InsightSolver API client

Release 0.1.0

Noé Aubin-Cadot

Nov 18, 2024

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 for advanced rule mining and data insights, powered by a centralized rule-mining engine. It helps organizations uncover hidden patterns, generate actionable insights, and make data-driven decisions.

1.2 Access Options

InsightSolver can be accessed through the following options:

- The API Client, designed for seamless integration with Python applications and data workflows.
- The **Web App** (not yet available), which will provide an intuitive and interactive interface for exploring rule-mining results, visualization, and analysis.

This documentation specifically covers the API Client. The 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

The API client requires a valid `.json` service key. This key authenticates your identity and ensures secure communication with the InsightSolver API server. If you do not yet have a service key, please contact support@insightsolver.com for assistance.

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 .
```

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

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.

3.1 Quick Start Example

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

```
# Import some data
import pandas as pd
df = pd.read_csv('kaggle_titanic_train.csv') # Dataset here: https://www.kaggle.com/
↪competitions/titanic/data

# 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', # Could be 'multiclass' (i.e. unordered) or
↪'continuous' (i.e. ordered)
                  'Name'     : 'ignore',
                  'Sex'      : 'binary',
                  'Age'      : 'continuous',
                  'SibSp'    : 'continuous', # Could be 'multiclass' (i.e. unordered) or
↪'continuous' (i.e. ordered)
                  'Parch'    : 'continuous', # Could be '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 lift

0 2e-67   19.1%    2.47
                                86.2%      Sex             female
                                13.8%   Pclass             [1, 2]
1 6e-13    6.3%    2.19
                                52.7%      Age       [0.42, 15.0]  exclude
                                47.3%    SibSp             [0, 2]
2 3e-20   12.2%    2.06
                                81.4%   Pclass             [1, 1]
                                11.3%     Fare  [7.925, 512.3292]
                                 7.3%      Age        [4.0, 42.0]  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 +147% compared to the population of the Titanic.
- `(i=1)` : **Children (which we know the age) with not too many siblings.** This group covers 6.3% of the passengers and has a survival gain of +117% compared to the population of the Titanic.
- `(i=2)` : **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 +106% 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
```

(continues on next page)

(continued from previous page)

```
# 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 lift

0 7e-55   42.6%    1.46
                                78.8%      Sex             male
                                10.9%      Fare     [0.0, 26.0]
                                10.3%     Parch          [0, 0]
1 1e-12   12.9%    1.45
                                63.6%      Fare  [7.8875, 15.1]
                                36.4%       Age    [19.0, 26.0]  include
"""
```

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 +46% compared to the population of the Titanic.
- `(i=1)` : **Poor young adults (include missing ages).** This group covers 12.9% of the passengers and has a non-survival gain of +45% 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.

CHAPTER

FOUR

MODULES

4.1 insightsolver module

- *Project Name*: InsightSolver
- *Module Name*: insightsolver
- *Author*: Noé Aubin-Cadot
- *Organization*: InsightSolver
- *Email*: noe.aubin-cadot@insightsolver.com
- *Last Updated*: 2024-11-18
- *First Created*: 2024-09-09

4.1.1 Description

This module contains the `InsightSolver` class. It is meant to 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.`**`validate_class_integrity`**(*verbose: bool*, *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*) → None

This function aims to validate the integrity of the InsightSolver class.

`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 `class insightsolver.InsightSolver`(*df: DataFrame | None = 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*, *verbose: bool = False*)

Bases: `object`

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.

4.1.4 Attributes

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

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.

columns_types: dict
Types of the columns.

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

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.

4.1.5 Methods

validate_class_integrity: None
Validates the integrity of the class.

ingest_dict: None
Ingests a Python dict.

ingest_json_string: None
Ingests a JSON string.

fit: None
Fits the solver.

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

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

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).

4.1.6 Example

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

```python
# 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__`(*df: DataFrame | None = 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*, *verbose: bool = False*)

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.

target_threshold: float
Threshold used to convert a continuous target variable to a binary target variable.

M: int
Number of points, i.e. number of rows of df.

M0: int
Number of points carrying the value 0.

M1: int
Number of points carrying the value 1.

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

Returns

solver: InsightSolver
An instance of the class InsightSolver.

Example

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

```python
# 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
)
```

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.

fit(*verbose: bool = False*, *computing_source: str = 'auto'*, *service_key: str | None = None*, *api_source: str = 'auto'*, *do_compress_data: bool = False*) → None

This method aims to fit the solver.

ruleset_count() → int

This method returns the number of rules held in the InsightSolver.

i_to_rule(*i: int*) → dict

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_print(*i: int*, *indentation: str = ''*, *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*) → None

This method prints the content of the rule i in the InsightSolver.

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

This method gives the range of i in the InsightSolver. 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_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*, *mode: str = 'full'*) → None

This method prints the content of the InsightSolver.

print_light(*print_format: str = 'list'*) → None

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

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

print_dense() → None

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

insightsolver.**search_best_ruleset_from_API_public**(*df: DataFrame*, *computing_source: str = 'auto'*, *input_file_service_key: str | None = None*, *target_name: str | int | None = None*, *target_goal: str | Real | uint8 | None = None*, *columns_names_to_btypes: Dict | None = {}*, *threshold_M_max: int | None = None*, *specified_constraints: Dict | None = {}*, *top_N_rules: int | None = 10*, *verbose: bool = False*, *api_source: str = 'auto'*, *do_compress_data: bool = True*, *do_compute_memory_usage: bool = True*) → dict

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

4.1.7 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).

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.

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.

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.

4.1.8 Returns

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

4.2 api_utilities module

- *Project Name*: InsightSolver
- *Module Name*: api_utilities
- *Author*: Noé Aubin-Cadot
- *Organization*: InsightSolver
- *Email*: noe.aubin-cadot@insightsolver.com
- *Last Updated*: 2024-10-18
- *First Created*: 2024-09-16

4.2.1 Description

This module 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 module 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

- `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.
- `transform_dict`: Convert a dictionary for easier client-server communication.
- `untransform_dict`: Reverse the dictionary transformation to restore the original data format.
- `request_cloud_public_keys`: Request the server for public keys.
- `request_cloud_computation`: Request the server for computation.
- `generate_keys`: Generate RSA and ECDSA private and public keys.
- `search_best_ruleset_from_API_dict`: Make the API call.

4.2.3 License

Exclusive Use License - see LICENSE for details.

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.

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.

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'
```

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'
```

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
```

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'
```

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 'jsonpickle' (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',
# }
```

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}
```

`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.

4.2.28 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.

4.2.29 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.

4.2.30 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'
)

# 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']
```

4.2.31 Raises

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

`api_utilities.`**`request_cloud_computation`**(*computing_source: str*, *d_out_transformed: dict*, *input_file_service_key: str | None = None*, *timeout: int = 600*) → Response

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

4.2.32 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.

4.2.33 Returns

requests.Response
The response from the server after attempting the computation request.

4.2.34 Raises

requests.exceptions.RequestException
If the request to the server fails due to a network issue or server error.

`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`

4.2.35 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.

`api_utilities.search_best_ruleset_from_API_dict`(*d_out_original: dict*, *input_file_service_key: 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.

4.2.36 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.

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.

4.2.37 Returns

dict

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

CHAPTER FIVE

HELP

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

5.1 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.

5.2 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.

5.3 3. GitHub Issues

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

5.4 4. Contact Support

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

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

CHAPTER SIX

LICENSE

PYTHON MODULE INDEX

INDEX