Mangrove-Surface-Python-SDK Documentation

Release 0+untagged.1.g5704d0a

Mangrove

Apr 13, 2018

Contents

CHAPTER 1

Documentation

A complete documentation is available there: documentation

CHAPTER 2

Installation

Install the python SDK package:

```
pip install mangrove-surface
```

or using the git repository:

```
git clone https://github.com/mangroveai/mangrove-surface-python-sdk
python setup.py install
```

CHAPTER 3

(*optional*) Setup your environment variables

You can used the SDK with an explicit configuration of url instance and token or you can provide them as environment variables: `SURFACE_URL` and `SURFACE_TOKEN`.

For example on unix-like system:

```
$ export SURFACE_URL=http://your_mangrove.ai_url/api
$ export SURFACE_TOKEN='eyJ0eXAiOiJKV1QiLCJhbGciOiJ...'
```

Warning: Mangrove Surface URL have to end with `/api`.

Note: The explicit configuration overrides the implicit one.

Note: Token can be provided by the administrator using GUI (see documentation) or using the SDK (see `mangrove.Surface._Admin.create_token()`:)

CHAPTER 4

Test your installation

You can run tests with the following python lines:

- Test if it is properly installed:

```
>>> import mangrove_surface
>>> mangrove_surface.__version__
'2.0.0'
```

The python SDK is properly connected to your Mangrove Surface:

```
>>> from mangrove_surface import SurfaceClient
>>> # if environment variables are setup
>>> client = SurfaceClient()
>>> # otherwise
>>> # client = SurfaceClient(url="...", token="...")
>>> client.admin.versions()
[
    {
        'name': 'atlas',
        'version': '1.0.0'
    }, {
        'name': 'license_authority',
        'version': u'1.5.0'
    }, {
        'name': 'dmgr',
        'version': '1.0.0'
    }, {
        'name': 'modeler',
        'version': '1.0.0'
    }, {
        'name': 'exporter',
        'version': '1.0.0'
    }, {
        'name': 'mangrove-surface-sdk',
        'version': '1.0.0'
```

```
    }
]
```

It is well configured! Congratulation!

Let's begin with *Surface*.

CHAPTER 5

Support

Please refer to *Logger* section if you have any unexpected behavior using the SDK.

CHAPTER 6

Contents

6.1 Surface

class mangrove_surface.**SurfaceClient**(*url=None*, *token=None*, *username=None*, *password=None*)

Instanciate Mangrove.ai python SDK with instance url and identity access

Parameters

- **url** – (*optional*) url of the Mangrove.ai instance (by default environment variable `SURFACE_URL` is used)
- **token** – (*optional*) access token used to secure connection (by default, if `username`/`password` are given then those are used to generate an access token; otherwise environment variable `SURFACE_TOKEN` is used)
- **username** – (*optional*) username used (with `password`) to sign in (by default, a token is expected)
- **password** – (*optional*) password used to (with `username`) sign in (by default, a token is expected)

Note: Surface URL or access token can be explicitly provided as parameters or implicitly using environment variables (`SURFACE_URL` and `SURFACE_TOKEN`)

Raises

- **IOError** – if the endpoint doesn't answer correctly
- **AttributeError** – if url, username-password or token are not provided

Load Surface python SDK:

```
>>> from mangrove_surface import SurfaceClient
```

Instanciate with url and token are environment variables:

```
>>> client = SurfaceClient()
```

Or with url as environment variable and an explicit token:

```
>>> client = SurfaceClient(token='eyJ0eXAiOiJKV1QiLCJhbGciOiJ...')
```

Or with explicit url and token:

```
>>> client = SurfaceClient(
...     url='http://my.surface/api',
...     token='eyJ0eXAiOiJKV1QiLCJhbGciOiJ...'
... )
```

create_project(*name*, *schema*, *description=''*, *schema_test=None*, *tags=[]*, *default_classifier=True*, *force=False*)

Create a new project

Parameters

- **name** – project name
- **description** – (*optional*) project description
- **schema** – (*optional*) a data schema which contains train data sets and relations, like:

```
{
    "tags": ["dataset", "tag"],
    "datasets": [
        {
            "name": "Dataset Name",
            "filepath": "/path/to/dataset.csv",
            "tags": ["optional", "tags"],
            "central": True | False,
            "keys: ["index"], # optional if there is only
                              # one dataset
            "separator": ",", # could be `|`, `,`, `;` or `     `
        }, ...
    ],
    "outcome": "FIELD TARGETED",
    "outcome_modality": "main value targeted"
}
```

filepath is an absolute filepath or it could be a S3 uri, like:

```
{
    "type": "s3",
    "bucket": "mang-model-producer-samples",
    "key": "CAR_INSURANCE/CHAT_SESSION_CONTENT_TRAIN.csv"
}
```

- **tags** – (*optional*) list of project tags
- **schema_test** – (*optional*) a data schema which contains test data sets and relations, like **schema**
- **default_classifier** – (*default*: `True`) indicates if a default classifier it provided at the project creation

- **force** – (*default*: `False`) indicates if a project with the same name exists then it is replaced or not

project (*name*)
Return project named `name`

Parameters **name** – the name of the wished project

projects ()
List all projects

6.1.1 Surface admin

class `SurfaceClient.`**_Admin**
Administration methods:

create_token (*token_name*, *expiration_date*)
Create a new token

Parameters

- **token_name** –
- **date** (*expiration*) – should be a *datetime.datetime* object

```
>>> from datetime import datetime
>>> expire = datetime(2020, 1, 1)
>>> mang.admin.create_token('token_jbp', expire)
{
    'created_at': '2018-03-29T09:45:06.067Z',
    'expires_at': 1577833200,
    'id': '40a6a8b6-65b1-465e-b6c7-6c3021c30952',
    'name': 'token_jbp',
    'token': 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJzdWIiOiI4MjI0OTg4NC05M2UwLTQwN2MtYmQ3OS02NTllMWE4MzQ2NTUiLCJzY3AiOiJ1c2VyIiwiaWF0IjoxNT
↪looosUk2TuXOVXREmAvPoVnOx0kLaSLOT4TlOMK_yTA',
    'updated_at': '2018-03-29T09:45:06.067Z'
}
```

create_user (*username*, *password*)
Create a user

Parameters

- **username** – new username
- **password** – it password

delete_users (**usernames*)
Delete users

Parameters ***usernames** – arbitrary number of usernames

```
>>> mang.admin.delete_users("Alice", "Bob", "Oscar")
```

license_information (*request_code=False*)
Retrieve license information

Parameters **request_code** – (default *False*) if *True* it adds the request code to obtain a new license (only for *byol*)

```
>>> mang.admin.license_information()
[{
    'expires_at': '2018-08-29 00:00:00 UTC',
    'service_level': 'full',
    'system_information': None,
    'updated_at': '2018-03-29 09:49:34 UTC'
}]

>>> mang.admin.license_information(request_code=True)
[{
    'expires_at': '2018-08-29 00:00:00 UTC',
    'service_level': 'full',
    'system_information': {
        'request_code': 'kzho-isiA-8dwy-gZyq-gFNb-EC5l-od7s-JBai-RcaF-2hMb-
↪cirj-52rS-P4M3-2sRg-fuZa-/S5W-FkRn-RDSo-srVa-0xlX-q7KO-NkMY-380Y-dmW4-JfHG-
↪Q01x-so3N-NhdO-MoMj-Xw+B-bUdb-Q7VI-K+Hy-gSMF-kVpD-kCkO-v3Ay-a2/f-To9v-Lnxw-
↪3EdE-FEPa-yVMI-x/U4-EsUV-T1eq-LQsM-C88E-yPOS-RtVp-vDtD-zwEn-PAS7-/pSl-MGJ+-
↪jnUq-JllG-uxO+-seDZ-6X+v-rXBI-zHUx-go3p-K2ZO'
    },
    'updated_at': '2018-03-29 09:49:34 UTC'
}]
```

Raises **`MangroveError`** – if the license has expired

```
{
    "status": 402,
    "status_text": "Payment Required",
    "errors": [
        {
            "code": "LICENSE-402-002",
            "type": "license",
            "metadata": {
                "reason": "License not found",
                "request_code": "P6jC-ns9H-X5ph-KZVg-finF-ttgv-2Jtl-ygPn-Ie/z-
↪VUBc-hYYz-GeLT-yTb4-UrkS-tr/s-w3uf-lzlG-Av34-3fnx-0gl2-8SnL-jaQt-0BJ+-bhKU-
↪zgWl-tu6j-kM4r-i84u-s2Qo-SR4P-JyEH-AIRh-psnw-d0zd-R+Nf-zrl+-8hWy-l3Db-HeD9-
↪6DY7-gwlO-1Zjp-Opvu-pp5I-mxWQ-qDtS-WWTo-xjlK-hE9q-sukL-YEeK-OPWz-aaJl-0ZzB-
↪0sN2-6Gqz-soPd-lEXR-USDl-vDzJ-JltE-RWX+-HfZs-2Njd",
                "type": "private",
                "autofetch": False
            }
        }
    ]
}
```

`new_license`(*license_code*)

Update the licence

Parameters **`license_code`** – string provided by the Mangrove team

```
>>> mang.admin.new_license('fxbE-D9pK-h0t7-x+5r-B3G7-bY+x-AG6x-dzYW-tccq-HAtl-
↪Bkzb-JPVw-jsFd-zcvN-Nr15-vkIZ-ZK4J-yafW-niK9-RbaV-FGS9-oks5-zsLJ-yweZ-fg3K-
↪SAeT-jDWP-pDnj-bJ8P-ZjKh-Tskp-I/1A-Ymow-fV6s-fvXK-dliu-cHCY-1Orf-pBY0-VDgm-
↪IBaP-3Dz3-CiYS-4MVR-hQsO-KNfu-WK7d-7/6w-CTNW-A0HA-9rnB-im62-evcd-j7HS-KnnL-
↪K/aD-UNlU-5vO5-K9g=')
{
    'expires_at': '2018-03-30 00:00:00 UTC',
```

```
    'service_level': 'full',
    'system_information': None,
    'updated_at': '2018-03-31 11:47:54 UTC'
}
```

Raises **HTTPError** – if you provide a wrong license

(Please use GUI)

tokens()

List all tokens

```
>>> mang.admin.tokens()
[{
    'created_at': '2018-03-29T09:45:06.067Z',
    'expires_at': 1577833200,
    'id': '40a6a8b6-65b1-465e-b6c7-6c3021c30952',
    'name': 'token_jbp',
    'token': 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJzdWIiOiI4MjI0OTg4NC05M2UwLTQwN2MtYmQ3OS02NTllMWE4MzQ2NTUiLCJzY3AiOiJ1c2VyIiwiaWF0IjoxNT
↪looosUk2TuXOVXREmAvPoVnOx0kLaSLOT4TlOMK_yTA',
    'updated_at': '2018-03-29T09:45:06.067Z'
}]
```

users()

Retrieve all users

```
>>> mang.admin.users()
[admin(admin), Toto, Gillou]
```

versions()

Version information

```
>>> mang.admin.versions()
[
    {
        'name': 'atlas',
        'version': '0.0.1-alpha.1'
    }, {
        'name': 'license_authority',
        'version': u'1.3.2-rc1'
    }, {
        'name': 'dmgr',
        'version': '0.0.1'
    }, {
        'name': 'modeler',
        'version': '0.0.1-alpha.1'
    }, {
        'name': 'exporter',
        'version': '0.0.1'
    }, {
        'name': 'mang_sdk',
        'version': '1.2.1-30-ge792bea-dirty'
    }
]
```

6.2 Project

class mangrove_surface.wrapper.project.**ProjectWrapper**

Project resource

classifier(*name*)

Return classifier named `name`

Parameters **name** – project name

Raises **ClassifierDoesNotExist** – if there is no classifier named `name`

classifiers()

List all classifiers

```
>>> pj.classifiers()
[
    Project_2018-03-20T15:39:18.120Z,
    Project_2018-03-20T15:40:02.880Z,
    Project_2018-03-20T15:40:45.242Z,
    MyClassifier
]
```

collection(*name*)

Return the collection named `name`

Parameters **name** – collection name

collections()

List all collections

create_collection()

Create a new collection

A collection stores similar schemas of data sets.

Warning: Expert method: it should be only use to store new data set schemas

default_feature_set()

Return the default feature set

(see: `mangrove.wrapper.feature_set`)

description()

Project description

schemas(*type=None*)

List all schemas

Parameters **type** – (*optional*) `type` could be `train`, `test` or `export`. It is to filter schemas of type `type`. By default all schemas are listed.

tags()

Return the project tags

update_description(*new_description*)

Update the project description

Parameters **new_description** – the new project description

update_name (*new_name*)
Update the project name

Parameters **new_name** – the new project name

update_tags (*new_tags=[]*)
Update the project tags

Parameters **new_tags** – the list of new tags

6.3 Classifier

class mangrove_surface.wrapper.classifier.**ClassifierWrapper**
Classifier resource

A classifier provides

- the list relevants features (including level, weight, discretization attributes)
- the assessments over each train/test schemas
- method to export scores over
- method to improve classifier

add_schema (*type_schm*, *schema*, *name=None*)
Upload a new schema of datasets

Parameters

- **type_schm** – `train`, `test` or `export`
- **schema** – a python dictionary recording datasets like this

```
{
    "tags": ["dataset", "tag"],
    "datasets": [
        {
            "name": "Dataset Name",
            "filepath": "/path/to/dataset.csv",
            "tags": ["optional", "tags"],
            "central": True | False,
            "keys: ["index"], # optional if there is only
                              # one dataset
            "separator": ",", # could be `|`, `,`, `;` or ` `
        }, ...
    ]
}
```

add_schema_and_export (*schema*, *name=None*, *modalities=[]*, *bin_format='label'*, *raw_variables=[]*, *binned_variables=[]*, *predicted_modality=False*)
Upload a new schema and export it

Parameters

- **schema** – a python dictionary of datasets (see *add_schema()*:)
- **name** – (*optional*) the schema name
- **modalities** – (*optional*) the modalities scored. If no modality is provided then scores are not provided (only variables)

- **raw_variables** – the list of variables to export as raw value
- **binned_variables** – the list of variables to export as binned value
- **bin_format** – (*default*: `label`) select how to express the binned variables. `label` (*default*) to express value as its intervals or groups, or `id` to express value as a concise value
- **predicted_modality** – provided a column with the predicted value if `predicted_modality==True` (*default* `predicted_modality==False`)

compatible_schemas(*test=True*, *export=True*)
List compatible schemas (with there type)

compute_assessments(*schm_name*, *outcome_modality=None*)
Compute assessment over schema named `schm_name` (focus on modality `outcome_modality`)

Parameters

- **schm_name** – name of the schema used to compute assessments
- **outcome_modality** – the modality used to compute assessments (by *default* assessments is computed over the main modality)

compute_export(*schm_name*, *export_name=None*, *modalities=[]*, *bin_format='label'*, *raw_variables=[]*, *binned_variables=[]*, *predicted_modality=False*)
Compute a new export

Parameters

- **schm_name** – the dataset schema which is exported
- **export_name** – name of the export
- **modalities** – (*optional*) the modalities scored. If no modality is provided then scores are not provided (only variables)
- **raw_variables** – the list of variables to export as raw value
- **binned_variables** – the list of variables to export as binned value
- **bin_format** – (*default*: `label`) select how to express the binned variables. `label` (*default*) to express value as its intervals or groups, or `id` to express value as a concise value
- **predicted_modality** – provided a column with the predicted value if `predicted_modality==True` (*default* `predicted_modality==False`)

discretization_attribute(**args*, ***kwargs*)
Return the discretization attribute of the contributive feature `name`

Parameters **name** – feature name

```
>>> classifier.discretization_attribute("Car_Type")
[
    {
        'coverage': 0.0248497,
        'frequency': 529,
        'target_distribution': {
            '0': 0.837429,
            '1': 0.162571
        },
        'value_list': ['Full-size luxury car']
    },
```

```
    ...
]
```

download(**args*, ***kwargs*)
Download the classifier

Parameters **filepath** – the filepath where store the classifier

exports()
List all exports

feature(**args*, ***kwargs*)
Information about feature `name`

It returns level, weight, discretization attributes.

Parameters **name** – feature name

```
>>> classifier.feature('Car_Type')
{
    'level': 0.103459,
     'maximum_a_posteriori': True,
     'name': 'Car_Type',
     'nb_parts': 4,
     'parts': [
         {
            'coverage': 0.0248497,
            'frequency': 529,
            'target_distribution': {
                '0': 0.837429,
                '1': 0.162571
            },
            'value_list': ['Full-size luxury car']
        },
        ...
    ],
    'weight': 0.832425
}
```

feature_set(**args*, ***kwargs*)
Return the underlying feature set

Note: This feature set could be used to change type, unused some features

features(**args*, ***kwargs*)
List all the features used by the current classifier

```
>>> classifier.features()
[
    {
        'level': 0.103459,
         'maximum_a_posteriori': True,
         'name': 'Car_Type',
         'nb_parts': 4,
         'parts': [
             {
                'coverage': 0.0248497,
                'frequency': 529,
```

```
                'target_distribution': {
                    '0': 0.837429,
                    '1': 0.162571
                },
                'value_list': ['Full-size luxury car']
            },
            ...
        ],
        'weight': 0.832425
    },
    ...
]
```

improve(*name=None*, *tags=[]*, *nb_aggregates=None*, *maximum_features=None*)
Create a new classifier

Parameters

- **name** – (*optional*) classifier name
- **tags** – (*optional*) list of project tag
- **nb_aggregates** – (*optional*) number of aggregates generated for the new classifier
- **maximum_features** – (*optional*) maximal number of features used by the new classifier

Raises **MangroveError** – if the number of requested aggregates is provided and it is smaller than `.nb_aggregates()`

level(**args*, ***kwargs*)
Return the level of the feature named `name`

Parameters **name** – feature name

The level indicates the correlation between the feature and the outcome

nb_aggregates()
Return the number of aggregates

outcome()
Outcome field predicted by the current classifier

set_unused(**args*, ***kwargs*)
Set feature `name` unused

Parameters **name** – feature name

update_name(*new_name*)
Update the classifier name

Parameters **new_name** – new classifier name

weight(**args*, ***kwargs*)
Return the weight of the feature named `name`

Parameters **name** – feature name

The weight indicates how the feature discriminates more than others relevant features (with level > 0)

6.3.1 Assessment

class mangrove_surface.wrapper.classifier_evaluation_report.**ClassifierEvaluationReportWrapp**
Classifier Evaluation Report resource

ACC()
Accuracy

Note:

This method has some alias:

- ACC

AUC(**args*, ***kwargs*)
Area under curve

DOR()
Diagnostic odds ratio

Note:

This method has some alias:

- DOR

F1_score(*outcome_modality=None*)
F1 score

Parameters **outcome_modality** – (*optional*) the modality

FDR(*outcome_modality*)
False discovery rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- FDR

FNR(*outcome_modality=None*)
False negative rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- FNR
- miss_rate

FOR(*outcome_modality*)
False omission rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- FOR

FPR(*outcome_modality=None*)
False positive rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- FPR
- fall_out

LRm()
Negative Likehood ratio

Note:

This method has some alias:

- LRp

LRp()
Positive Likehood ratio

Note:

This method has some alias:

- LRp

NPV(*outcome_modality*)
Negative predictive value

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- NPV

PPV(*outcome_modality=None*)
Precision

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- `positive_predictive_value`

SPC(*outcome_modality=None*)
True negative rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- `TNR`
- `specificity`
- `SPC`

TNR(*outcome_modality=None*)
True negative rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- `TNR`
- `specificity`
- `SPC`

TPR(*outcome_modality=None*)
True positive rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- `recall`
- `TPR`
- `sensitivity`
- `probability_of_detection`

accuracy()
Accuracy

Note:

This method has some alias:

- ACC

---

area_under_curve(**args*, ***kwargs*)
Area under curve

auc(**args*, ***kwargs*)
Area under curve

confusion_matrix(**args*, ***kwargs*)
Confusion matrix

::

```
>>> ass.confusion_matrix()
{
    'matrix': [
        [13376, 1393],
        [  683, 4084]
    ],
    'modalities': ['N', 'Y']
}
```

diagnostic_odds_ratio()
Diagnostic odds ratio

Note:

This method has some alias:

- DOR

---

fall_out(*outcome_modality=None*)
False positive rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- FPR
- fall_out

---

false_discovery_rate(*outcome_modality*)
False discovery rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- FDR

---

false_negative(*outcome_modality*)
Number of false negative errors of the `outcome_modality`

False negative = incorrectly rejected

Parameters **outcome_modality** – (*optional*) compute the number of incorrect rejection of the modality

false_negative_rate(*outcome_modality=None*)
False negative rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- `FNR`
- `miss_rate`

---

false_omission_rate(*outcome_modality*)
False omission rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- `FOR`

---

false_positive(*outcome_modality=None*)
Number of incorrect predictions

False positive = incorrectly identified

Parameters **outcome_modality** – (*optional*) compute the number of incorrect prediction associated to this modality

Raises **KeyError** – if the `outcome_modality` does not exist

```
>>> ass.false_positive()
2076

>>> ass.false_positive('Y')
4084
```

false_positive_rate(*outcome_modality=None*)
False positive rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- `FPR`
- `fall_out`

---

gini()
Gini coefficient

instances(*outcome_modality=None*)
Number of instances evaluated

lift_curve(**args*, ***kwargs*)
Lift curve over the schema

Parameters **using** – is `classifier` or `optimal`; by default the lift curve associated to the classifier.

miss_rate(*outcome_modality=None*)
False negative rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- `FNR`
- `miss_rate`

---

negative_likehood_ratio()
Negative Likehood ratio

Note:

This method has some alias:

- `LRp`

---

negative_predictive_value(*outcome_modality*)
Negative predictive value

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- `NPV`

---

positive_likehood_ratio()
Positive Likehood ratio

Note:

This method has some alias:

- `LRp`

---

positive_predictive_value(*outcome_modality=None*)
Precision

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- positive_predictive_value

---

precision(*outcome_modality=None*)
Precision

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- positive_predictive_value

---

prevalence()
Prevalence

probability_of_detection(*outcome_modality=None*)
True positive rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- recall
- TPR
- sensitivity
- probability_of_detection

---

recall(*outcome_modality=None*)
True positive rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- recall
- TPR
- sensitivity
- probability_of_detection

---

sensitivity(*outcome_modality=None*)
True positive rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- recall
- TPR
- sensitivity
- probability_of_detection

---

specificity(*outcome_modality=None*)
True negative rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- TNR
- specificity
- SPC

---

target_rate(*outcome_modality*)
Target rate of the modality outcome_modality

Parameters **outcome_modality** – a modality

true_negative(*outcome_modality*)
Number of true negative errors of the outcome_modality

True negative = correctly rejected

Parameters **outcome_modality** – (*optional*) compute the number of correct rejection of the modality

true_negative_rate(*outcome_modality=None*)
True negative rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- TNR
- specificity
- SPC

---

true_positive(*outcome_modality=None*)
Number of correct predictions

True positive = correctly identified

Parameters **outcome_modality** – (*optional*) compute the number of correct prediction associated to this modality

Raises **KeyError** – if the `outcome_modality` does not exist

```
>>> ass.true_positive()
17460

>>> ass.true_positive('Y')
4084
```

true_positive_rate(*outcome_modality=None*)
True positive rate

Parameters **outcome_modality** – (*optional*) the modality

Note:

This method has some alias:

- `recall`
- `TPR`
- `sensitivity`
- `probability_of_detection`

6.4 Export

class mangrove_surface.wrapper.export.**ExportWrapper**

binned_variables()
List the binned variables

classifier()
Return the classifier used

download(**args*, ***kwargs*)
Download the export

Parameters **filepath** – the filepath where store the classifier

instances_submitted(**args*, ***kwargs*)
Number of instances submitted to export

push_s3(*bucket*, *key*)
Push the current export to S3

Parameters

- **bucket** – the S3 bucket
- **key** – the S3 key

raw_variables()
List the raw variables

6.5 Feature set

class mangrove_surface.wrapper.feature_set.**FeatureSetWrapper**(*feature_set_resource*, *collection*)

Feature set resource

A feature set is a set of frames (one for each data set). A frame contains variables and its metadata (type, use or not).

It is used to customize data, generate aggregates and to train classifiers.

central(**args*, ***kwargs*)
Return the central frame

The central frame is the one used to train classifiers.

clone(*new_name=None*, *tags=None*)
Clone the current feature set.

fit_classifier(*name=None*, *tags=[]*, *nb_aggregates=None*, *maximum_features=None*)
Fit a new classifier

Parameters

- **name** – (*optional*) classifier name (by default the name will be the project name concatenated with the current time
- **tags** – the classifier tags
- **nb_aggregates** – used to generates `nb_aggregates` aggregates on the central frame used to train the classifier
- **maximum_features** – used to allow at most `maximum_features` features in the new classifier

frame(**args*, ***kwargs*)
Return the frame named `name`

Parameters **name** – data (set) frame name

frames(**args*, ***kwargs*)
List all frames

generate_aggregates(**args*, ***kwargs*)
Generate a new feature set with `n` aggregates

Parameters **n** – number of aggregates requested (a non-negative integer)

is_modified(**args*, ***kwargs*)
Indicates if the current feature set has been modified

save(**args*, ***kwargs*)
Save all the modifications (change variables type, set unused, *etc.*)

> **Warning:** If `clone = False` the method overrides the current feature set resource

Raises **Exception** – if `clone = False` and the current feature set is the default one.

6.5.1 Frame

class FeatureSetWrapper.**_Frame**(*dataset*, *change_type*, *fs*)

features(*filt=<function <lambda>>*, *id=False*)

List features of the current frame

```
>>> fs.features()
[
    {
        'name': 'Flag_Prospect',
        'type': 'categorical',
        'use': True
    },
    {
        'name': 'LABEL',
        'type': 'continuous',
        'use': True
    },
    ...
]
```

Parameters **filt** – (*optional*) a function that can be used to filter features

```
>>> fs.features(filter=lambda feat: fs.is_categorical(feat))
[
    {
        'name': 'Flag_Prospect',
        'type': 'categorical',
        'use': True
    },
    ...
]
```

or:

```
>>> fs.features(filter=lambda feat: feat["name"].startswith("Foo"))
[
    {
        'name': 'FooBar',
        'type': 'categorical',
        'use': True
    },
    {
        'name': 'FooFoo',
        'type': 'continuous',
        'use': False
    },
    ...
]
```

is_categorical(*variable*)

Indicates if the feature `variable` is categorical or not

:param variable:: feature name

is_central()
Indicates if the frame is central

is_change_type_allowed()
Indicate if the frame allows to change feature type

It is forbidden to change type of peripheral frame features if there is aggregates in the central frame

is_continuous(*variable*)
Indicates if the feature `variable` is continuous or not

:param variable:: feature name

is_modified()
Indicates if the frame has been modified

is_used(*variable*)
Return if the feature is used or not

modalities(*name*)
List modalities of the feature `name`

Parameters **name** – feature name

set_categorical(*variable*)
Change the type of the feature `variable` to categorical

Parameters **variable** – the feature name

Raises **MangroveChangeForbidden** – if change type is not allowed

set_continuous(*variable*)
Change the type of the feature `variable` to continuous

Parameters **variable** – the feature name

Raises **MangroveChangeForbidden** – if change type is not allowed

set_unused(*variable*)
Set unused the feature `variable`

Parameters **variable** – the feature name

set_used(*variable*)
Set used the feature `variable`

Parameters **variable** – the feature name

type(*variable*)
Return the type of the feature `variable` The type could be `categorical` or `continuous` (other types can be provided like `timestamps` but they are not managed)

Parameters **variable** – the feature

6.6 Collection

class mangrove_surface.wrapper.collection.**CollectionWrapper**
Collection

A collection is a set of dataset schemas which are similars.

create_schema(*type_schm*, *schema*, *name=None*, *check=True*)
Create a new schema into the current collection

Parameters

- **type_schm** – `train`, `test` or `export`
- **schema** – a python dictionary recording datasets like this

```
{
    "tags": ["dataset", "tag"],
    "datasets": [
        {
            "name": "Dataset Name",
            "filepath": "/path/to/dataset.csv",
            "tags": ["optional", "tags"],
            "central": True | False,
            "keys: ["index"], # optional if there is only
                              # one dataset
            "separator": ",", # could be `|`, `,`, `;` or ` `
        }, ...
    ]
}
```

schema (*name*)
Return schema named `name`

Parameters **name** – the name of the requested schema

schemas()
List all schemas of the current collection

6.7 Logger

This logger is configured to log SDK behaviors.

Each api request and answer are logged at level `DEBUG` and each api resource created is logged at level `INFO`. Warnings and errors are logged at the expected level: `WARNING` and `ERROR`.

If you meet some unexpected behavior or bugs then please use the following lines to store in a file the behavior:

```
>>> import os.path as path
>>> path_logfile = path.join( # similar to /home/my_nickname/.mang-sdk.log
...     path.expanduser('~'),
...     '.mang-sdk.log'
... )
>>> from mangrove.logger import logger, logging
>>> from logging.handlers import RotatingFileHandler
>>> file_handler = RotatingFileHandler(
...     path_to_your_logfile, 'a', 1000000, 1
... )
>>> file_handler.setLevel(logging.DEBUG)
>>> logger.addHandler(file_handler)
```

Run your script/code (with the unexpected behavior) and send it to our support (support@mangrove.ai).

6.8 Indices and tables

- genindex

- modindex
- search

Python Module Index

m

Index

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W