OpenML Cheat Sheet (Python)

config

Find your API key (required for uploads):

- `www.openml.org` > Your profile > API Authentication

Main OpenML servers:

- Public: `https://www.openml.org/api/v1` (default)
- Test: `https://test.openml.org/api/v1`

datasets

`list_datasets(offset=None, size=None, tag=None)`

- returns ID -> dataset dict mapping
- `offset` and `size` for paging results
- `tag` to filter datasets (e.g. 'uci')

`get_dataset(dataset_id)`

- returns **OpenMLDataset** object
- automatically downloads and caches the data itself

`OpenMLDataset`

`.features`: list of features and their properties
`.qualities`: list of all dataset properties

`.get_data`(target,return_attribute_names=False,return_categorical_indicator=Fa
returns numpy arrays (or sparse matrices) with features and targets, optionally with attribute names and which are categorical
`.retrieve_class_labels(target_name='class')`: return all class labels for the given target attribute

Upload new datasets

- Create a new OpenML dataset with all relevant information
- Call **`.publish()`** to upload
- Note: use test server for testing

tasks

`list_tasks(task_type_id=None, offset=None, size=None, tag=None)`

- returns ID -> task dict mapping (task IDs do not match dataset IDs)
- `offset` and `size` for paging results, `tag` to filter tags
- `task_type_id`: 1=Classification, 2=Regression,...

`OpenMLTask`
`.get_dataset()`: downloads associated dataset
`.download_split()`: downloads train/test splits

```
# General imports
from openml import datasets, tasks, runs, flows, config
import os, pandas, sklearn, arff, pprint, numpy
```

Set server, API key and cache directory (default: `~/.openml/cache`)

```
config.apikey = 'qxlfpbe...ebairtd'
config.server = 'https://...'
config.set_cache_directory(os.path.expanduser('~/mycache'))
```

Or, create a config file called `~/.openml/config` and add these lines:

```
server=https://www.openml.org/api/v1
apikey=qxlfpbeaudtprb23985hcqlfoebairtd
cachedir=/homedir/.openml/cache
```

```
dlist = datasets.list_datasets(size=100)
pandas.DataFrame.from_dict(dlist, orient='index')[
['name','NumberOfInstances', 'NumberOfFeatures']][:3]
```

	name	NumberOfInstances	NumberOfFeatures
2	anneal	898	39
3	kr-vs-kp	3196	37
4	labor	57	17

```
odata = datasets.get_dataset(1471)
print(odata.name, "Target: "+ odata.default_target_attribute
      odata.description[260:308], sep='\n')
```

```
eeg-eye-state
Target: Class
All data is from one continuous EEG measurement
```

```
X, y, attribute_names = odata.get_data(
    target=odata.default_target_attribute,
    return_attribute_names=True)
pandas.DataFrame(X, columns=attribute_names)[:2]
```

	V1	V2	V3	V4	V5
0	4329.229980	4009.229980	4289.229980	4148.209961	43
1	4324.620117	4004.620117	4293.850098	4148.720215	43

```
# Train a scikit-learn classifier on this data
sklearn.linear_model.LinearRegression().fit(X, y)
```

```
LinearRegression(copy_X=True, fit_intercept=Tru
e, n_jobs=1, normalize=False)
```

```
mydata = datasets.OpenMLDataset(data_file='test.arff',
name='t', description='t', version='1', format='ARFF',
licence='Public', visibility='public',
default_target_attribute='class')
response = mydata.publish()
print("New dataset ID: " + str(response.dataset_id))
```

```
tlist = tasks.list_tasks(size=100)
pandas.DataFrame.from_dict(tlist, orient='index')[
['name','task_type','estimation_procedure']][:3]
```

	name	task_type	estimation_procedure
2	anneal	Supervised Classification	10-fold Crossvalidation
3	kr-vs-kp	Supervised Classification	10-fold Crossvalidation
4	labor	Supervised Classification	10-fold Crossvalidation

Upload new tasks
Under development

`get_task(task_id)`

- returns **OpenMLTask** object
  - includes estimation procedure, target name, cost matrix,...
- automatically caches the task description

```
task = tasks.get_task(14951)
pprint.pprint(task.estimation_procedure)
```

```
{'data_splits_url': 'https://www.openml.org/api
_splits/get/14951/Task_14951_splits.arff',
 'parameters': {'number_folds': '10',
                'number_repeats': '1',
                'percentage': '',
                'stratified_sampling': 'true'},
 'type': 'crossvalidation'}
```

flows

`list_flows(offset=None, size=None, tag=None)`

- returns ID -> flow dict mapping
- `offset` and `size` for paging results, `tag` to filter tags

```
flist = flows.list_flows(size=200)
pandas.DataFrame.from_dict(flist, orient='index')[
    ['name','version','external_version']][100:102]
```

	name	version	external_version
101	moa.WEKAClassifier_REPTree	1	Moa_2014.03_1.0
102	weka.REPTree	2	Weka_3.7.5_9378

`sklearn_to_flow(sklearn_estimator)`

- converts a scikit-learn estimator or pipeline to an OpenML Flow

`publish()`

- uploads the flow to the server. Returns the flow ID

```
lr = sklearn.linear_model.LinearRegression().fit(X, y)
flow = flows.sklearn_to_flow(lr)

pipe = sklearn.pipeline.Pipeline(steps=[
    ('Imputer', sklearn.preprocessing.Normalizer()),
    ('Classifier', sklearn.linear_model.LinearRegression())]
flow2 = flows.sklearn_to_flow(pipe)
# flows.publish(flow)
```

runs

`list_runs(offset=None, size=None, tag=None, id=None, task=None, flow=None, uploader=None, display_errors=False)`

- returns ID -> run dict mapping
- `offset` and `size` for paging results, `tag` to filter tags
- `id`: list of run IDs to filter on, e.g. [1,2,3]
- `task`: list of task IDs to filter on, e.g. [1,2,3]
- `flow`: list of flow IDs to filter on, e.g. [1,2,3]
- `uploader`: list of uploader IDs to filter on, e.g. [1,2,3]
- `display_errors`: whether to return failed runs

```
rlist = runs.list_runs(task=[14951],size=100)
pandas.DataFrame.from_dict(rlist, orient='index')[1:3]
```

	task_id	setup_id	flow_id	run_id	uploader
544514	14951	5540	3404	544514	2
595116	14951	6436	4074	595116	2

```
rlist = runs.list_runs(id=[1,2,3])
pandas.DataFrame.from_dict(rlist, orient='index')[1:3]
```

	task_id	setup_id	flow_id	run_id	uploader
2	72	16	75	2	1
3	95	8	63	3	1

`get_run(run_id)`

- returns **OpenMLRun** object
  - includes the exact task, exact flow, and all evaluations
- automatically caches the run description

```
rlist = runs.list_runs(task=[14951],size=100)
scores = []
for id, _ in rlist.items():
    run = runs.get_run(id)
    scores.append({"flow":run.flow_name,
                   "score":run.evaluations['area_under_roc_c
pandas.DataFrame.from_dict(scores)[5:8]
```

	flow	score
5	sklearn.neighbors.classification.KNeighborsCla...	0.980389
6	mlr.classif.rpart(11)	0.693537
7	sklearn.ensemble.forest.RandomForestClassifier...	0.966248

OpenMLRun
.uploader_name: full name of the run author
.flow_name: full name of the flow
.parameter_settings: hyperparameters of the flow
.evaluations: key-value pairs of metric and score
.fold_evaluations: dict of per-fold evaluations

`run_flow_on_task(task, flow)`

- Runs the flow on the task
- Trains and tests the flow of all train/test splits defined by the task
- Returns an **OpenMLRun** model with all information

`publish()`

- Publishes the run on OpenML

```
task = tasks.get_task(14951)
clf = sklearn.linear_model.LogisticRegression()
flow = flows.sklearn_to_flow(clf)
run = runs.run_flow_on_task(task, flow)
run.fold_evaluations['predictive_accuracy'][0]
```

```
{0: 0.63618157543391185,
 1: 0.65153538050734316,
 2: 0.63284379172229643,
 3: 0.63618157543391185,
 4: 0.6435246995994659,
 5: 0.64552736982643522,
 6: 0.64085447263017359,
 7: 0.63618157543391185,
 8: 0.63484646194926564,
 9: 0.64753004005340453}
```