InsightSolver API client

Release 0.1.50

Noé Aubin-Cadot

Sep 17, 2025

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Welcome to the technical documentation for the **InsightSolver API Client**.

1.1 About InsightSolver

InsightSolver is a SaaS (Software as a Service) solution designed for advanced rule mining and data insights. Powered by a centralized rule-mining engine, it enables organizations to uncover hidden patterns and generate actionable insights for data-driven decision-making.

1.2 Access Options

InsightSolver can be accessed through the following options:

- The InsightSolver API Client, designed for seamless integration with Python applications and data workflows.
- The InsightSolver Web App, which provides an intuitive and interactive interface for exploring rule-mining results, visualization, and analysis.

This documentation specifically covers the InsightSolver API Client. The InsightSolver Web App will have its own documentation once available.

1.3 The InsightSolver API client

The InsightSolver API Client offers a Python-based interface for direct interaction with our rule-mining engine. Designed for data engineers, scientists, and developers, this client integrates InsightSolver's functionalities into custom workflows, applications, and automated pipelines. The API client allows users to configure rule-mining parameters, send encrypted data to the server and retrieve results efficiently.

1.4 Who Should Use This Documentation?

This documentation is for:

- Developers integrating our rule-mining API.
- Engineers needing setup guides and technical references.
- Users looking for examples of effective API use.

1.5 Accessing the API

To use the InsightSolver API, you need the following:

- **A valid service key** This is a `.json` file that authenticates your identity and secures communication with the InsightSolver API server.
- **Available credits** API usage is metered based on the size of your dataset. See the *Credits* section for more details on how credits are calculated and usage tips.

To request a service key and/or obtain additional credits, please contact us at support@insightsolver.com.

CHAPTER

TWO

INSTALLATION

This guide provides instructions to install and set up the InsightSolver API client on your local machine.

2.1 Prerequisites

- **Python 3.9 or higher**: Ensure Python is installed on your system. You can check your Python version with:

```
python --version
```

- **pip**: Python's package installer, which is typically included with Python 3 installations.

2.2 Installation Steps

You can install the InsightSolver API client in different ways, depending on your setup:

A. **Install directly using pip (100% CLI)**

If you have git installed and don't need a local copy of the repository, run:

```
pip install git+https://github.com/insightsolver/insightsolver.git
```

B. **Clone the repository and install locally (100% CLI)**

If you have git installed and want to also keep a local copy of the repository, run:

```
git clone https://github.com/insightsolver/insightsolver.git
cd insightsolver
pip install .
```

C. **Download via browser and install (50% GUI + 50% CLI)**

If you don't have git installed, follow these steps:

- Open a browser and go to https://github.com/insightsolver/insightsolver.
- Click on the green button <> `Code v` and select `Download ZIP`.
- Extract the ZIP file to a folder on your machine.
- Open a terminal and navigate to the extracted folder using:

```
cd path/to/unzipped-folder
```

- Then install the package with:

```
pip install .
```

Warning for Anaconda users: When using a virtual environment managed by Anaconda, the installation of the `insightsolver` library as specified above could install dependencies (specified in the file `requirements.txt`) via pip that are not handled by Anaconda. There are two options available. The first option is to do as specified above, which lets pip install dependencies, but risk that the virtual environment is no longer handled by Anaconda. The second option is to add a `--no-deps` flag to the pip install, e.g. `pip install --no-deps .`. This last command would install the scripts of the `insightsolver` module without installing the dependencies. This prevents breaking the Anaconda environment but could result in `insightsolver` not finding all the required dependencies at runtime. These required dependencies should therefore either be installed manually from within the Anaconda application or either using the `environment.yml` file.

2.3 Testing the Installation

To verify that the installation was successful, you can run a quick test in Python to ensure all dependencies are correctly installed and functioning.

1. Open a terminal and start a Python interpreter by typing:

   ```
   python
   ```

2. Once inside the Python shell, try importing the `InsightSolver` class with the following command:

   ```
   from insightsolver import InsightSolver
   ```

3. If the installation was successful, there should be no errors, and the Python shell should return to the prompt. If you encounter an *ImportError*, ensure that you have installed the package in the correct environment.

4. Exit the Python shell by typing:

   ```
   quit()
   ```

If you encounter any errors or need assistance, please refer to the *Help section*.

CHAPTER

THREE

USAGE

The following sections provide examples on how to use the InsightSolver API client.

3.1 Quick Start Example

This section provides a quick example of how to use the InsightSolver API client. Before running the example script, please ensure that:

1. You have completed the steps in the *Installation Guide*.
2. You have obtained a valid service key.
3. You have credits available.

The following example demonstrates the basic usage of the InsightSolver API client, showing how to initialize the solver and generate insights.

```
# Import Pandas
import pandas as pd

# Import some data: https://www.kaggle.com/competitions/titanic/data
df = pd.read_csv('kaggle_titanic_train.csv')

# Specify the name of the target variable
target_name = 'Survived' # We are interested in whether the passengers survived or not

# Specify the target goal
target_goal = 1 # We are searching rules that describe survivors

# Choose how features should be interpreted
columns_types = {
   'Survived' : 'binary',
   'Pclass'   : 'continuous', # 'multiclass' (i.e. unordered) or 'continuous' (i.e. ordered)
   'Name'     : 'ignore',
   'Sex'      : 'binary',
   'Age'      : 'continuous',
   'SibSp'    : 'continuous', # 'multiclass' (i.e. unordered) or 'continuous' (i.e. ordered)
   'Parch'    : 'continuous', # 'multiclass' (i.e. unordered) or 'continuous' (i.e. ordered)
   'Ticket'   : 'ignore',
   'Fare'     : 'continuous',
   'Cabin'    : 'ignore',
   'Embarked' : 'multiclass',
}
```

(continues on next page)

(continued from previous page)

```
# Import the class InsightSolver from the module insightsolver
from insightsolver import InsightSolver

# Create an instance of the class InsightSolver
solver = InsightSolver(
   df            = df,            # A dataset
   target_name   = target_name,   # Name of the target variable
   target_goal   = target_goal,   # Target goal
   columns_types = columns_types, # Columns types
)

# Specify the service key
service_key = 'name_of_your_service_key.json'

# Fit the solver
solver.fit(
   service_key  = service_key, # Use your API service key here
)

# Print the rule mining results
solver.print(mode='dense')
"""
                                  contribution variable               rule     nans
i p_value coverage gain     cohen_d

0 2e-67   19.1%    +146.73% 84.76
                                        86.2%      Sex             female
                                        13.8%   Pclass             [1, 2]
1 3e-20   12.2%    +105.55% 19.80
                                        81.4%   Pclass             [1, 1]
                                        11.3%     Fare  [7.925, 512.3292]
                                         7.3%      Age        [4.0, 42.0]  exclude
2 7e-12    7.9%    +100.98% 8.37
                                        57.4%    Parch             [1, 6]
                                        31.4%    SibSp             [0, 1]
                                        11.2%      Age       [0.42, 25.0]  exclude
"""
```

In this specific example, the InsightSolver API gives us three rules in which we find more survivors of the Titanic:

- `(i=0)` : **Women in 1st or 2nd class.** This group covers 19.1% of the passengers and has a survival gain of +146.73% compared to the population of the Titanic.
- `(i=1)` : **Rich 1st class that are not too old (which we know the age).** This group covers 12.2% of the passengers and has a survival gain of +105.55% compared to the population of the Titanic.
- `(i=2)` : **Children (which we know the age) with not too many siblings.** This group covers 7.9% of the passengers and has a survival gain of +100.98% compared to the population of the Titanic.

Note that there could be a survivor bias in the two rules `i=1` and `i=2` because we know the age of the survivors more than we know the age of the non-survivors of the Titanic. We could also use `target_goal = 0` to look for passengers that did not survive the Titanic:

```
# Specify the target goal
target_goal = 0 # We are searching rules that describe non-survivors

# Create an instance of the class InsightSolver
solver = InsightSolver(
   df            = df,            # A dataset
   target_name   = target_name,   # Name of the target variable
   target_goal   = target_goal,   # Target goal
   columns_types = columns_types, # Columns types
)

# Fit the solver
solver.fit(
   service_key = service_key,
)

# Print the rule mining results
solver.print(mode='dense')

"""
                                  contribution variable            rule     nans
i p_value coverage gain    cohen_d

0 7e-55   42.6%    +45.64% 60.04
                                         80.1%      Sex            male
                                         11.1%     Fare    [0.0, 26.25]
                                          8.8%    Parch          [0, 0]
1 2e-14    9.2%    +56.36% 10.32
                                         41.1%     Fare  [7.8875, 14.5]
                                         35.0%      Age    [19.0, 25.0]  include
                                         23.9%   Pclass          [3, 3]
"""
```

In this specific example, the InsightSolver API gives us two rules in which we find more non-survivors of the Titanic:

- `(i=0)` : **Poor males without a family.** This group covers 42.6% of the passengers and has a non-survival gain of +45.64% compared to the population of the Titanic.
- `(i=1)` : **Poor young third class adults (include missing ages).** This group covers 9.2% of the passengers and has a non-survival gain of +56.36% compared to the population of the Titanic.

Note that there could be a survivor bias in the rule `i=1` because we know the age of the non-survivors less than we know the age of the survivors of the Titanic. In conclusion, using the InsightSolver API, we know that *Rose DeWitt Bukater* (young rich female, 1st class, with her family) had a higher chance to survive the Titanic than *Jack Dawson* (3rd class young male without a family). For more technical details about the API, please refer to the detailed documentation.

3.2 Advanced usage

This section provides a deeper look at how to use the InsightSolver API client.

Let's revisit the *Titanic* demo. Once the `solver` is fitted, we can do more than simply `print` the results. This becomes particularly important when integrating the InsightSolver API client into a Python pipeline.

3.2.1 Conventions

In InsightSolver, the parameter `target_goal` specifies the target modality of the target variable for which rules should capture a large number of 1's. By convention:

- A data point is considered a `1` if it matches the target modality specified by `target_goal`.
- A data point is considered a `0` otherwise.

It is important to note that these `0`'s and `1`'s are conventions used internally by InsightSolver and should not be confused with the actual values or modalities of the target variable in the dataset.

For instance, consider the Titanic dataset used in the *Titanic* example:

- Dataset: `kaggle_titanic_train.csv`.
- Target variable: `target_name='Survived'`.
- Target modalities: `0` (non-survivor), `1` (survivor).
- Target goals: either `target_goal=0` (looking for non-survivors) either `target_goal=1` (looking for survivors).

Here, the modalities `0` and `1` are specific to the Titanic dataset and represent whether a passenger survived or not.

- Total passengers: `891` rows.
- Non-survivors (`Survived=0`): 549 rows.
- Survivors (`Survived=1`): 342 rows.

Case 1: Looking for Survivors (`target_goal=1`)

When the goal is to identify survivors:

- `M=891`: Total population.
- `M0=549`: Number of `0`'s, representing non-survivors (`Survived=0`).
- `M1=342`: Number of `1`'s, representing survivors (`Survived=1`).

Case 2: Looking for Non-Survivors (`target_goal=0`)

When the goal is to identify non-survivors:

- `M=891`: Total population.
- `M0=342`: Number of `0`'s, representing survivors (`Survived=1`).
- `M1=549`: Number of `1`'s, representing non-survivors (`Survived=0`).

InsightSolver operates under the principle of capturing `1`'s and rejecting `0`'s, regardless of the specific meaning of these values in a given dataset.

3.2.2 Attributes of the solver

The `solver` object includes several relevant attributes, which are described exhaustively here. For now, let's take a brief look at the most important ones:

- `M`: The total number of points in the population.
- `M0`: The number of points classified as `0` in the population.
- `M1`: The number of points classified as `1` in the population.
- `rule_mining_results`: A dictionary containing the results of the rule mining process. Below, we'll explore methods to access and parse specific aspects of these results.

- `benchmark_scores`: A dictionary containing the best scores obtained on shuffled data. This is useful to compare the scores of the rules found in the real data against the scores of the rules found in random data.

3.2.3 Counting the number of rules

To obtain the number of rules found by the solver, we can use the `ruleset_count` method:

```
solver.ruleset_count() # 3

# 3 rules are found by the solver
```

Each rule in the solver is indexed by an integer, conventionally denoted as `i`.

3.2.4 Getting the index of the rules

To retrieve the range of rule indices, we can use the `get_range_i` method:

```
solver.get_range_i() # [0, 1, 2]
```

This shows that the index `i` can take the values `0`, `1` or `2`. Knowing this range is useful when iterating over individual rules in the solver.

3.2.5 Exhaustive dictionary of a given rule

Let's take a closer look at the rule at position `i=0`. We can retrieve an exhaustive dictionary of the rule at position `i=0` as follows:

```
    solver.i_to_rule(i=0)

# {
#     "m": 170,
#     "m0": 9,
#     "m1": 161,
#     "coverage": 0.19079685746352412,
#     "m1/M1": 0.47076023391812866,
#     "mu_rule": 0.9470588235294117,
#     "mu_pop": 0.3838383838383838,
#     "sigma_pop": 0.48659245426485753,
#     "lift": 2.4673374613003096,
#     "p_value": 1.925558554763681e-67,
#     "F_score": 0.62890625,
#     "Z_score": 16.767366956025956,
#     "rule_S": {
#         "Sex": "female",
#         "Pclass": [
#             1,
#             2
#         ]
#     },
#     "complexity_S": 2,
#     "F1_pop": 0.5547445255474452,
#     "G_bad_class": 0.17059483726150393,
#     "G_information": 0.24588549145241542,
#     "G_gini": 0.14958927829841417,
```

(continues on next page)

(continued from previous page)

```
#     "p_value_ratio_S": {
#         "Pclass": 5.359920512293736e-08,
#         "Name": 1.0,
#         "Sex": 5.022554571114061e-46,
#         "Age": 1.0,
#         "SibSp": 1.0,
#         "Parch": 1.0,
#         "Ticket": 1.0,
#         "Fare": 1.0,
#         "Cabin": 1.0,
#         "Embarked": 1.0
#     },
#     "F_score_ratio_S": {
#         "Pclass": 1.2812499999999998,
#         "Name": 1.0,
#         "Sex": 0.9302417652027029,
#         "Age": 1.0,
#         "SibSp": 1.0,
#         "Parch": 1.0,
#         "Ticket": 1.0,
#         "Fare": 1.0,
#         "Cabin": 1.0,
#         "Embarked": 1.0
#     },
#     "subrules_S": [
#         {
#             "M": 891,
#             "M0": 549,
#             "M1": 342,
#             "mu_pop": 0.3838383838383838,
#             "sigma_pop": 0.48659245426485753,
#             "F1_pop": 0.5547445255474452,
#             "m": 314,
#             "m0": 81,
#             "m1": 233,
#             "coverage": 0.35241301907968575,
#             "m1/M1": 0.6812865497076024,
#             "mu_rule": 0.7420382165605095,
#             "lift": 1.9332048273550118,
#             "mc": 577,
#             "m0c": 468,
#             "m1c": 109,
#             "p_value": 3.592513266469419e-60,
#             "F_score": 0.7103658536585366,
#             "Z_score": 16.20063097451895,
#             "G_bad_class": 0.17059483726150393,
#             "G_information": 0.21766010666061436,
#             "G_gini": 0.13964795747285225,
#             "complexity": 1,
#             "subrule_S": {
#                 "Sex": "female"
#             },
```

(continues on next page)

(continued from previous page)

```
#             "var_name": "Sex",
#             "var_rule": "female",
#             "p_value_ratio": 5.022554571114061e-46,
#             "shuffling_scores": {
#                 "p_value": {
#                     "cohen_d": 75.86463627446636,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 },
#                 "Z_score": {
#                     "cohen_d": 37.71820414056423,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 },
#                 "F_score": {
#                     "cohen_d": 51.16755087975539,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 }
#             }
#         },
#         {
#             "M": 891,
#             "M0": 549,
#             "M1": 342,
#             "mu_pop": 0.3838383838383838,
#             "sigma_pop": 0.48659245426485753,
#             "F1_pop": 0.5547445255474452,
#             "m": 170,
#             "m0": 9,
#             "m1": 161,
#             "coverage": 0.19079685746352412,
#             "m1/M1": 0.47076023391812866,
#             "mu_rule": 0.9470588235294117,
#             "lift": 2.4673374613003096,
#             "mc": 721,
#             "m0c": 540,
#             "m1c": 181,
#             "p_value": 1.925558554763681e-67,
#             "F_score": 0.62890625,
#             "Z_score": 16.767366956025956,
#             "G_bad_class": 0.17059483726150393,
#             "G_information": 0.24588549145241542,
#             "G_gini": 0.14958927829841417,
#             "complexity": 2,
#             "subrule_S": {
#                 "Sex": "female",
#                 "Pclass": [
#                     1,
#                     2
#                 ]
#             },
```

(continues on next page)

(continued from previous page)

```
#             "var_name": "Pclass",
#             "var_rule": [
#                 1,
#                 2
#             ],
#             "p_value_ratio": 5.359920512293736e-08,
#             "shuffling_scores": {
#                 "p_value": {
#                     "cohen_d": 85.9832032893128,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 },
#                 "Z_score": {
#                     "cohen_d": 39.52974355288319,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 },
#                 "F_score": {
#                     "cohen_d": 23.36359433204899,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 }
#             }
#         }
#     ],
#     "feature_contributions_S": {
#         "rule_S": {
#             "Sex": "female",
#             "Pclass": "[1, 2]"
#         },
#         "p_value_contribution": {
#             "Sex": 0.8616919700920942,
#             "Pclass": 0.13830802990790583
#         },
#         "F_score_contribution": {
#             "Sex": 1.0,
#             "Pclass": 0.0
#         },
#         "Z_score_contribution": {
#             "Sex": 0.9417181496074654,
#             "Pclass": 0.05828185039253464
#         },
#         "G_bad_class_contribution": {
#             "Sex": 1.0,
#             "Pclass": 0.0
#         },
#         "G_information_contribution": {
#             "Sex": 0.857675593215252,
#             "Pclass": 0.142324406784748
#         },
#         "G_gini_contribution": {
#             "Sex": 0.9099458875412633,
```

(continues on next page)

(continued from previous page)

```
#             "Pclass": 0.0900541124587367
#         }
#     },
#     "shuffling_scores": {
#         "p_value": {
#             "cohen_d": 85.9832032893128,
#             "effect_size": "6. huge",
#             "wy_ratio": 0.0
#         },
#         "Z_score": {
#             "cohen_d": 39.52974355288319,
#             "effect_size": "6. huge",
#             "wy_ratio": 0.0
#         },
#         "F_score": {
#             "cohen_d": 23.36359433204899,
#             "effect_size": "6. huge",
#             "wy_ratio": 0.0
#         }
#     }
# }
```

This dictionary contains detailed information and statistics about the rule at position `i=0`. Here are some of the key entries:

- `"m": 170`: This is the number of points captured by the rule. The rule contains 170 points in total.
- `"m0": 9`: This is the number of `0` captured by the rule. The rule contains 9 non-survivors.
- `"m1": 161`: This is the number of `1` captured by the rule. The rule contains 161 survivors.
- `"coverage": 0.19079685746352412`: This is the coverage of the rule, i.e. the ratio `m/M`. The rule covers 19.1% of the population.
- `"m1/M1": 0.47076023391812866`: This is the sensitivity of the rule, i.e. the capture rate of `1`. The rule captures 47.1% of the survivors.
- `"mu_rule": 0.9470588235294117`: This is the average of the target variable in the rule, i.e. the ratio `m1/m`. Here we have a survival rate of 94.7% in the rule.
- `"mu_pop": 0.3838383838383838`: This the average of the target variable in the population, i.e. the ration `M1/M`. Here we have a survival rate of 38.4% in the population.
- `"sigma_pop": 0.48659245426485753`: This is the standard deviation of the target variable in the population.
- `"lift": 2.4673374613003096`: This is the lift of the rule, i.e. the ratio `mu_rule/mu_pop`.
- `"p_value": 1.925558554763681e-67`: This is the p-value (according to the hypergeometric probability law, not the chi-squared) of the rule.
- `"F_score": 0.62890625`: This is the F1-score of the rule.
- `"Z_score": 16.767366956025956`: This is the Z-score of the rule.
- `"rule_S": {"Sex": "female","Pclass": [1,2]}`: The rule reads *Females in first or second class.*
- `"complexity_S": 2`: The complexity of the rule is 2, i.e. two variables are involded in the rule (`"Sex"` and `"Pclass"`).

- `"F1_pop": 0.5547445255474452`: This is the F1-score of the population.
- `"G_bad_class": 0.17059483726150393`: This is the bad classification gain of the rule.
- `"G_information": 0.24588549145241542`: This is the information gain of the rule.
- `"G_gini": 0.14958927829841417`: This is the Gini gain of the rule.
- `"shuffling_scores"`: This contains the scores that measure how strong is the rule compared to what would be found in shuffled data.

3.2.6 DataFrame of subrules

We can retrieve a DataFrame of the subrules for the rule at position `i=0` as follow:

```
    solver.i_to_subrules_dataframe(i=0)

#    p_value_ratio variable    rule  complexity       p_value   F_score  ...  m0c  m1c␣
↪G_bad_class  G_information    G_gini            subrule_S
# 0   5.022555e-46      Sex  female           1  3.592513e-60  0.710366  ...  468  109␣
↪  0.170595       0.217660  0.139648    {'Sex': 'female'}
# 1   5.359921e-08   Pclass  [1, 2]           2  1.925559e-67  0.628906  ...  540  181␣
↪  0.170595       0.245885  0.149589  {'Sex': 'female'...
```

The DataFrame of subrules begins with a rule of complexity 1 (e.g. `{"Sex": "female"}`) and progresses to higher complexities, such as complexity 2 (e.g. `{"Sex": "female","Pclass": [1,2]}`). As we can observe, increasing the complexity from 1 to 2 improves the p-values and the information gain but degrades the F1-score.

The purpose of the subrules DataFrame is to assist in deciding the optimal level of rule complexity based on various metrics.

3.2.7 DataFrame of features contributions

We can retrieve a DataFrame showing the contributions of the features for the rule at position `i=0` as follows:

```
    solver.i_to_feature_contributions_S(i=0)

#                p_value  F_score   Z_score  G_bad_class  G_information    G_gini
# feature_name
# Sex           0.861692      1.0  0.941718          1.0       0.857676  0.909946
# Pclass        0.138308      0.0  0.058282          0.0       0.142324  0.090054
```

As we can observe, the variable `"Sex"` provides the largest contribution. The variable `"Pclass"` adds a slight positive contribution to both the p-value and the information gain, as including it in the rule improves these metrics. However, the contribution of `"Pclass"` is zero for the F-score. This indicates that it does not enhance the F-score (in fact, it degrades it, but by convention, contributions are kept nonnegative).

3.2.8 Printing modes

Earlier in *Titanic* we saw the `dense` printing mode. There are three printing modes:

- `full`: A full print of the results.
- `light`: A lighter version of the full print.
- `dense`: A very compact version of the print.

3.2.9 Column types

The columns of a Pandas DataFrame are associated with a type known as a *dtype*, such as `int64`, `float64`, `object`, and so on. In addition to these, InsightSolver introduces a complementary layer of types called *btype*.

While *dtypes* describe the encoding of the data (e.g., integers or floats), *btypes* define how the data should be interpreted when mining for rules. These *btypes* include:

- `binary`: The variable is treated as a binary categorical variable, and rule mining will focus on finding subsets.
- `multiclass`: The variable is treated as a multiclass categorical variable, and rule mining will aim to find subsets.
- `continuous`: The variable is treated as an ordered variable, and rule mining will focus on identifying meaningful intervals.
- `ignore`: The variable is excluded from rule mining.

The *btypes* of the columns are automatically detected in InsightSolver, so its not mandatory to explicitly specify a *btype* for each variable. However, if the user wishes to specify the *btype* for some or all variables, this can be done using the `columns_types` dictionary (a key is a column name, a value is a *btype*). The `columns_types` dictionary can be passed as a parameter of the `solver`.

CHAPTER

FOUR

MODULES

4.1 insightsolver module

- *Organization*: InsightSolver Solutions Inc.
- *Project Name*: InsightSolver
- *Module Name*: insightsolver
- *File Name*: insightsolver.py
- *Author*: Noé Aubin-Cadot
- *Email*: noe.aubin-cadot@insightsolver.com
- *Last Updated*: 2025-08-28
- *First Created*: 2024-09-09

4.1.1 Description

This file contains the `InsightSolver` class. This class is meant to ingest data, specify rule mining parameters and make rule mining API calls.

4.1.2 Note

A service key is necessary to use the API client.

4.1.3 License

Exclusive Use License - see LICENSE for details.

`insightsolver.insightsolver.`**`compute_admissible_btypes`**(*M: int*, *dtype: str*, *nunique: int*, *name: str*) → list[str]

This function computes the admissible *btypes* a column can take. The *btypes*:

- `'binary'`
- `'multiclass'`
- `'continuous'`
- `'ignore'`

insightsolver.insightsolver.**compute_columns_names_to_admissible_btypes**(*df: DataFrame*) → dict[str, list[str]]

This function computes a dict that maps the column names of `df` to lists of admissible btypes.

insightsolver.insightsolver.**validate_class_integrity**(*df: DataFrame | None*, *target_name: str | int | None*, *target_goal: str | Real | uint8 | None*, *columns_types: Dict | None*, *columns_descr: Dict | None*, *threshold_M_max: int | None*, *specified_constraints: Dict | None*, *top_N_rules: int | None*, *filtering_score: str*, *n_benchmark_original: int*, *n_benchmark_shuffle: int*, *do_strict_types: bool = False*, *verbose: bool = False*) → dict

This function validates the integrity of the parameter values passed during the instantiation of the InsightSolver class.

4.1.4 Parameters

df: DataFrame
The DataFrame that contains the data to analyse (a target column and various feature columns).

target_name: str
Name of the column of the target variable.

target_goal: str (or other modality of the target variable)
Target goal.

columns_types: dict
Types of the columns.

columns_descr: dict
Descriptions of the columns.

threshold_M_max: int
Threshold on the maximum number of observations to consider, above which we sample observations.

specified_constraints: dict
Dictionary of the specified constraints on m_min, m_max, coverage_min, coverage_max.

top_N_rules: int
An integer that specifies the maximum number of rules to get from the rule mining.

filtering_score: str
A string that specifies the filtering score to be used when selecting rules.

n_benchmark_original: int
An integer that specifies the number of benchmarking runs to execute where the target is not shuffled.

n_benchmark_shuffle: int
An integer that specifies the number of benchmarking runs to execute where the target is shuffled.

do_strict_types: bool (default False)
A boolean that specifies if we want a strict evaluation of types.

verbose: bool (default False)
Verbosity.

4.1.5 Returns

columns_types: dict
A dict of the columns types after adjusting the types.

insightsolver.insightsolver.**format_value**(*value*, *format_type='scientific'*, *decimals=1*, *verbose=False*)

This function formats values depending on the type of values (`float` or `mpmath`) and the type of the format to show:

- `'percentage'`: shows the values as percentage (default).
- `'scientific'`: shows the values in scientific notation with 4 decimals.
- `'scientific_no_decimals'`: shows the values in scientific notation without decimals.

insightsolver.insightsolver.**S_to_index_points_in_rule**(*solver*, *S: dict*, *verbose: bool = False*, *df: DataFrame | None = None*) → Index

This function takes a rule `S` and returns the index of the points inside the rule of a DataFrame. If no DataFrame is provided, the one used to train the solver is used.

insightsolver.insightsolver.**resolve_language**(*language: str = 'auto'*, *default_language: str = 'english'*) → str

insightsolver.insightsolver.**search_best_ruleset_from_API_public**(*df: DataFrame*, *computing_source: str = 'auto'*, *input_file_service_key: str | None = None*, *user_email: str | None = None*, *target_name: str | int | None = None*, *target_goal: str | Real | uint8 | None = None*, *columns_names_to_btypes: Dict | None = {}*, *columns_names_to_descr: Dict | None = {}*, *threshold_M_max: int | None = None*, *specified_constraints: Dict | None = {}*, *top_N_rules: int | None = 10*, *verbose: bool = False*, *filtering_score: str = 'auto'*, *api_source: str = 'auto'*, *do_compress_data: bool = True*, *do_compute_memory_usage: bool = True*, *n_benchmark_original: int = 5*, *n_benchmark_shuffle: int = 20*, *do_llm_readable_rules: bool = False*, *llm_source: str = 'remote_gemini'*, *llm_language: str = 'auto'*, *do_store_llm_cache: bool = True*, *do_check_llm_cache: bool = True*) → dict

This function is meant to make a rule mining API call.

4.1.6 Parameters

df: DataFrame
The DataFrame that contains the data to analyse (a target column and various feature columns).

computing_source: str
If the rule mining should be computed locally or remotely.

input_file_service_key: str
The string that specifies the path to the service key (necessary to use the remote Cloud Function from outside GCP).

user_email: str
The email of the user (necessary to use the remote Cloud Function from inside GCP).

target_name: str
Name of the column of the target variable.

target_goal: str (or other modality of the target variable)
Target goal.

columns_names_to_btypes: dict
A dict that specifies the btypes of the columns.

columns_names_to_descr: dict
A dict that specifies the descriptions of the columns.

threshold_M_max: int
Threshold on the maximum number of points to use during the rule mining (max. 10000 pts in the public API).

specified_constraints: dict
A dict that specifies contraints to be used during the rule mining.

top_N_rules: int
An integer that specifies the maximum number of rules to get from the rule mining.

verbose: bool
Verbosity.

filtering_score: str
A string that specifies the filtering score to be used when selecting rules.

api_source: str
A string used to identify the source of the API call.

do_compress_data: bool
A boolean that specifies if we want to compress the data.

do_compute_memory_usage: bool
A bool that specifies if we want to get the memory usage of the computation.

n_benchmark_original: int
Number of benchmarking runs to execute where the target is not shuffled.

n_benchmark_shuffle: int
Number of benchmarking runs to execute where the target is shuffled.

do_llm_readable_rules: bool
If we want to convert the rules to a readable format using a LLM.

llm_source: str
Source where the LLM is running

do_store_llm_cache: bool
If we want to store the result of the LLM in the cache (makes futur LLM calls faster).

do_check_llm_cache: bool
If we want to check if the results of the prompt are found in the cache (makes LLM calls faster).

4.1.7 Returns

response: requests.models.Response
A response object obtained from the API call that contains the rule mining results.

insightsolver.insightsolver.**get_credits_available**(*computing_source: str = 'auto'*, *service_key: str | None = None*, *user_email: str | None = None*)

This function is meant to retrieve from the server the amount of credits available.

class insightsolver.insightsolver.**InsightSolver**(*verbose: bool = False*, *df: DataFrame = None*, *target_name: str | int | None = None*, *target_goal: str | Real | uint8 | None = None*, *columns_types: Dict | None = {}*, *columns_descr: Dict | None = {}*, *threshold_M_max: int | None = 10000*, *specified_constraints: Dict | None = {}*, *top_N_rules: int | None = 10*, *filtering_score: str = 'auto'*, *n_benchmark_original: int = 5*, *n_benchmark_shuffle: int = 20*)

Bases: `Mapping`

The class `InsightSolver` is meant to :

1. Take input data.
2. Make an insightsolver API calls to the server.
3. Present the results of the rule mining.

4.1.8 Attributes

df: DataFrame
The DataFrame that contains the data to analyse.

target_name: str (default None)
Name of the target variable (by default it's the first column).

target_goal: (str or int)
Target goal.

target_threshold: (int or float)
Threshold used to convert a continuous target variable to a binary target variable.

M: int
Number of points in the population.

M0: int
Number of points 0 in the population.

M1: int
Number of points 1 in the population.

columns_types: dict
Types of the columns.

columns_descr: dict
Textual descriptions of the columns.

other_modalities: dict
Modalities that are mapped to the modality 'other'.

threshold_M_max: int (default 10000)
Threshold on the maximum number of observations to consider, above which we under sample the observations.

specified_constraints: dict
Dictionary of the specified constraints on `m_min`, `m_max`, `coverage_min`, `coverage_max`.

top_N_rules: int (default 10)
An integer that specifies the maximum number of rules to get from the rule mining.

filtering_score: str (default 'auto')
A string that specifies the filtering score to be used when selecting rules.

n_benchmark_original: int (default 5)
An integer that specifies the number of benchmarking runs to execute where the target is not shuffled.

n_benchmark_shuffle: int (default 20)
An integer that specifies the number of benchmarking runs to execute where the target is shuffled.

monitoring_metadata: dict
Dictionary of monitoring metadata.

benchmark_scores: dict
Dictionary of the benchmarking scores against shuffled data.

rule_mining_results: dict
Dictionary that contains the results of the rule mining.

_is_fitted: bool
Boolean that tells if the solver is fitted.

4.1.9 Methods

__init__: None
Initialization of an instance of the class InsightSolver.

__str__: None
Converts the solver to a string as provided by the print method.

ingest_dict: None
Ingests a Python dict.

ingest_json_string: None
Ingests a JSON string.

is_fitted: Bool
Returns a boolean that tells if the solver is fitted.

fit: None
Fits the solver.

S_to_index_points_in_rule: Pandas Index
Returns the index of the points in a rule S.

S_to_s_points_in_rule: Pandas Series
Returns a boolean Pandas Series that tells if the point is in the rule S.

S_to_df_filtered: Pandas DataFrame
Returns the filtered df of rows that are in the rule S.

ruleset_count: int
Counts the number of rules held by the InsightSolver.

i_to_rule: dict
Gives the rule i of the InsightSolver.

i_to_S: dict
Returns the rule S for the rule at index i.

i_to_subrules_dataframe: Pandas DataFrame
Returns a DataFrame containing the informations about the subrules of the rule i.

i_to_feature_contributions_S: Pandas DataFrame
Returns a DataFrame of the feature contributions of the variables in the rule S at position i.

i_to_readable_text: str
Returns the readable text of the rule i if it is available.

i_to_print: None
Prints the content of the rule i in the InsightSolver.

get_range_i: list
Gives the range of i in the InsightSolver.

print: None
Prints the content of the InsightSolver.

print_light: None
Prints the content of the InsightSolver ('light' mode).

print_dense: None
Prints the content of the InsightSolver ('dense' mode).

to_dict: dict
Exports the content of the InsightSolver object to a Python dict.

to_json_string: str
Exports the content of the InsightSolver object to a JSON string.

to_dataframe: Pandas DataFrame
Exports the rule mining results to a Pandas DataFrame.

to_csv: str
Exports the rule mining results to a CSV string and/or a local CSV file.

to_excel: None
Exports the rule mining results to a Excel file.

to_excel_string: str
Exports the rule mining results to a Excel string.

get_credits_needed_for_computation: int
Get the number of credits needed for the fitting computation of the solver.

get_df_credits_infos: Pandas DataFrame
Get a DataFrame of the transactions involving credits.

get_credits_available: int
Get the number of credits available.

convert_target_to_binary: pd.Series
Converts the target variable to a binary {0,1}-valued Pandas Series.

compute_mutual_information: pd.Series
Computes a Pandas Series of the mutual information between features and the target variable.

show_all_mutual_information: None
Generates a bar plot of the mutual information between the features and the target variable.

show_feature_distributions_of_S: None
Generates bar plots of the distributions of the points in the specified rule S.

show_feature_contributions_of_i: None
Generates a horizontal bar plot of the feature contributions of the rule at position `i`.

show_all_feature_contributions: None
Generates the feature contributions and feature distributions for all rules found in the solver.

show_all_feature_contributions_and_distributions: None
Generates the feature contributions and feature distributions for all rules found in the solver.

4.1.10 Example

Here's a sample code to use the class `InsightSolver`:

```
# Specify the service key
service_key = 'name_of_your_service_key.json'

# Import some data
import pandas as pd
df = pd.read_csv('kaggle_titanic_train.csv')

# Specify the name of the target variable
target_name = 'Survived' # We are interested in whether the passengers survived or␣
↪not

# Specify the target goal
target_goal = 1 # We are searching rules that describe survivors

# Import the class InsightSolver from the module insightsolver
from insightsolver import InsightSolver

# Create an instance of the class InsightSolver
solver = InsightSolver(
        df          = df,          # A dataset
        target_name = target_name, # Name of the target variable
        target_goal = target_goal, # Target goal
)

# Fit the solver
solver.fit(
        service_key = service_key, # Use your API service key here
)

# Print the rule mining results
solver.print()
```

__init__(*verbose: bool = False*, *df: DataFrame = None*, *target_name: str | int | None = None*, *target_goal: str | Real | uint8 | None = None*, *columns_types: Dict | None = {}*, *columns_descr: Dict | None = {}*, *threshold_M_max: int | None = 10000*, *specified_constraints: Dict | None = {}*, *top_N_rules: int | None = 10*, *filtering_score: str = 'auto'*, *n_benchmark_original: int = 5*, *n_benchmark_shuffle: int = 20*)

The initialization occurs when an `InsightSolver` class instance is created.

Parameters

verbose: bool (default False)
If we want the initialization to be verbose.

df: DataFrame
The DataFrame that contains the data to analyse (a target column and various feature columns).

target_name: str
Name of the column of the target variable.

target_goal: str (or other modality of the target variable)
Target goal.

columns_types: dict
Types of the columns.

columns_descr: dict
Descriptions of the columns.

threshold_M_max: int
Threshold on the maximum number of observations to consider, above which we sample observations.

specified_constraints: dict
Dictionary of the specified constraints on m_min, m_max, coverage_min, coverage_max.

top_N_rules: int (default 10)
An integer that specifies the maximum number of rules to get from the rule mining.

filtering_score: str (default 'auto')
A string that specifies the filtering score to be used when selecting rules.

n_benchmark_original: int (default 5)
An integer that specifies the number of benchmarking runs to execute where the target is not shuffled.

n_benchmark_shuffle: int (default 20)
An integer that specifies the number of benchmarking runs to execute where the target is shuffled.

Returns

solver: InsightSolver
An instance of the class InsightSolver.

Example

Here's a sample code to instantiante the class `InsightSolver`:

```
# Import the class InsightSolver from the module insightsolver
from insightsolver import InsightSolver

# Create an instance of the class InsightSolver
solver = InsightSolver(
```

(continues on next page)

(continued from previous page)

```
        df          = df,          # A Pandas DataFrame
        target_name = target_name, # Name of the target variable
        target_goal = target_goal, # Target goal
)
```

ingest_dict(*d: dict*, *verbose: bool = False*) → None

This method aims to ingest a Python dict in the solver.

ingest_json_string(*json_string: str*, *verbose: bool = False*) → None

This method aims to ingest a JSON string in the solver.

is_fitted()

This method returns a boolean that tells if the solver is fitted.

fit(*verbose: bool = False*, *computing_source: str = 'auto'*, *service_key: str | None = None*, *user_email: str | None = None*, *api_source: str = 'auto'*, *do_compress_data: bool = True*, *do_compute_memory_usage: bool = True*, *do_check_enough_credits: bool = False*, *do_llm_readable_rules: bool = True*, *llm_source: str = 'auto'*, *llm_language: str = 'auto'*, *do_store_llm_cache: bool = True*, *do_check_llm_cache: bool = True*) → None

This method aims to fit the solver.

Parameters

verbose: bool (default False)
If we want the fitting to be verbose.

computing_source: str (default 'auto')
Specify where the rule mining computation is done ('local_cloud_function' or 'remote_cloud_function').

service_key: str (default None)
Path+name of the service key.

user_email: str (default None)
User email.

api_source: str (default 'auto')
Source of the API call.

do_compress_data: bool (default True)
If we want to compress the data for the communications with the server.

do_compute_memory_usage: bool (default True)
If we want to monitor the first thread memory usage on the server side.

do_check_enough_credits: bool (default False)
Check if there are enough credits to fit the solver.

do_llm_readable_rules: bool (default True)
If we want to convert the rules to a readable format using a LLM.

llm_source: str (default 'auto')
Source where the LLM is running.

llm_language: str (default 'auto')
Language of the LLM.

do_store_llm_cache: bool (default True)
If we want to store the result of the LLM in the cache (makes futur LLM calls faster).

do_check_llm_cache: bool (default True)
If we want to check if the results of the prompt are found in the cache (makes LLM calls faster).

S_to_index_points_in_rule(*S: dict*, *verbose: bool = False*, *df: DataFrame | None = None*) → Index

This method returns the index of the points inside a rule `S`.

S_to_s_points_in_rule(*S: dict*, *verbose: bool = False*, *df: DataFrame | None = None*) → Series

This method returns a boolean Series that tells if the points are in the rule `S` or not.

S_to_df_filtered(*S: dict*, *verbose: bool = False*, *df: DataFrame | None = None*)

This method returns the DataFrame of rows of `df` that lie inside a rule `S`.

ruleset_count() → int

This method returns the number of rules held in an instance of the solver.

i_to_rule(*i: int*) → dict

i_to_S(*i*)

This method returns the rule `S` at position `i`.

i_to_subrules_dataframe(*i: int = 0*) → DataFrame

This method returns a DataFrame which contains the informations about the subrules of the rule `i`.

i_to_feature_contributions_S(*i: int*, *do_rename_cols: bool = True*, *do_ignore_col_rule_S: bool = True*) → DataFrame

This method returns a DataFrame of the feature contributions of the variables in the rule `S` at position `i`.

i_to_readable_text(*i*) → str | None

Returns the readable text of the rule `i` if it is available.

i_to_print(*i: int*, *indentation: str = ''*, *do_print_shuffling_scores: bool = True*, *do_print_rule_DataFrame: bool = False*, *do_print_subrules_S: bool = True*, *do_show_coverage_diff: bool = False*, *do_show_cohen_d: bool = True*, *do_show_wy_ratio: bool = True*, *do_print_feature_contributions_S: bool = True*) → None

This method prints the content of the rule `i` in the solver.

Parameters

i: int
Index of the rule to print.

indentation: str
Indentation of some printed elements.

do_print_shuffling_scores: bool
If we want to print the shuffling scores.

do_print_rule_DataFrame: bool
If we want to print a DataFrame of the rule.

do_print_subrules_S: bool
If we want to print the DataFrame of subrules.

do_show_coverage_diff: bool
If we want to show the differences of coverage in the DataFrame of subrules.

do_show_cohen_d: bool
If we want to show the Cohen d in the DataFrame of subrules.

do_show_wy_ratio: bool
If we want to show the WY ratio in the DataFrame of subrules.

do_print_feature_contributions_S: bool
If we want to print the DataFrame of feature contributions.

`get_range_i`(*complexity_max: int | None = None*) → list

This method gives the range of `i` in the solver. If the integer `complexity_max` is specified, return only this number of elements.

`print`(*verbose: bool = False*, *r: int | None = None*, *do_print_dataset_metadata: bool = True*, *do_print_monitoring_metadata: bool = False*, *do_print_benchmark_scores: bool = True*, *do_print_shuffling_scores: bool = True*, *do_show_cohen_d: bool = True*, *do_show_wy_ratio: bool = True*, *do_print_rule_mining_results: bool = True*, *do_print_rule_DataFrame: bool = False*, *do_print_subrules_S: bool = True*, *do_show_coverage_diff: bool = False*, *do_print_feature_contributions_S: bool = True*, *separation_width_between_rules: int | None = 79*, *do_print_last_separator: bool = True*, *mode: str = 'full'*) → None

This method prints the content of the `InsightSolver` solver.

`print_light`(*print_format: str = 'list'*, *do_print_shuffling_scores: bool = True*, *do_print_last_separator: bool = True*) → None

This method does a 'light' print of the solver.

Two formats:

- `'list'`: shows the rules via a loop of prints.
- `'compact'`: shows the rules in a single DataFrame.

`print_dense`(*do_print_shuffling_scores: bool = True*) → None

This method is aimed at printing a 'dense' version of the solver.

`to_dict`() → dict

This method aims to export the content of the solver to a dictionary.

`to_json_string`(*verbose=False*) → str

This method aims to export the content of the solver to a JSON string.

`to_dataframe`(*verbose=False*, *do_append_datetime=False*, *do_rename_cols=False*) → DataFrame

This method aims to export the content of the solver to a DataFrame.

`to_csv`(*output_file=None*, *verbose=False*, *do_rename_cols=False*) → str

This method is meant to export the content of the solver to a CSV file.

`to_excel`(*output_file*, *verbose=False*, *do_rename_cols=False*) → None

This method is meant to export the solver to a Excel file.

`to_excel_string`(*verbose=False*, *do_rename_cols=False*) → str

This method is meant to export the solver to a Excel string.

`get_credits_needed_for_computation`() → int

This method is meant to compute the number of credits for the computation during the fitting of the solver.

`get_df_credits_infos`(*computing_source: str = 'auto'*, *service_key: str | None = None*, *user_email: str | None = None*) → DataFrame

This method is meant to retrieve from the server the transactions involving credits.

get_credits_available(*computing_source: str = 'auto'*, *service_key: str | None = None*, *user_email: str | None = None*) → int

This method is meant to retrieve from the server the amount of credits available.

convert_target_to_binary()

This method converts the target variable to a binary {0,1}-valued Pandas Series.

To use this method, the attribute `solver.target_goal` must be populated because it specifies how to convert the target variable to binary. As a reminder, the target goal must be one of the following:

- A modality of the target variable in the case of a categorical (i.e. `'binary'` or `'multiclass'`) target variable.
- `'min'`, `'min_q0'`, `'min_q1'`, `'min_q2'`, `'min_q3'`, `'min_c00'`, `'min_c01'`, ..., `'min_c98'`, `'min_c99'`.
- `'max'`, `'max_q1'`, `',max_q2'`, `'max_q3'`, `'max_q4'`, `'max_c01'`, `'max_c02'`, ..., `'max_c99'`, `'max_c100'`.

Returns

s_target: pd.Series
A {0,1}-valued Pandas Series representing the target variable.

compute_mutual_information(*n_samples=1000*)

This method computes the mutual information between the features and the target variable.

Parameters

n_samples: int
An integer that specifies the number of data rows to use in the computation of the mutual information.

Returns

s_mi: pd.Series
A Pandas Series that contains the mutual information of the features with the target variable.

show_all_mutual_information(*n_samples: int | None = 1000*, *n_cols: int | None = 20*) → None

This method generates a bar plot of the mutual information between the features and the target variable.

Parameters

n_samples: int
An integer that specifies the number of data rows to use in the computation of the mutual information.

n_cols: int
An integer that specifies the maximum number of features to show

show_feature_distributions_of_S(*S: dict*, *padding_y: int = 5*, *do_show_kde: bool = False*, *do_show_vertical_lines: bool = False*) → None

This method generates bar plots of the distributions of the points in the specified rule S.

Parameters

S
[dict] The rule S that we wish to visualize.

padding_y: int
The padding used for the ylim.

do_show_kde: bool
Boolean to show the KDE of the continuous features.

`show_feature_contributions_of_i`(*i: int*, *a: float = 0.5*, *b: float = 1*, *fig_width: float = 12*, *language: str = 'en'*, *do_grid: bool = True*, *do_title: bool = False*, *do_banner: bool = True*, *bar_annotations: str = 'p_value_ratio'*, *loss: float | None = None*) → None

This method generates a horizontal bar plots of the feature constributions of the rule at position `i`.

Parameters

i: int
The index of the rule to show.

a: float
Height per bar.

b: float
Added height to the figure.

fig_width: float
Width of the figure

language: str
Language of the figure ('fr' or 'en').

do_grid: bool
If we want to show a vertical grid behind the horizontal bars.

do_title: bool
If we want to show a title.

do_banner: bool
If we want to show the banner.

bar_annotations: str
Type of values to show at the end of the bars (can be 'p_value_ratio', 'p_value_contribution' or None)

loss: float
If we want to show a loss.

`show_all_feature_contributions`(*a: float = 0.5*, *b: float = 1*, *fig_width: float = 12*, *language: str = 'en'*, *do_grid: bool = True*, *do_title: bool = False*, *do_banner: bool = True*, *bar_annotations: str = 'p_value_ratio'*) → None

This method generates a horizontal bar plot of the feature contributions for each rule found in a solver.

Parameters

a: float
Height per bar.

b: float
Added height to the figure.

fig_width: float
Width of the figure

language: str
Language of the figure ('fr' or 'en').

do_grid: bool
If we want to show a vertical grid behind the horizontal bars.

do_title: bool
If we want to show a title.

do_banner: bool
If we want to show the banner.

bar_annotations: str
Type of values to show at the end of the bars (can be 'p_value_ratio', 'p_value_contribution' or None)

`show_all_feature_contributions_and_distributions`(*do_banner: bool = True*, *bar_annotations: str = 'p_value_ratio'*) → None

This method generates the feature contributions and feature distributions for all rules found in a fitted solver.

Parameters

do_banner: bool
If we want to show the banner.

bar_annotations: str
Type of values to show at the end of the bars (can be 'p_value_ratio', 'p_value_contribution' or None)

`plot`(*do_banner: bool = True*, *bar_annotations: str = 'p_value_ratio'*) → None

This method is an alias for the method .show_all_feature_contributions_and_distributions()

Parameters

do_banner: bool
If we want to show the banner.

bar_annotations: str
Type of values to show at the end of the bars (can be 'p_value_ratio', 'p_value_contribution' or None)

4.2 api_utilities module

- *Organization*: InsightSolver Solutions Inc.
- *Project Name*: InsightSolver
- *Module Name*: insightsolver
- *File Name*: api_utilities.py
- *Author*: Noé Aubin-Cadot
- *Email*: noe.aubin-cadot@insightsolver.com
- *Last Updated*: 2025-09-08
- *First Created*: 2024-09-16

4.2.1 Description

This file provides essential utility functions to secure and streamline client-server communication within the API. It includes functions for data compression, encryption, decryption, and transformations of data structures, all designed to facilitate efficient and protected message exchange between the client and server.

While all communications are secured via HTTPS, this file goes a step further by adding an additional layer of encryption, using RSA-4096 and ECDSA-SECP521R1 for secure key exchange and AES-256 for data encryption. These functions are particularly useful for scenarios requiring enhanced data privacy and integrity.

4.2.2 Functions provided

- `hash_string`: Computes the hash of a string.
- `convert_bytes_to_base64_string`: Convert bytes to a base64 string.
- `convert_base64_string_to_bytes`: Convert a base64 string to bytes.
- `compress_string`: Compress a string using gzip.
- `decompress_string`: Decompress a gzip-compressed string.
- `compress_and_encrypt_string`: Compress and encrypt a string for secure transmission.
- `decrypt_and_decompress_string`: Decrypt an encrypted string.
- `encode_obj`: Takes an object and encode it to a new object compatible with json serialization.
- `convert_dict_to_json_string`: Convert a dict to a json string.
- `decode_obj`: Inverse operation from `encode_obj`.
- `convert_json_string_to_dict`: Convert a json string to a dict.
- `transform_dict`: Convert a dictionary for easier client-server communication.
- `untransform_dict`: Reverse the dictionary transformation to restore the original data format.
- `generate_keys`: Generate RSA and ECDSA private and public keys.
- `compute_credits_from_df`: Compute the amount of credits consumed for a given DataFrame.
- `request_cloud_credits_infos`: Request the server for informations about the credits available.
- `request_cloud_public_keys`: Request the server for public keys.
- `request_cloud_computation`: Request the server for computation.
- `search_best_ruleset_from_API_dict`: Make the API call.

4.2.3 License

`insightsolver.api_utilities.`**`hash_string`**`(`*`string`*`)`

A function to compute the hash of a string using hashlib.

`insightsolver.api_utilities.`**`convert_bytes_to_base64_string`**`(`*`data: bytes`*`)` → str

Convert a bytes object to a base64-encoded string.

4.2.4 Parameters

data
: [bytes] The byte data to encode.

4.2.5 Returns

str
: The base64-encoded string.

insightsolver.api_utilities.**convert_base64_string_to_bytes**(*string: str*) → bytes

Convert a base64-encoded string to a bytes object.

4.2.6 Parameters

string
: [str] The base64-encoded string.

4.2.7 Returns

bytes
: The decoded byte data.

insightsolver.api_utilities.**compress_string**(*original_string: str*) → str

Compress a string using gzip and then encode it to base64.

4.2.8 Parameters

original_string
: [str] The original string to be compressed.

4.2.9 Returns

str
: The compressed string.

4.2.10 Example

```
original_string = "This is a test string"
compressed_string = compress_string(original_string)
print(compressed_string)  # Example output: 'H4sIAA01/2YC/
↪wvJyCxWAKJEhZLU4hKF4pKizLx0AG3zTmsVAAAA'
```

insightsolver.api_utilities.**decompress_string**(*compressed_string: str*) → str

Decompress a base64-encoded string that was previously compressed using gzip.

This function takes a base64-encoded string, decodes it, and then decompresses the resulting data using gzip to return the original string.

4.2.11 Parameters

compressed_string
: [str] The base64-encoded string that contains the compressed data.

4.2.12 Returns

str
The original uncompressed string.

4.2.13 Example

```
compressed_string = 'H4sIAA01/2YC/wvJyCxWAKJEhZLU4hKF4pKizLx0AG3zTmsVAAAA'
original_string = decompress_string(compressed_string)
print(original_string) # 'This is a test string'
```

insightsolver.api_utilities.**compress_and_encrypt_string**(*original_string: str*, *symmetric_key: bytes*) → tuple[str, str]

Compress and encrypt a string using AES-256-GCM.

This function compresses the given string using gzip and then encrypts it using AES-256 in GCM mode. A nonce is used in the encryption process for AES-GCM, and the result is base64-encoded for easy transfer over networks.

Security: - AES-256 encryption - GCM (Galois/Counter Mode) with authentication

4.2.14 Parameters

original_string
[str] The original string to be compressed and encrypted.

symmetric_key
[bytes] The 32-byte symmetric key used for encryption.

4.2.15 Returns

tuple[str, str]
A tuple containing the base64-encoded encrypted compressed string and the base64-encoded nonce used.

4.2.16 Example

```
transformed_string, nonce_string = compress_and_encrypt_string(
        original_string = "Secret data",
        symmetric_key   = token_bytes(32),
)
print(transformed_string, nonce_string) # 'Base64_encoded_result', nonce_string
```

insightsolver.api_utilities.**decrypt_and_decompress_string**(*transformed_string: str*, *symmetric_key: bytes*, *nonce: bytes*) → str

Decrypt and decompress a string using AES-256-GCM.

This function takes a base64-encoded encrypted string, decrypts it using AES-256 in GCM mode with the provided symmetric key and nonce, and then decompresses the result using gzip.

Security: - AES-256 encryption - GCM (Galois/Counter Mode) with authentication

4.2.17 Parameters

transformed_string
[str] The base64-encoded string that contains the encrypted and compressed data.

symmetric_key
[bytes] The 32-byte symmetric key used for decryption.

nonce
[bytes] The nonce used for AES-GCM during encryption.

4.2.18 Returns

str
The original uncompressed and decrypted string.

4.2.19 Raises

Exception
If the decryption fails.

4.2.20 Example

```
original_string = decrypt_and_decompress_string(
        transformed_string = encrypted_compressed_string,
        symmetric_key      = token_bytes(32),
        nonce              = nonce
)
print(original_string) # 'Secret data'
```

insightsolver.api_utilities.**encode_obj**(*obj*)

This function takes an object and encode it to a new object compatible with json serialization.

insightsolver.api_utilities.**convert_dict_to_json_string**(*d: dict*) → str

This function converts a dict to a json string.

insightsolver.api_utilities.**decode_obj**(*obj*)

This function does the inverse operation from the function `encode_obj`.

insightsolver.api_utilities.**convert_json_string_to_dict**(*string: str*) → dict

This function takes a json string and converts it to a dict.

insightsolver.api_utilities.**transform_dict**(*d_original: dict*, *do_compress_data: bool = False*, *symmetric_key: bytes | None = None*, *json_format: str = 'json'*) → dict

Transform the contents of a dictionary by optionally compressing and encrypting its data.

This function takes a dictionary and converts it to a string. Depending on the options provided, it can compress the data using gzip, encrypt it using AES-256, or both. The resulting string is returned in a transformed dictionary format for easier transmission or storage.

4.2.21 Parameters

d_original
[dict] The original dictionary that needs to be transformed.

do_compress_data
[bool, optional] Whether or not to compress the dictionary data (default is False).

symmetric_key
[bytes, optional] A symmetric key. Typically generated using from secrets import token_bytes;symmetric_key = token_bytes(32). If provided, the data will be encrypted (default is None).

json_format
[str, optional] The format to convert the dictionary to a string. Can be 'json' or 'json_extended' (default is 'json').

4.2.22 Returns

dict
A dictionary containing the transformed string, the transformations applied, and the json format.

4.2.23 Example

```
d_original = {'A':1, 'B':2, 'C':3}
from secrets import token_bytes
symmetric_key = token_bytes(32) # b'\x1a\xef&\x0bR\xe1\x95\xfa\x90\x10r\x93\x1a\xaeN\
↪xc2\xba\x80\xf1\x1a\x0fG\xf4(\x0e#\xd4\xaf`\x81q\xf4'
d_transformed = transform_dict(
        d_original       = d_original,
        do_compress_data = True,
        symmetric_key    = symmetric_key,
        json_format      = 'json',
)
print(d_transformed)
# {
#       'transformations': 'encrypted_gzip_base64',
#       'json_format': 'json',
#       'transformed_string':
↪'q30qPkK19Z3sENnfk77t4CnpzWKV+gdHLLSpNNgU3DjdmEbLcZWj+AjZyFmUquuUmh6obZmTh8k=',
#       'nonce_string': '7PpTvoc0Ksx8whRy',
# }
```

insightsolver.api_utilities.**untransform_dict**(*d_transformed: dict*, *symmetric_key: bytes | None = None*, *verbose: bool = False*) → dict

Decompress and decrypt the contents of a transformed dictionary.

This function takes a dictionary that has been transformed (e.g., compressed, encrypted), and restores its original contents by reversing the transformations. Depending on the transformation type, it may decrypt and/or decompress the data.

4.2.24 Parameters

d_transformed
[dict] The transformed dictionary containing the compressed/encrypted string, the transformations applied, and the json format used.

symmetric_key
[bytes, optional] A symmetric key. Typically generated using `from secrets import token_bytes; symmetric_key = token_bytes(32)`. If provided, the data will be decrypted using this key (default is None).

verbose
[bool, optional] If True, additional debug information will be printed (default is False).

4.2.25 Returns

dict
The original dictionary with its content restored.

4.2.26 Raises

Exception
If an invalid transformation type or JSON format is provided.

4.2.27 Example

```
d_transformed = {
        'transformations'    : 'encrypted_gzip_base64',
        'json_format'        : 'json',
        'transformed_string' :
↪'q30qPkK19Z3sENnfk77t4CnpzWKV+gdHLLSpNNgU3DjdmEbLcZWj+AjZyFmUquuUmh6obZmTh8k='
        'nonce_string'       : '7PpTvoc0Ksx8whRy',
}
d_untransformed = untransform_dict(
        d_transformed = d_transformed,
        symmetric_key = symmetric_key, # b'\x1a\xef&\x0bR\xe1\x95\xfa\x90\x10r\x93\
↪x1a\xaeN\xc2\xba\x80\xf1\x1a\x0fG\xf4(\x0e#\xd4\xaf^\x81q\xf4'
)
print(d_untransformed) # {'A': 1, 'B': 2, 'C': 3}
```

insightsolver.api_utilities.**generate_keys**()

This function generates RSA and ECDSA private and public keys. The generated keys:

- rsa_private_key
- ecdsa_private_key
- rsa_public_key_pem_bytes
- ecdsa_public_key_pem_bytes

4.2.28 Returns

tuple
A tuple containing four elements:

- rsa_private_key: The generated RSA private key.
- ecdsa_private_key: The generated ECDSA private key.
- rsa_public_key_pem_bytes: The RSA public key serialized in PEM format.
- ecdsa_public_key_pem_bytes: The ECDSA public key serialized in PEM format.

insightsolver.api_utilities.**generate_url_headers**(*computing_source: str*, *input_file_service_key: str | None = None*) → Tuple[str, Dict[str, Any] | None]

This function generates the url and the headers for the POST request.

4.2.29 Parameters

computing_source
[str] Where the server is.

input_file_service_key
[optional] The client's service key, needed if the server is remote. Default is *None*.

insightsolver.api_utilities.**compute_credits_from_df**(*df: DataFrame*, *columns_names_to_btypes: dict = {}*) → int

This function computes the number of credits consumed by a rule mining via the API. This number is based on the size of the DataFrame sent to the API.

Remark: The amount of credits debited is m*n where: - m is the number of rows of df (excluding the header). - n is the number of features to explore (i.e. the number of columns less the index, less the target variable, less the ignored features).

4.2.30 Parameters

df
[pd.DataFrame] Input DataFrame whose size is used to compute credits.

columns_names_to_btypes: dict
The dict that specifies how to handle the variables.

4.2.31 Returns

int
The computed number of credits consumed.

insightsolver.api_utilities.**request_cloud_credits_infos**(*computing_source: str*, *d_out_credits_infos: dict*, *input_file_service_key: str | None = None*, *user_email: str | None = None*, *timeout: int = 60*) → dict

Send a dict that specifies which infos about the credits are asked for.

4.2.32 Parameters

computing_source
[str] Where the server is.

d_out_credits_infos
[dict] A dictionary containing the infos about the credits that are asked for. The dictionary format is:

- `private_key_id`: private_key_id of the service_key.
- `user_email`: Email of the user.
- `do_compute_credits_available`: A boolean that specifies where the number of credits available is requested.
- `do_compute_df_credits_infos`: A boolean that specifies if a DataFrame containing all credits transactions is asked for.

input_file_service_key
[optional] The client's service key, needed if the server is remote. Default is *None*.

timeout
[int, optional] The timeout duration for the request, in seconds. Default is 60 seconds, as this operation is typically fast and does not involve computation.

insightsolver.api_utilities.**request_cloud_public_keys**(*computing_source: str*, *d_client_public_keys: dict*, *input_file_service_key: str | None = None*, *timeout: int = 60*) → dict

Send the client's public keys to the server and receive the server's public keys in response.

This function establishes a secure connection to the specified server (`computing_source`) and sends the client's public keys (`d_client_public_keys`). The server responds with its own set of public keys, which are returned in a dictionary format.

4.2.33 Parameters

computing_source
[str] Where the server is.

d_client_public_keys
[dict] A dictionary containing the client's public keys to be sent to the server. The dictionary format is:

- `alice_rsa_public_key_pem_base64`: Client's RSA public key, encoded in base64.
- `alice_ecdsa_public_key_pem_base64`: Client's ECDSA public key, encoded in base64.

input_file_service_key
[optional] The client's service key, needed if the server is remote. Default is *None*.

timeout
[int, optional] The timeout duration for the request, in seconds. Default is 60 seconds, as this operation is typically fast and does not involve computation.

4.2.34 Returns

dict
A dictionary containing the server's public keys and a unique session identifier. The dictionary format is as follows:

- `session_id`: A unique identifier for the session.
- `bob_rsa_public_key_pem_base64`: Server's RSA public key, encoded in base64.
- `bob_ecdsa_public_key_pem_base64`: Server's ECDSA public key, encoded in base64.

4.2.35 Example

```
# Client's public keys
d_client_public_keys = {
        'alice_rsa_public_key_pem_base64': '<base64-encoded RSA public key>',
        'alice_ecdsa_public_key_pem_base64': '<base64-encoded ECDSA public key>',
}

# Request server public keys
d_server_public_keys = request_cloud_public_keys(
        computing_source='https://server-address.com',
        d_client_public_keys=d_client_public_keys,
        input_file_service_key='client_service_key'
```

(continues on next page)

(continued from previous page)

```
)

# Access the session ID and server's public keys
session_id = d_server_public_keys['session_id']
bob_rsa_public_key = d_server_public_keys['bob_rsa_public_key_pem_base64']
bob_ecdsa_public_key = d_server_public_keys['bob_ecdsa_public_key_pem_base64']
```

4.2.36 Raises

Exception
If the request fails or the server does not return the expected keys.

insightsolver.api_utilities.**request_cloud_computation**(*computing_source: str*, *d_out_transformed: dict*, *input_file_service_key: str | None = None*, *timeout: int = 600*, *verbose: bool = False*) → dict

Send the transformed dict to the server for it to compute the rule mining.

4.2.37 Parameters

computing_source
[str] The computing source.

d_out_transformed
[dict] The transformed dict to send to the server.

input_file_service_key
[str, optional] The client's service key, needed if the server is remote. Default is *None*.

timeout
[int, optional] Timeout for the request, in seconds. Default is 600 seconds, as computation may take longer.

4.2.38 Returns

dict
The dict that contains the rule mining results.

insightsolver.api_utilities.**search_best_ruleset_from_API_dict**(*d_out_original: dict*, *input_file_service_key: str | None = None*, *user_email: str | None = None*, *computing_source: str = 'remote_cloud_function'*, *do_compress_data: bool = True*, *do_compute_memory_usage: bool = True*, *verbose: bool = False*) → dict

Search for the best ruleset where the computation is done from the server.

4.2.39 Parameters

d_out_original: dict
The original dict, pre-transformation, that contains the necessary data for the server to do rule mining.

input_file_service_key: str, optional
The service key of the client.

user_email: str, optional
Email of the user (only for use inside a Google Cloud Run container).

computing_source: str, optional
The computing source.

do_compress_data: bool, optional
If we want to compress the data to reduce transmission size.

do_compute_memory_usage: bool, optional
If we want to compute the memory usage.

verbose: bool, optional
Verbosity.

4.2.40 Returns

dict
The dict that contain the output of the rule mining from the server.

4.3 visualization module

- *Organization*: InsightSolver Solutions Inc.
- *Project Name*: InsightSolver
- *Module Name*: insightsolver
- *File Name*: visualization.py
- ***Authors*: Noé Aubin-Cadot <noe.aubin-cadot@insightsolver.com>,**
  Arthur Albo <arthur.albo@insightsolver.com>
- *Last Updated*: 2025-09-10
- *First Created*: 2025-04-24

4.3.1 Description

This file contains some visualization functions, some of which are integrated as a method of the InsightSolver class.

4.3.2 Functions provided

- show_all_mutual_information
- classify_variable_as_continuous_or_categorical
- compute_feature_label
- truncate_label
- show_feature_distributions_of_S_feature
- show_feature_distributions_of_S
- p_value_to_p_text
- generate_insightsolver_banner
- wrap_text_with_word_boundary

- show_feature_contributions_of_i
- show_all_feature_contributions
- show_feature_contributions_and_distributions_of_i
- show_all_feature_contributions_and_distributions

4.3.3 License

Exclusive Use License - see LICENSE for details.

---

insightsolver.visualization.**show_all_mutual_information**(*solver*, *n_samples: int | None = 1000*, *n_cols: int | None = 20*)

This function generates a bar plot of the mutual information between the features and the target variable.

4.3.4 Parameters

n_samples: int
An integer that specifies the number of data rows to use in the computation of the mutual information.

n_cols: int
An integer that specifies the maximum number of features to show

insightsolver.visualization.**classify_variable_as_continuous_or_categorical**(*s: Series*, *unique_ratio_threshold: float = 0.1*, *max_categories: int = 20*) → str

Classify a pandas Series as 'continuous' or 'categorical'.

Heuristic: - If dtype is object/string/bool → categorical - If all values are equal → categorical - If all values are integers:

- Few unique values (<= max_categories) → categorical
- Low unique ratio (<= unique_ratio_threshold) → categorical
- Otherwise → continuous

4.3.5 Parameters

s
[pd.Series] Input series.

unique_ratio_threshold
[float, optional] Threshold for ratio (#unique / #non-missing) to treat integers as categorical.

max_categories
[int, optional] Absolute cap for number of unique categories to treat as categorical.

4.3.6 Returns

str
"categorical" or "continuous"

insightsolver.visualization.**compute_feature_label**(*solver*, *feature_name: str*, *S: dict*) → [<class 'str'>, <class 'str'>]

This function computes the label of a feature in a rule S.

4.3.7 Parameters

solver: InsightSolver
The solver.

feature_name: str
The name of the feature.

S: dict
The rule S.

4.3.8 Returns

feature_label: str
The label of the feature.

feature_relationship: str
The relationship of the feature to the constraints.

insightsolver.visualization.**truncate_label**(*label*, *max_length=30*, *asterisk=False*)

This function truncates a string if it exceeds a specified length, adding an ellipsis.

4.3.9 Parameters

label: string
the feature rule's modalities.

max_length: int
the maximum number of character accepted.

asterisk: bool
whether we want an asterisk to appear after the truncation.

4.3.10 Returns

truncated_label: str
The truncated label.

insightsolver.visualization.**show_feature_distributions_of_S_feature**(*solver*, *df_filtered: DataFrame*, *S: dict*, *feature_name: str*, *missing_value: str = False*, *ax: str = None*, *language: str = 'en'*, *padding_y: int = 5*, *do_show_kde: bool = False*, *do_show_vertical_lines: bool = False*) → None

This function generates bar plots of the distributions of the points in the specified rule S for a given feature.

4.3.11 Parameters

solver
[InsightSolver] The solver object.

df_filtered: pd.DataFrame
The filtered data according to the rule S.

S
[dict] The rule S that we wish to visualize.

feature_name
[str] The name of the column

missing_value: bool
If we want to show the graph for the present values or the missing values.

ax: matplotlib.axes
Axes to be used if provided.

language: str
Language to be used.

padding_y: int
The padding used for the ylim.

do_show_kde: bool
Boolean to show the KDE of the continuous features.

do_show_vertical_lines: bool
If we want to show vertical lines.

insightsolver.visualization.**show_feature_distributions_of_S**(*solver*, *S: dict*, *language: str = 'en'*, *padding_y: int = 5*, *do_show_kde: bool = False*, *do_show_vertical_lines: bool = False*) → None

This function generates bar plots of the distributions of the points in the specified rule S.

4.3.12 Parameters

solver
[InsightSolver] The solver object.

S
[dict] The rule S that we wish to visualize.

language: str
Language to use.

padding_y: int
The padding used for the ylim.

do_show_kde: bool
Boolean to show the KDE of the continuous features.

do_show_vertical_lines: bool
If we want to show some vertical lines.

insightsolver.visualization.**p_value_to_p_text**(*p_value*, *precision_p_values: str*) → str

This function converts the p-value to a string.

4.3.13 Parameters

p_value: float or mpmath.mpf
The p-value to convert.

precision_p_values: str
The precision of the p-values.

4.3.14 Returns

p_text: str
The p_value formatted as a string.

insightsolver.visualization.**generate_insightsolver_banner**(*solver*, *i: int*, *loss: float | None = None*)

This function returns an image containing the parameters for p-value, purity, lift, coverage, size and loss value of a specified rule.

4.3.15 Parameters

solver: InsightSolver
The solver.

i: int
Index of the rule.

loss: float
Some loss to show in the banner.

4.3.16 Returns

image: Image
Image of the banner (with the values).

insightsolver.visualization.**wrap_text_with_word_boundary**(*text: str*, *max_line_length: int = 150*) → str

Wraps a text string into multiple lines by inserting line breaks around a target character width, while preserving word boundaries whenever possible.

- If the next word would cause the line to exceed *max_line_length*, a line break is inserted *before* that word.
- If a single word is longer than *max_line_length*, the word is split with a hyphen followed by a line break.

4.3.17 Parameters

text
[str] The input text to wrap.

max_line_length
[int, optional] The maximum allowed line length before wrapping occurs (default is 150).

4.3.18 Returns

str
The wrapped string, with line breaks (and occasional hyphenation) inserted at appropriate positions.

insightsolver.visualization.**show_feature_contributions_of_i**(*solver*, *i: int*, *a: float = 0.5*, *b: float = 1*, *fig_width: float = 12*, *language: str = 'en'*, *do_grid: bool = True*, *do_title: bool = False*, *do_banner: bool = True*, *bar_annotations: str = 'p_value_ratio'*, *loss: float | None = None*) → None

This function generates a horizontal bar plots of the feature constributions of a specified rule S.

4.3.19 Parameters

solver: InsightSolver
The fitted solver that contains the identified rules.

i: int
The index of the rule to show.

a: float
Height per bar.

b: float
Added height to the figure.

fig_width: float
Width of the figure

language: str
Language of the figure ('fr' or 'en').

do_grid: bool
If we want to show a vertical grid behind the horizontal bars.

do_title: bool
If we want to show a title.

do_banner: bool
If we want to show the banner.

bar_annotations: str
Type of values to show at the end of the bars (can be 'p_value_ratio', 'p_value_contribution' or None)

loss: float
If we want to show a loss.

insightsolver.visualization.**show_all_feature_contributions**(*solver*, *a: float = 0.5*, *b: float = 1*, *fig_width: float = 12*, *language: str = 'en'*, *do_grid: bool = True*, *do_title: bool = False*, *do_banner: bool = True*, *bar_annotations: str = 'p_value_ratio'*) → None

This function generates a horizontal bar plot of the feature contributions for each rule found in a solver.

4.3.20 Parameters

solver: InsightSolver
The fitted solver that contains the identified rules.

a: float
Height per bar.

b: float
Added height to the figure.

fig_width: float
Width of the figure

language: str
Language of the figure ('fr' or 'en').

do_grid: bool
If we want to show a vertical grid behind the horizontal bars.

do_title: bool
If we want to show a title.

do_banner: bool
If we want to show the banner.

bar_annotations: str
Type of values to show at the end of the bars (can be 'p_value_ratio', 'p_value_contribution' or None)

insightsolver.visualization.**show_feature_contributions_and_distributions_of_i**(*solver*, *i: int*, *language: str = 'en'*, *do_banner: bool = True*, *loss: float | None = None*, *bar_annotations: str = 'p_value_ratio'*) → None

This function returns a bar plot of the feature contributions and a distribution of the points in the rule i.

4.3.21 Parameters

solver: InsightSolver
The fitted solver that contains the identified rules.

i: int
The index of the rule to show.

language: str
Language to use.

do_banner: bool
If we want to show the banner.

loss: float
If we want to show a loss.

bar_annotations: str
Type of values to show at the end of the bars (can be 'p_value_ratio', 'p_value_contribution' or None)

insightsolver.visualization.**show_all_feature_contributions_and_distributions**(*solver*, *language: str = 'en'*, *do_banner: bool = True*, *bar_annotations: str = 'p_value_ratio'*) → None

This function generates the feature contributions and feature distributions for all rules found in a fitted solver.

4.3.22 Parameters

solver: InsightSolver
The fitted solver that contains the identified rules.

language: str
Language to use.

do_banner: bool
If we want to show the banner.

bar_annotations: str
Type of values to show at the end of the bars (can be 'p_value_ratio', 'p_value_contribution' or None)

CHAPTER

FIVE

CREDITS

The InsightSolver API charges **credits** based on the **size of the dataset** you submit.

5.1 Credit Calculation

The number of credits consumed per request is computed using the formula:

```
credits = ceil(m * n / 10000)
```

where `m` and `n` depend on the shape of your dataset:

- `m` is the number of rows (excluding the header),
- `n` is the number of feature columns (i.e. excluding the index column, the target column and other ignored columns),
- `ceil` is the mathematical ceiling function, rounding up to the next integer.

For example:

Rows (`m`)	Columns (`n`)	Computation	Credits Charged
1000	10	ceil(1000*10 / 10000)	1
10000	25	ceil(10000*25 / 10000)	25
20000	100	ceil(20000*100 / 10000)	200

5.2 Example Dataset

The Titanic training dataset from Kaggle contains **891 rows** and **9 feature columns** (excluding the index column `PassengerId` and the target column `Survived`). This results in the following credit usage:

```
ceil(891 * 9 / 10000) = 1 credit
```

You can think of **1 credit as roughly equivalent to "one Titanic"** in size.

5.3 Usage Tips

To reduce credit consumption:

- Remove unused or irrelevant columns before sending data,
- Filter your dataset (e.g., by time window or category),
- Use a representative sample when full data isn't necessary.

5.4 Getting Credits

If you need additional credits or want to upgrade your plan, please contact us at support@insightsolver.com.

CHAPTER

SIX

HELP

If you need assistance with the InsightSolver API client, please follow the resources below in the suggested order:

6.1 1. Technical Documentation

The technical documentation provides detailed guidance on installation, usage, and troubleshooting. Start here if you are encountering issues.

- **Installation Guide**: Refer to the *Installation Guide* for step-by-step instructions.
- **Quickstart Usage Example**: Check out the *Quickstart Example* for a simple and practical example to get started quickly. This example demonstrates how to use the InsightSolver API client and may inspire you to adapt it to your specific needs.
- **API Reference**: Explore the *API Reference* for detailed information on available methods and parameters.

6.2 2. Frequently Asked Questions (FAQ)

The FAQ section addresses common questions and issues.

- **How do I install the API client?**

  Please see the *Installation Guide*.

- **What should I do if I encounter a connection error?**

  Ensure your service key is valid, and check that your network permits outgoing connections.

6.3 3. GitHub Issues

For reporting bugs or exploring existing issues, visit our GitHub Issues page.

6.4 4. Contact Support

If the above resources do not resolve your issue, you can contact us directly at:

- **Email**: support@insightsolver.com

CHAPTER

SEVEN

LICENSE

```
Exclusive Use License:

This script is provided under an exclusive use license.
The holder of this license is authorized to use the script for internal
purposes only. Any modification, distribution, or sharing of the script or
parts thereof is strictly prohibited without prior written consent from the author.

This script is provided "as is", without warranty of any kind, express
or implied, including but not limited to the warranties of merchantability,
fitness for a particular purpose, or non-infringement.
```

PYTHON MODULE INDEX

i

INDEX

Symbols

C

D

E

F

G

H

I

M

P

R

S

T

U

V

W