OpenML in Python

OpenML is an online collaboration platform for machine learning:

- Find or share interesting, well-documented datasets
- Define research / modelling goals (tasks)
- Explore large amounts of machine learning algorithms, with APIs in Java, R, Python
- Log and share reproducible experiments, models, results
- Works seamlessly with scikit-learn and other libraries
- Large scale benchmarking, compare to state of the art

Authentication

- Create an OpenML account (free) on http://www.openml.org (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 'YOURKEY' with your API key.

It all starts with data

Explore thousands of datasets, or share your own

List datasets

```
import openml as oml
openml_list = oml.datasets.list_datasets()
```

```
First 10 of 2550 datasets...
```

Out[2]:

	did	name	NumberOfInstances	NumberOfFeatures	NumberOf
2	2	anneal	898.0	39.0	5.0
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	13.0
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	0.0
9	9	autos	205.0	26.0	6.0
10	10	lymph	148.0	19.0	4.0
11	11	balance-scale	625.0	5.0	3.0

Filter datasets

```
openml_list = oml.datasets.list_datasets(
    number_instances = '10000..20000',
    number_features = '10..20')
```

```
Found 11 datasets
```

Out[3]:

	did	name	NumberOfInstances	N
6	6	letter	20000	1
32	32	pendigits	10992	1
216	216	elevators	16599	1
846	846	elevators	16599	1
977	977	letter	20000	1
1019	1019	pendigits	10992	1
1120	1120	MagicTelescope	19020	1
1199	1199	BNG(echoMonths)	17496	1
1222	1222	letter-challenge-unlabeled.arff	20000	1
1414	1414	Kaggle_bike_sharing_demand_challange	10886	1
1471	1471	eeg-eye-state	14980	1

Search datasets by name

```
openml_list = oml.datasets.list_datasets(
    data_name = 'eeg-eye-state')
```

Out[4]:

	did	name	NumberOfInstances	NumberOfFeatures	NumberOfCl
1471	1471	eeg-eye-state	14980	15	2

Download datasets

This is done based on the dataset ID ('did').

```
dataset = oml.datasets.get_dataset(1471)
```

```
This is dataset 'eeg-eye-state', the target feature is 'Class'
URL: https://www.openml.org/data/v1/download/1587924/eeg-eye-state.arff
**Author**: Oliver Roesler
**Source**: [UCI](https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State),
Baden-Wuerttemberg, Cooperative State University (DHBW), Stuttgart, German
y
**Please cite**: [UCI](https://archive.ics.uci.edu/ml/citation_policy.htm
l)

All data is from one continuous EEG measurement with the Emotiv EEG Neuroh
eadset. The duration of the measurement was 117 seconds. The eye state was
detected via a camera during the EEG measurement and added later manually
to the file after
```

Get the actual data.

```
X, y, attribute_names = dataset.get_data(
    return_attribute_names=True)
eeg = pd.DataFrame(X, columns=attribute_names)
```

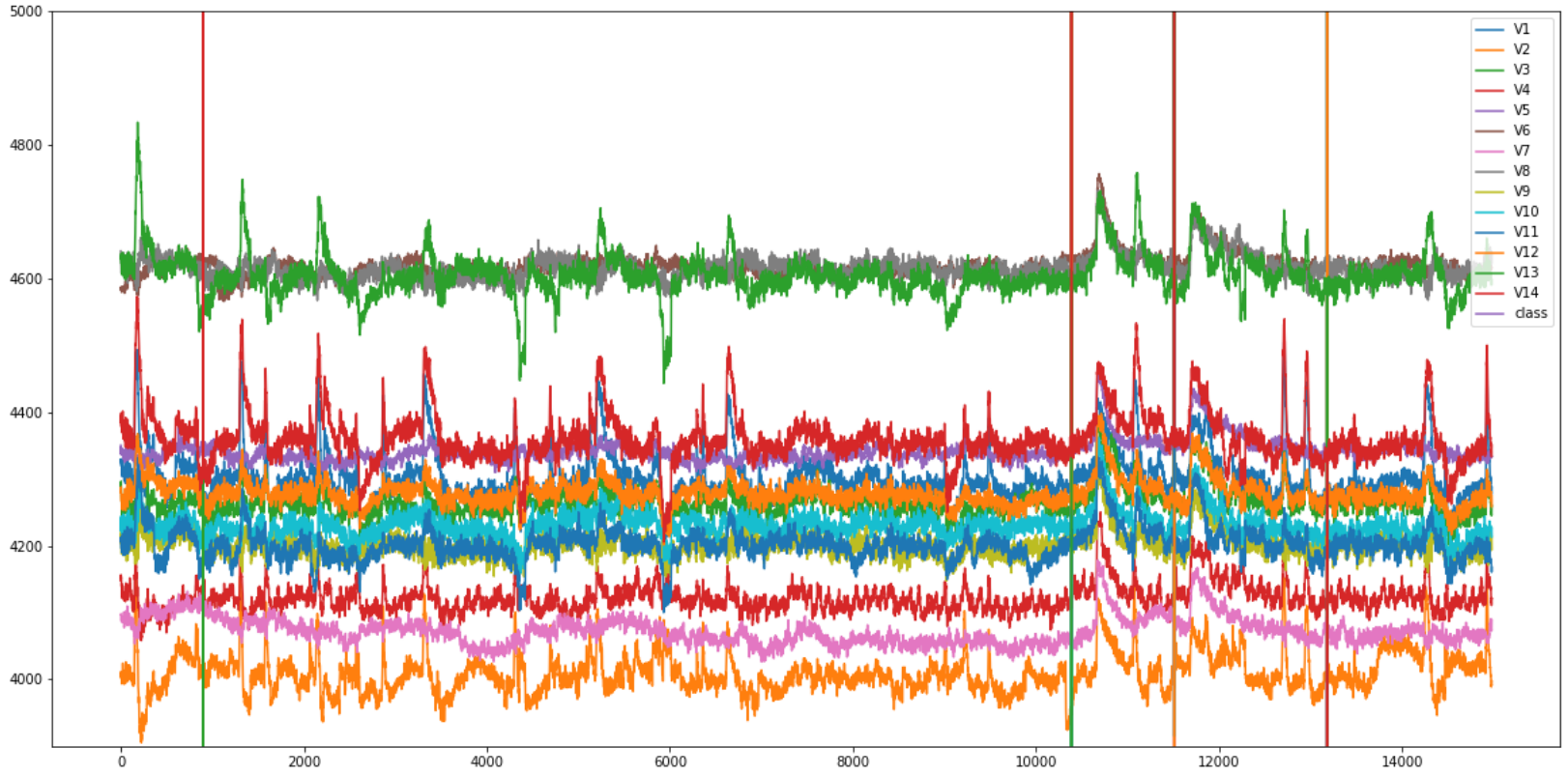
	V1	V2	V3	V4	V5	\
0	4329.229980	4009.229980	4289.229980	4148.209961	4350.259766	
1	4324.620117	4004.620117	4293.850098	4148.720215	4342.049805	
2	4327.689941	4006.669922	4295.379883	4156.410156	4336.919922	
3	4328.720215	4011.790039	4296.410156	4155.899902	4343.589844	
4	4326.149902	4011.790039	4292.310059	4151.279785	4347.689941	
5	4321.029785	4004.620117	4284.100098	4153.330078	4345.640137	
6	4319.490234	4001.030029	4280.509766	4151.790039	4343.589844	
7	4325.640137	4006.669922	4278.459961	4143.080078	4344.100098	
8	4326.149902	4010.770020	4276.410156	4139.490234	4345.129883	
9	4326.149902	4011.280029	4276.919922	4142.049805	4344.100098	

	V6	V7	V8	V9	V10	\
0	4586.149902	4096.919922	4641.029785	4222.049805	4238.459961	
1	4586.669922	4097.439941	4638.970215	4210.770020	4226.669922	
2	4583.589844	4096.919922	4630.259766	4207.689941	4222.049805	
3	4582.560059	4097.439941	4630.770020	4217.439941	4235.379883	
4	4586.669922	4095.899902	4627.689941	4210.770020	4244.100098	
5	4587.180176	4093.330078	4616.919922	4202.560059	4232.819824	
6	4584.620117	4089.739990	4615.899902	4212.310059	4226.669922	
7	4583.080078	4087.179932	4614.870117	4205.640137	4230.259766	
8	4584.100098	4091.280029	4608.209961	4187.689941	4229.740234	
9	4582.560059	4092.820068	4608.720215	4194.359863	4228.720215	

	V11	V12	V13	V14	class
0	4211.279785	4280.509766	4635.899902	4393.850098	0
1	4207.689941	4279.490234	4632.819824	4384.100098	0
2	4206.669922	4282.049805	4628.720215	4389.229980	0
3	4210.770020	4287.689941	4632.310059	4396.410156	0
4	4212.819824	4288.209961	4632.819824	4398.459961	0
5	4209.740234	4281.029785	4628.209961	4389.740234	0
6	4201.029785	4269.740234	4625.129883	4378.459961	0
7	4195.899902	4266.669922	4622.049805	4380.509766	0

Visualize data

```
eeg.plot()
```



Over 100 data qualities

Statistical and other properties of each dataset

```
dataset.qualities
```

Out[25]:

```
{'AutoCorrelation': 0.5034013605442177,
 'CfsSubsetEval_DecisionStumpAUC': 0.7924545850419331,
 'CfsSubsetEval_DecisionStumpErrRate': 0.23648648648648649,
 'CfsSubsetEval_DecisionStumpKappa': 0.5474401537655076,
 'CfsSubsetEval_NaiveBayesAUC': 0.7924545850419331,
 'CfsSubsetEval_NaiveBayesErrRate': 0.23648648648648649,
 'CfsSubsetEval_NaiveBayesKappa': 0.5474401537655076,
 'CfsSubsetEval_kNN1NAUC': 0.7924545850419331,
 'CfsSubsetEval_kNN1NErrRate': 0.23648648648648649,
 'CfsSubsetEval_kNN1NKappa': 0.5474401537655076,
 'ClassEntropy': 1.2276775019465804,
 'DecisionStumpAUC': 0.7715656536027917,
 'DecisionStumpErrRate': 0.24324324324324326,
 'DecisionStumpKappa': 0.5316455696202532,
 'Dimensionality': 0.12837837837837837,
 'EquivalentNumberOfAtts': 9.37680223405617,
 'J48.00001.AUC': 0.8035040133716935,
 'J48.00001.ErrRate': 0.24324324324324326,
 'J48.00001.Kappa': 0.55,
 'J48.0001.AUC': 0.8035040133716935,
 'J48.0001.ErrRate': 0.24324324324324326,
 'J48.0001.Kappa': 0.55,
 'J48.001.AUC': 0.8035040133716935,
 'J48.001.ErrRate': 0.24324324324324326,
 'J48.001.Kappa': 0.55,
 'MajorityClassPercentage': 54.729729729729726,
 'MajorityClassSize': 81.0,
 'MaxAttributeEntropy': 2.527125737973009,
 'MaxKurtosisOfNumericAtts': 29.749465128075876,
 'MaxMeansOfNumericAtts': 2.6013513513513518,
 'MaxMutualInformation': 0.40188387586188,
 'MaxNominalAttDistinctValues': 8.0,
 'MaxSkewnessOfNumericAtts': 5.442361694493849,
```

```
'MaxStdDevOfNumericAtts': 1.9050233089611373,
'MeanAttributeEntropy': 1.1174061851513224,
'MeanKurtosisOfNumericAtts': 9.883463404163178,
'MeanMeansOfNumericAtts': 2.045045045045045,
'MeanMutualInformation': 0.13092709767170999,
'MeanNoiseToSignalRatio': 7.534567748176438,
'MeanNominalAttDistinctValues': 3.0,
'MeanSkewnessOfNumericAtts': 2.326489482053779,
'MeanStdDevOfNumericAtts': 1.0184023049334343,
'MinAttributeEntropy': 0.2748031957462935,
'MinKurtosisOfNumericAtts': -0.5040960482425287,
'MinMeansOfNumericAtts': 1.060810810810811,
'MinMutualInformation': 0.02911996300275,
'MinNominalAttDistinctValues': 2.0,
'MinSkewnessOfNumericAtts': 0.33379516180165014,
'MinStdDevOfNumericAtts': 0.3135565426849874,
'MinorityClassPercentage': 1.3513513513513513,
'MinorityClassSize': 2.0,
'NaiveBayesAUC': 0.9083282647773021,
'NaiveBayesErrRate': 0.1554054054054054,
'NaiveBayesKappa': 0.7014820661229503,
'NumberOfBinaryFeatures': 9.0,
'NumberOfClasses': 4.0,
'NumberOfFeatures': 19.0,
'NumberOfInstances': 148.0,
'NumberOfInstancesWithMissingValues': 0.0,
'NumberOfMissingValues': 0.0,
'NumberOfNumericFeatures': 3.0,
'NumberOfSymbolicFeatures': 16.0,
'PercentageOfBinaryFeatures': 47.368421052631575,
'PercentageOfInstancesWithMissingValues': 0.0,
'PercentageOfMissingValues': 0.0,
'PercentageOfNumericFeatures': 15.789473684210526,
'PercentageOfSymbolicFeatures': 84.21052631578947,
```

Train machine learning models

```
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)
```

Out[8]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=1, p=2,
           weights='uniform')
```

Import directly via scikit-learn

```
from sklearn.datasets import fetch_openml
mice_data = fetch_openml(name='miceprotein',
                         version=4)
mice_data.details
```

Out[28]:

```
{'default_target_attribute': 'class',
 'file_id': '17928620',
 'format': 'ARFF',
 'id': '40966',
 'ignore_attribute': ['Genotype', 'Treatment', 'Behavior'],
 'licence': 'Public',
 'md5_checksum': '3c479a6885bfa0438971388283a1ce32',
 'name': 'MiceProtein',
 'processing_date': '2018-10-04 00:49:58',
 'row_id_attribute': 'MouseID',
 'status': 'active',
 'tag': ['OpenML-CC18', 'study_135', 'study_98', 'study_99'],
 'upload_date': '2017-11-08T16:00:15',
 'url': 'https://www.openml.org/data/v1/download/17928620/MiceProtein.arf
f',
 'version': '4',
 'visibility': 'public'}
```

You can also ask for meta-data to automatically preprocess the data

- e.g. categorical features -> do feature encoding

```
dataset = oml.datasets.get_dataset(10)
X, y, categorical = dataset.get_data(
    return_categorical_indicator=True)
enc = preprocessing.OneHotEncoder(
    categorical_features=categorical)
X = enc.fit_transform(X)
clf.fit(X, y)
```

Listing tasks

- Tasks define how models should be evaluated
- Specific target, train-test splits, ...

```
task_list = oml.tasks.list_tasks(size=5000)
```

```
First 5 of 5000 tasks:
```

Out[10]:

	tid	did	name	task_type	estimation_procedure	evaluation_m
2	2	2	anneal	Supervised Classification	10-fold Crossvalidation	predictive_ac
3	3	3	kr-vs-kp	Supervised Classification	10-fold Crossvalidation	NaN
4	4	4	labor	Supervised Classification	10-fold Crossvalidation	predictive_ac
5	5	5	arrhythmia	Supervised Classification	10-fold Crossvalidation	predictive_ac
6	6	6	letter	Supervised Classification	10-fold Crossvalidation	NaN

Listing/filtering same as datasets

```
mytasks.query('name=="eeg-eye-state"')
```

Out[11]:

	tid	did	name	task_type	estimation_procedure	evaluation
9983	9983	1471	eeg-eye-state	Supervised Classification	10-fold Crossvalidation	NaN
14951	14951	1471	eeg-eye-state	Supervised Classification	10-fold Crossvalidation	NaN

Download tasks

```
task = oml.tasks.get_task(14951)
```

```
{'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_s
plits/get/14951/Task_14951_splits.arff',
                          'parameters': {'number_folds': '10',
                                         'number_repeats': '1',
                                         'percentage': '',
                                         'stratified_sampling': 'true'},
                          'type': 'crossvalidation'},
 'evaluation_measure': None,
 'split': None,
 'target_name': 'Class',
 'task_id': 14951,
 'task_type': 'Supervised Classification',
 'task_type_id': 1}
```

Runs: Easily train models on tasks

```
# Get a task
task = oml.tasks.get_task(14951)

# Build any classifier or pipeline
clf = tree.ExtraTreeClassifier()

# Create a flow
flow = oml.flows.sklearn_to_flow(clf)

# Run the flow
run = oml.runs.run_flow_on_task(task, flow)
```

Share the run on the OpenML server

```
run = run.publish()
```

```
Uploaded to http://www.openml.org/r/10154294
```

Train and share pipelines

```
from sklearn import pipeline, ensemble, preprocessing
from openml import tasks,runs, datasets
task = tasks.get_task(59)
pipe = pipeline.Pipeline(steps=[
            ('Imputer', preprocessing.Imputer()),
            ('OneHotEncoder', preprocessing.OneHotEncoder(),
            ('Classifier', ensemble.RandomForestClassifier())
           ])
flow = oml.flows.sklearn_to_flow(pipe)
run = oml.runs.run_flow_on_task(task, flow)
myrun = run.publish()
```

Train and share Keras models

```
import keras
model = Sequential()
model.add(Conv2D(20, (5, 5), input_shape=(28,28,1),
          activation='relu'))
model.add(Dense(10))
model.add(Activation('softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

task = tasks.get_task(3573) #MNIST
flow = oml.flows.keras_to_flow(model)
run = oml.runs.run_flow_on_task(task, flow)
myrun = run.publish()
```

Download previous results

You can download all your results anytime, as well as everybody else's

```
myruns = oml.evaluations.list_evaluations(
    task=[14951],
    function='area_under_roc_curve')
sns.violinplot(x="score", y="flow", data=pd.DataFrame(scores), scale="wi
dth", palette="Set3", cut=0);
```