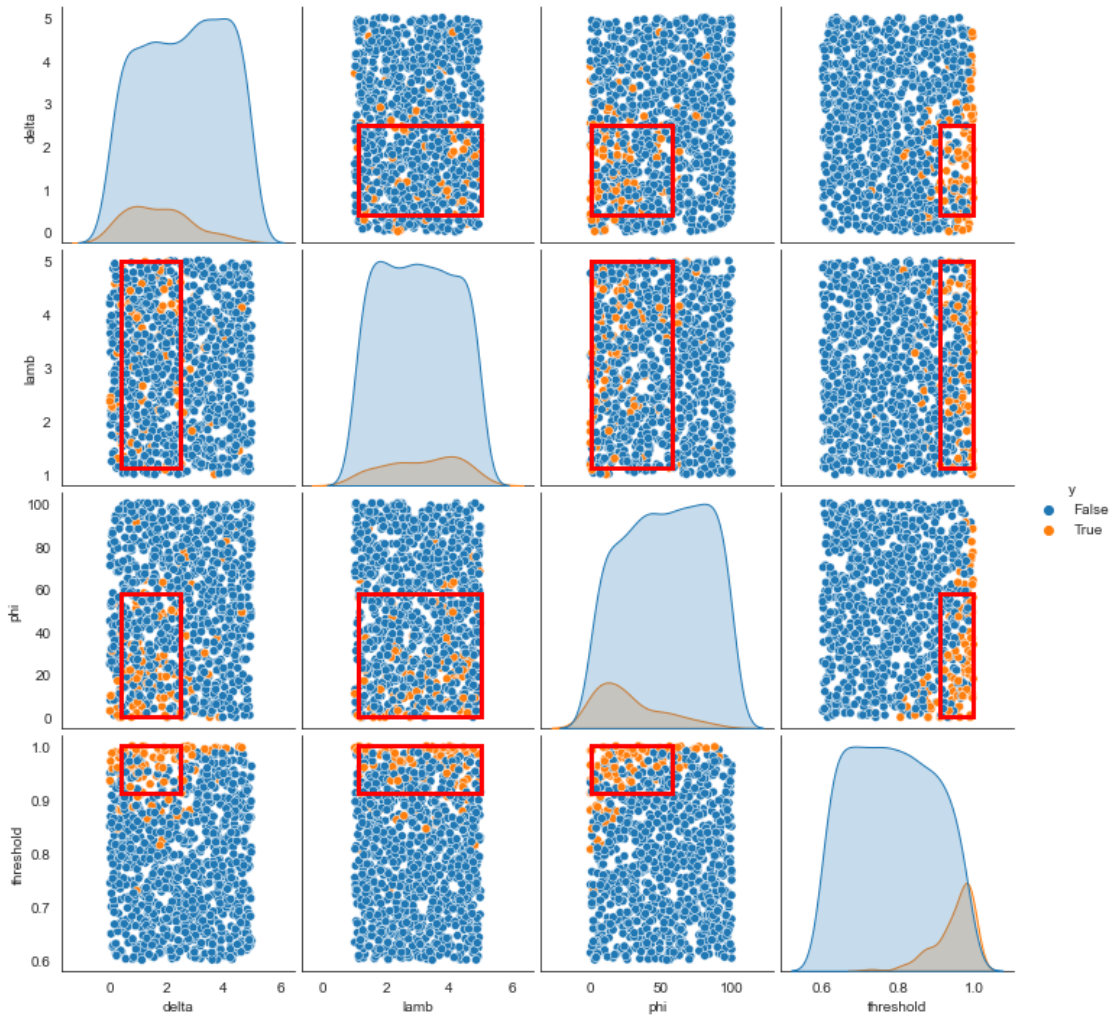
driverlog


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4

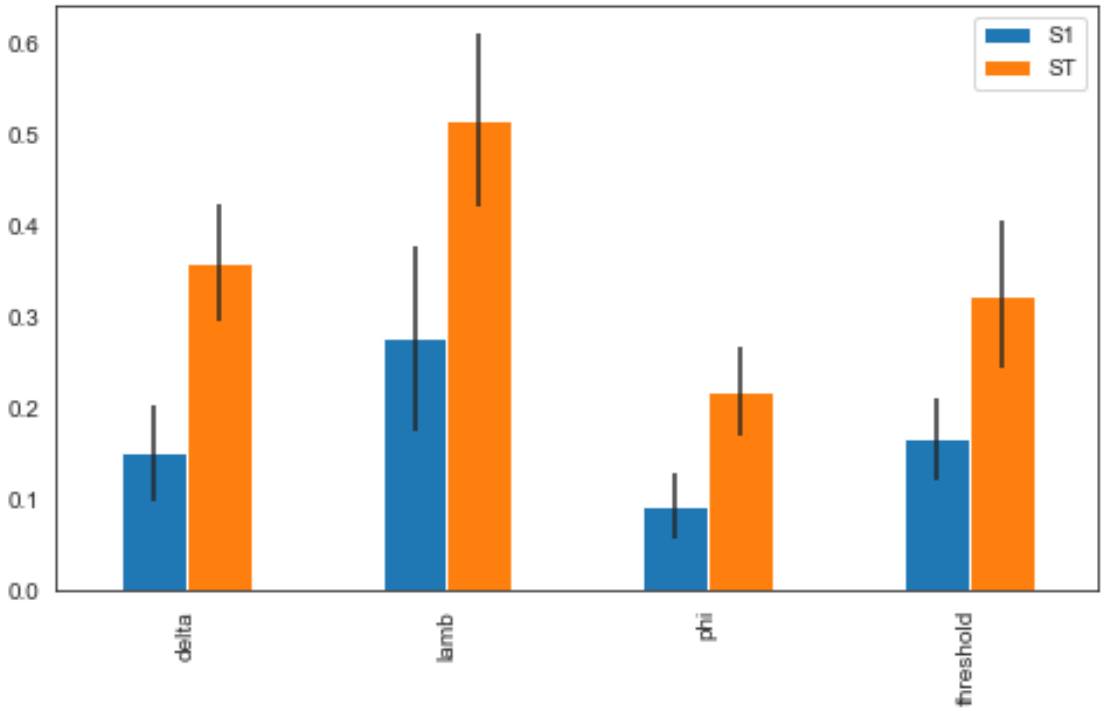
```
coverage        0.42
density     0.823529
id                55
mass           0.051
mean        0.823529
res_dim            4
Name: 55, dtype: object
```

box 55	min	max	qp values
phi	0.000000	57.500000	[-1.0, 0.0007902148082662998]
threshold	0.912177	0.999646	[1.6406392086458046e-18, -1.0]
delta	0.393675	2.479597	[0.10963883066482047, 6.952570541033969e-07]
lamb	1.118845	4.998966	[0.49754775586406597, -1.0]

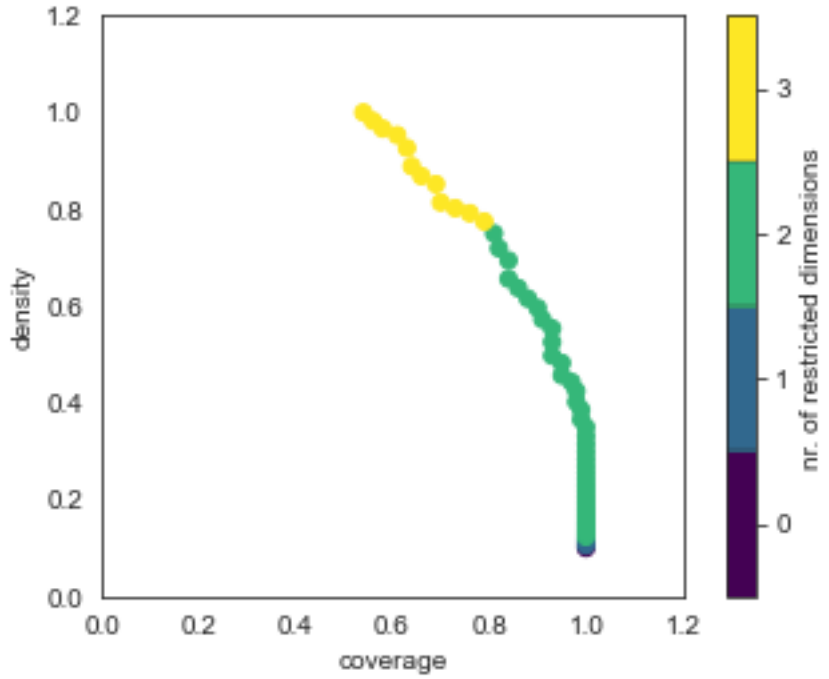




avg: accuracy

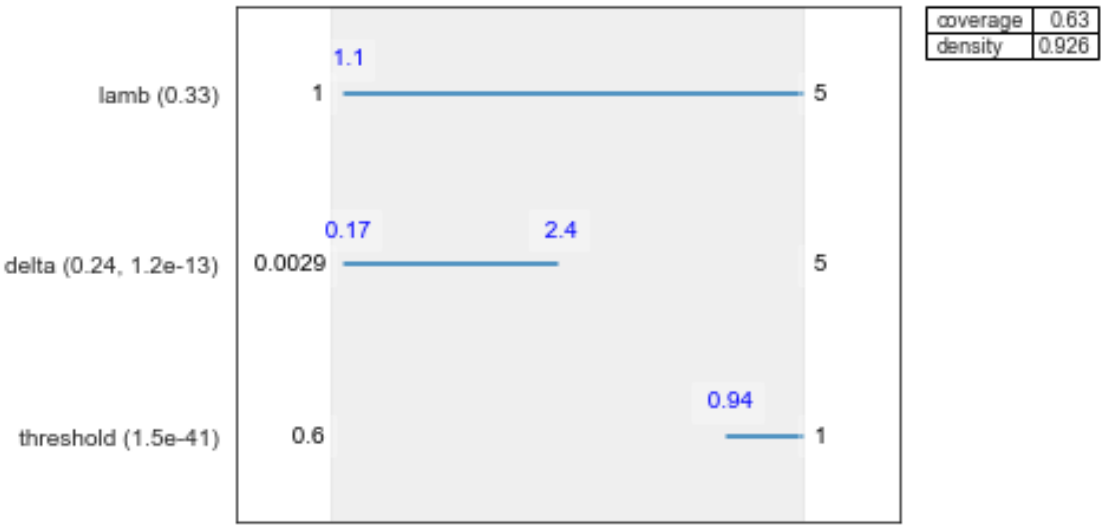

S1
ST
0.6
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

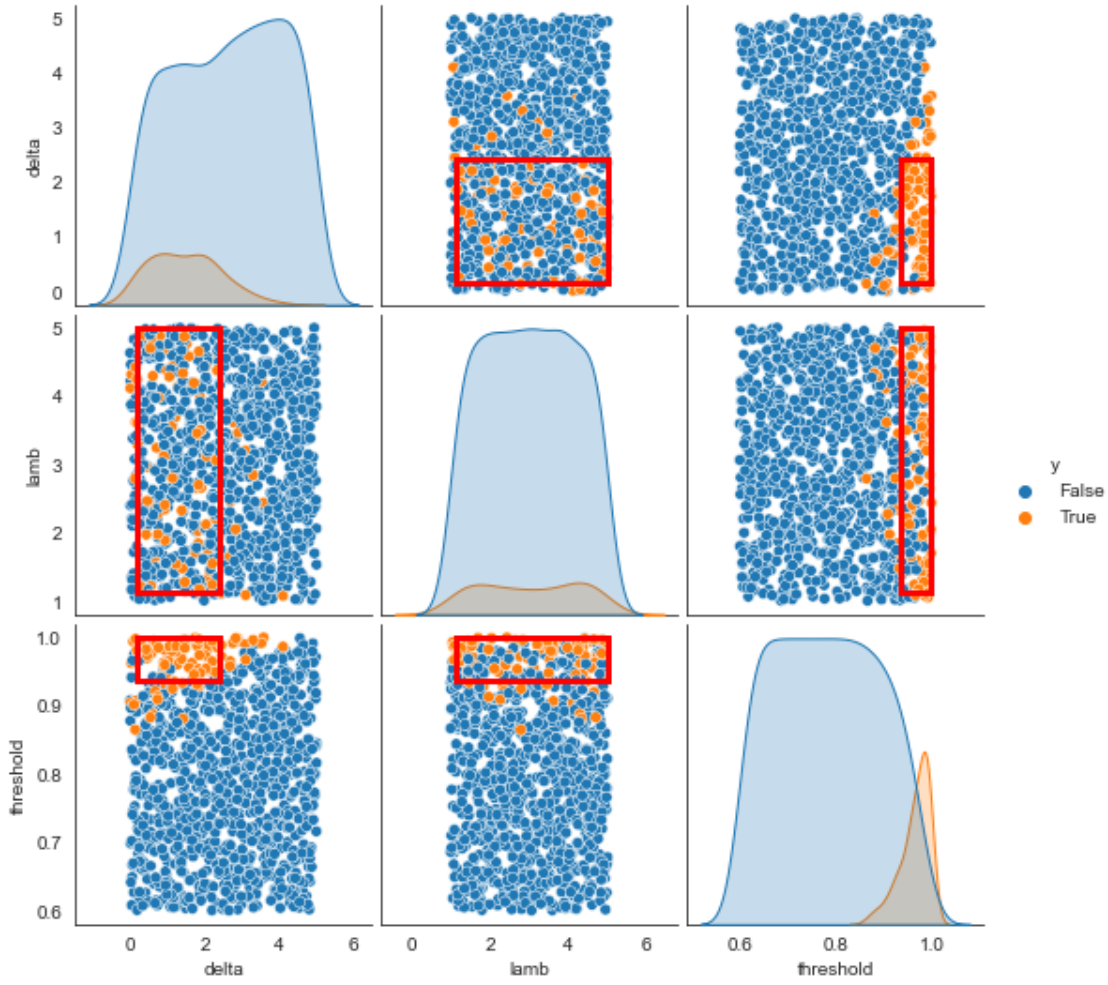
dwr


1.2
1.0
0.8
0.6
0.4
0.2
0.0
density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
coverage
3
2
1
0
nr. of restricted dimensions

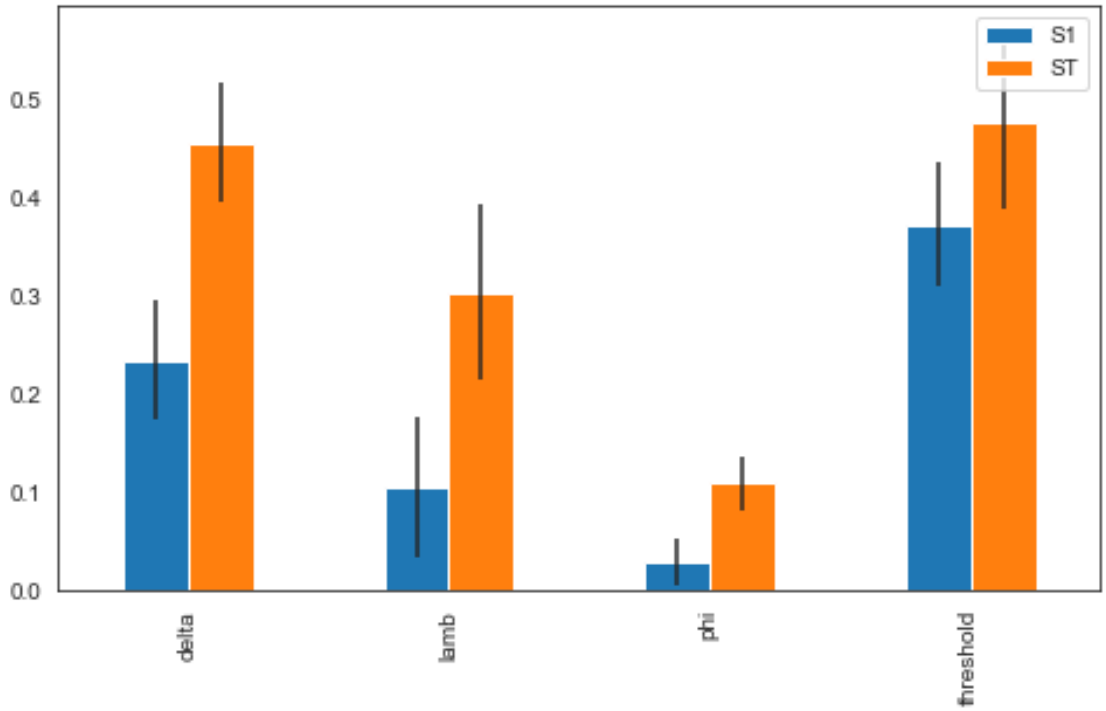
```
coverage        0.63
density     0.926471
id                50
mass           0.068
mean        0.926471
res_dim            3
Name: 50, dtype: object
```

	box 50		
	min	max	qp values
threshold	0.935421	0.999857	[1.4610288370512375e-41, -1.0]
delta	0.165411	2.406049	[0.24268535678601247, 1.2363838764110285e-13]
lamb	1.124228	4.996235	[0.32776936949069213, -1.0]

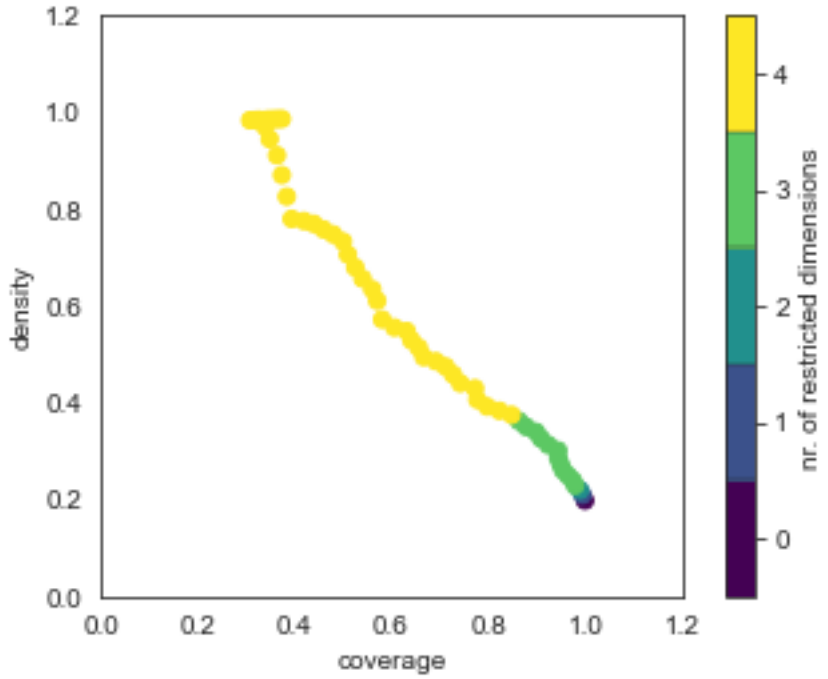





```
avg: accuracy
```

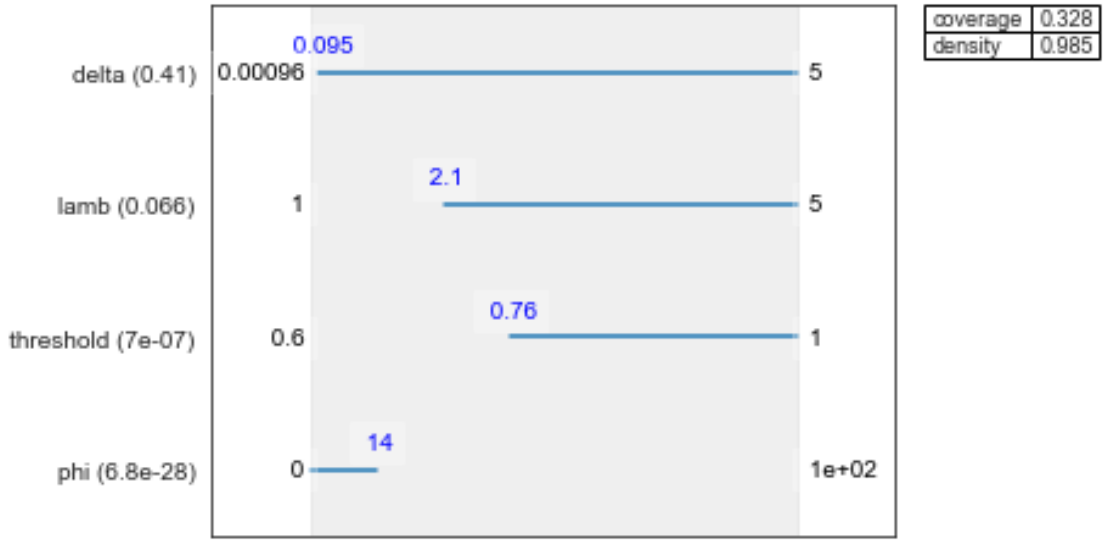


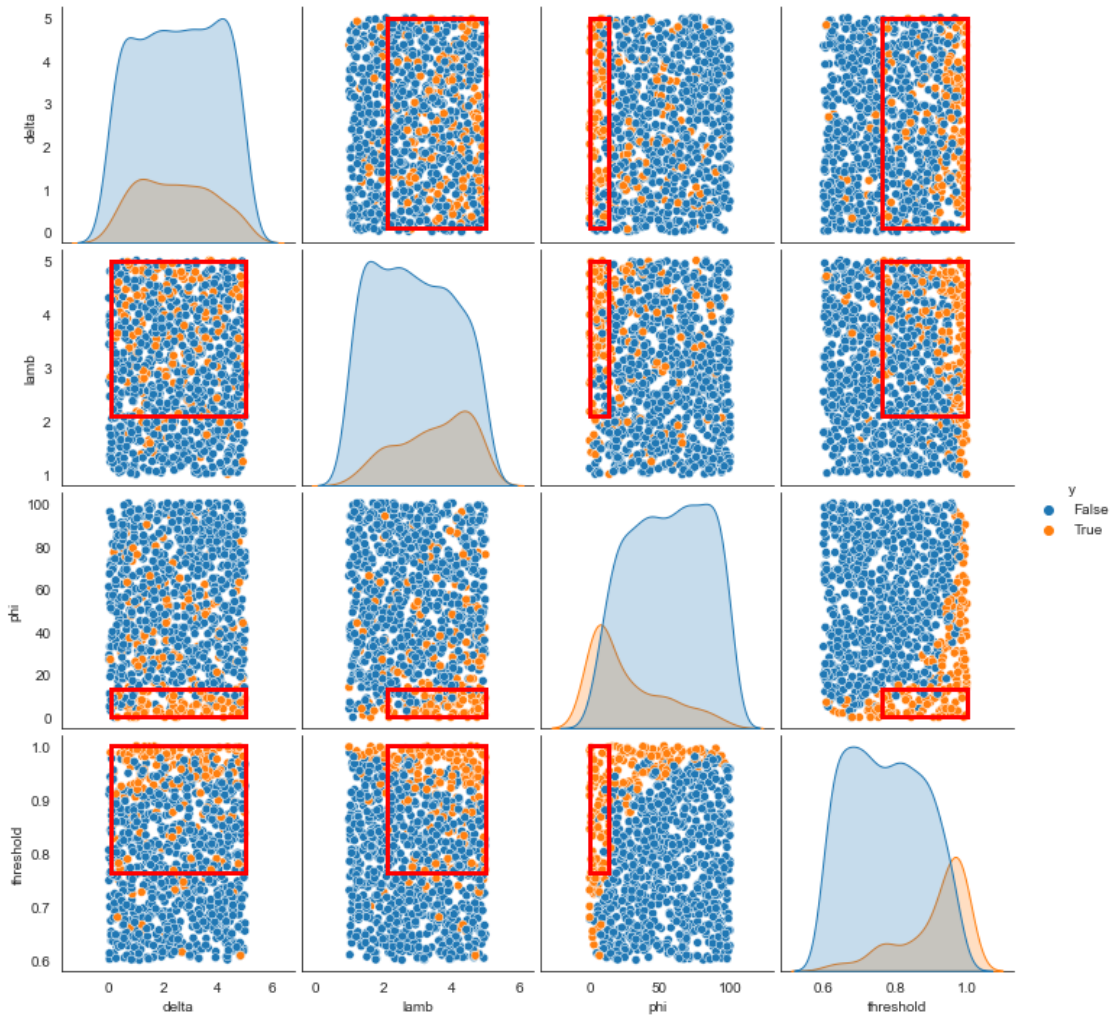
`easy-ipc-grid`



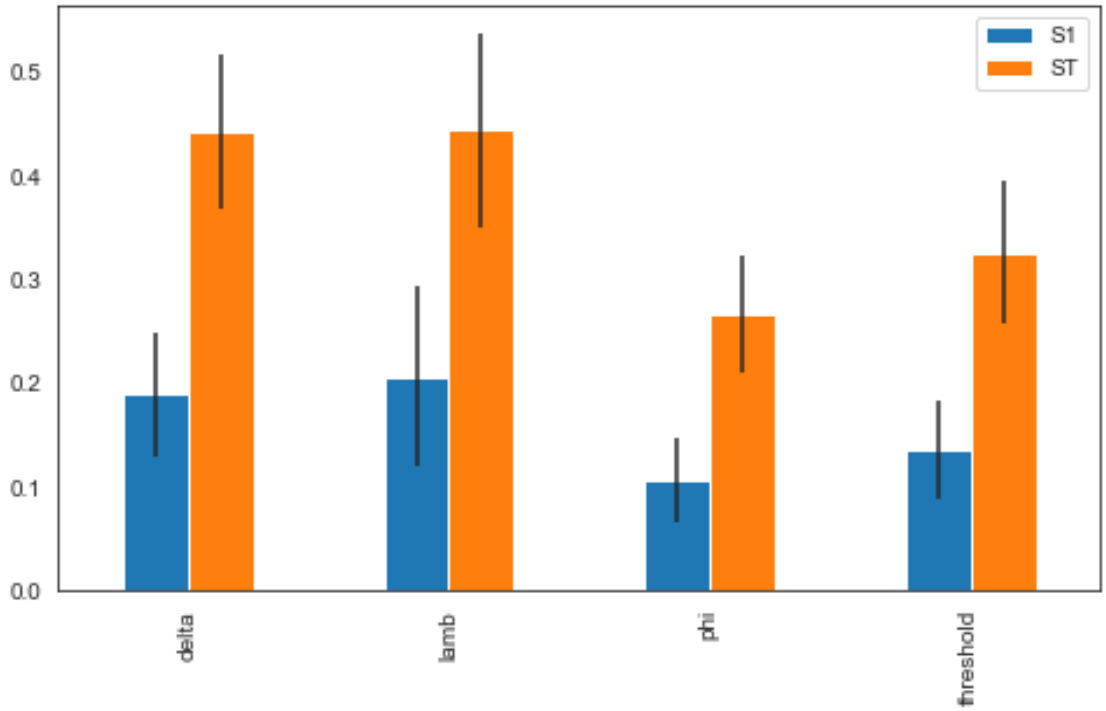
```
coverage    0.328283
density     0.984848
id                50
mass           0.066
mean        0.984848
res_dim            4
Name: 50, dtype: object
```

	box 50		
	min	max	qp values
phi	0.000000	13.500000	[-1.0, 6.821697812702756e-28]
threshold	0.763970	0.999991	[7.031594586001354e-07, -1.0]
lamb	2.100676	4.997264	[0.0664431876388809, -1.0]
delta	0.094708	4.996166	[0.4101072337089218, -1.0]

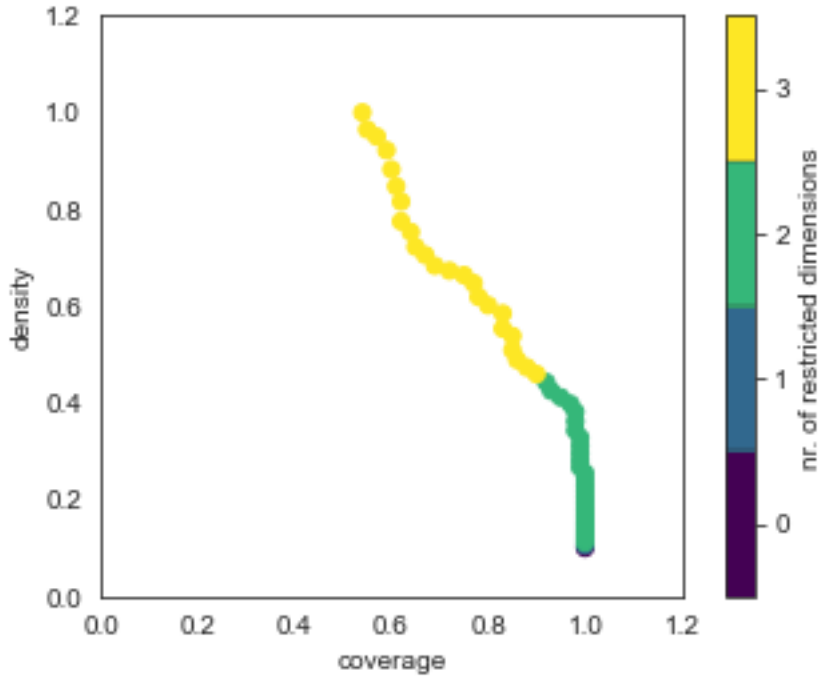





```
avg: accuracy
```

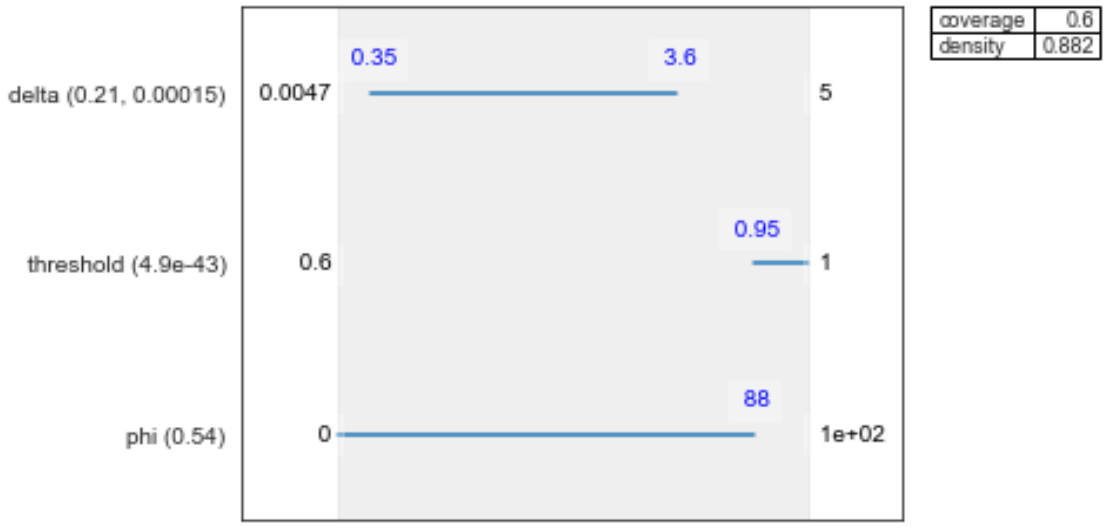

S1
ST
0.0
0.1
0.2
0.3
0.4
0.5
delta
lamb
phi
threshold

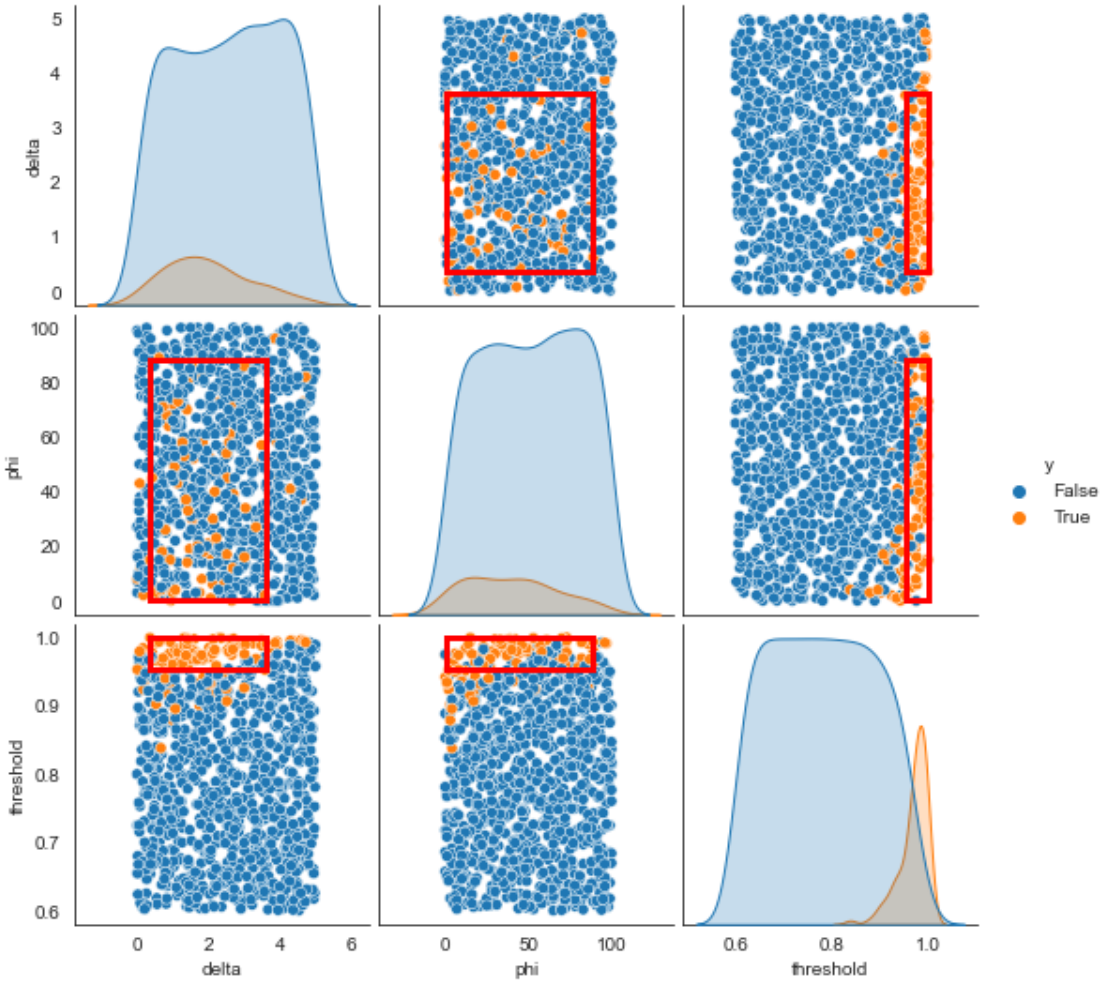
ferry


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3

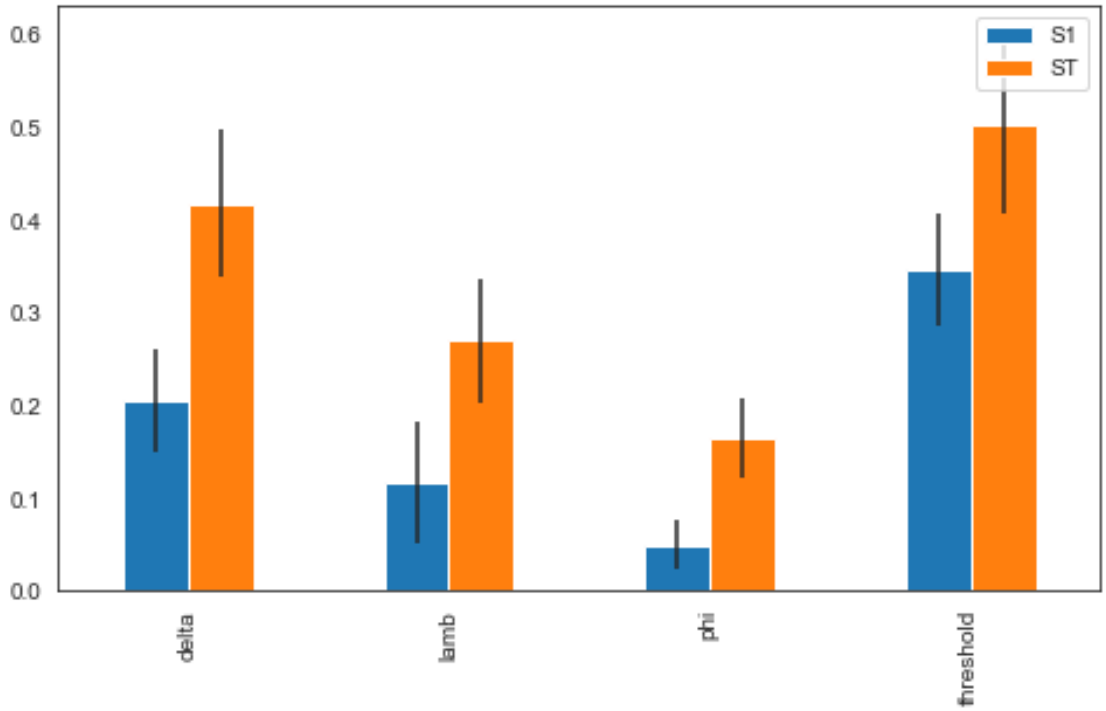
```
coverage         0.6
density     0.882353
id                50
mass           0.068
mean        0.882353
res_dim            3
Name: 50, dtype: object
```

box 50	min	max	qp values
phi	0.000000	88.500000	[-1.0, 0.535752582852719]
threshold	0.952656	0.999630	[4.857133270393899e-43, -1.0]
delta	0.353303	3.610603	[0.20611344542981067, 0.00014772758880682464]

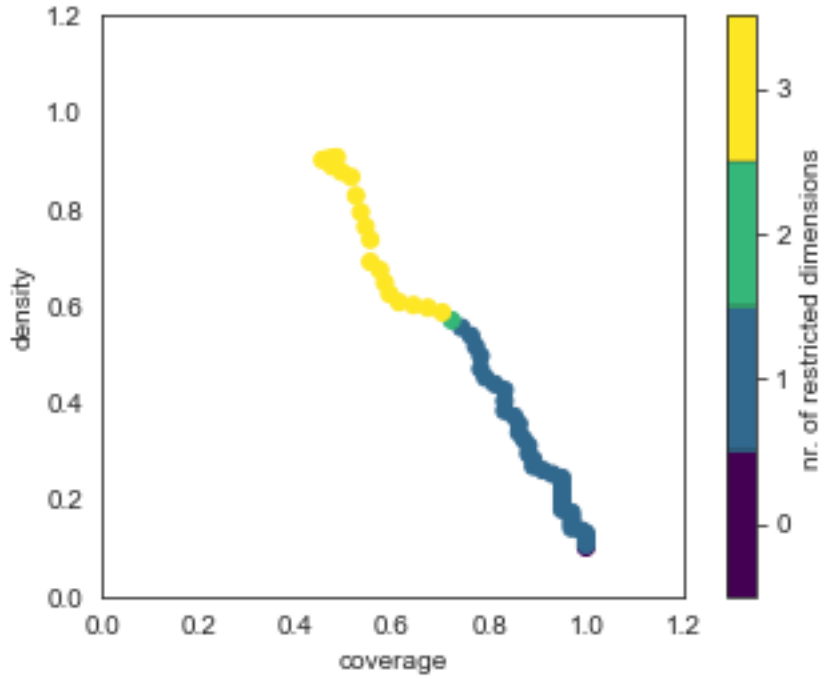





```
avg: accuracy
```

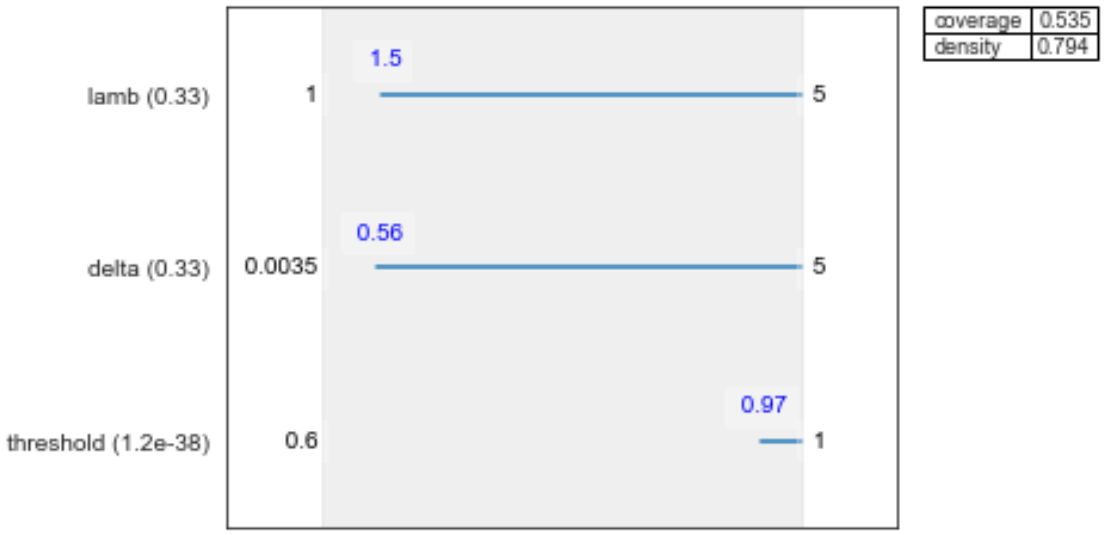


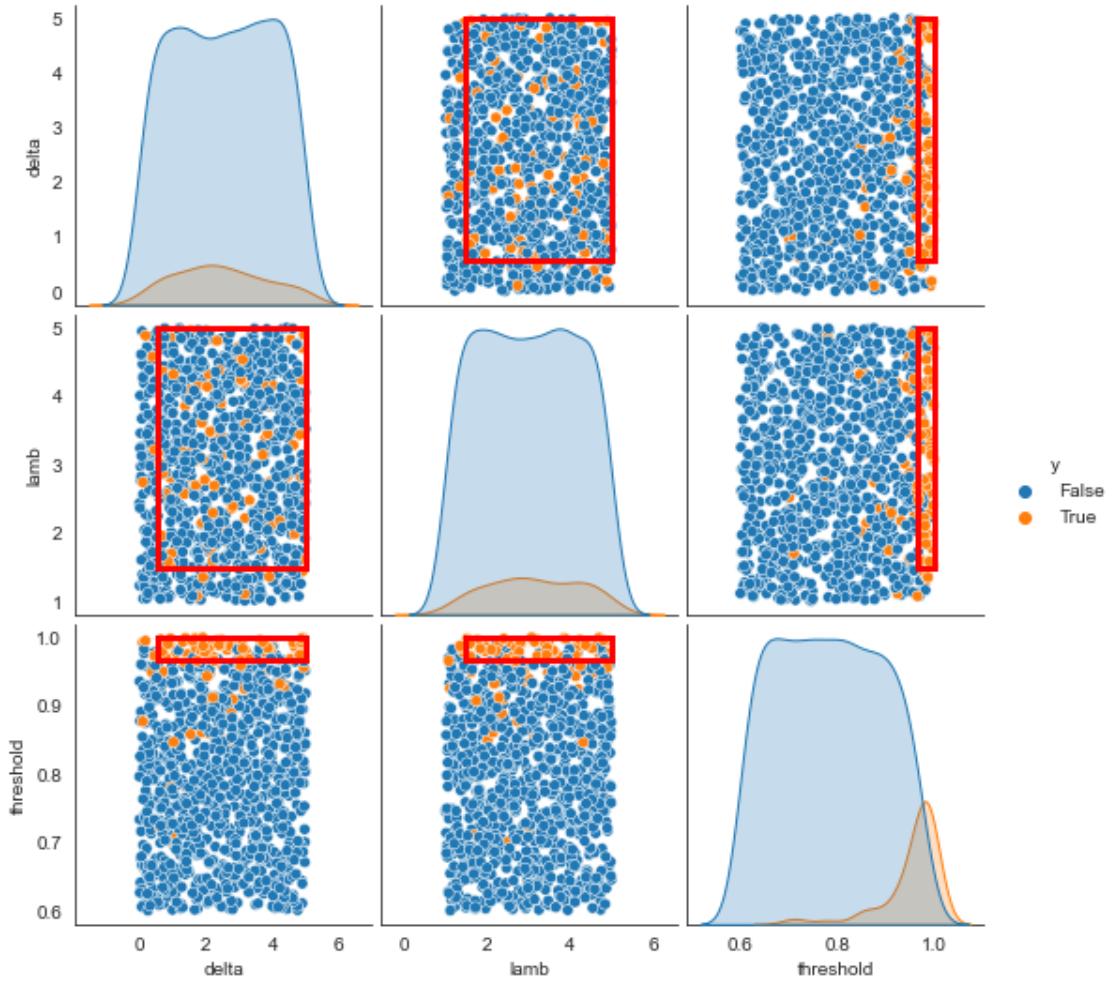
intrusion-detection



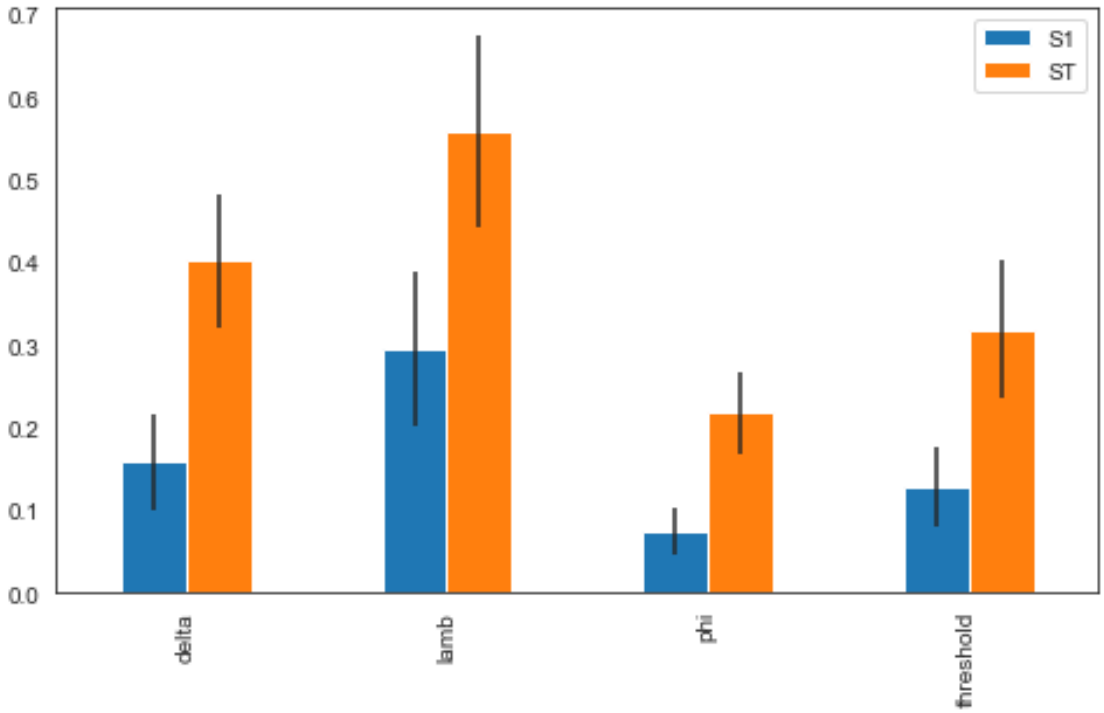
```
coverage    0.534653
density     0.794118
id                50
mass           0.068
mean        0.794118
res_dim            3
Name: 50, dtype: object
```

	box 50		
	min	max	qp values
threshold	0.966153	0.999695	[1.23026342186239e-38, -1.0]
delta	0.558982	4.997110	[0.3309678068897105, -1.0]
lamb	1.490211	4.996453	[0.3309678068897105, -1.0]

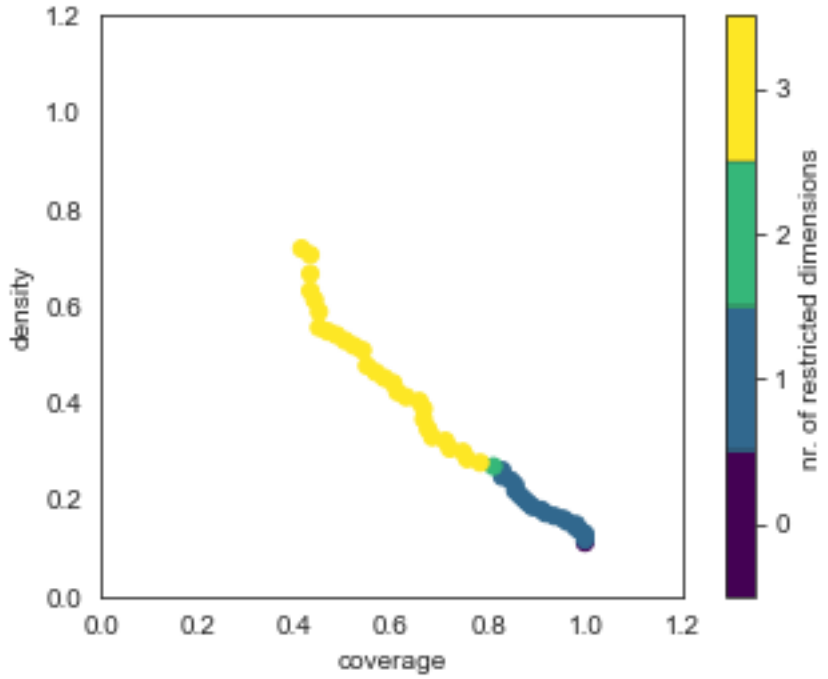




avg: accuracy

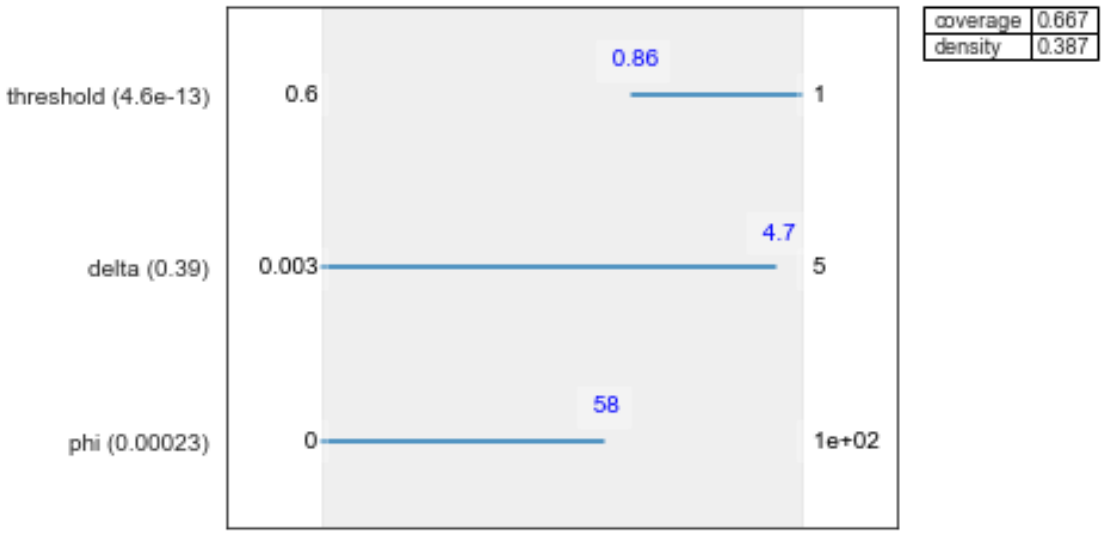

S1
ST
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
delta
lamb
phi
threshold

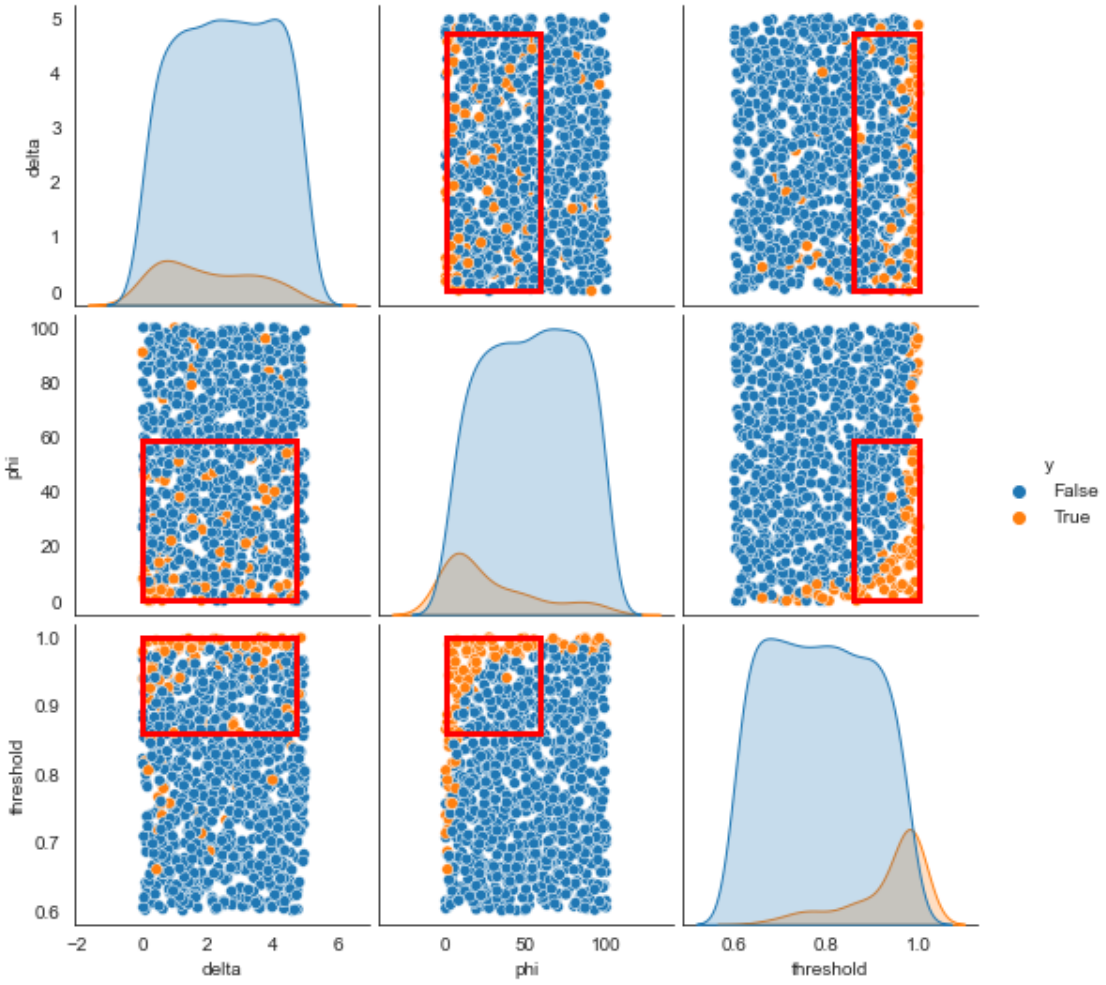
kitchen


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3

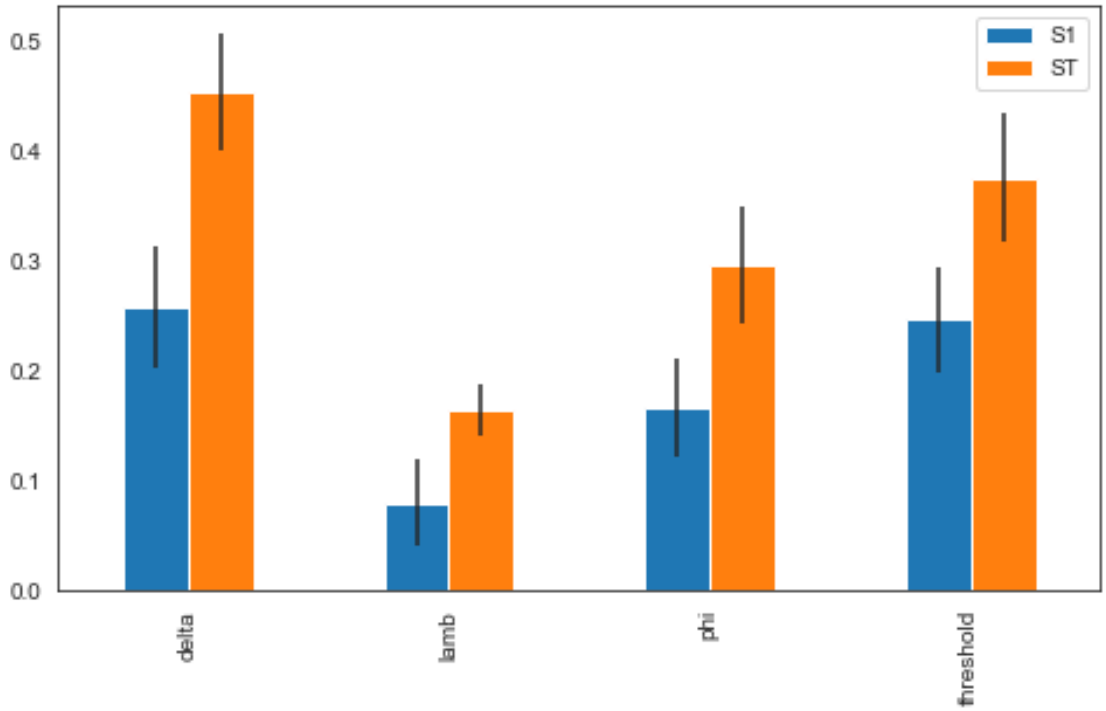
```
coverage    0.666667
density     0.387435
id                30
mass           0.191
mean        0.387435
res_dim            3
Name: 30, dtype: object
```

box 30	min	max	qp values
phi	0.000000	58.500000	[-1.0, 0.0002325832966247555]
delta	0.003013	4.732179	[-1.0, 0.3874883149043474]
threshold	0.858698	0.999771	[4.58290142601594e-13, -1.0]

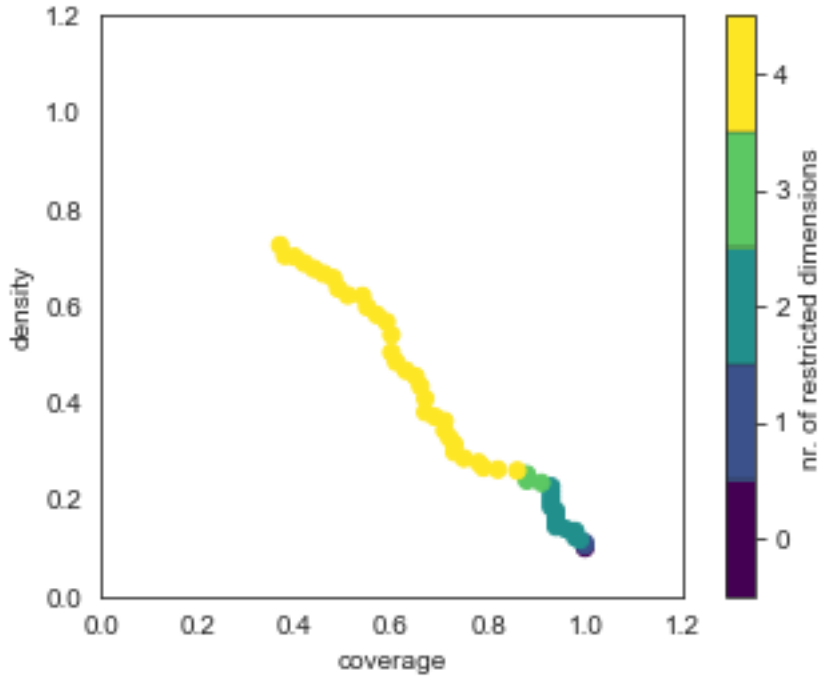





```
avg: accuracy
```

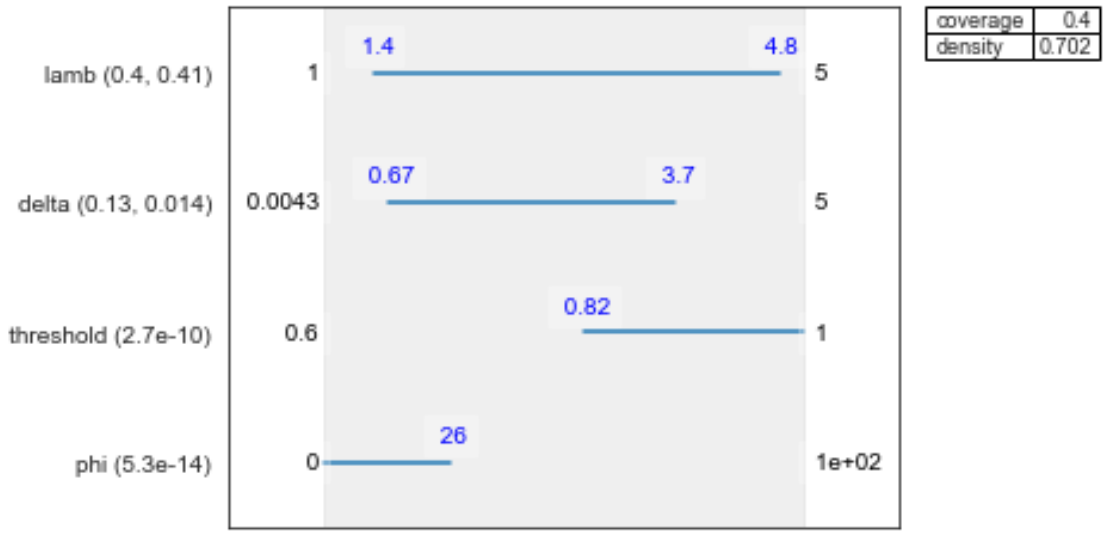

S1
ST
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

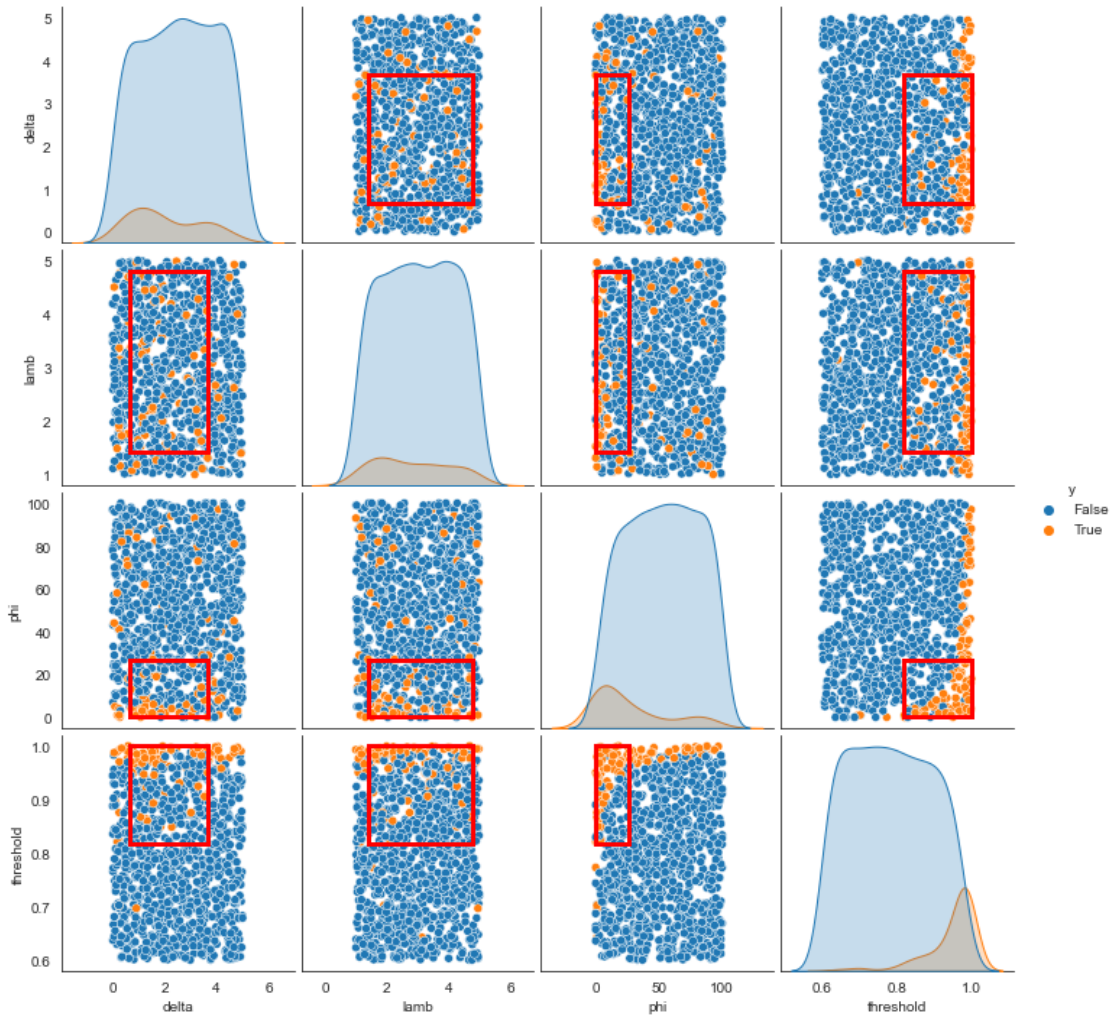
logistics


1.2
1.0
0.8
0.6
0.4
0.2
0.0
density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
coverage
4
3
2
1
0
nr. of restricted dimensions

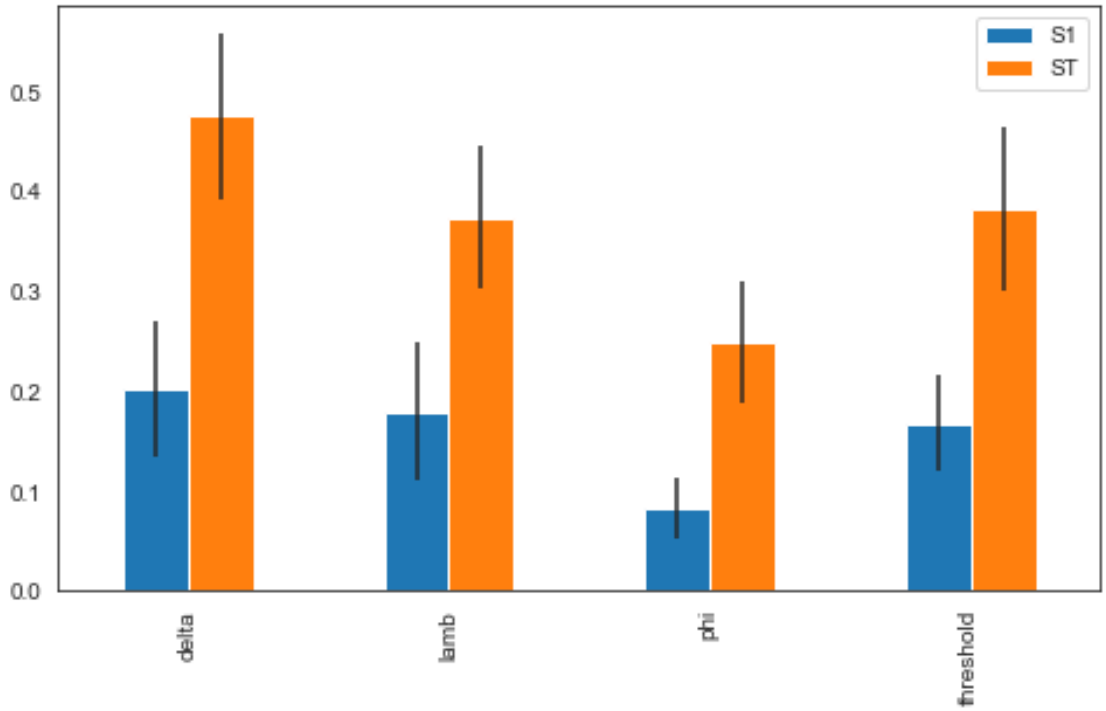
```
coverage         0.4
density     0.701754
id                50
mass           0.057
mean        0.701754
res_dim            4
Name: 50, dtype: object
```

	box 50		
	min	max	qp values
phi	0.000000	26.500000	[-1.0, 5.3347693634223285e-14]
threshold	0.817235	0.999944	[2.7330357641051353e-10, -1.0]
delta	0.668168	3.660067	[0.12642876896295208, 0.014215248405283735]
lamb	1.416186	4.802064	[0.4039644610013883, 0.4105067721298188]

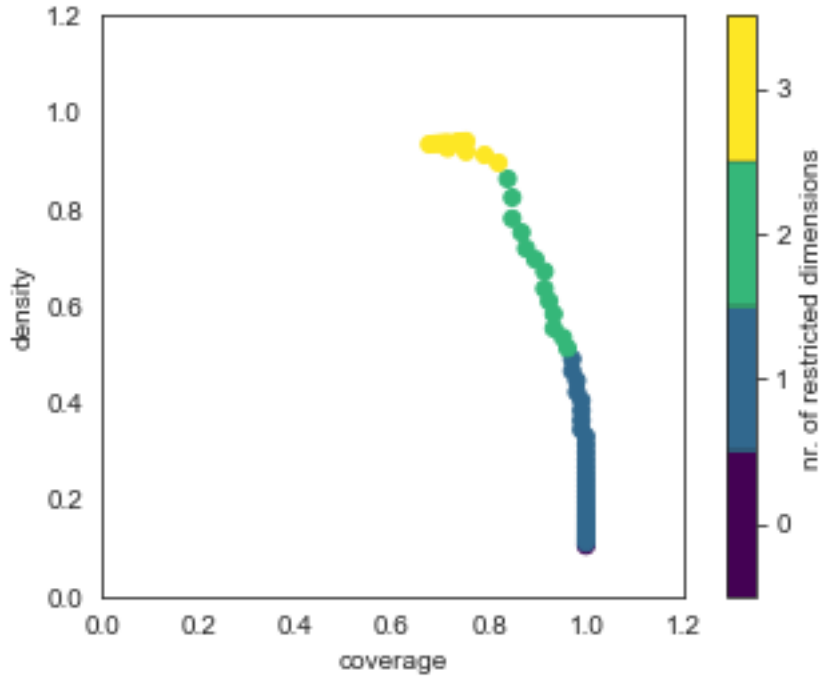





```
avg: accuracy
```

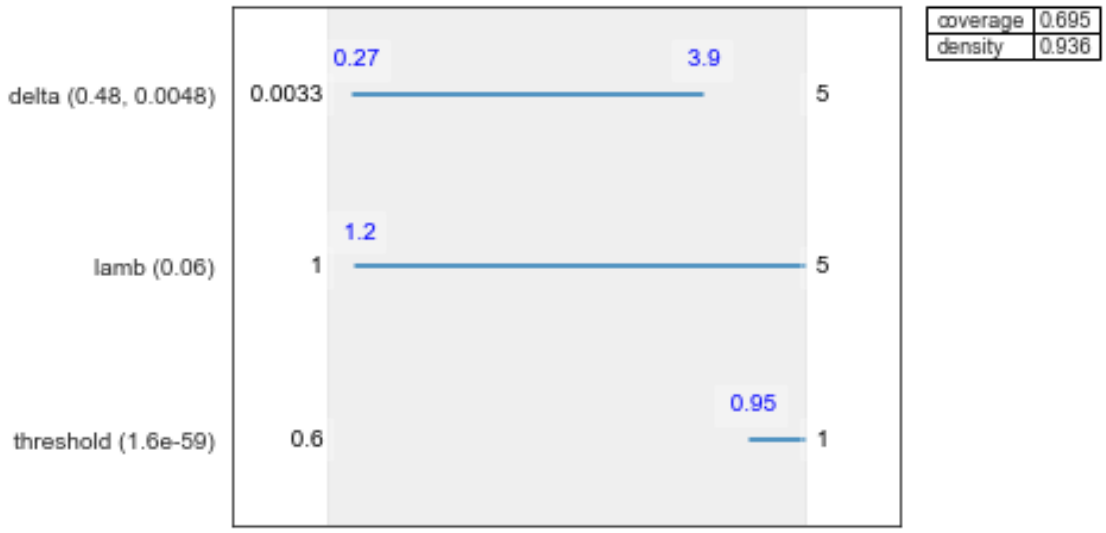

S1
ST
0.0
0.1
0.2
0.3
0.4
0.5
delta
lamb
phi
threshold

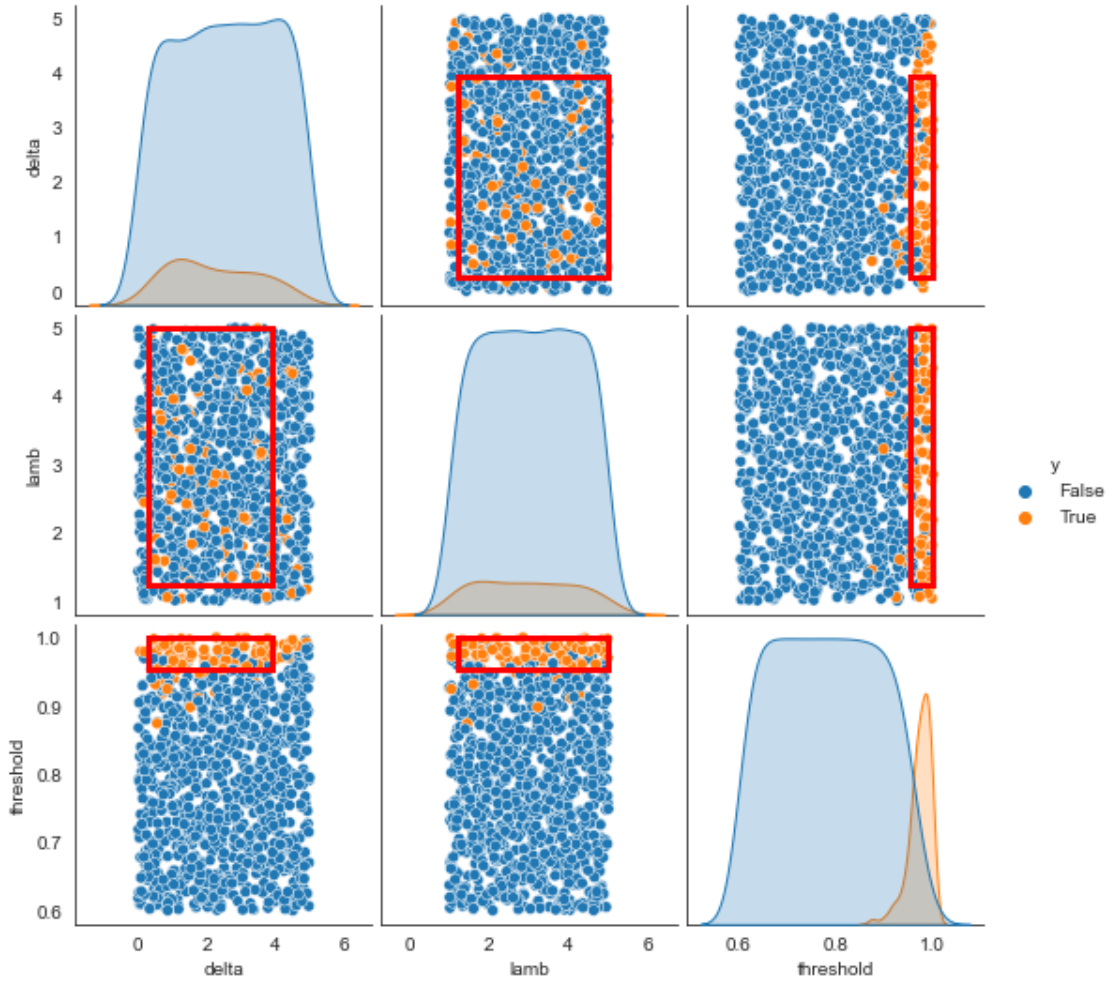
```
miconic
```


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3

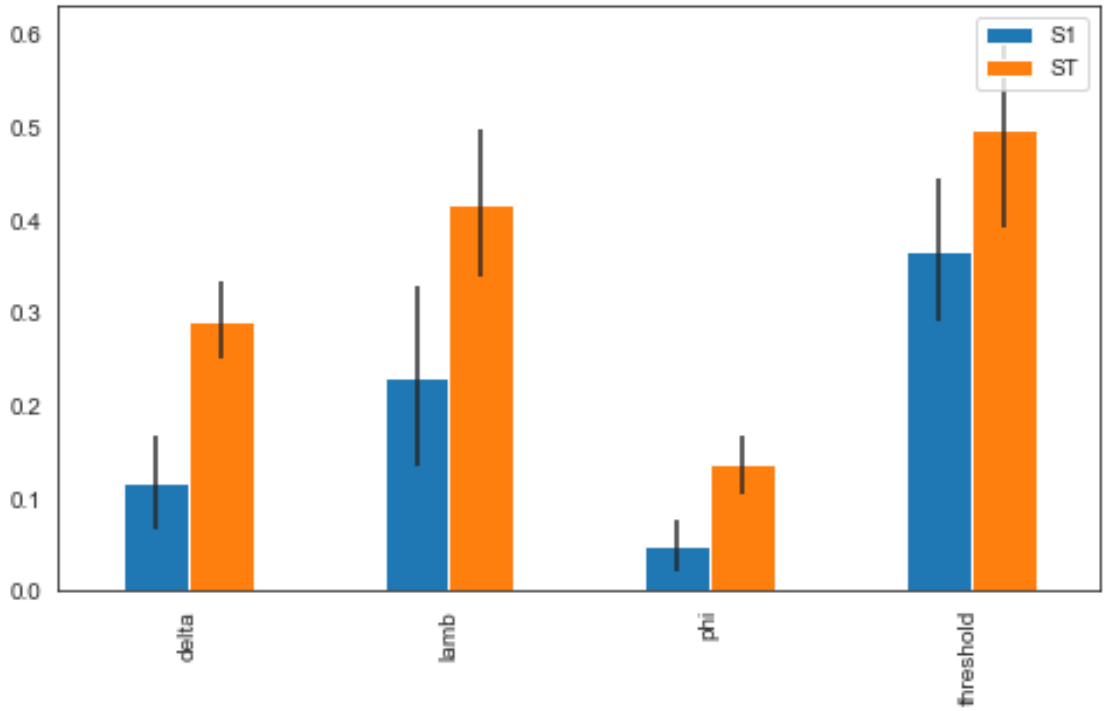
```
coverage    0.695238
density     0.935897
id                50
mass           0.078
mean        0.935897
res_dim            3
Name: 50, dtype: object
```

	box 50		
	min	max	qp values
threshold	0.954132	0.999936	[1.5978591310829218e-59, -1.0]
lamb	1.233999	4.999281	[0.05991234816245654, -1.0]
delta	0.273098	3.922304	[0.4770174949785684, 0.004789633930689048]

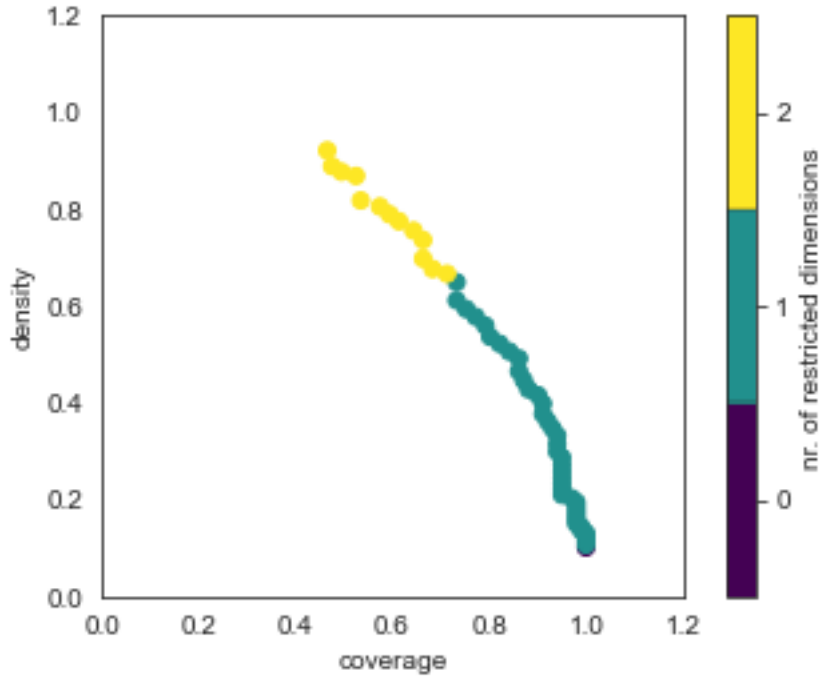




avg: accuracy

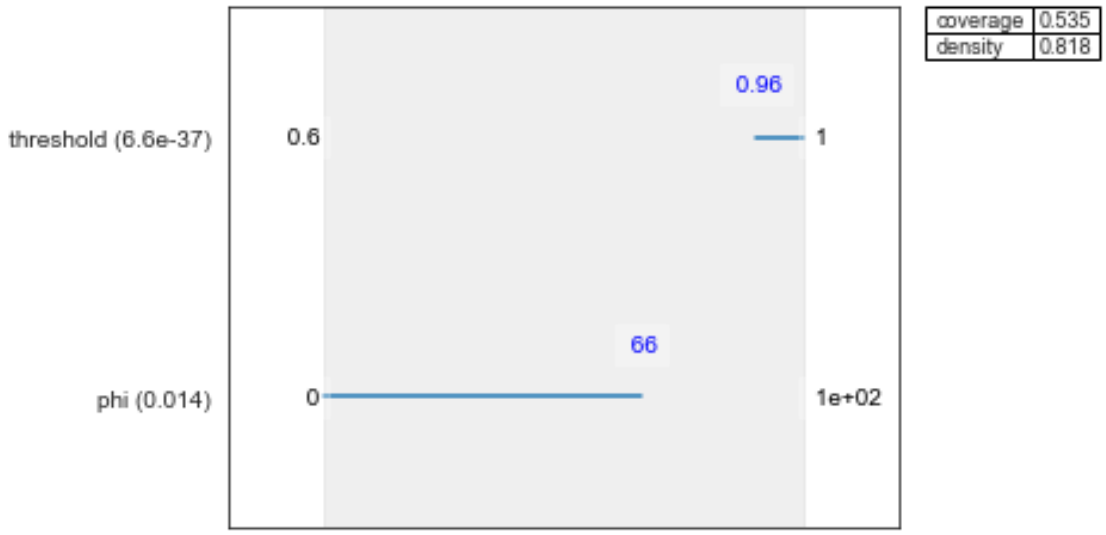

S1
ST
0.6
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

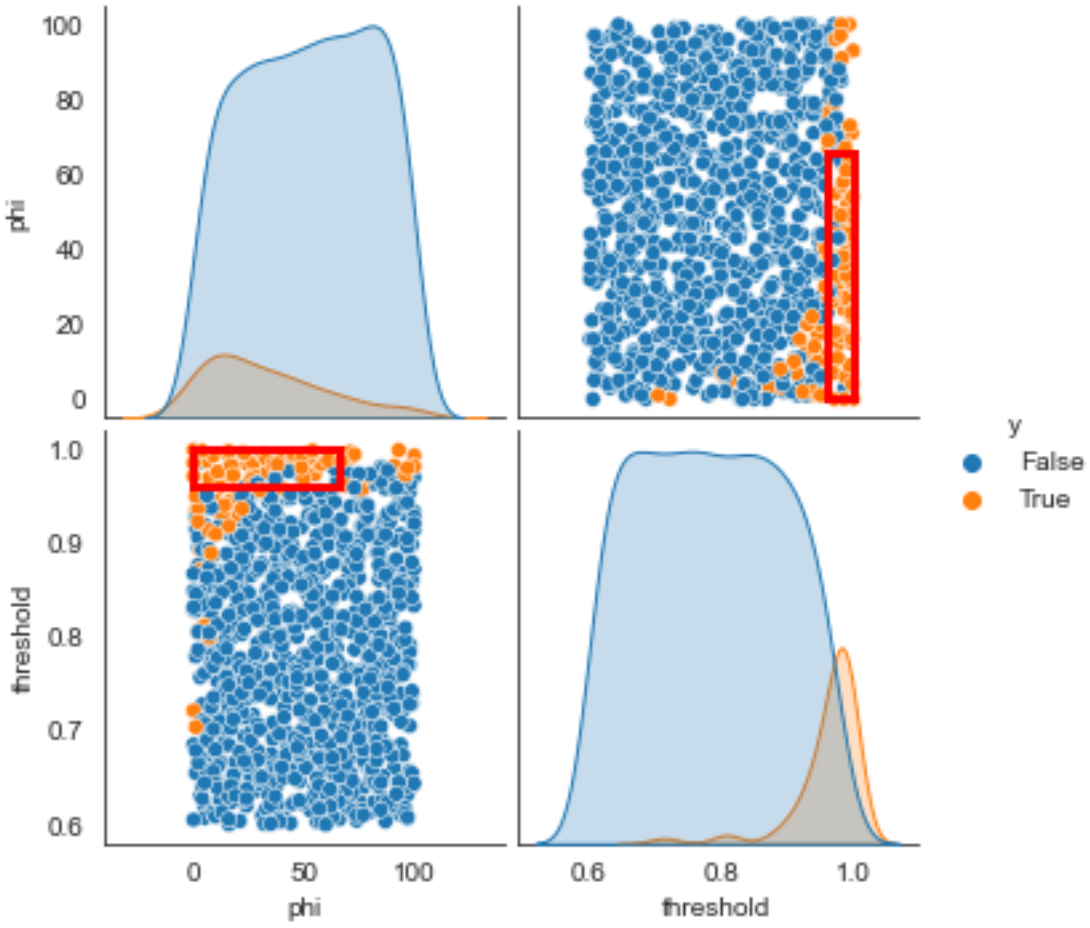
rovers


1.2
1.0
0.8
0.6
0.4
0.2
0.0
density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
coverage
2
1
0
nr. of restricted dimensions

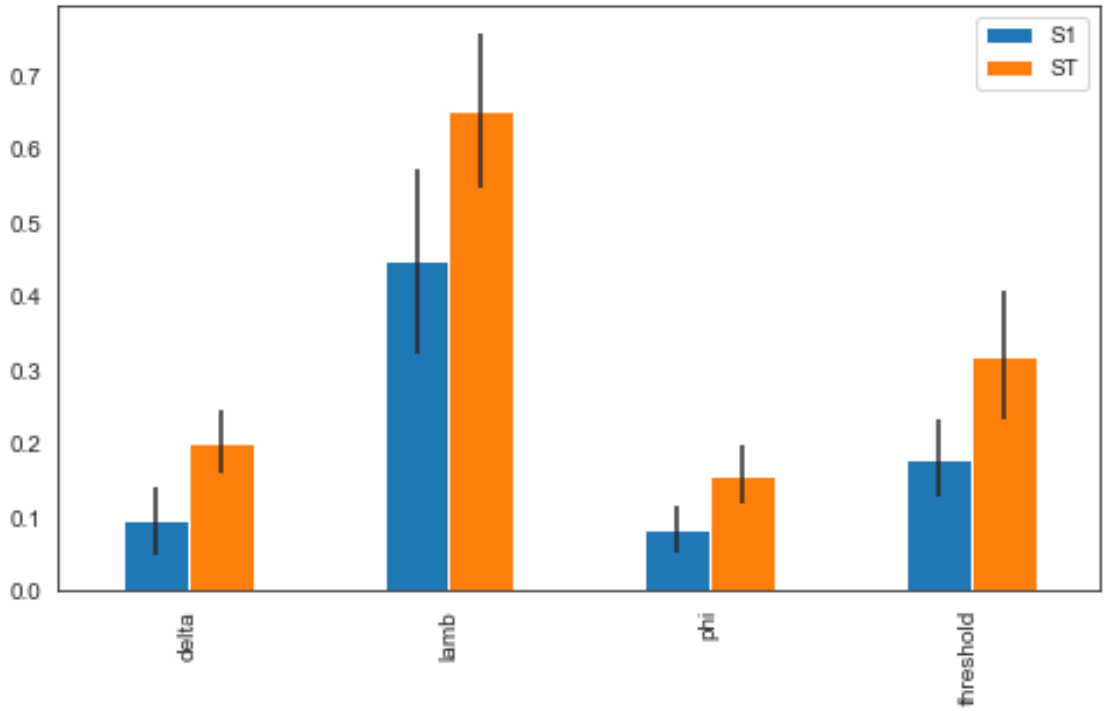
```
coverage    0.534653
density     0.818182
id                50
mass           0.066
mean        0.818182
res_dim            2
Name: 50, dtype: object
```

box 50	min	max	qp values
phi	0.000000	66.000000	[-1.0, 0.014018959882765126]
threshold	0.960073	0.999779	[6.579580452179475e-37, -1.0]

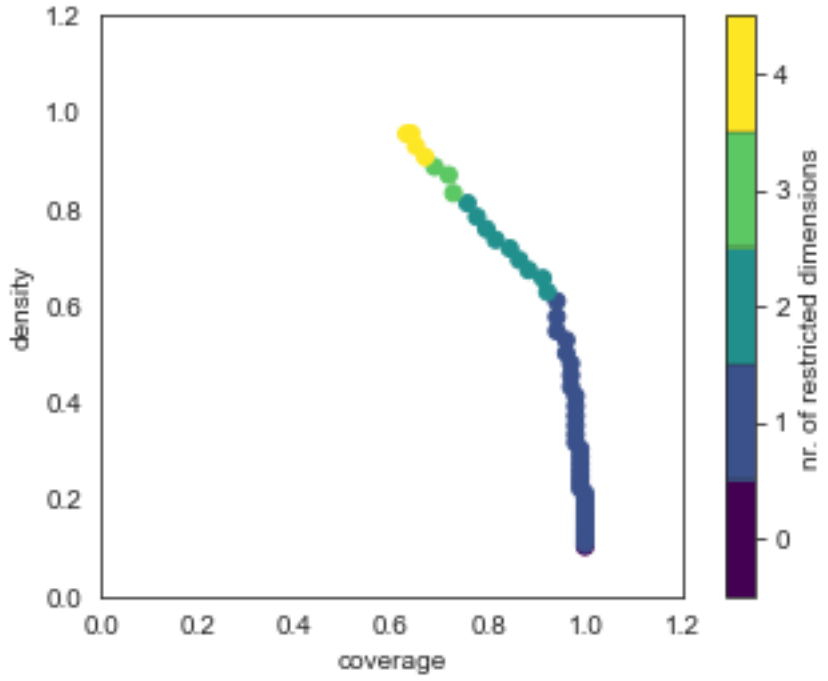





```
avg: accuracy
```

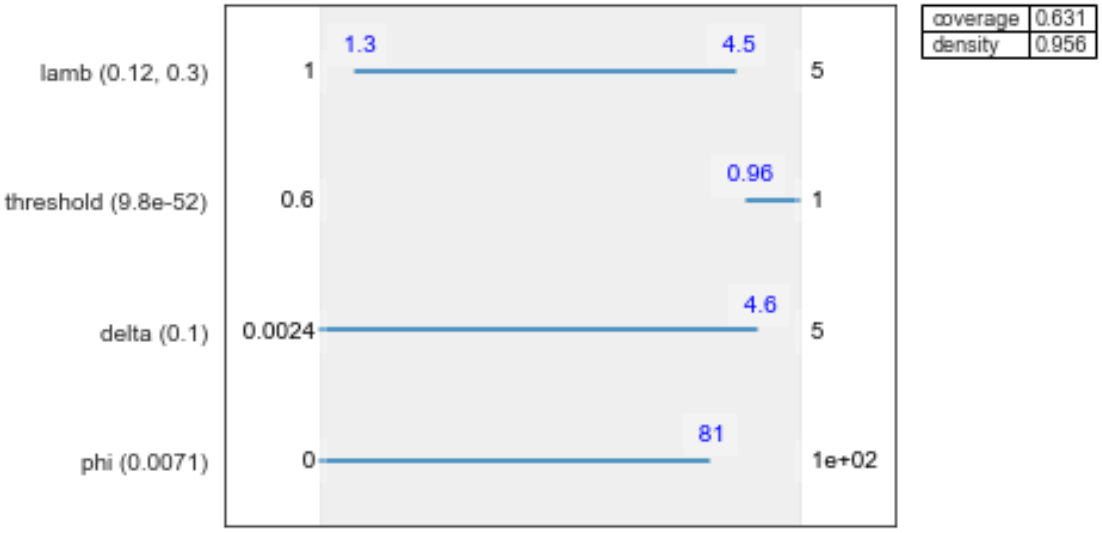

S1
ST
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
delta
lamb
phi
threshold

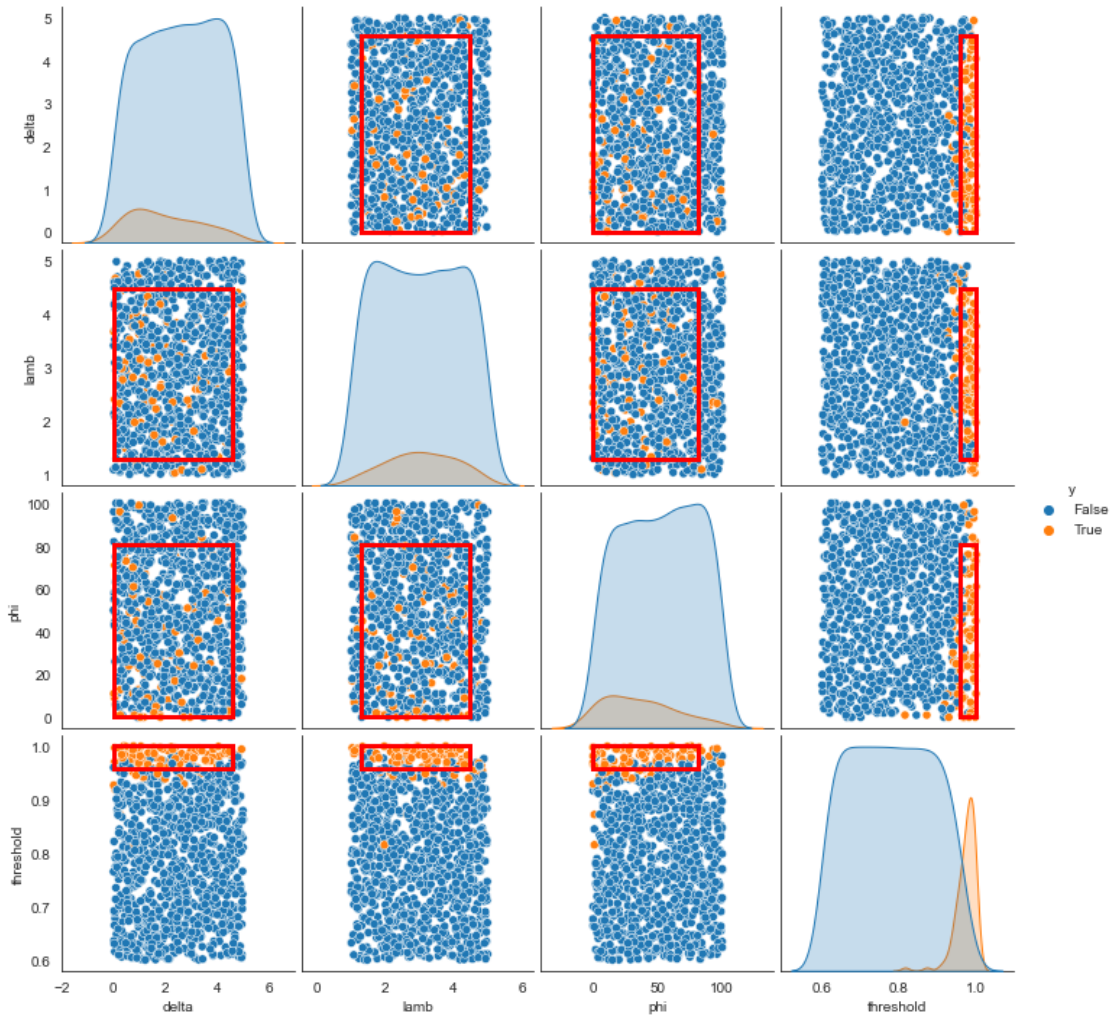
satellite


density
coverage
nr. of restricted dimensions
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4

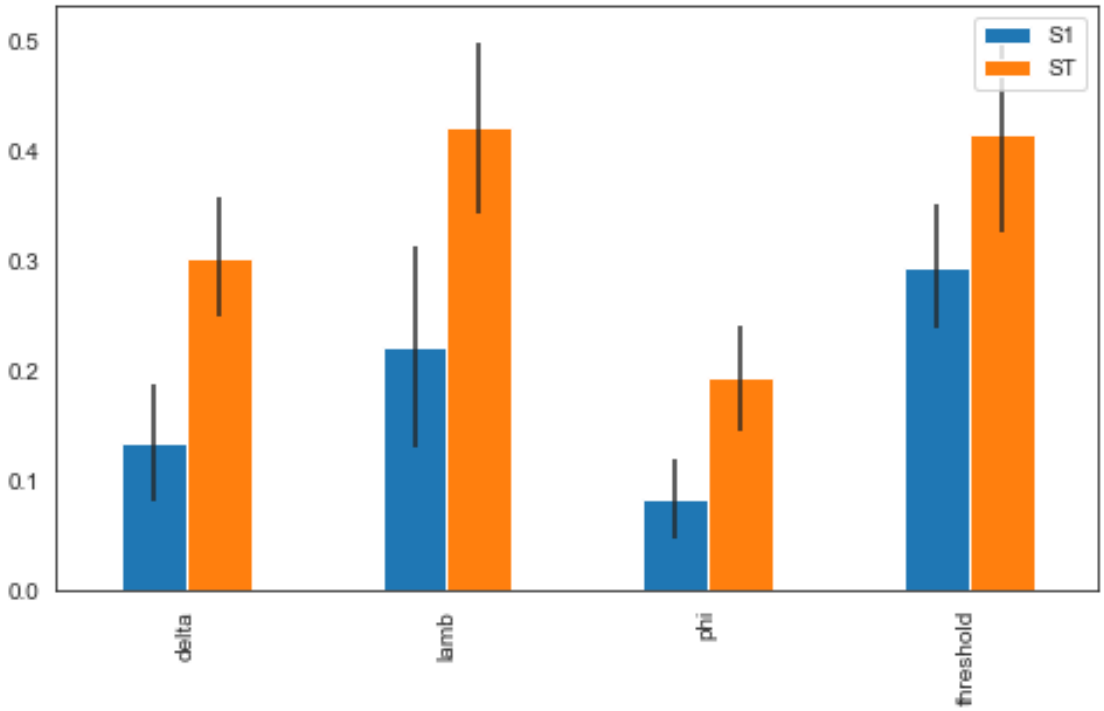
```
coverage     0.631068
density      0.955882
id                 50
mass            0.068
mean         0.955882
res_dim             4
Name: 50, dtype: object
```

box 50	min	max	qp values
phi	0.000000	81.000000	[-1.0, 0.007056184858491923]
delta	0.002380	4.571105	[-1.0, 0.10475397014768047]
threshold	0.957283	0.999750	[9.802706556340941e-52, -1.0]
lamb	1.298928	4.465743	[0.1172155665422155, 0.29652484105792665]

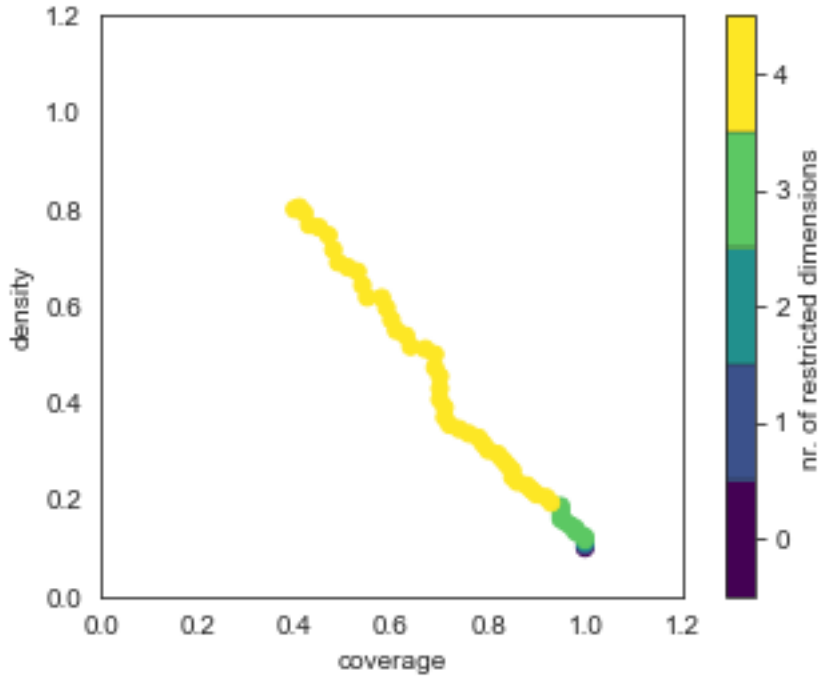




avg: accuracy

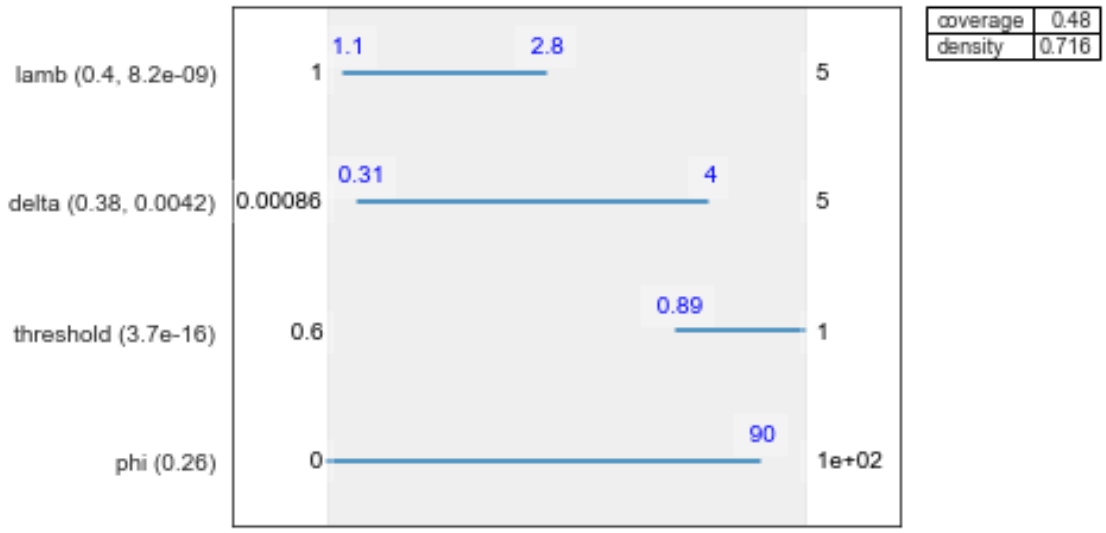

S1
ST
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

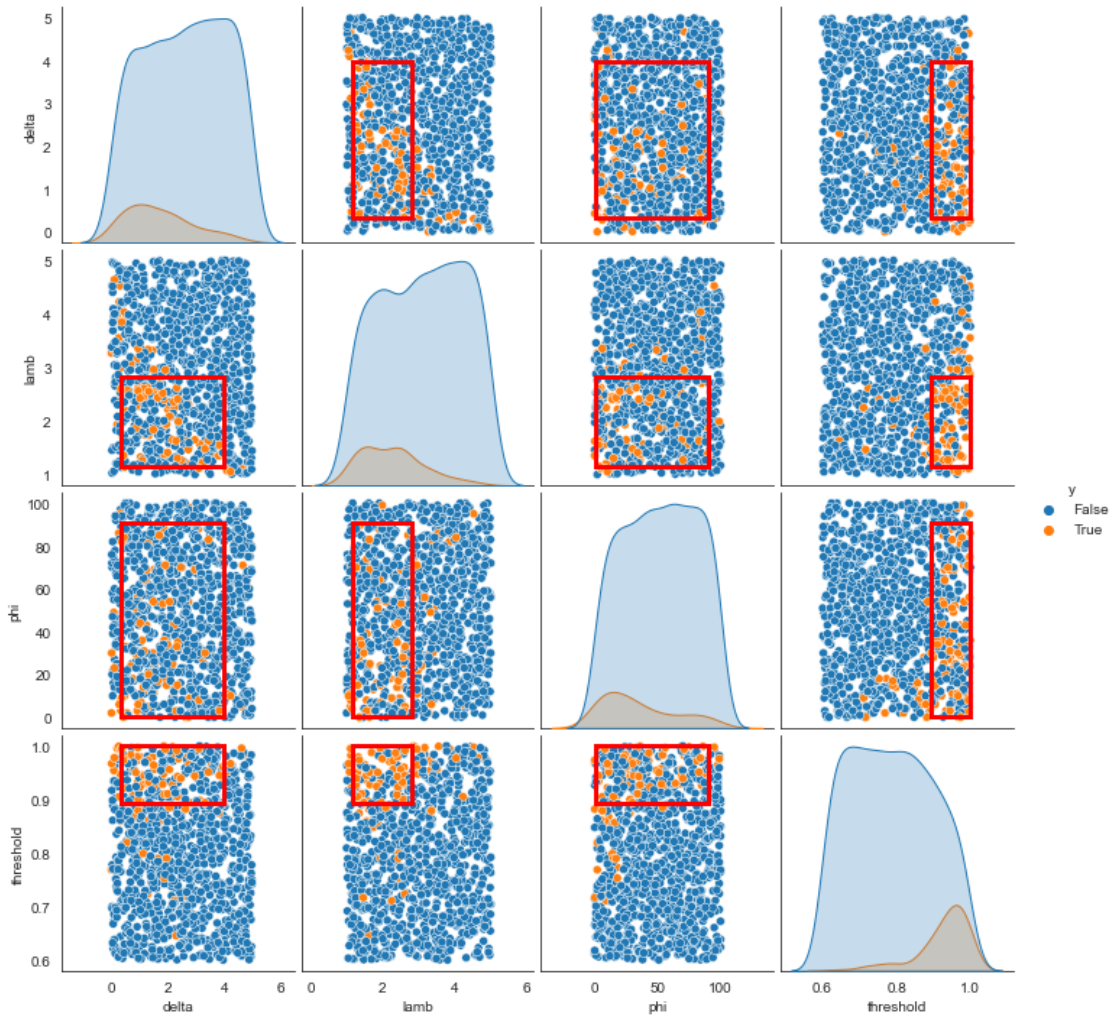
sokoban


1.2
1.0
0.8
0.6
0.4
0.2
0.0
density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
coverage
4
3
2
1
0
nr. of restricted dimensions

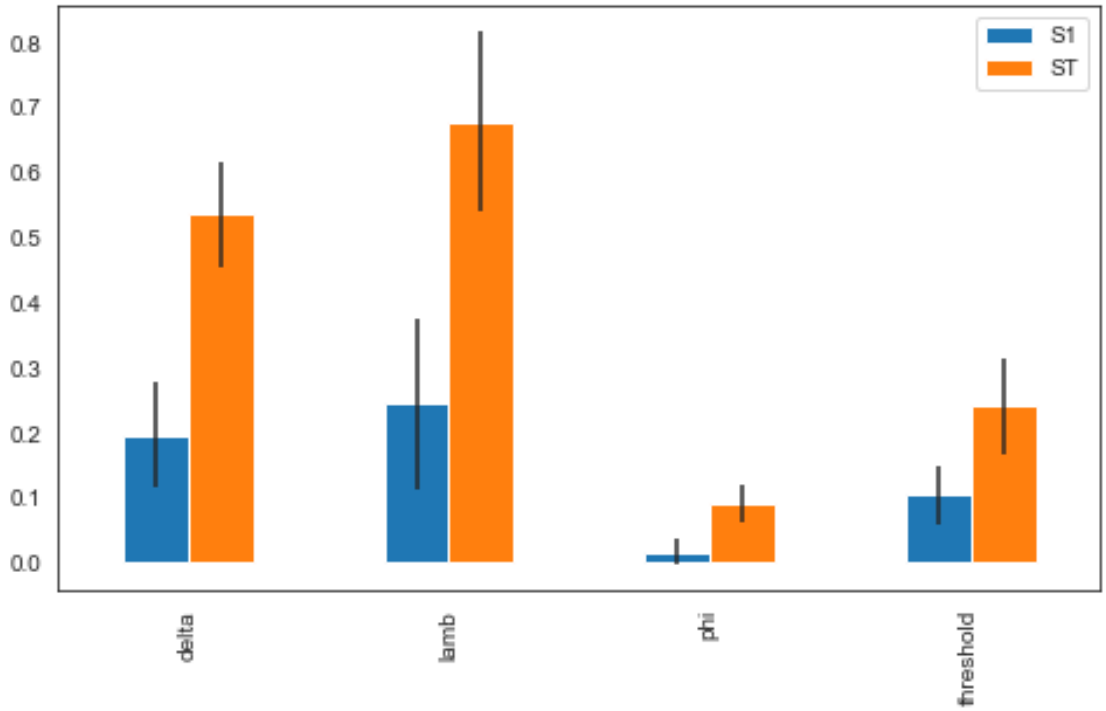
```
coverage        0.48
density     0.716418
id                50
mass           0.067
mean        0.716418
res_dim            4
Name: 50, dtype: object
```

box 50	min	max	qp values
phi	0.000000	90.500000	[-1.0, 0.25740137754695797]
threshold	0.892190	0.999889	[3.6924019823425327e-16, -1.0]
delta	0.314841	3.973954	[0.37541441915255575, 0.004228446740785658]
lamb	1.144148	2.823730	[0.40466722861694526, 8.166445984241457e-09]

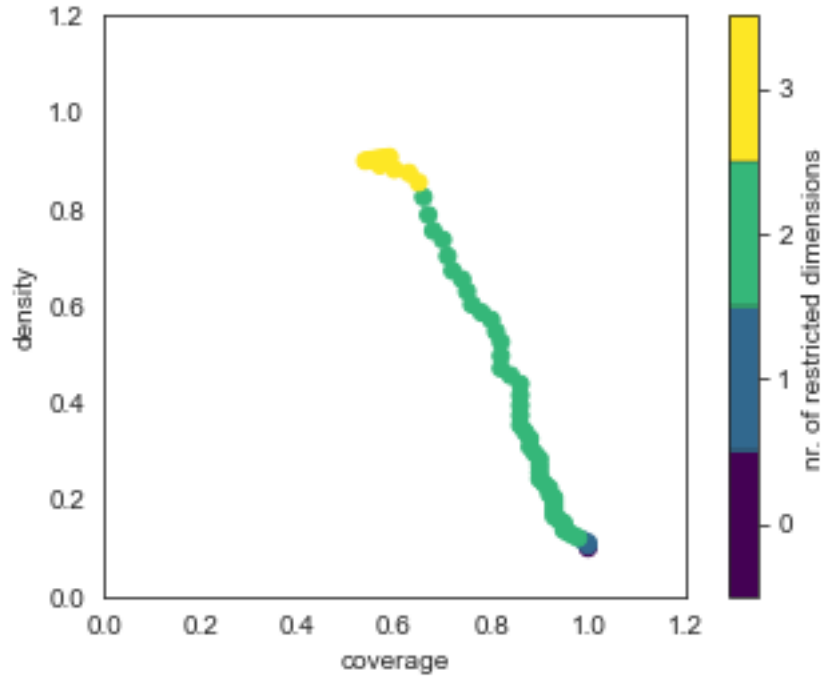




avg: accuracy

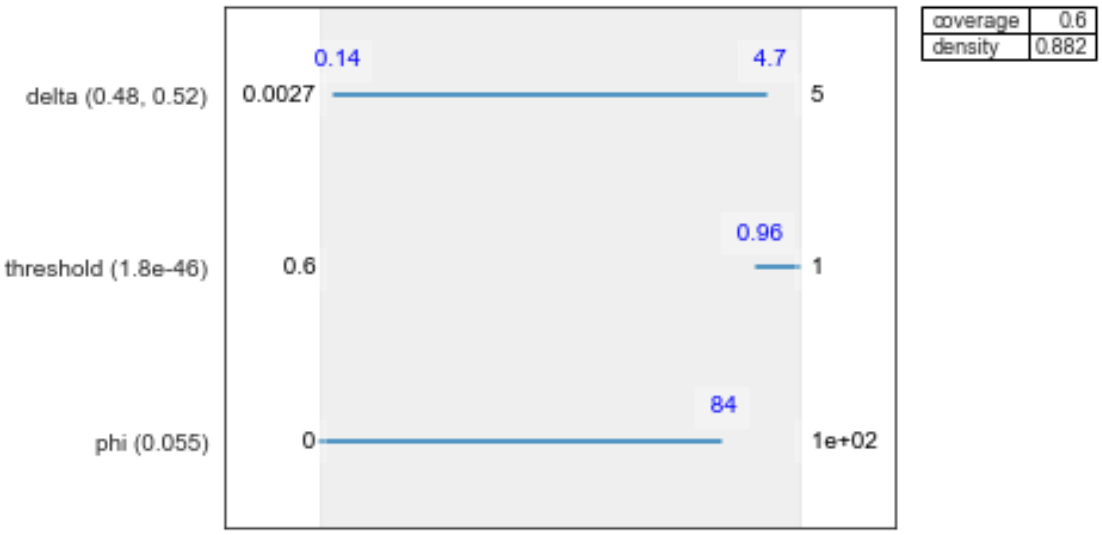

S1
ST
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

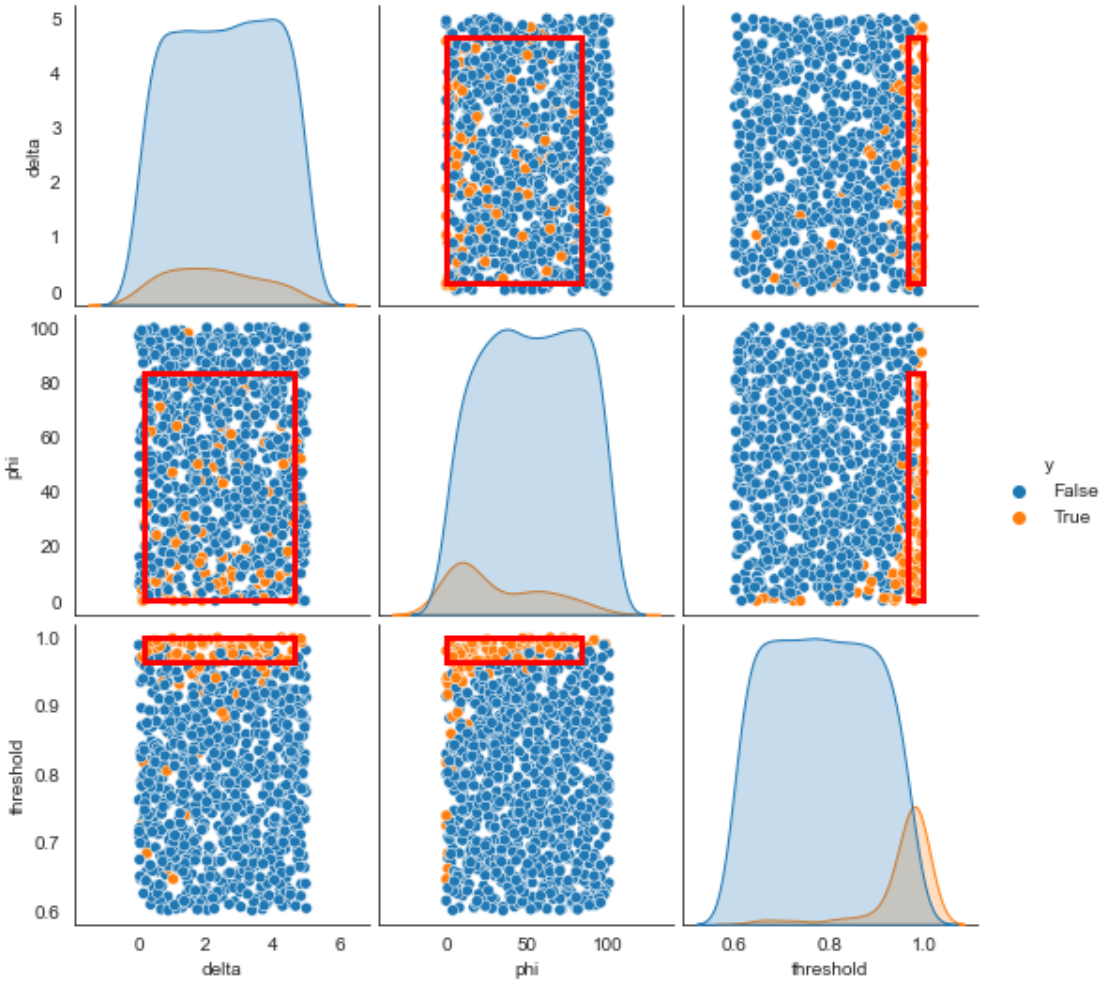
zeno-travel


1.2
1.0
0.8
0.6
0.4
0.2
0.0
density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
coverage
3
2
1
0
nr. of restricted dimensions

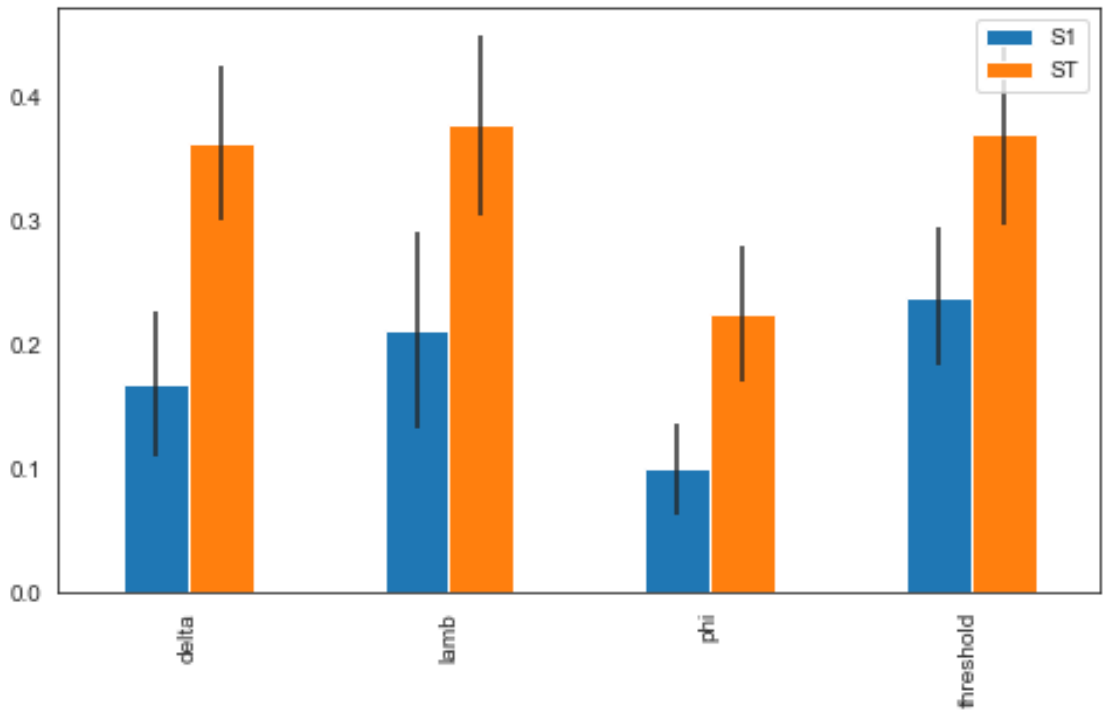
```
coverage         0.6
density     0.882353
id                50
mass           0.068
mean        0.882353
res_dim            3
Name: 50, dtype: object
```

	box 50		
	min	max	qp values
phi	0.000000	83.500000	[-1.0, 0.05476180730086377]
threshold	0.964518	0.999696	[1.8288661934704696e-46, -1.0]
delta	0.142859	4.651094	[0.4830500081787835, 0.5185574432063291]






```
avg: accuracy
```


S1
ST
0.4
0.3
0.2
0.1
0.0
delta
lamb
phi
threshold

[ ]: