OpenML in Python

OpenML is an online collaboration platform for machine learning:

- Share/reuse machine learning datasets, algorithms, models, experiments
- Well documented/annotated datasets, uniform access
- APIs in Java, R, Python*,... to download/upload everything
- Better reproducibility of experiments, reuse of machine learning models
- Works well with machine learning libraries such as scikit-learn
- Large scale benchmarking, compare to state of the art

```
[ ]: # Install OpenML (developer version)
     pip install git+https://github.com/renatopp/liac-arff@master
     pip install git+https://github.com/openml/openml-python.git@develop

     # Import and set key
     import openml as oml
     oml.config.apikey = 'YOURKEY'
```

```
<IPython.core.display.HTML object>
```

Authentication

- Create an OpenML account (free) on http://www.openml.org.
- After logging in, open your account page (avatar on the top right)
- Open 'Account Settings', then 'API authentication' to find your API key.

There are two ways to authenticate:

- Create a plain text file `~/.openml/config` with the line 'apikey=MYKEY', replacing MYKEY with your API key.
- Run the code below, replacing 'MYKEY' with your API key.

```
[1]: # Uncomment and run this to authenticate. Don't share your API key!
     oml.config.apikey = os.environ.get('OPENMLKEY','MYKEY')
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call la

<ipython-input-1-06722a42f500> in <module>()
  1 # Uncomment and run this to authenticate. Don't share your API key!
----> 2 oml.config.apikey = os.environ.get('OPENMLKEY','MYKEY')


NameError: name 'os' is not defined
```

Data sets

We can list, select, and download all OpenML datasets

List datasets

```
[4]: datalist = oml.datasets.list_datasets() # Returns a dict
     datalist = pd.DataFrame.from_dict(datalist, orient='index') # Create a Data
     print("First 10 of %s datasets..." % len(datalist))
     datalist[:10][['did','name','NumberOfInstances',
                    'NumberOfFeatures','NumberOfClasses']]
```

```
First 10 of 19528 datasets...
```

	did	name	NumberOfInstances	NumberOfFeatures	NumberOfClasses
1	1	anneal	898.0	39.0	6.0
2	2	anneal	898.0	39.0	6.0
3	3	kr-vs-kp	3196.0	37.0	2.0
4	4	labor	57.0	17.0	2.0
5	5	arrhythmia	452.0	280.0	16.0
6	6	letter	20000.0	17.0	26.0
7	7	audiology	226.0	70.0	24.0
8	8	liver-disorders	345.0	7.0	-1.0
9	9	autos	205.0	26.0	7.0
10	10	lymph	148.0	19.0	4.0

There are many properties that we can query

```
[5]: list(datalist)
     datalist = datalist[['did','name','NumberOfInstances',
                    'NumberOfFeatures','NumberOfClasses']]
```

```
['NumberOfInstances',
 'NumberOfClasses',
 'MajorityClassSize',
 'NumberOfMissingValues',
 'did',
 'NumberOfSymbolicFeatures',
 'NumberOfFeatures',
 'format',
 'name',
 'NumberOfInstancesWithMissingValues',
 'status',
 'MinorityClassSize',
 'NumberOfNumericFeatures',
 'MaxNominalAttDistinctValues']
```

and we can filter or sort on all of them

```
[6]: datalist[datalist.NumberOfInstances>10000
             ].sort(['NumberOfInstances'])[:20]
```

	did	name	NumberOfInstances	\
23515	23515	sulfur	10081.0	

```
372      372                        internet_usage            10108.0
981      981                    kdd_internet_usage            10108.0
1536    1536                          volcanoes-b6            10130.0
4562    4562                         InternetUsage            10168.0
1531    1531                          volcanoes-b1            10176.0
1534    1534                          volcanoes-b4            10190.0
1459    1459                artificial-characters            10218.0
1478    1478                                   har            10299.0
1533    1533                          volcanoes-b3            10386.0
1532    1532                          volcanoes-b2            10668.0
1053    1053                                   jm1            10885.0
1414    1414  Kaggle_bike_sharing_demand_challange            10886.0
1044    1044                        eye_movements            10936.0
32        32                            pendigits            10992.0
1019    1019                            pendigits            10992.0
4534    4534                      PhishingWebsites            11055.0
399      399                             ohscal.wc            11162.0
310      310                          mammography            11183.0
1568    1568                              nursery            12958.0

       NumberOfFeatures  NumberOfClasses
23515               7.0             -1.0
372                72.0             46.0
981                69.0              2.0
1536                4.0              5.0
4562               72.0             -1.0
1531                4.0              5.0
1534                4.0              5.0
1459                8.0             10.0
1478              562.0              6.0
1533                4.0              5.0
1532                4.0              5.0
1053               22.0              2.0
1414               12.0             -1.0
1044               28.0              3.0
32                 17.0             10.0
1019               17.0              2.0
4534               31.0              2.0
399             11466.0             10.0
310                 7.0              2.0
1568                9.0              4.0
```

or find specific ones

```
[7]: datalist.query('name == "eeg-eye-state"')

       did           name  NumberOfInstances  NumberOfFeatures  \
1471  1471  eeg-eye-state            14980.0              15.0

      NumberOfClasses
1471              2.0
```

```
[8]: datalist.query('NumberOfClasses > 50')
```

```
       did                          name  NumberOfInstances  NumberOfFeatures  \
1491  1491   one-hundred-plants-margin             1600.0              65.0
1492  1492    one-hundred-plants-shape             1600.0              65.0
1493  1493  one-hundred-plants-texture             1599.0              65.0
4546  4546                      Plants            44940.0              16.0
4552  4552          BachChoralHarmony             5665.0              17.0

      NumberOfClasses
1491            100.0
1492            100.0
1493            100.0
4546             57.0
4552            102.0
```

Download a specific dataset. This is done based on the dataset ID (called 'did').

```
[9]: dataset = oml.datasets.get_dataset(1471)

     print("This is dataset '%s', the target feature is '%s'" %
           (dataset.name, dataset.default_target_attribute))
     print("URL: %s" % dataset.url)
     print(dataset.description[:500])
```

```
This is dataset 'eeg-eye-state', the target feature is 'Class'
URL: https://www.openml.org/data/download/1587924/eeg-eye-state.ARFF
**Author**: Oliver Roesler, it12148'@'lehre.dhbw-stuttgart.de
**Source**: [UCI](https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State), Baden-
**Please cite**:

All data is from one continuous EEG measurement with the Emotiv EEG Neuroheadset
```

Convert the data to a DataFrame for easier processing/plotting

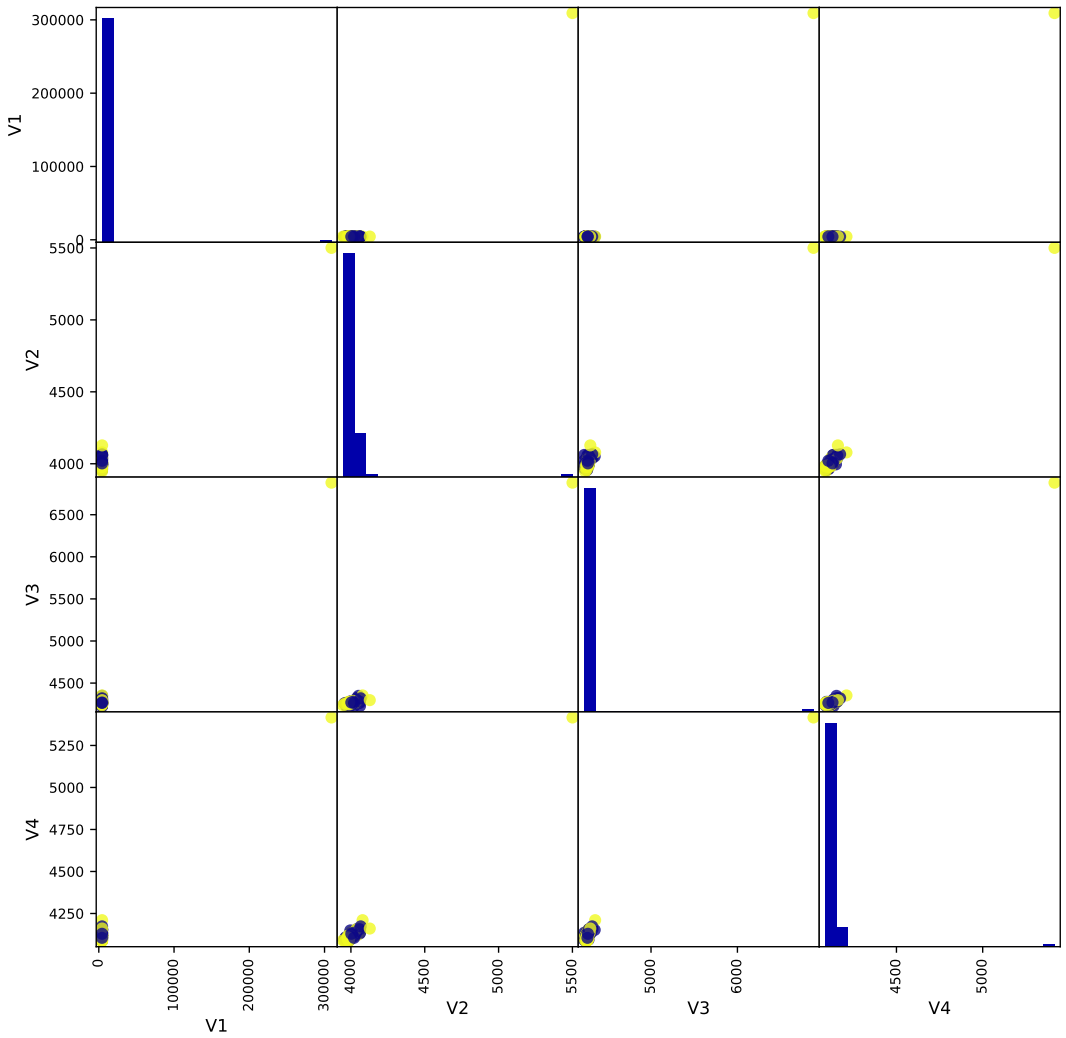
```
[10]: X, y, attribute_names = dataset.get_data(
          target=dataset.default_target_attribute,
          return_attribute_names=True)
      eeg = pd.DataFrame(X, columns=attribute_names)
      eeg['class'] = y
      print(eeg[:10])
```

```
        V1       V2       V3       V4  ...       V12      V13      V14  class
0  4329.23  4009.23  4289.23  4148.21  ...   4280.51  4635.90  4393.85      0
1  4324.62  4004.62  4293.85  4148.72  ...   4279.49  4632.82  4384.10      0
2  4327.69  4006.67  4295.38  4156.41  ...   4282.05  4628.72  4389.23      0
3  4328.72  4011.79  4296.41  4155.90  ...   4287.69  4632.31  4396.41      0
4  4326.15  4011.79  4292.31  4151.28  ...   4288.21  4632.82  4398.46      0
5  4321.03  4004.62  4284.10  4153.33  ...   4281.03  4628.21  4389.74      0
6  4319.49  4001.03  4280.51  4151.79  ...   4269.74  4625.13  4378.46      0
7  4325.64  4006.67  4278.46  4143.08  ...   4266.67  4622.05  4380.51      0
```

```
8   4326.15   4010.77   4276.41   4139.49   ...    4273.85   4627.18   4389.74       0
9   4326.15   4011.28   4276.92   4142.05   ...    4277.95   4637.44   4393.33       0

[10 rows x 15 columns]
```

```
[11]: eegs = eeg.sample(n=1000)
      _ = pd.scatter_matrix(eegs.iloc[:100,:4], c=eegs[:100]['class'], figsize=(
                            marker='o', hist_kwds={'bins': 20},
                            alpha=.8, cmap='plasma')
```



Train models

Train a scikit-learn model on the data manually

```
[12]: from sklearn import neighbors

      dataset = oml.datasets.get_dataset(1471)
      X, y = dataset.get_data(target=dataset.default_target_attribute)
```

```
        clf = neighbors.KNeighborsClassifier(n_neighbors=1)
        clf.fit(X, y)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Note: This is the same as downloading the dataset manually from OpenML and doing the following

(This is just part of the complexity that OpenML hides)

```
[13]: # Download the data from https://www.openml.org/d/1471
      # We'll assume it is stored in the current directory as eeg-eye-state.arff
      import arff
      import io

      # You also need to look up the name and datatype of the target attribute(s
      arff_filename = 'eeg-eye-state.arff'
      arff_target = ['Class']
      arff_target_dtype = int

      # Read data and extract properties
      fh = io.open(arff_filename, encoding='utf8')
      data = arff.ArffDecoder().decode(fh, encode_nominal=True, return_type=arff
      categorical = [False if type(type_) != list else True
                     for name, type_ in data['attributes']]
      attribute_names = [name for name, type_ in data['attributes']]
      targets = np.array([True if column in arff_target else False for column in

      # Encode the data
      if isinstance(data['data'], tuple):
          X = data['data']
          X_shape = (max(X[1]) + 1, max(X[2]) + 1)
          X = scipy.sparse.coo_matrix(
              (X[0], (X[1], X[2])), shape=X_shape, dtype=np.float32)
          X = X.tocsr()
      elif isinstance(data['data'], list):
          X = np.array(data['data'], dtype=np.float32)

      # Get the features and targets
      x = X[:, ~targets]
      y = X[:, targets].astype(arff_target_dtype)

      # Build the model
      clf = neighbors.KNeighborsClassifier(n_neighbors=1)
      clf.fit(X, y)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

You can also ask which features are categorical to do your own encoding

```
[14]: from sklearn import preprocessing
      dataset = oml.datasets.get_dataset(10)
      X, y, categorical = dataset.get_data(
          target=dataset.default_target_attribute,
          return_categorical_indicator=True)
      print("Categorical features: %s" % categorical)
      enc = preprocessing.OneHotEncoder(categorical_features=categorical)
      X = enc.fit_transform(X)
      clf.fit(X, y)
```

```
Categorical features: [True, True, True, True, True, True, True, True, False, Fa
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Tasks

To run benchmarks consistently (also across studies and tools), OpenML offers Tasks, which include specific train-test splits and other information to define a scientific task. Tasks are typically created via the website by the dataset provider.

Listing tasks

```
[15]: task_list = oml.tasks.list_tasks(size=5000) # Get first 5000 tasks

      mytasks = pd.DataFrame(task_list).transpose()
      print("First 5 of %s tasks:" % len(mytasks))
      print(mytasks.columns)
```

```
First 5 of 4998 tasks:
Index(['MajorityClassSize', 'MaxNominalAttDistinctValues', 'MinorityClassSize',
       'NumberOfClasses', 'NumberOfFeatures', 'NumberOfInstances',
       'NumberOfInstancesWithMissingValues', 'NumberOfMissingValues',
       'NumberOfNumericFeatures', 'NumberOfSymbolicFeatures', 'cost_matrix',
       'did', 'estimation_procedure', 'evaluation_measures', 'name',
       'number_samples', 'quality_measure', 'source_data',
       'source_data_labeled', 'status', 'target_feature',
       'target_feature_event', 'target_feature_left', 'target_feature_right',
       'target_value', 'task_type', 'tid', 'time_limit', 'ttid'],
      dtype='object')
```

```
[16]: mytasks = mytasks[['tid','did','name','task_type','estimation_procedure','
      print(mytasks.head())
```

```
  tid did        name                    task_type     estimation_procedure  \
1   1   1      anneal  Supervised Classification  10-fold Crossvalidation
```

```
2   2   2      anneal  Supervised Classification  10-fold Crossvalidation
3   3   3    kr-vs-kp  Supervised Classification  10-fold Crossvalidation
4   4   4       labor  Supervised Classification  10-fold Crossvalidation
5   5   5  arrhythmia  Supervised Classification  10-fold Crossvalidation

   evaluation_measures
1  predictive_accuracy
2  predictive_accuracy
3  predictive_accuracy
4  predictive_accuracy
5  predictive_accuracy
```

Search for the tasks you need

```
[17]: print(mytasks.query('name=="eeg-eye-state"'))
```

```
         tid   did           name                   task_type  \
9983    9983  1471  eeg-eye-state  Supervised Classification
14951  14951  1471  eeg-eye-state  Supervised Classification

          estimation_procedure  evaluation_measures
9983   10-fold Crossvalidation  predictive_accuracy
14951  10-fold Crossvalidation                  NaN
```

Download tasks

```
[18]: task = oml.tasks.get_task(14951)
      pprint(vars(task))
```

```
{'class_labels': ['1', '2'],
 'cost_matrix': None,
 'dataset_id': 1471,
 'estimation_parameters': {'number_folds': '10',
                           'number_repeats': '1',
                           'percentage': '',
                           'stratified_sampling': 'true'},
 'estimation_procedure': {'data_splits_url': 'https://www.openml.org/api_splits/
                          'parameters': {'number_folds': '10',
                                         'number_repeats': '1',
                                         'percentage': '',
                                         'stratified_sampling': 'true'},
                          'type': 'crossvalidation'},
 'evaluation_measure': None,
 'target_name': 'Class',
 'task_id': 14951,
 'task_type': 'Supervised Classification'}
```

Runs: Train models on tasks

We can run (many) scikit-learn algorithms on (many) OpenML tasks.

```
[19]: task = oml.tasks.get_task(14951)
      clf = neighbors.KNeighborsClassifier(n_neighbors=1)
      run = oml.runs.run_task(task, clf)
      run.model
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

Share the run on the OpenML server

```
[20]: myrun = run.publish()
      print("Uploaded to http://www.openml.org/r/" + str(myrun.run_id))
```

```
Uploaded to http://www.openml.org/r/2275738
```

It also works with pipelines

```
[21]: from sklearn import pipeline, ensemble, preprocessing
      from openml import tasks,runs, datasets
      task = tasks.get_task(59)
      pipe = pipeline.Pipeline(steps=[
                   ('Imputer', preprocessing.Imputer(strategy='median')),
                   ('OneHotEncoder', preprocessing.OneHotEncoder(sparse=False, ha
                   ('Classifier', ensemble.RandomForestClassifier())
                  ])
      run = runs.run_task(task, pipe)
      myrun = run.publish()
      print("Uploaded to http://www.openml.org/r/" + str(myrun.run_id))
```

```
Uploaded to http://www.openml.org/r/2275739
```

All together

Train any model on any OpenML dataset and upload to OpenML in a few lines of code

```
[22]: from sklearn.linear_model import LogisticRegression

      task = oml.tasks.get_task(145677)
      clf = LogisticRegression()
      run = oml.runs.run_task(task, clf)
      run.model
      myrun = run.publish()
      print("Uploaded to http://www.openml.org/r/" + str(myrun.run_id))
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)


Uploaded to http://www.openml.org/r/2275740
```

A Challenge

We'll see many machine learning algorithms in this course. Try to build the best possible models on several OpenML tasks, and compare your results with the rest of the class, and learn from them. Some tasks you could try (or browse openml.org):

- EEG eye state: data_id:1471, task_id:14951
- Volcanoes on Venus: data_id:1527, task_id:10103
- Walking activity: data_id:1509, task_id: 9945, 150k instances
- Covertype (Satellite): data_id:150, task_id: 218. 500k instances
- Higgs (Physics): data_id:23512, task_id:52950. 100k instances, missing values

Easy benchmarking:

```
[23]: import openml as oml
      from sklearn import neighbors

      for task_id in [14951,10103,9945]:
          task = oml.tasks.get_task(task_id)
          data = oml.datasets.get_dataset(task.dataset_id)
          clf = neighbors.KNeighborsClassifier(n_neighbors=5)
          run = oml.runs.run_task(task, clf)
          myrun = run.publish()
          print("kNN on %s: http://www.openml.org/r/%d" % (data.name, myrun.run_
```

```
kNN on eeg-eye-state: http://www.openml.org/r/2275741
kNN on volcanoes-a1: http://www.openml.org/r/2275742
kNN on walking-activity: http://www.openml.org/r/2275743
```

Other possibilities

OpenML's Python API is currently still under development. To be added soon:

- Support for uploading pipelines
- Organizing data sets, algorithms, and experiments into studies
- Downloading previous experiments, evaluations and models
- Uploading new datasets to OpenML
- Filters for listings (e.g. filter by author, tags, other properties)

All of this is already possible with the R and Java API.