prevision-python Documentation

Prevision.io

Jun 04, 2021

Contents:

Prevision.io is an automated SaaS machine learning platform that enables you to create and deploy powerful predictive models and business applications in one-click.

This documentation focuses on how to use Prevision.io's Python SDK for a direct usage in your data science scripts.

To take a quick peek at the available features, look at the *Getting started* guide.

For more in-depth explanations, dive into the extended documentation with *Tutorials & Samples*.

If you'd rather examine the Python API directly, here is the direct *API Reference*.

The compatibility version between Prevision.io's Python SDK and Prevision Platform works as follows:

Table 1: Compatibility matrix

	Previsions 10.10	Previsions 10.11	Previsions 10.12	Previsions 10.13	Previsions 10.14	Previsions 10.15	Previsions 10.16	Previsions 10.17	Previsions 10.18	Previsions 10.19
Prevision Python SDK 10.10	✓	✓	✓							
Prevision Python SDK 10.11	✓	✓	✓							
Prevision Python SDK 10.12	✓	✓	✓							
Prevision Python SDK 10.13					✓	✓	✓	✓	✓	
Prevision Python SDK 10.14				✓	✓	✓	✓	✓	✓	
Prevision Python SDK 10.15				✓	✓	✓	✓	✓	✓	
Prevision Python SDK 10.16				✓	✓	✓	✓	✓	✓	
Prevision Python SDK 10.17				✓	✓	✓	✓	✓	✓	
Prevision Python SDK 10.18				✓	✓	✓	✓	✓	✓	
Prevision Python SDK 10.19										✓

CHAPTER 1

Getting started

1.1 Pre-requisites

You need to have an account at cloud.prevision.io or on an on-premise version installed in your company. Contact us or your IT manager for more info.

You will be working on a specific "instance". This instance corresponds to the subdomain at the beginning of the url in your prevision.io address: `https://<your instance>.prevision.io`.

1.2 Get the package

1. clone the git repo:

```
git clone https://github.com/previsionio/prevision-python.git
```

2. install as a Python package:

```
cd prevision-python
python setup.py install
```

1.3 Setup your client

Prevision.io's SDK client uses a specific master token to authenticate with the instance's server and allow you to perform various requests. To get your master token, log in the online interface on your instance, navigate to the admin page and copy the token.

You can either set the token and the instance name as environment variables, by specifying `PREVISION_URL` and `PREVISION_MASTER_TOKEN`, or at the beginning of your script:

```
import previsionio as pio

# We initialize the client with our master token and the url of the prevision.io␣
↪server
# (or local installation, if applicable)
url = """https://<your instance>.prevision.io"""
token = """<your token>"""
pio.client.init_client(url, token)
```

1.4 A small example

1.4.1 Getting some data

First things first, to train a usecase, you need to gather some training data. This data can be passed a `pandas DataFrame` or a string representing a path to a file.

```
# load some data from a CSV file
data_path = 'data/titanic.csv'
dataset = pio.Dataset.new(name='helloworld', file_name=data_path)

# or use a pandas DataFrame
dataframe = pd.read_csv(data_path)
dataset = pio.Dataset.new(name='helloworld', dataframe=dataframe)
```

This will automatically read the given data and upload it as a new dataset on your Prevision.io's instance. If you go to the online interface, you will see this new dataset in the list of datasets (in the "Data" tab).

You can also load in your script a dataset that has already been uploaded on the platform; to do so, either use its name or its unique id:

```
# load a dataset by name
dataset = pio.Dataset.from_name('helloworld')

# or by unique id
dataset = pio.Dataset.from_id('5ebaad70a7271000e7b28ea0')
```

Note: If you want to list all of the available datasets on your instance, simply use:

```
datasets = pio.Dataset.list()
```

1.4.2 Configuring a usecase

If you want, you can also specify some training parameters, such as which models are used, which transformations are applied, and how the models are optimized.

```
uc_config = pio.TrainingConfig(models=[pio.Model.XGBoost],
                               features=pio.Feature.Full,
                               profile=pio.Profile.Quick)
```

For a full details on training config and training parameters, see the training config documentation.

1.4.3 Starting training

You can then create a new usecase based on :

- a usecase name
- a dataset
- a column config
- (optional) a metric type
- (optional) a training config

```
uc = pio.Classification.fit('helloworld_classif',
                            dataset,
                            metric=pio.metrics.Classification.AUC,
                            training_config=uc_config)
```

Note: For more complex usecase setups (for example with an image dataset), refer to the *Starting a usecase* guide.

1.4.4 Monitoring training

You can retrieve at any moment the number of models trained so far and the current error score, as well as some additional info.

```
>>> uc.score
0.0585

>>> len(uc)
4

>>> uc.print_info()
scores_cv: 0.0585
nbPreds: None
datasetId: 5b4f039f997e790001081657
nbModels: 4
state: RUNNING
```

You can also wait until a certain condition is reached, such as a number of models or a certain score:

```
# will block until there are more than 3 models
uc.wait_until(lambda usecase: len(usecase) > 3)

# will block until error is lower than 0.3 (warning, it may never reach it and wait␣
↪forever)
uc.wait_until(lambda usecase: usecase.score < .3)
```

The `wait_until` method takes a function that takes the usecase as an argument, and can therefore access any info relative to the usecase. For now, `len(usecase)` and `usecase.score` are the most useful ones, but other will come with time.

1.4.5 Making predictions

Once we have at least a model, we can start making predictions. We don't need to wait until the complete training process is done, and we'll always have access to the best model trained so far.

```
# we have some test data here:
data_path = 'data/titanic_test.csv'
test_dataset = pio.Dataset.new(name='helloworld_test', file_name=data_path)

# we can predict asynchronously (just like training, either from a file path or a␣
↪dataframe):
predict_id = uc.predict_from_dataset(test_dataset)

# We wait until they're ready
uc.wait_for_prediction(predict_id)

# and retrieve them once they're done
preds = uc.download_prediction(predict_id)

# If the data is not too large, we can predict synchronously, scikit-learn style:
df = pd.read_csv(data_path)
preds = uc.predict(df)
```

1.5 Additional util methods

1.5.1 Retrieving a use case

Since a use case can be somewhat long to train, it can be useful to separate the training, monitoring and prediction phases.

To do that, we need to be able to recreate a usecase object in python from its name:

```
uc = pio.Supervised.from_name('titanic')
# or
uc = pio.Supervised.from_id('<a usecase id>')
# uc now has all the same methods as a usecase created directly from a file or a␣
↪dataframe
>>> uc.print_info()
scores_cv: 0.0585
nbPreds: None
datasetId: 5b4f039f997e790001081657
nbModels: 4
state: RUNNING
```

1.5.2 Stopping and deleting

Once you're satisfied with model performance, don't want to wait for the complete training process to be over, or need to free up some resources to start a new training, you can stop the model simply:

```
uc.stop()
```

You'll still be able to make predictions and get info, but the performance won't improve anymore. Note: there's no difference in state between a stopped usecase and a usecase that has completed its training completely.

You can also decide to completely delete the usecase:

```
uc.delete()
```

However, be careful because, in that case, any detail about the usecase will be removed, and you won't be able to make predictions anymore.

CHAPTER 2

Tutorials & Samples

2.1 Tutorials

2.1.1 Configuring the client

To connect with Prevision.io's Python SDK onto your instance, you need to have your master token. This token can be found either:

- by going to the online web interface, in the "Administration & API key" page
- by calling:

```
client.init_client_with_login(prevision_url, email, password)
```

Then, to use your client credentials, you have 2 options:

- set the credentials as environment variables so that they are automatically reloaded when you run your scripts: you need to set `PREVISION_URL` to your instance url (i.e. something of the form: `https://<instance_name>.prevision.io`) and `PREVISION_MASTER_TOKEN` to the master token you just retrieved
- set the credentials at the beginning of your script, using the `init_client()` method:

```
import previsionio as pio

url = """https://<your instance>.prevision.io"""
token = """<your token>"""
pio.client.init_client(url, token)
```

For a full description of Prevision.io's client API, check out the *API Reference*.

2.1.2 Loading & fetching datasets

Loading up data

To use a dataset in a Prevision.io's usecase, you need to upload it on the platform. This can be done on the online platform, in the "Data" page, or through the SDK.

When using the SDK, you can reference a file path directly, or use a pre-read `pandas` dataframe, to easily create a new Prevision.io dataset on the platform:

```
# load some data from a CSV file
data_path = 'helloworld.csv'
dataset = pio.Dataset.new(name='helloworld', file_name=data_path)

# or use a pandas DataFrame
dataframe = pd.read_csv(data_path)
dataset = pio.Dataset.new(name='helloworld', dataframe=dataframe)
```

If you have a datasource you to take a snapshot of to create a dataset (see *Managing datasources & connectors*), then use the SDK resource object in your arguments:

```
datasource = pio.Dataset.from_name('my_datasource')
dataset = pio.Dataset.new(name='helloworld', datasource=datasource)
```

Listing available datasets

To get a list of all the datasets currently available on the platform (in your workspace), use the `list()` method:

```
datasets = pio.Dataset.list()
for dataset in datasets:
    print(dataset.name)
```

Fetching data from the platform

If you already uploaded a dataset on the platform and want to grab it locally to perform some preprocessing, or a train/test split, simply use the `from_name()` or `from_id()` SDK methods:

```
# load a dataset by name
dataset = pio.Dataset.from_name('helloworld')

# or by unique id
dataset = pio.Dataset.from_id('5ebaad70a7271000e7b28ea0')
```

2.1.3 Starting a usecase

To run a use case and train models on your training subset, you need to define some configuration parameters, and then simply use the SDK's *BaseUsecase*-derived methods to have the platform automatically take care of starting everything for you.

Configuring the use case columns

In order for the platform to know what your training target is, or whether you have some specific id columns that should not be taken into account during computation, you need to specify some "column configuration" for your use case.

These columns are bundled together into a *ColumnConfig* instance; there are 5 interesting parameters:

- `target_column`: the name of the target column in the dataset
- `id_column` (optional): the name of an id column that has no value for the model (it doesn't have any true signal) but is just a handy list of references for example; this column should thus be ignored during training (but it will eventually be rematched to the prediction sample to give you back the full data)
- `fold_column` (optional): if you want to perform a custom stratification to improve the quality of your predictions (which is sometimes better than regular cross-validation), you can pass a specific column name to use as reference; if none is provided, a random stratification will be used and will try to force the same distribution of the target between folds
- `weight_column` (optional): sometimes, a numerical does not contain an actual feature but rather an indication of how important each row is — if that is the case, you can pass the name of this column as *weight_column* (the higher the weight, the more important the row — by default, all rows are considered to be of equal importance); note that if this is provided, the optimised metric will become weighted
- `drop_list` (optional): you can pass a list of column names that you wish to exclude from the training (they will simply be ignored)

There are additional columns required in case of a timeseries or image-based usecase: take a look at the *ColumnConfig* API reference for more details.

Here is an example of a very basic column configuration instance:

```
column_config = pio.ColumnConfig(target_column='TARGET', id_column='ID')
```

Configuring the training profile

You can also fine-tune your use case options by configuring a training profile. This ensemble of variables will decide several things for your use case: what models are tested out, what metric is used, the desired types of feature engineering…

The function offers you a range of options to choose from, among which some that are used quite often:

- `models`: the list of "full" models you want to add to your training pipeline chosen among "LR", "RF", "ET", "XGB", "LGB" and "NN"
- `simple_models`: the list of "simple" models you want to add to your training pipeline chosen among "LR" and "DT"
- `fe_selected_list`: the list of feature engineering blocks to integrate in the pipeline (these will be applied on your dataset during training to extract relevant information and transform it in the best possible way for the models fit step)
- `profile`: this Prevision.io specific is a way of setting a global run mode that determines both training time and performance. You may choose between 3 profiles:
  1. the "quick" profile runs very fast but has a lower performance (it is recommended for early trials)
  2. the "advanced" profile runs slower but has increased performance (it is usually for optimization steps at the end of your project)
  3. the "normal" profile is something in-between to help you investigate an interesting result
- `with_blend`: if you turn this setting on, you will allow Prevision.io's training pipeline to append additional "blend" models at the end that are based on some cherry-picked already-trained models and proceed to further optimization to usually get even better results

A common "quick-test" training config could be:

```
training_config = pio.TrainingConfig(models=[pio.Model.XGBoost, pio.Model.
↪RandomForest],
                                     features=pio.Feature.Full,
                                     profile=pio.Profile.Quick,
                                     with_blend=False)
```

Starting the use case!

To create the usecase and start your training session, you need to call the `fit()` function of one of the SDK's usecase classes. The class you pick depends on the type of problem your usecase uses: regression, (binary) classification, multiclassification or timeseries; and on whether it uses a simple tabular dataset or images.

For a full list of the usecase objects and their API, check out the *API Reference*.

You also need to provide the API with the dataset you want to use (for a tabular usecase) or the CSV reference dataset and ZIP image dataset (for an image usecase).

The following example shows how to start a regression on a simple tabular dataset:

```
uc = pio.Regression.fit('helloworld reg',
                        dataset,
                        column_config=column_config,
                        training_config=training_config)
```

If you are running an image usecase, then you need to pass the two datasets as a tuple:

The following example shows how to start a regression on a simple tabular dataset (where the CSV reference dataset is a *Dataset* instance and the ZIP image dataset is a *DatasetImages* instance):

```
uc = pio.RegressionImages.fit('helloworld images reg',
                              (dataset_csv, dataset_zip),
                              column_config=column_config,
                              training_config=training_config)
```

When you start your usecase, you can either let the SDK pick a default metric according to your usecase type, or you can choose one yourself from the list of available *Metrics*.

2.1.4 Managing datasources & connectors

Datasources and connectors are Prevision.io's way of keeping a link to a source of data and taking snapshots when needed. The distant data source can be an FTP server, a database, an Amazon bucket...

Connectors hold the credentials to connect to the distant data source and datasources specify the exact resource to resource from it (be it the path to the file to load, the name of the database table to parse...).

For more info on all the options of connectors and datasources, check out the *API Reference*.

Listing available connectors and datasources

Like all SDK API resources, connectors and datasources already registered on the platform can be listed using the `list()` method:

```
connectors = pio.Connector.list()
for connector in connectors:
    print(connector.name)

datasources = pio.Datasource.list()
for datasource in datasources:
    print(datasource.name)
```

Creating a connector

To create a connector, use the `new()` method of the connector class you want to use. For example, to create a connector to an SQL database, use the *SQLConnector* and pass in your credentials:

```
connector = pio.SQLConnector.new('my_sql_connector',
                                 'https://myserver.com',
                                 port=3306,
                                 username='username',
                                 password='password')
```

Creating a datasource

After you've created a connector, you need to use a datasource to actually refer to and fetch a resource in the distant data source. To create a datasource, you need to link the matching connector and to supply the relevant info, depending on the connector type.

```
datasource = pio.Datasource.new(connector,
                                'my_sql_datasource',
                                database='my_db',
                                table='table1')
```

You can then create datasets from this datasource as explained in the guide on *Loading & fetching datasets*.

2.2 Samples

2.2.1 Getting started

This piece of code shows how to:

- initialize a connection to your instance and authentify with your token
- load some data
- start a usecase
- get info about the usecase and its model
- make some predictions

```
import previsionio as pio
import pandas as pd

# CLIENT INITIALIZATION ----------------------------------------
url = """https://<your instance>.prevision.io"""
```

(continues on next page)

(continued from previous page)

```
token = """<your token>"""
pio.client.init_client(url, token)

# DATA LOADING ------------------------------------------------
# load data from a CSV
dataframe = pd.read_csv('helloworld_train.csv')
# upload it to the platform
dataset = pio.Dataset.new(name='helloworld_train', dataframe=dataframe)

# USECASE TRAINING --------------------------------------------
# setup usecase
uc_config = pio.TrainingConfig(models=[pio.Model.XGBoost],
                               features=pio.Feature.Full,
                               profile=pio.Profile.Quick)

# run training
uc = pio.Classification.fit('helloworld_classif',
                            dataset,
                            metric=pio.metrics.Classification.AUC,
                            training_config=uc_config)

# (block until there is at least 1 model trained)
uc.wait_until(lambda usecase: len(usecase) > 0)

# check out the usecase status and other info
uc.print_info()
print('Current number of models:', len(uc))
print('Current (best model) score:', uc.score)

# PREDICTIONS -------------------------------------------------
# load up test data
test_datapath = 'helloworld_test.csv'
test_dataset = pio.Dataset.new(name='helloworld_test', file_name=test_datapath)

# 1. use an ASYNC prediction
predict_id = uc.predict_from_dataset(test_dataset)
uc.wait_for_prediction(predict_id)
preds = uc.download_prediction(predict_id)
print(preds)

# 2. or use a SYNC prediction (Scikit-learn blocking style)
# WARNING: should only be used for small datasets
df = pd.read_csv(test_datapath)
preds = uc.predict(df)
```

2.2.2 Setting logging

Prevision.io's SDK can provide with more detailed information if you change the logging level. By default, it will only output warnings and errors.

To change the logging level, use `pio.verbose()` method:

previsionio.__init__.**verbose**(*v*, *debug=False*, *event_log=False*)

Set the SDK level of verbosity.

Parameters

- **v** (*bool*) – Whether to activate info logging
- **debug** (*bool, optional*) – Whether to activate detailed debug logging (default: `False`)
- **event_log** (*bool, optional*) – Whether to activate detailed event managers debug logging (default: `False`)

For example:

```
import previsionio as pio

# CHANGE LOGGING LEVEL ----------------------------------------
pio.verbose(True, debug=True) # (add event_log=True
                              # for events logging)

# CLIENT INITIALIZATION ---------------------------------------
url = """https://<your instance>.prevision.io"""
token = """<your token>"""
pio.client.init_client(url, token)

# TESTING LOGS ------------------------------------------------
# fetching a dataset from the platform
dataset = pio.Dataset.from_name('helloworld')

# fetching a usecase from the platform
uc = pio.Supervised.from_name('helloworld classif')
```

CHAPTER 3

API Reference

This section gathers all the available classes, functions and tools offered by Prevision.io's Python SDK.

3.1 Client

Prevision.io's SDK client uses a specific master token to authenticate with the instance's server and allow you to perform various requests. To get your master token, log in the online interface, navigate to the admin page and copy the token.

You can either set the token and the instance name as environment variables, by specifying `PREVISION_URL` and `PREVISION_MASTER_TOKEN`, or at the beginning of your script:

```
import previsionio as pio

# We initialize the client with our master token and the url of the prevision.io␣
↪server
# (or local installation, if applicable)
url = """https://<your instance>.prevision.io"""
token = """<your token>"""
pio.client.init_client(url, token)
```

class previsionio.prevision_client.**Client**

Client class to interact with the Prevision.io platform and manage authentication.

init_client(*prevision_url*, *token*)

Init the client (and check that the connection is valid).

Parameters

- **prevision_url** (*str*) – URL of the Prevision.io platform. Should be https://cloud.prevision.io if you're in the cloud, or a custom IP address if installed on-premise.
- **token** (*str*) – Your Prevision.io master token. Can be retrieved on /dashboard/infos on the web interface or obtained programmatically through:

```
client.init_client_with_login(prevision_url, email, password)
```

request(*endpoint*, *method*, *files=None*, *data=None*, *allow_redirects=True*, *content_type=None*, *no_retries=False*, ***requests_kwargs*)

Make a request on the desired endpoint with the specified method & data.

Requires initialization.

Parameters

- **endpoint** – (str): api endpoint (e.g. /usecases, /prediction/file)
- **method** (*requests.{get,post,delete}*) – requests method
- **files** (*dict*) – files dict
- **data** (*dict*) – for single predict
- **content_type** (*str*) – force request content-type
- **allow_redirects** (*bool*) – passed to requests method
- **no_retries** (*bool*) – force request to run the first time, or exit directly

Returns request response

Raises Exception – Error if url/token not configured

3.2 Base API Resource

All resource objects you will be using in Prevision.io's Python SDK inherit from this base parent class.

In the SDK, a resource is an object that can be fetched from the platform, used in your code, updated, deleted... *previsionio.usecase.BaseUsecase*, *previsionio.dataset.Dataset* and *previsionio.model.Model* are all resources.

class previsionio.api_resource.**ApiResource**(**args*, ***params*)

Base parent class for all SDK resource objects.

delete()

Delete a resource from the actual [client] workspace.

Raises PrevisionException – Any error while deleting data from the platform

edit(***kwargs*)

Edit a resource on the platform. You simply pass the function a dictionary of all the fields you want to update (as kwargs), with the name of the field as key and the new data for the field as value.

Note: The parameters you can update can be listed by calling:

```
print(my_resource.resource_params)
```

Returns Updated resource data

Return type dict

Raises PrevisionException – Any error while updating data on the platform or parsing result

classmethod from_id(*_id=None*, *specific_url=None*)
Get a resource from the platform by its unique id. You must provide either an `_id` or a `specific_url`.

Parameters

- **_id** (*str, optional*) – Unique id of the resource to retrieve
- **specific_url** (*str, optional*) – Specific (already parametrized) url to fetch the resource from

Returns Fetched resource

Return type *ApiResource*

Raises

- `Exception` – If neither an `_id` nor a `specific_url` was provided
- `PrevisionException` – Any error while fetching data from the platform or parsing result

classmethod from_name(*name*, *raise_if_non_unique=False*, *partial_match=False*)
Get a resource from the platform by its name.

Parameters

- **name** (*str*) – Name of the resource to retrieve
- **raise_if_non_unique** (*bool, optional*) – Whether or not to raise an error if duplicates are found (default: `False`)
- **partial_match** (*bool, optional*) – If true, resources with a name containing the requested name will also be returned; else, only perfect matches will be found (default: `False`)

Raises `PrevisionException` – Error if duplicates are found and the `raise_if_non_unique` is enabled

Returns Fetched resource

Return type *ApiResource*

classmethod list(*all=False*)
List all available instances of this resource type on the platform.

Parameters **all** (*boolean, optional*) – Whether to force the SDK to load all items of the given type (by calling the paginated API several times). Else, the query will only return the first page of result.

Returns Fetched resources

Return type dict

update_status(*specific_url=None*)
Get an update on the status of a resource.

Parameters **specific_url** (*str, optional*) – Specific (already parametrized) url to fetch the resource from (otherwise the url is built from the resource type and unique `_id`)

Returns Updated status info

Return type dict

class previsionio.api_resource.**ApiResourceType**
All the different resource types and matching API endpoints.

3.3 Dataset

class previsionio.dataset.**Dataset**(*_id*, *name*, *datasource=None*, *_data=None*, ***kwargs*)
Bases: *previsionio.api_resource.ApiResource*

Dataset objects represent data resources that will be explored by Prevision.io platform.

In order to launch an auto ml process (see *BaseUsecase* class), we need to have the matching dataset stored in the related workspace.

Within the platform they are stored in tabular form and are derived:

- from files (CSV, ZIP)
- or from a Data Source at a given time (snapshot)

data
Load in memory the data content of the current dataset into a pandas DataFrame.

Returns Dataframe for the data object

Return type pd.DataFrame

Raises PrevisionException – Any error while fetching or parsing the data

delete()
Delete a dataset from the actual [client] workspace.

Raises

- PrevisionException – If the dataset does not exist
- requests.exceptions.ConnectionError – Error processing the request

classmethod download(*dataset_name=None*, *download_path=None*)
Download the dataset from the platform locally.

Parameters

- **dataset_name** (*str*) – Name of the dataset to download
- **download_path** (*str, optional*) – Target local directory path (if none is provided, the current working directory is used)

Returns Path the data was downloaded to

Return type str

Raises PrevisionException – If dataset does not exist or if there was another error fetching or parsing data

classmethod get_by_name(*name=None*, *version='last'*)
Get an already registered dataset from the platform (using its registration name).

Parameters

- **name** (*str*) – Name of the dataset to retrieve
- **version** (*int, str, optional*) – Specific version of the dataset (can be an int, or 'last' - default - to get the latest version of the dataset)

Raises

- AttributeError – if dataset_name is not given
- PrevisionException – If dataset does not exist or if there was another error fetching or parsing data

Returns Fetched dataset

Return type *Dataset*

get_embedding()
Gets the embeddings analysis of the dataset from the actual [client] workspace

Raises

- PrevisionException – DatasetNotFoundError
- requests.exceptions.ConnectionError – request error

classmethod getid_from_name(*name=None*, *version='last'*)
Return the dataset id corresponding to a given name.

Parameters

- **name** (*str*) – Name of the dataset
- **version** (*int, str, optional*) – Specific version of the dataset (can be an int, or 'last' - default - to get the latest version of the dataset)

Raises PrevisionException – If dataset does not exist, version number is out of range or there is another error fetching or parsing data

classmethod list(*all=<built-in function all>*)
List all the available datasets in the current active [client] workspace.

> **Warning:** Contrary to the parent list() function, this method returns actual *Dataset* objects rather than plain dictionaries with the corresponding data.

Parameters **all** (*boolean, optional*) – Whether to force the SDK to load all items of the given type (by calling the paginated API several times). Else, the query will only return the first page of result.

Returns Fetched dataset objects

Return type list(*Dataset*)

classmethod new(*name*, *datasource=None*, *file_name=None*, *dataframe=None*)
Register a new dataset in the workspace for further processing. You need to provide either a datasource, a file name or a dataframe (only one can be specified).

Note: To start a new use case on a dataset, it has to be already registred in your workspace.

Parameters

- **name** (*str*) – Registration name for the dataset
- **datasource** (*DataSource*, optional) – A DataSource object used to import a remote dataset (if you want to import a specific dataset from an existent database, you need a datasource connector (*Connector* object) designed to point to the related data source)
- **file_name** (*str, optional*) – Path to a file to upload as dataset
- **dataframe** (*pd.DataFrame, optional*) – A pandas dataframe containing the data to upload

Raises

- Exception – If more than one of the keyword arguments datasource, file_name, dataframe was specified
- PrevisionException – Error while creating the dataset on the platform

Returns The registered dataset object in the current workspace.

Return type *Dataset*

start_embedding()
Starts the embeddings analysis of the dataset from the actual [client] workspace

Raises

- PrevisionException – DatasetNotFoundError
- requests.exceptions.ConnectionError – request error

to_pandas() → pandas.core.frame.DataFrame
Load in memory the data content of the current dataset into a pandas DataFrame.

Returns Dataframe for the data object

Return type pd.DataFrame

Raises PrevisionException – Any error while fetching or parsing the data

class previsionio.dataset.**DatasetImages**(*_id*, *name*, *datasource=None*, *_data=None*, ***kwargs*)
Bases: *previsionio.dataset.Dataset*

DatasetImages objects represent image data resources that will be used by Prevision.io's platform.

In order to launch an auto ml process (see *BaseUsecase* class), we need to have the matching dataset stored in the related workspace.

Within the platform, image folder datasets are stored as ZIP files and are copied from ZIP files.

classmethod new(*name*, *file_name*)
Register a new image dataset in the workspace for further processing (in the image folders group).

Note: To start a new use case on a dataset, it has to be already registred in your workspace.

Parameters

- **name** (*str*) – Registration name for the dataset
- **file_name** (*str*) – Path to the zip file to upload as image dataset

Raises PrevisionException – Error while creating the dataset on the platform

Returns The registered dataset object in the current workspace.

Return type *Dataset*

to_pandas() → pandas.core.frame.DataFrame
Invalid method for a *DatasetImages* object.

Raises ValueError – Folder datasets cannot be converted to a pandas dataframe

3.4 DataSource

class previsionio.datasource.**DataSource**(*_id*, *connector*, *name*, *path=None*, *database=None*, *table=None*, *request=None*, *gCloud=None*, ***kwargs*)

Bases: *previsionio.api_resource.ApiResource*, previsionio.api_resource.UniqueResourceMixin

A datasource to access a distant data pool and create or fetch data easily. This resource is linked to a *Connector* resource that represents the connection to the distant data source.

Parameters

- **_id** (*str*) – Unique id of the datasource
- **connector** (*Connector*) – Reference to the associated connector (the resource to go through to get a data snapshot)
- **name** (*str*) – Name of the datasource
- **path** (*str, optional*) – Path to the file to fetch via the connector
- **database** (*str, optional*) – Name of the database to fetch data from via the connector
- **table** (*str, optional*) – Name of the table to fetch data from via the connector
- **request** (*str, optional*) – Direct SQL request to use with the connector to fetch data

classmethod from_id(*_id*)

Get a datasource from the instance by its unique id.

Parameters **_id** (*str*) – Unique id of the resource to retrieve

Returns The fetched datasource

Return type *DataSource*

Raises PrevisionException – Any error while fetching data from the platform or parsing the result

classmethod list(*all=False*)

List all the available datasources in the current active [client] workspace.

> **Warning:** Contrary to the parent `list()` function, this method returns actual *DataSource* objects rather than plain dictionaries with the corresponding data.

Parameters **all** (*boolean, optional*) – Whether to force the SDK to load all items of the given type (by calling the paginated API several times). Else, the query will only return the first page of result.

Returns Fetched datasource objects

Return type list(*DataSource*)

classmethod new(*connector*, *name*, *path=None*, *database=None*, *table=None*, *bucket=None*, *request=None*, *gCloud=None*)

Create a new datasource object on the platform.

Parameters

- **connector** (*Connector*) – Reference to the associated connector (the resource to go through to get a data snapshot)
- **name** (*str*) – Name of the datasource
- **path** (*str, optional*) – Path to the file to fetch via the connector
- **database** (*str, optional*) – Name of the database to fetch data from via the connector
- **table** (*str, optional*) – Name of the table to fetch data from via the connector
- **request** (*str, optional*) – Direct SQL request to use with the connector to fetch data

Returns The registered datasource object in the current workspace

Return type *DataSource*

Raises

- `PrevisionException` – Any error while uploading data to the platform or parsing the result
- `Exception` – For any other unknown error

3.5 Connector

In all the specific connectors, the parameters for the `new()` method are the same as the ones in the *Connector._new()*.

class previsionio.connector.**Connector**(*_id*, *name*, *host=None*, *port=None*, *type=None*, *username=''*, *password=''*, *googleCredentials=None*, ***kwargs*)

Bases: *previsionio.api_resource.ApiResource*, previsionio.api_resource.UniqueResourceMixin

A connector to interact with a distant source of data (and easily get data snapshots using an associated *DataSource* resource).

Parameters

- **_id** (*str*) – Unique reference of the connector on the platform
- **name** (*str*) – Name of the connector
- **host** (*str*) – Url of the connector
- **port** (*int*) – Port of the connector
- **conn_type** (*str*) – Type of the connector, among "FTP", "SFTP", "SQL", "S3", "HIVE", "HBASE", "GCP"
- **username** (*str, optional*) – Username to use connect to the remote data source
- **password** (*str, optional*) – Password to use connect to the remote data source

classmethod _new(*name*, *host*, *port*, *conn_type*, *username=None*, *password=None*, *googleCredentials=None*)

Create a new connector object on the platform.

Parameters

- **name** (*str*) – Name of the connector

- **host** (*str*) – Url of the connector
- **port** (*int*) – Port of the connector
- **conn_type** (*str*) – Type of the connector, among "FTP", "SFTP", "SQL", "S3", "HIVE", "HBASE" or "GCP"
- **username** (*str, optional*) – Username to use connect to the remote data source
- **password** (*str, optional*) – Password to use connect to the remote data source

Returns Newly create connector object

Return type *Connector*

classmethod list(*all=False*)

List all the available connectors in the current active [client] workspace.

> **Warning:** Contrary to the parent `list()` function, this method returns actual *Connector* objects rather than plain dictionaries with the corresponding data.

Parameters **all** (*boolean, optional*) – Whether to force the SDK to load all items of the given type (by calling the paginated API several times). Else, the query will only return the first page of result.

Returns Fetched connector objects

Return type list(*Connector*)

test()

Test a connector already uploaded on the platform.

Returns Test results

Return type dict

class previsionio.connector.**DataBaseConnector**(*_id*, *name*, *host=None*, *port=None*, *type=None*, *username=''*, *password=''*, *googleCredentials=None*, ***kwargs*)

Bases: *previsionio.connector.Connector*

A specific type of connector to interact with a database client (containing databases and tables).

list_databases()

List all available databases for the client.

Returns Databases information

Return type dict

list_tables(*database*)

List all available tables in a specific database for the client.

Parameters **database** (*str*) – Name of the database to find tables for

Returns Tables information

Return type dict

class previsionio.connector.**FTPConnector**(*_id*, *name*, *host=None*, *port=None*, *type=None*, *username=''*, *password=''*, *googleCredentials=None*, ***kwargs*)

Bases: *previsionio.connector.Connector*

A specific type of connector to interact with a FTP client (containing files).

class previsionio.connector.**GCPConnector**(*_id*, *name*, *host=None*, *port=None*, *type=None*, *username=''*, *password=''*, *googleCredentials=None*, ***kwargs*)

Bases: *previsionio.connector.Connector*

A specific type of connector to interact with a GCP database client (containing databases and tables or buckets).

class previsionio.connector.**HiveConnector**(*_id*, *name*, *host=None*, *port=None*, *type=None*, *username=''*, *password=''*, *googleCredentials=None*, ***kwargs*)

Bases: *previsionio.connector.DataBaseConnector*

A specific type of connector to interact with a Hive database client (containing databases and tables).

class previsionio.connector.**S3Connector**(*_id*, *name*, *host=None*, *port=None*, *type=None*, *username=''*, *password=''*, *googleCredentials=None*, ***kwargs*)

Bases: *previsionio.connector.Connector*

A specific type of connector to interact with an Amazon S3 client (containing buckets with files).

class previsionio.connector.**SFTPConnector**(*_id*, *name*, *host=None*, *port=None*, *type=None*, *username=''*, *password=''*, *googleCredentials=None*, ***kwargs*)

Bases: *previsionio.connector.Connector*

A specific type of connector to interact with a secured FTP client (containing files).

class previsionio.connector.**SQLConnector**(*_id*, *name*, *host=None*, *port=None*, *type=None*, *username=''*, *password=''*, *googleCredentials=None*, ***kwargs*)

Bases: *previsionio.connector.DataBaseConnector*

A specific type of connector to interact with a SQL database client (containing databases and tables).

3.6 Usecases

Prevision.io's Python SDK enables you to very easily run usecases of different types: regression, (binary) classification, multiclassification or timeseries.

All these classes inherit from the base *previsionio.usecase.BaseUsecase* class, and then from the *previsionio.supervised.Supervised* class.

When starting a usecase, you also need to specify a training configuration.

Take a look at the specific documentation pages for a more in-depth explanation of each layer and of the usecase configuration options:

3.6.1 Base usecase

class previsionio.usecase.**BaseUsecase**(***usecase_info*)

Bases: *previsionio.api_resource.ApiResource*

Base parent class for all usecases objects.

best_model

Get the model with the best predictive performance over all models (including Blend models), where the best performance corresponds to a minimal loss.

Returns Model with the best performance in the usecase, or `None` if no model matched the search filter.

Return type (*Model*, None)

best_single
Get the model with the best predictive performance (the minimal loss) over single models (excluding Blend models), where the best performance corresponds to a minimal loss.

Returns Single (non-blend) model with the best performance in the usecase, or `None` if no model matched the search filter.

Return type *Model*

correlation_matrix
Get the correlation matrix of the features (those constitute the dataset on which the usecase was trained).

Returns Correlation matrix as a `pandas` dataframe

Return type `pd.DataFrame`

delete()
Delete a usecase from the actual [client] workspace.

Returns Deletion process results

Return type dict

delete_prediction(*prediction_id*)
Delete a prediction in the list for the current usecase from the actual [client] workspace.

Parameters **prediction_id** (*str*) – Unique id of the prediction to delete

Returns Deletion process results

Return type dict

delete_predictions()
Delete all predictions in the list for the current usecase from the actual [client] workspace.

Returns Deletion process results

Return type dict

drop_list
Get the list of drop columns in the usecase.

Returns Names of the columns dropped from the dataset

Return type list(str)

fastest_model
Returns the model that predicts with the lowest response time

Returns Model object – corresponding to the fastest model

fe_selected_list
Get the list of selected feature engineering modules in the usecase.

Returns Names of the feature engineering modules selected for the usecase

Return type list(str)

feature_stats

- feature types distribution
- feature information list

- list of dropped features

Returns General features information

Return type dict

Type Get the general description of the usecase's features, such as

classmethod from_id(*_id*, *version=1*)
Get a usecase from the platform by its unique id.

Parameters

- **_id** (*str*) – Unique id of the usecase to retrieve
- **version** (*int, optional*) – Specific version of the usecase to retrieve (default: 1)

Returns Fetched usecase

Return type *BaseUsecase*

Raises PrevisionException – Any error while fetching data from the platform or parsing result

get_cv(*use_best_single=False*) → pandas.core.frame.DataFrame
Get the cross validation dataset from the best model of the usecase.

Parameters **use_best_single** (*bool, optional*) – Whether to use the best single model instead of the best model overall (default: False)

Returns Cross validation dataset

Return type pd.DataFrame

get_feature_info(*feature_name=None*)
Return some information about the given feature, such as:

- name: the name of the feature as it was given in the feature_name parameter
- type: linear, categorical, ordinal...
- stats: some basic statistics such as number of missing values, (non missing) values count, plus additional information depending on the feature type:
  - for a linear feature: min, max, mean, std and median
  - for a categorical/textual feature: modalities/words frequencies, list of the most frequent tokens
- role: whether or not the feature is a target/fold/weight or id feature (and for time series usecases, whether or not it is a group/apriori feature - check the Prevision.io's timeseries documentation)
- importance_value: scores reflecting the importance of the given feature

Parameters

- **feature_name** (*str*) – Name of the feature to get informations about
- **warning:** (.) – The feature_name is case-sensitive, so "age" and "Age" are different features!

Returns Dictionary containing the feature information

Return type dict

Raises PrevisionException – If the given feature name does not match any feaure

get_model_from_id(*id*)
Get a model of the usecase by its unique id.

Note: The model is only searched through the models that are done training.

Parameters **id** (*str*) – Unique id of the model resource to search for

Returns Matching model resource, or `None` if none with the given id could be found

Return type (*Model*, None)

get_model_from_name(*model_name=None*)
Get a model of the usecase by its name.

Note: The model is only searched through the models that are done training.

Parameters **name** (*str*) – Name of the model resource to search for

Returns Matching model resource, or `None` if none with the given name could be found

Return type (*Model*, None)

get_predictions(*full=False*)
Retrieves the list of predictions for the current usecase from client workspace (with the full predictions object if necessary) :param full: If true, return full prediction objects (else only metadata) :type full: boolean

lite_models_list
Get the list of selected lite models in the usecase.

Returns Names of the lite models selected for the usecase

Return type list(str)

models
Get the list of models generated for the current use case. Only the models that are done training are retrieved.

Returns List of models found by the platform for the usecase

Return type list(*Model*)

normal_models_list
Get the list of selected normal models in the usecase.

Returns Names of the normal models selected for the usecase

Return type list(str)

predict(*df*, *confidence=False*, *use_best_single=False*) → pandas.core.frame.DataFrame
Get the predictions for a dataset stored in the current active [client] workspace using the best model of the usecase with a Scikit-learn style blocking prediction mode.

Warning: For large dataframes and complex (blend) models, this can be slow (up to 1-2 hours). Prefer using this for simple models and small dataframes, or use option `use_best_single = True`.

Parameters

- **df** (pd.DataFrame) – pandas DataFrame containing the test data
- **confidence** (*bool, optional*) – Whether to predict with confidence values (default: False)
- **use_best_single** (*bool, optional*) – Whether to use the best single model instead of the best model overall (default: False)

Returns Prediction data (as pandas dataframe) and prediction job ID.

Return type tuple(pd.DataFrame, str)

predict_from_dataset(*dataset*, *use_best_single=False*, *confidence=False*, *dataset_folder=None*) → pandas.core.frame.DataFrame

Get the predictions for a dataset stored in the current active [client] workspace using the best model of the usecase.

Parameters

- **dataset** (*Dataset*) – Reference to the dataset object to make predictions for
- **use_best_single** (*bool, optional*) – Whether to use the best single model instead of the best model overall (default: False)
- **confidence** (*bool, optional*) – Whether to predict with confidence values (default: False)
- **dataset_folder** (*Dataset*) – Matching folder dataset for the predictions, if necessary

Returns Predictions as a pandas dataframe

Return type pd.DataFrame

predict_proba(*df*, *confidence=False*, *use_best_single=False*) → pandas.core.frame.DataFrame

Get the predictions for a dataset stored in the current active [client] workspace using the best model of the usecase with a Scikit-learn style blocking prediction mode, and returns the probabilities.

> **Warning:** For large dataframes and complex (blend) models, this can be slow (up to 1-2 hours). Prefer using this for simple models and small dataframes, or use option use_best_single = True.

Parameters

- **df** (pd.DataFrame) – pandas DataFrame containing the test data
- **confidence** (*bool, optional*) – Whether to predict with confidence values (default: False)
- **use_best_single** (*bool, optional*) – Whether to use the best single model instead of the best model overall (default: False)

Returns Prediction probabilities data (as pandas dataframe) and prediction job ID.

Return type tuple(pd.DataFrame, str)

predict_single(*use_best_single=False*, *confidence=False*, *explain=False*, ***predict_data*)

Get a prediction on a single instance using the best model of the usecase.

Parameters

- **use_best_single** (*bool, optional*) – Whether to use the best single model instead of the best model overall (default: `False`)
- **confidence** (*bool, optional*) – Whether to predict with confidence values (default: `False`)
- **explain** (*bool*) – Whether to explain prediction (default: `False`)

Returns

Dictionary containing the prediction.

Note: The format of the predictions dictionary depends on the problem type (regression, classification...)

Return type dict

print_info()
Print all info on the usecase.

running
Get a flag indicating whether or not the usecase is currently running.

Returns Running status

Return type bool

schema
Get the data schema of the usecase.

Returns Usecase schema

Return type `pd.DataFrame`

score
Get the current score of the usecase (i.e. the score of the model that is currently considered the best performance-wise for this usecase).

Returns Usecase score (or infinity if not available).

Return type float

share(*with_users*)
Share a usecase in the actual [client] workspace with other users (specified via their emails).

Parameters **with_users** (*list(str)*) – List of emails of the users to share the usecase with

simple_models_list
Get the list of selected simple models in the usecase.

Returns Names of the simple models selected for the usecase

Return type list(str)

stop()
Stop a usecase (stopping all nodes currently in progress).

train_dataset
Get the *Dataset* object corresponding to the training dataset of the usecase.

Returns Associated training dataset

Return type *Dataset*

unshare(*from_users*)
Unshare a usecase in the actual [client] workspace from other users (specified via their emails).

Parameters **from_users** (*list(str)*) – List of emails of the users to unshare the usecase from

update_status()
Get an update on the status of a resource.

Parameters **specific_url** (*str, optional*) – Specific (already parametrized) url to fetch the resource from (otherwise the url is built from the resource type and unique `_id`)

Returns Updated status info

Return type dict

versions
Get the list of all versions for the current use case.

Returns List of the usecase versions (as JSON metadata)

Return type list(dict)

wait_until(*condition*, *raise_on_error=True*, *timeout=10800*)
Wait until condition is fulfilled, then break.

Parameters

- **(func** (*condition*) – (*BaseUsecase*) -> bool.): Function to use to check the break condition
- **raise_on_error** (*bool, optional*) – If true then the function will stop on error, otherwise it will continue waiting (default: `True`)
- **timeout** (*float, optional*) – Maximal amount of time to wait before forcing exit

Raises `PrevisionException` – If the resource could not be fetched or there was a timeout.

3.6.2 Supervised usecases

class previsionio.supervised.**Classification**(***usecase_info*)
Bases: *previsionio.supervised.Supervised*

A (binary) classification usecase for a categorical target with exactly 2 modalities using a basic dataset.

predict_proba(*df*, *use_best_single=False*, *confidence=False*) → pandas.core.frame.DataFrame
Get the predictions for a dataset stored in the current active [client] workspace using the best model of the usecase with a Scikit-learn style blocking prediction mode, and returns the probabilities.

> **Warning:** For large dataframes and complex (blend) models, this can be slow (up to 1-2 hours). Prefer using this for simple models and small dataframes, or use option `use_best_single = True`.

Parameters

- **df** (`pd.DataFrame`) – `pandas` DataFrame containing the test data
- **confidence** (*bool, optional*) – Whether to predict with confidence values (default: `False`)
- **use_best_single** (*bool, optional*) – Whether to use the best single model instead of the best model overall (default: `False`)

Returns Prediction probabilities data (as `pandas` dataframe) and prediction job ID.

Return type tuple(pd.DataFrame, str)

class previsionio.supervised.**ClassificationImages**(***usecase_info*)
Bases: *previsionio.supervised.SupervisedImages*

A (binary) classification usecase for a categorical target with exactly 2 modalities using an image dataset.

class previsionio.supervised.**MultiClassification**(***usecase_info*)
Bases: *previsionio.supervised.Supervised*

A multiclassification usecase for a categorical target with strictly more than 2 modalities using a basic dataset.

class previsionio.supervised.**MultiClassificationImages**(***usecase_info*)
Bases: *previsionio.supervised.SupervisedImages*

A multiclassification usecase for a categorical target with strictly more than 2 modalities using an image dataset.

class previsionio.supervised.**Regression**(***usecase_info*)
Bases: *previsionio.supervised.Supervised*

A regression usecase for a numerical target using a basic dataset.

class previsionio.supervised.**RegressionImages**(***usecase_info*)
Bases: *previsionio.supervised.SupervisedImages*

A regression usecase for a numerical target using an image dataset.

class previsionio.supervised.**Supervised**(***usecase_info*)
Bases: *previsionio.usecase.BaseUsecase*

A supervised usecase.

classmethod fit(*name*, *dataset*, *column_config*, *metric=None*, *holdout_dataset=None*, *training_config=<previsionio.usecase_config.TrainingConfig object>*, *type_problem=None*, ***kwargs*)
Start a supervised usecase training with a specific training configuration (on the platform).

Parameters

- **name** (*str*) – Name of the usecase to create
- **dataset** (*Dataset*, *DatasetImages*) – Reference to the dataset object to use for as training dataset
- **column_config** (*ColumnConfig*) – Column configuration for the usecase (see the documentation of the *ColumnConfig* resource for more details on each possible column types)
- **metric** (*str, optional*) – Specific metric to use for the usecase (default: `None`)
- **holdout_dataset** (*Dataset*, optional) – Reference to a dataset object to use as a holdout dataset (default: `None`)
- **training_config** (*TrainingConfig*) – Specific training configuration (see the documentation of the *TrainingConfig* resource for more details on all the parameters)
- **type_problem** (*str, optional*) – Specific problem type to train (default: `None`)

Returns Newly created supervised usecase object

Return type *Supervised*

classmethod from_id(*_id*, *version=1*)
Get a supervised usecase from the platform by its unique id.

Parameters

- **_id** (*str*) – Unique id of the usecase to retrieve
- **version** (*int, optional*) – Specific version of the usecase to retrieve (default: 1)

Returns Fetched usecase

Return type *Supervised*

Raises PrevisionException – Invalid problem type or any error while fetching data from the platform or parsing result

classmethod from_name(*name*, *raise_if_non_unique=False*, *partial_match=False*)
Get a supervised usecase from the platform by its name.

Parameters

- **name** (*str*) – Name of the usecase to retrieve
- **raise_if_non_unique** (*bool, optional*) – Whether or not to raise an error if duplicates are found (default: False)
- **partial_match** (*bool, optional*) – If true, usecases with a name containing the requested name will also be returned; else, only perfect matches will be found (default: False)

Raises PrevisionException – Error if duplicates are found and the raise_if_non_unique is enabled

Returns Fetched usecase

Return type *Supervised*

new_version(***fit_params*)
Start a supervised usecase training to create a new version of the usecase (on the platform): the training config is copied from the current version and then overridden for the given parameters.

Parameters **fit_params** (*kwargs*) – Training config parameters to change for the new version (compared to the current version)

Returns Newly created supervised usecase object (new version)

Return type *Supervised*

class previsionio.supervised.**SupervisedImages**(***usecase_info*)
Bases: *previsionio.supervised.Supervised*

A supervised usecase with an image dataset.

class previsionio.timeseries.**TimeSeries**(***usecase_info*)
Bases: *previsionio.usecase.BaseUsecase*

A TimeSeries usecase.

model_class
alias of *previsionio.model.RegressionModel*

class previsionio.timeseries.**TimeWindow**(*derivation_start*, *derivation_end*, *forecast_start*, *forecast_end*)
Bases: previsionio.usecase_config.UsecaseConfig

A time window object for representing either feature derivation window periods or forecast window periods

exception previsionio.timeseries.**TimeWindowException**
Bases: Exception

3.6.3 Usecase configuration

class previsionio.usecase_config.**ColumnConfig**(*target_column=None*, *filename_column=None*, *id_column=None*, *fold_column=None*, *weight_column=None*, *time_column=None*, *group_columns=()*, *apriori_columns=()*, *drop_list=()*)

Column configuration for starting a usecase: this object defines the role of specific columns in the dataset (and optionally the list of columns to drop).

Parameters

- **target_column** (*str, optional*) – Name of the target column in the dataset
- **id_column** (*str, optional*) – Name of the id column in the dataset that does not have any signal and will be ignored for computation
- **fold_column** (*str, optional*) – Name of the fold column used that should be used to compute the various folds in the dataset
- **weight_column** (*str, optional*) – Name of the weight column used to assign non-equal importance weights to the various rows in the dataset
- **filename_column** (*str, optional*) – Name of the filename column in the dataset for an image-based usecase
- **time_column** (*str, optional*) – Name of the time column in the dataset for a time-series usecase
- **group_columns** (*str, optional*) – Name of the target column in the dataset for a timeseries usecase
- **apriori_columns** (*str, optional*) – Name of the target column in the dataset for a timeseries usecase
- **drop_list** (*list(str), optional*) – Names of all the columns that should be dropped from the dataset while training the usecase

class previsionio.usecase_config.**Feature**

Types of feature engineering that can be applied to a dataset with Prevision.io. The `Full` member is a shortcut to get all available feature engineering modules at once. To just drop a feature engineering module from a list of modules, use:

```
Feature.drop(Feature.xxx)
```

Counts = 'Counter'
Value type counting

DateTime = 'Date'
Date transformation

Frequency = 'freq'
Frequency encoding

Full = ['Counter', 'Date', 'freq', 'text_tfidf', 'text_word2vec', 'text_embedding', 't
Full feature engineering

KMeans = 'kmean'
K-Means clustering

PCA = 'pca'
Principal component analysis

PolynomialFeatures = 'poly'
Polynomial feature

TargetEncoding = 'tenc'
Target encoding

TextEmbedding = 'text_embedding'
Sentence embedding

TextTfidf = 'text_tfidf'
Statistical analysis

TextWord2vect = 'text_word2vec'
Word embedding

class previsionio.usecase_config.**LiteModel**
Types of lite models that can be trained with Prevision.io. The `Full` member is a shortcut to get all available models at once. To just drop a single model from a list of models, use:

```
Model.drop(Model.xxx)
```

ExtraTrees = 'ET'
ExtraTrees

Full = ['LGB', 'XGB', 'NN', 'ET', 'LR', 'RF', 'NBC']
Evaluate all models

LightGBM = 'LGB'
LightGBM

LinReg = 'LR'
Linear Regression

NaiveBayesClassifier = 'NBC'
Random Forest

NeuralNet = 'NN'
NeuralNet

RandomForest = 'RF'
Random Forest

XGBoost = 'XGB'
XGBoost

class previsionio.usecase_config.**Model**
Types of normal models that can be trained with Prevision.io. The `Full` member is a shortcut to get all available models at once. To just drop a single model from a list of models, use:

```
LiteModel.drop(LiteModel.xxx)
```

ExtraTrees = 'ET'
ExtraTrees

Full = ['LGB', 'XGB', 'NN', 'ET', 'LR', 'RF']
Evaluate all models

LightGBM = 'LGB'
LightGBM

LinReg = 'LR'
Linear Regression

NeuralNet = 'NN'
NeuralNet

RandomForest = 'RF'
Random Forest

XGBoost = 'XGB'
XGBoost

class previsionio.usecase_config.**ParamList**
A list of params to be passed to a usecase.

class previsionio.usecase_config.**Profile**
Training profile type.

Advanced = 'advanced'
Slowest profile, for maximal optimization

Normal = 'normal'
Normal profile, best balance

Quick = 'quick'
Quickest profile, lowest predictive performance

class previsionio.usecase_config.**SimpleModel**
Types of simple models that can be trained with Prevision.io. The `Full` member is a shortcut to get all available simple models at once. To just drop a single model from a list of simple models, use:

```
SimpleModel.drop(SimpleModel.xxx)
```

DecisionTree = 'DT'
DecisionTree

Full = ['DT', 'LR']
Evaluate all simple models

LinReg = 'LR'
Linear Regression

class previsionio.usecase_config.**TrainingConfig**(*profile='normal', normal_models=['LGB', 'XGB', 'NN', 'ET', 'LR', 'RF'], lite_models=['LGB', 'XGB', 'NN', 'ET', 'LR', 'RF', 'NBC'], simple_models=['DT', 'LR'], features=['Counter', 'Date', 'freq', 'text_tfidf', 'text_word2vec', 'text_embedding', 'tenc', 'poly', 'pca', 'kmean'], with_blend=False, fe_selected_list=[]*)
Training configuration that holds the relevant data for a usecase description: the wanted feature engineering, the selected models, the training speed...

Parameters

- **profile** (*str*) – Type of training profile to use:
  - "quick": this profile runs very fast but has a lower performance (it is recommended for early trials)

  - ”advanced”: this profile runs slower but has increased performance (it is usually for optimization steps at the end of your project)
  - the “normal” profile is something in-between to help you investigate an interesting result
- **models** (*list(str), optional*) – Names of the (normal) models to use in the usecase (among: “LR”, “RF”, “ET”, “XGB”, “LGB” and “NN”)
- **simple_models** (*list(str), optional*) – Names of the (normal) models to use in the usecase (among: “LR” and “DT”)
- **features** (*list(str), optional*) – Names of the feature engineering modules to use (among: “Counter”, “Date”, “freq”, “text_tfidf”, “text_word2vec”, “text_embedding”, “tenc”, “ee”, “poly”, “pca” and “kmean”)
- **with_blend** (*bool, optional*) – If true, Prevision.io’s pipeline will add “blend” models at the end of the training by cherry-picking already trained models and fine-tuning hyperparameters (usually gives even better performance)
- **fe_selected_list** (*list(str), optional*) – Override for the features list, to restrict it only this list

class previsionio.usecase_config.**TypeProblem**
Type of supervised problems available with Prevision.io.

Classification = 'classification'
Classification

Clustering = 'clustering'
Clustering

MultiClassification = 'multiclassification'
Multi Classification

Regression = 'regression'
Regression

3.7 Model

class previsionio.model.**Model**(*_id*, *uc_id*, *uc_version*, *name=None*, ***other_params*)
Bases: *previsionio.api_resource.ApiResource*

A Model object is generated by Prevision AutoML plateform when you launch a use case. All models generated by Prevision.io are deployable in our Store

With this Model class, you can select the model with the optimal hyperparameters that responds to your buisiness requirements, then you can deploy it as a real-time/batch endpoint that can be used for a web Service.

Parameters

- **_id** (*str*) – Unique id of the model
- **uc_id** (*str*) – Unique id of the usecase of the model
- **uc_version** (*str, int*) – Version of the usecase of the model (either an integer for a specific version, or “last”)
- **name** (*str, optional*) – Name of the model (default: None)

chart()
Return chart analysis information for a model.

Returns Chart analysis results

Return type dict

Raises `PrevisionException` – Any error while fetching data from the platform or parsing the result

cross_validation
Get model's cross validation dataframe.

Returns Cross-validation dataframe

Return type `pd.Dataframe`

deploy() → previsionio.deployed_model.DeployedModel
(Not Implemented yet) Deploy the model as a REST API app.

Keyword Arguments {enum} -- it can be 'model', 'notebook', 'shiny', 'dash' or 'node' application (*app_type*) –

Returns Path of the deployed application

Return type str

feature_importance
Return a dataframe of feature importances for the given model features, with their corresponding scores (sorted by descending feature importance scores).

Returns Dataframe of feature importances

Return type `pd.DataFrame`

Raises `PrevisionException` – Any error while fetching data from the platform or parsing the result

hyperparameters
Return the hyperparameters of a model.

Returns Hyperparameters of the model

Return type dict

predict(*df*, *confidence=False*) → pandas.core.frame.DataFrame
Make a prediction in a Scikit-learn blocking style.

> **Warning:** For large dataframes and complex (blend) models, this can be slow (up to 1-2 hours). Prefer using this for simple models and small dataframes or use option `use_best_single = True`.

Parameters

- **df** (`pd.DataFrame`) – A `pandas` dataframe containing the testing data
- **confidence** (*`bool, optional`*) – Whether to predict with confidence estimator (default: `False`)

Returns Prediction results dataframe

Return type `pd.DataFrame`

predict_from_dataset(*dataset*, *confidence=False*, *dataset_folder=None*) → pandas.core.frame.DataFrame
Make a prediction for a dataset stored in the current active [client] workspace (using the current SDK dataset object).

Parameters

- **dataset** (*Dataset*) – Dataset resource to make a prediction for
- **confidence** (*bool, optional*) – Whether to predict with confidence values (default: `False`)
- **dataset_folder** (*Dataset*, None) – Matching folder dataset resource for the prediction, if necessary

Returns Prediction results dataframe

Return type `pd.DataFrame`

predict_from_dataset_name(*dataset_name*, *confidence=False*) → pandas.core.frame.DataFrame

Make a prediction for a dataset stored in the current active [client] workspace (referenced by name).

Parameters

- **dataset_name** (*str*) – Name of the dataset to make a prediction for (if there is more than one dataset having the given name, the first one will be used)
- **confidence** (*bool, optional*) – Whether to predict with confidence values (default: `False`)

Returns Prediction results dataframe

Return type `pd.DataFrame`

predict_single(*confidence=False*, *explain=False*, ***predict_data*)

Make a prediction for a single instance. Use *predict_from_dataset_name()* or predict methods to predict multiple instances at the same time (it's faster).

Parameters

- **confidence** (*bool, optional*) – Whether to predict with confidence values (default: `False`)
- **explain** (*bool, optional*) – Whether to explain prediction (default: `False`)
- ****predict_data** – Features names and values (without target feature) - missing feature keys will be replaced by nans

---

Note: You can set both `confidence` and `explain` to true.

Returns

Dictionary containing the prediction result

Note: The prediction format depends on the problem type (regression, classification, etc...)

Return type dict

wait_for_prediction(*predict_id*)

Wait for a specific prediction to finish.

Parameters **predict_id** (*str*) – Unique id of the prediction to wait for

class previsionio.model.**ClassificationModel**(*_id*, *uc_id*, *name=None*, ***other_params*)
Bases: *previsionio.model.Model*

A model object for a (binary) classification usecase, i.e. a usecase where the target is categorical with exactly 2 modalities.

Parameters

- **_id** (*str*) – Unique id of the model
- **uc_id** (*str*) – Unique id of the usecase of the model
- **uc_version** (*str, int*) – Version of the usecase of the model (either an integer for a specific version, or "last")
- **name** (*str, optional*) – Name of the model (default: None)

get_dynamic_performances(*threshold=0.5*)
Get model performance for the given threshold.

Parameters **threshold** (*float, optional*) – Threshold to check the model's performance for (default: 0.5)

Returns

Model classification performance dict with the following keys:

- confusion_matrix
- accuracy
- precision
- recall
- f1_score

Return type dict

Raises PrevisionException – Any error while fetching data from the platform or parsing the result

optimal_threshold
Get the value of threshold probability that optimizes the F1 Score.

Returns Optimal value of the threshold (if it not a classification problem it returns None)

Return type float

Raises PrevisionException – Any error while fetching data from the platform or parsing the result

predict_proba(*df*, *confidence=False*)
Make a prediction in a Scikit-learn blocking style and return probabilities.

> **Warning:** For large dataframes and complex (blend) models, this can be slow (up to 1-2 hours). Prefer using this for simple models and small dataframes or use option use_best_single = True.

Parameters

- **df** (pd.DataFrame) – A pandas dataframe containing the testing data
- **confidence** (*bool, optional*) – Whether to predict with confidence estimator (default: False)

Returns (Formatted) prediction results dataframe

Return type pd.DataFrame

predict_single(*confidence=False*, *explain=False*, ***predict_data*)
Make a prediction for a single instance.

Parameters

- **confidence** (*bool, optional*) – Whether to predict with confidence values (default: False)
- **explain** (*bool, optional*) – Whether to explain prediction (default: False)
- ****predict_data** – Features names and values (without target feature) - missing feature keys will be replaced by nans

Note: You can set both confidence and explain to true.

Returns Predictions probability, predictions class, predictions confidence and predictions explanation

Return type tuple

class previsionio.model.**MultiClassificationModel**(*_id*, *uc_id*, *uc_version*, *name=None*, ***other_params*)
Bases: *previsionio.model.Model*

A model object for a multi-classification usecase, i.e. a usecase where the target is categorical with strictly more than 2 modalities.

Parameters

- **_id** (*str*) – Unique id of the model
- **uc_id** (*str*) – Unique id of the usecase of the model
- **uc_version** (*str, int*) – Version of the usecase of the model (either an integer for a specific version, or "last")
- **name** (*str, optional*) – Name of the model (default: None)

predict(*df*, *confidence=False*)
Make a prediction in a Scikit-learn blocking style.

Warning: For large dataframes and complex (blend) models, this can be slow (up to 1-2 hours). Prefer using this for simple models and small dataframes or use option use_best_single = True.

Parameters

- **df** (pd.DataFrame) – A pandas dataframe containing the testing data
- **confidence** (*bool, optional*) – Whether to predict with confidence estimator (default: False)

Returns (Formatted) prediction results dataframe

Return type pd.DataFrame

predict_proba(*df*, *confidence=False*)
Make a prediction in a Scikit-learn blocking style and return probabilities.

> **Warning:** For large dataframes and complex (blend) models, this can be slow (up to 1-2 hours). Prefer using this for simple models and small dataframes or use option `use_best_single = True`.

Parameters

- **df** (`pd.DataFrame`) – A `pandas` dataframe containing the testing data
- **confidence** (*bool, optional*) – Whether to predict with confidence estimator (default: `False`)

Returns (Formatted) prediction results dataframe

Return type `pd.DataFrame`

class previsionio.model.**RegressionModel**(*_id*, *uc_id*, *uc_version*, *name=None*, ***other_params*)
Bases: *previsionio.model.Model*

A model object for a regression usecase, i.e. a usecase where the target is numerical.

Parameters

- **_id** (*str*) – Unique id of the model
- **uc_id** (*str*) – Unique id of the usecase of the model
- **uc_version** (*str, int*) – Version of the usecase of the model (either an integer for a specific version, or "last")
- **name** (*str, optional*) – Name of the model (default: `None`)

predict(*df*, *confidence=False*)
Make a prediction in a Scikit-learn blocking style.

> **Warning:** For large dataframes and complex (blend) models, this can be slow (up to 1-2 hours). Prefer using this for simple models and small dataframes or use option `use_best_single = True`.

Parameters

- **df** (`pd.DataFrame`) – A `pandas` dataframe containing the testing data
- **confidence** (*bool, optional*) – Whether to predict with confidence estimator (default: `False`)

Returns Prediction results dataframe

Return type `pd.DataFrame`

3.8 Metrics

The metric of a usecase is the function that will be used to assess for the efficiency of its models. The metrics you can choose depends on the type of usecase you are training.

class previsionio.metrics.**Classification**
Metrics for classification projects Available metrics in prevision:

auc, log_loss, error_rate_binary

AUC = 'auc'
Area Under ROC Curve

error_rate = 'error_rate_binary'
Error rate

log_loss = 'log_loss'
Logarithmic Loss

class previsionio.metrics.**Clustering**
Metrics for clustering projects

calinski_harabaz = 'calinski_harabaz'
Clustering calinski_harabaz metric

silhouette = 'silhouette'
Clustering silhouette metric

class previsionio.metrics.**MultiClassification**
Metrics for multiclassification projects

error_rate = 'error_rate_multi'
Multi-class Error rate

log_loss = 'log_loss'
Logarithmic Loss

class previsionio.metrics.**Regression**
Metrics for regression projects Available metrics in prevision:

rmse, mape, rmsle, mse, mae

MAE = 'mae'
Mean Average Error

MAPE = 'mape'
Mean Average Percentage Error

MSE = 'mse'
Mean squared Error

RMSE = 'rmse'
Root Mean Squared Error

RMSLE = 'rmsle'
Root Mean Squared Logarithmic Error

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X