OpenML Python Tutorial

Joaquin Vanschoren and the OpenML Team

This tutorial explains how you can connect to the OpenML platform for exchanging machine learning datasets, pipelines, models, and evaluations.

Use it to collaborate online, log your experiments, and/or share your work in a reproducible and reusable way.

Prerequisites

You will need a Python working environment with:

- Python 3 or higher
- numpy, scipy, matplotlib, pandas,...
- scikit-learn 0.20 or higher (for building models)
- jupyter (to run these notebooks)
- openml 0.8.0 or higher

Option 1: Google Colab

Google Colab allows you to run this notebook in your Google Drive. Hence, you can run it in your browser without installing anything.

- Open this notebook (http://goo.gl/VwbKb4)
- Use `File > Make a copy in Drive` to create your own copy to work with.

Option 2: Binder

An open source alternative to run this notebook in your browser is *Binder*

- Go to https://mybinder.org/ (https://mybinder.org/)
- Copy-paste the repository name: https://github.com/openml/openml-tutorial (https://github.com/openml/openml-tutorial)
- Click launch.

Option 3: Anaconda

The easiest way to set things up locally (especially for Windows) is to install an [Anaconda (https://www.continuum.io/downloads)](https://www.continuum.io/downloads) environment. Choose Python 3.

After installation, run the following on the commandline:

```
conda install numpy scipy matplotlib pandas scikit-learn seaborn pprint jupyter
```

Option 4: pip

You can also install everything via pip, ideally in a virtual environment (http://docs.python-guide.org/en/latest/dev/virtualenvs/).

After installing pip, run the following on the commandline:

```
pip install numpy scipy matplotlib pandas scikit-learn seaborn pprint jupyter
```

Running this notebook locally

For a local setup, use git to clone the tutorial repository (https://github.com/openml/openml-tutorial) and start jupyter notebooks.

```
git clone https://github.com/openml/openml-tutorial
cd openml-tutorial
jupyter notebook
```

Installing OpenML

You can install the OpenML API via pip. In will be pre-installed if you use Binder.

In your Anaconda or custom environment, run

```
pip install openml
```

In Google colab or Jupyter, install by running (note the '!')

```
!pip install openml
```

Authentication

To upload new datasets, experiments,... to the OpenML server, you first need to find your API key.

- Create an OpenML account (free) on http://www.openml.org (http://www.openml.org).
- Log in, click your avatar/picture, open 'API authentication'.
- Your API key is a secret 32-character string

You can copy this API key into your code (but only if you never share it):

In [1]:
```
# Uncomment and set your OpenML key.
import openml as oml
#oml.config.apikey = 'YOUR_KEY'
```

Config file

It is safer to set your API key in a config file. By default this is created in `~/.openml/config` and loaded when you import openml.

It has the following settings (and defaults):

```
apikey=YOUR_KEY
server=https://www.openml.org/api/v1
cachedir=/HOME/.openml/cache
verbosity=1
confirm.upload=FALSE
```

Guides and cheat sheets

You are now ready to start playing with OpenML.

We also provide this handy cheat sheet (OpenML Python cheat sheet) with the most common commands.

Also, you can browse the official OpenML Python docs (https://openml.github.io/openml-python) for further examples and guidance.

Datasets

OpenML aims to allow *frictionless* sharing of data:

- Explore and search many thousands of datasets
- Every dataset is imported directly as an array/dataframe
- Rich and uniform meta-data.

Exploring datasets

`datasets.list_datasets()` returns a dict with all datasets.

In [5]:

```
data_dict = oml.datasets.list_datasets() # Returns a dict
data_list = pd.DataFrame.from_dict(data_dict, orient='index') # dataframe
print("First 10 of %s datasets..." % len(data_list))
data_list[columns][:10]
```

```
First 10 of 2588 datasets...
```

Out[5]:

	did	name	NumberOfInstances	NumberOfFeatures	NumberOfClas
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

You can filter datasets by:

- `data_name` and `data_version`
- verification `status` ('active', 'in_preparation', 'deactivated')
  - Default: 'active'
- `tag` (tags added by you or other users)
- `number_instances`, `number_features`, `number_classes`, `number_missing_values`

In [6]:
```
d = oml.datasets.list_datasets(number_instances = '10000..20000',
                               number_features = '10..20')
pd.DataFrame.from_dict(d, orient='index')[columns][:10]
```

Out[6]:

	did	name	NumberOfInstances	Num
6	6	letter	20000	17
32	32	pendigits	10992	17
216	216	elevators	16599	19
846	846	elevators	16599	19
977	977	letter	20000	17
1019	1019	pendigits	10992	17
1120	1120	MagicTelescope	19020	12
1199	1199	BNG(echoMonths)	17496	10
1222	1222	letter-challenge-unlabeled.arff	20000	17
1414	1414	Kaggle_bike_sharing_demand_challange	10886	12

Alternatively, download the whole list as a dataframe and query the meta-data

In [7]:
```
data_list.query('NumberOfInstances > 10000 & NumberOfFeatures > 10'
               ).sort_values(['name'])[columns][:10]
```

Out[7]:

	did	name	NumberOfInstances	NumberOfFeatures
274	274	20_newsgroups.drift	399940.0	1002.0
40517	40517	20_newsgroups.drift	399940.0	1001.0
727	727	2dplanes	40768.0	11.0
215	215	2dplanes	40768.0	11.0
41138	41138	APSFailure	76000.0	171.0
296	296	Ailerons	13750.0	41.0
1240	1240	AirlinesCodrnaAdult	1076790.0	30.0
1197	1197	BNG(2dplanes)	177147.0	11.0
1207	1207	BNG(Ailerons)	1000000.0	41.0
1205	1205	BNG(Australian)	1000000.0	15.0

This also allows to search for terms in the dataset name

In [8]:
```
data_list.query('name.str.contains("eeg")', engine='python')[columns]
```

Out[8]:

	did	name	NumberOfInstances	NumberOfFeatures	NumberOfClasse
1471	1471	eeg-eye-state	14980.0	15.0	2.0

Download datasets

`datasets.get_dataset(data_id)` returns an `OpenMLData` object with the dataset and meta-data.

In [9]:
```
dataset = oml.datasets.get_dataset(1471)
print(dataset.description[:500])
```

```
**Author**: Oliver Roesler
**Source**: [UCI](https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State), Baden
-Wuerttemberg, Cooperative State University (DHBW), Stuttgart, Germany
**Please cite**: [UCI](https://archive.ics.uci.edu/ml/citation_policy.html)

All data is from one continuous EEG measurement with the Emotiv EEG Neuroheadse
t. 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 af
ter
```

Get the data itself

`OpenMLData.getdata()` returns the actual data as numpy arrays.

```
X = dataset.get_data()
```

Optional arguments:

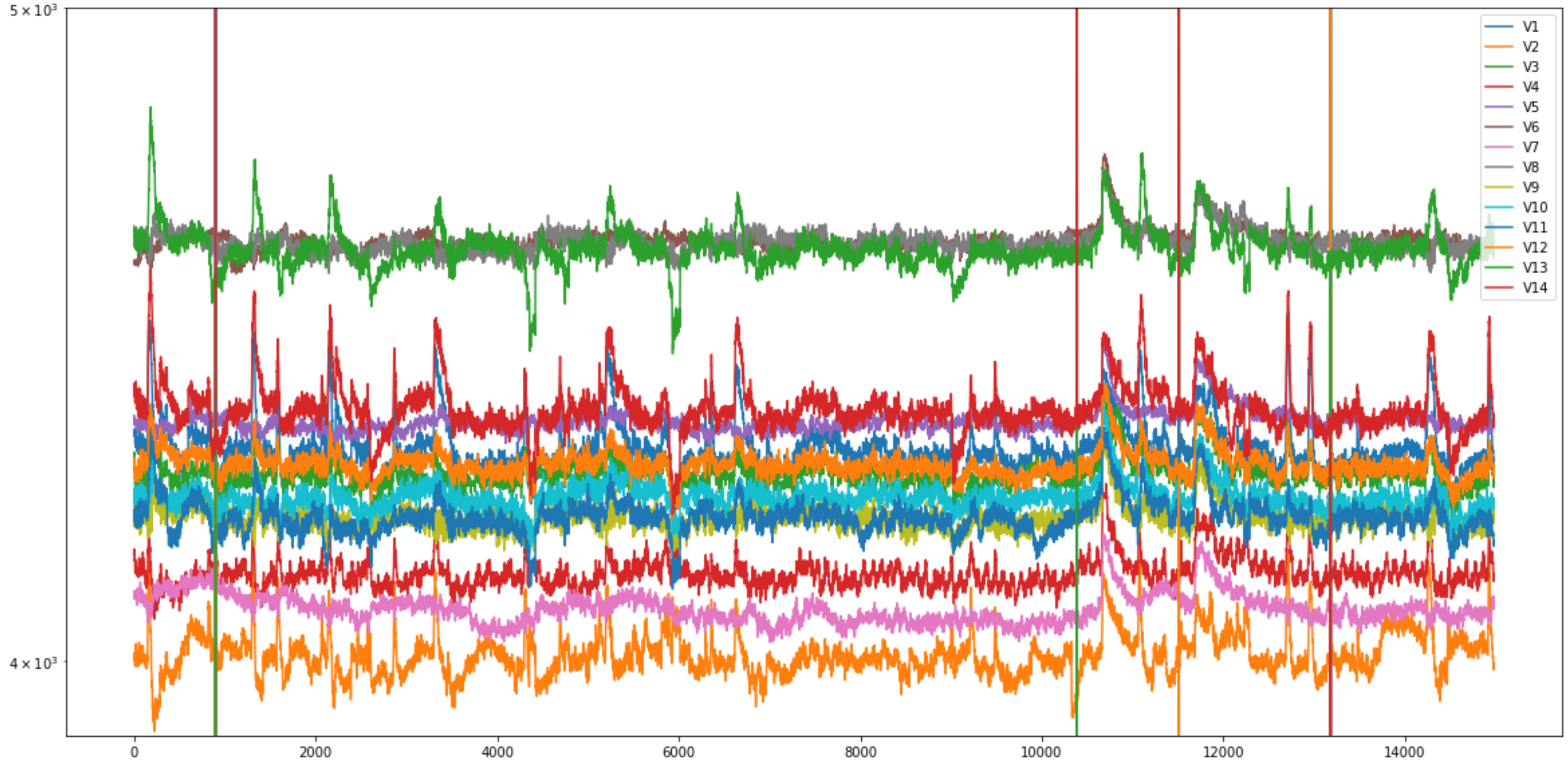
- `target=dataset.default_target_attribute` also return the target (y) values
- `return_attribute_names=True` also return the attributes names
- `return_categorical=True` return a boolean array indicating categorical attributes

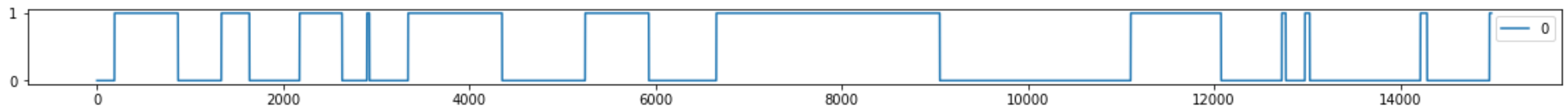
```
In [10]: X, y, attribute_names = dataset.get_data(
             target=dataset.default_target_attribute,
             return_attribute_names=True)
         eeg = pd.DataFrame(X, columns=attribute_names)
         eeg[:10]
```

Out[10]:

	V1	V2	V3	V4	V5	V6
0	4329.229980	4009.229980	4289.229980	4148.209961	4350.259766	4586.14990
1	4324.620117	4004.620117	4293.850098	4148.720215	4342.049805	4586.66992
2	4327.689941	4006.669922	4295.379883	4156.410156	4336.919922	4583.58984
3	4328.720215	4011.790039	4296.410156	4155.899902	4343.589844	4582.56005
4	4326.149902	4011.790039	4292.310059	4151.279785	4347.689941	4586.66992
5	4321.029785	4004.620117	4284.100098	4153.330078	4345.640137	4587.18017
6	4319.490234	4001.030029	4280.509766	4151.790039	4343.589844	4584.62011
7	4325.640137	4006.669922	4278.459961	4143.080078	4344.100098	4583.08007
8	4326.149902	4010.770020	4276.410156	4139.490234	4345.129883	4584.10009
9	4326.149902	4011.280029	4276.919922	4142.049805	4344.100098	4582.56005

In [11]:
```
eeg.plot(logy=True,ylim=(3900,5000),figsize=(20,10))
pd.DataFrame(y).plot(figsize=(20,1));
```





Get the meta-data

Every dataset comes with rich meta-data:

- name, version, date, creator, licence, description, ...
- `dataset.qualities` returns 100+ statistical data properties
- `dataset.features` returns all variables and their data types
- tags added by the OpenML community

```
In [12]: vars(dataset)
```

```
Out[12]: {'_dataset': None,
 'citation': None,
 'collection_date': None,
 'contributor': None,
 'creator': None,
 'data_file': '/Users/joa/.openml/cache/org/openml/www/datasets/1471/dataset.ar
ff',
 'data_pickle_file': '/Users/joa/.openml/cache/org/openml/www/datasets/1471/dat
aset.pkl.py3',
 'dataset_id': 1471,
 'default_target_attribute': 'Class',
 'description': "**Author**: Oliver Roesler  \n**Source**: [UCI](https://archiv
e.ics.uci.edu/ml/datasets/EEG+Eye+State), Baden-Wuerttemberg, Cooperative State
University (DHBW), Stuttgart, Germany  \n**Please cite**: [UCI](https://archiv
e.ics.uci.edu/ml/citation_policy.html)  \n\nAll data is from one continuous EEG
measurement with the Emotiv EEG Neuroheadset. The duration of the measurement w
as 117 seconds. The eye state was detected via a camera during the EEG measurem
ent and added later manually to the file after analyzing the video frames. '1'
indicates the eye-closed and '0' the eye-open state. All values are in chronolo
gical order with the first measured value at the top of the data.\n\nThe featur
es correspond to 14 EEG measurements from the headset, originally labeled AF3,
F7, F3, FC5, T7, P, O1, O2, P8, T8, FC6, F4, F8, AF4, in that order.",
 'features': {0: [0 - V1 (numeric)],
  1: [1 - V2 (numeric)],
  2: [2 - V3 (numeric)],
  3: [3 - V4 (numeric)],
  4: [4 - V5 (numeric)],
  5: [5 - V6 (numeric)],
  6: [6 - V7 (numeric)],
  7: [7 - V8 (numeric)],
  8: [8 - V9 (numeric)],
  9: [9 - V10 (numeric)],
  10: [10 - V11 (numeric)],
  11: [11 - V12 (numeric)],

  12: [12 - V13 (numeric)]
```

```
 12: [12 - V13 (numeric)],
 13: [13 - V14 (numeric)],
 14: [14 - Class (nominal)]},
'format': 'ARFF',
'ignore_attributes': None,
'language': None,
'licence': 'Public',
'md5_checksum': '32086b7bec4daaa9cbe5f19efa63368c',
'name': 'eeg-eye-state',
'original_data_url': None,
'paper_url': None,
'qualities': {'AutoCorrelation': 0.9984645169904534,
 'CfsSubsetEval_DecisionStumpAUC': 0.8154762506306992,
 'CfsSubsetEval_DecisionStumpErrRate': 0.22930574098798398,
 'CfsSubsetEval_DecisionStumpKappa': 0.5342696977625635,
 'CfsSubsetEval_NaiveBayesAUC': 0.8154762506306992,
 'CfsSubsetEval_NaiveBayesErrRate': 0.22930574098798398,
 'CfsSubsetEval_NaiveBayesKappa': 0.5342696977625635,
 'CfsSubsetEval_kNN1NAUC': 0.8154762506306992,
 'CfsSubsetEval_kNN1NErrRate': 0.22930574098798398,
 'CfsSubsetEval_kNN1NKappa': 0.5342696977625635,
 'ClassEntropy': 0.992422373387609,
 'DecisionStumpAUC': 0.5730896889672722,
 'DecisionStumpErrRate': 0.41141522029372496,
 'DecisionStumpKappa': 0.12923993775163797,
 'Dimensionality': 0.0010013351134846463,
 'EquivalentNumberOfAtts': nan,
 'J48.00001.AUC': 0.8271462175860198,
 'J48.00001.ErrRate': 0.18631508678237652,
 'J48.00001.Kappa': 0.623155313134783,
 'J48.0001.AUC': 0.8271462175860198,
 'J48.0001.ErrRate': 0.18631508678237652,
 'J48.0001.Kappa': 0.623155313134783,
 'J48.001.AUC': 0.8271462175860198,
 'J48.001.ErrRate': 0.18631508678237652,
 'J48.001.Kappa': 0.623155313134783,
 'MajorityClassPercentage': 55.12016021361815,
 'MajorityClassSize': 8257.0,
 'MaxAttributeEntropy': nan
```

'MaxAttributeEntropy': nan,
'MaxKurtosisOfNumericAtts': 14979.178735371073,
'MaxMeansOfNumericAtts': 4644.02237917223,
'MaxMutualInformation': nan,
'MaxNominalAttDistinctValues': 2.0,
'MaxSkewnessOfNumericAtts': 122.38777688436316,
'MaxStdDevOfNumericAtts': 5891.2850425236575,
'MeanAttributeEntropy': nan,
'MeanKurtosisOfNumericAtts': 8904.72190163755,
'MeanMeansOfNumericAtts': 4316.882028546157,
'MeanMutualInformation': nan,
'MeanNoiseToSignalRatio': nan,
'MeanNominalAttDistinctValues': 2.0,
'MeanSkewnessOfNumericAtts': 71.73726620900297,
'MeanStdDevOfNumericAtts': 1767.2884824189532,
'MinAttributeEntropy': nan,
'MinKurtosisOfNumericAtts': 2056.5210594418404,
'MinMeansOfNumericAtts': 4009.767693591455,
'MinMutualInformation': nan,
'MinNominalAttDistinctValues': 2.0,
'MinSkewnessOfNumericAtts': -13.615160740498586,
'MinStdDevOfNumericAtts': 29.292603201776053,
'MinorityClassPercentage': 44.879839786381844,
'MinorityClassSize': 6723.0,
'NaiveBayesAUC': 0.5441157198060068,
'NaiveBayesErrRate': 0.4634846461949266,
'NaiveBayesKappa': 0.11130397334752887,
'NumberOfBinaryFeatures': 1.0,
'NumberOfClasses': 2.0,
'NumberOfFeatures': 15.0,
'NumberOfInstances': 14980.0,
'NumberOfInstancesWithMissingValues': 0.0,
'NumberOfMissingValues': 0.0,
'NumberOfNumericFeatures': 14.0,
'NumberOfSymbolicFeatures': 1.0,
'PercentageOfBinaryFeatures': 6.666666666666667,
'PercentageOfInstancesWithMissingValues': 0.0,
'PercentageOfMissingValues': 0.0,
'PercentageOfNumericFeatures': 93.33333333333333

```
'PercentageOfNumericFeatures': 93.33333333333333,
'PercentageOfSymbolicFeatures': 6.666666666666667,
'Quartile1AttributeEntropy': nan,
'Quartile1KurtosisOfNumericAtts': 2713.5598368066608,
'Quartile1MeansOfNumericAtts': 4193.079256508679,
'Quartile1MutualInformation': nan,
'Quartile1SkewnessOfNumericAtts': 22.475026893669664,
'Quartile1StdDevOfNumericAtts': 37.98467230942155,
'Quartile2AttributeEntropy': nan,
'Quartile2KurtosisOfNumericAtts': 9352.695219763142,
'Quartile2MeansOfNumericAtts': 4271.6276034712955,
'Quartile2MutualInformation': nan,
'Quartile2SkewnessOfNumericAtts': 84.61113200877784,
'Quartile2StdDevOfNumericAtts': 627.1558153704191,
'Quartile3AttributeEntropy': nan,
'Quartile3KurtosisOfNumericAtts': 14971.654606997874,
'Quartile3MeansOfNumericAtts': 4466.12820822263,
'Quartile3MutualInformation': nan,
'Quartile3SkewnessOfNumericAtts': 122.34170603745707,
'Quartile3StdDevOfNumericAtts': 3343.823788627256,
'REPTreeDepth1AUC': 0.8585707193015194,
'REPTreeDepth1ErrRate': 0.19072096128170896,
'REPTreeDepth1Kappa': 0.6126693923072585,
'REPTreeDepth2AUC': 0.8585707193015194,
'REPTreeDepth2ErrRate': 0.19072096128170896,
'REPTreeDepth2Kappa': 0.6126693923072585,
'REPTreeDepth3AUC': 0.8585707193015194,
'REPTreeDepth3ErrRate': 0.19072096128170896,
'REPTreeDepth3Kappa': 0.6126693923072585,
'RandomTreeDepth1AUC': 0.8092130970110125,
'RandomTreeDepth1ErrRate': 0.18845126835781043,
'RandomTreeDepth1Kappa': 0.6188451689674707,
'RandomTreeDepth2AUC': 0.8092130970110125,
'RandomTreeDepth2ErrRate': 0.18845126835781043,
'RandomTreeDepth2Kappa': 0.6188451689674707,
'RandomTreeDepth3AUC': 0.8092130970110125,
'RandomTreeDepth3ErrRate': 0.18845126835781043,
'RandomTreeDepth3Kappa': 0.6188451689674707,
'StdvNominalAttDistinctValues': 0.0
```

sklearn fetch_openml

You can also fetch OpenML datasets directly through scikit-learn

In [13]:
```
from sklearn.datasets import fetch_openml
eeg_data = fetch_openml(name='eeg-eye-state', version=1)
eeg_data.details
```

Out[13]:
```
{'default_target_attribute': 'Class',
 'file_id': '1587924',
 'format': 'ARFF',
 'id': '1471',
 'licence': 'Public',
 'md5_checksum': '32086b7bec4daaa9cbe5f19efa63368c',
 'name': 'eeg-eye-state',
 'processing_date': '2018-10-03 21:41:12',
 'status': 'active',
 'tag': ['brain',
  'EEG',
  'OpenML100',
  'study_123',
  'study_14',
  'study_34',
  'study_7',
  'time_series',
  'uci'],
 'upload_date': '2015-05-22T16:40:04',
 'url': 'https://www.openml.org/data/v1/download/1587924/eeg-eye-state.arff',
 'version': '1',
 'visibility': 'public'}
```

Upload datasets

You can easily share your own datasets by creating an `OpenMLData` object and publishing it. The data can be in the form of native lists, numpy arrays, pandas dataframes, or locally stored files.

See the documentation (https://openml.github.io/openml-python/master/examples/create_upload_tutorial.html#sphx-glr-examples-create-upload-tutorial-py) for detailed examples.

Helper function

`create_dataset` helps to create an OpenML dataset from data and an attribute description

```
dataset = datasets.functions.create_dataset(
    data=data, # data array
    attributes=attributes, # list of attributes
    name='..', # <128 characters, a-z, A-Z, 0-9, _, -, ., ()
    description='..', # Textual description
    creator='..', # Creator of this dataset
    licence='..', # Data licence
    default_target_attribute='..', # Optional target attribute(s)
    citation='..', # How to cite the dataset
    original_data_url='..', # Link to dataset elsewhere
)
```

Data format:

- `data` should be a single dataframe or array
- `attributes` is a list of names and data types (ARFF format)

```
attribute_names = [
  ('outlook', ['sunny', 'overcast', 'rainy']),
  ('temperature', 'REAL'),
  ('humidity', 'REAL'),
  ('windy', ['TRUE', 'FALSE']),
  ('play', ['yes', 'no']),
]
```

Dataset is a pandas DataFrame

In [18]:
```
df = pd.DataFrame(data, columns=[col_name for col_name, _ in attribute_names])
# enforce the categorical column to have a categorical dtype
df['outlook'] = df['outlook'].astype('category')
df['windy'] = df['windy'].astype('bool')
df['play'] = df['play'].astype('category')
df
```

Out[18]:

	outlook	temperature	humidity	windy	play
0	sunny	85	85	True	no
1	sunny	80	90	True	no
2	overcast	83	86	True	yes
3	rainy	70	96	True	yes
4	rainy	68	80	True	yes
5	rainy	65	70	True	no
6	overcast	64	65	True	yes
7	sunny	72	95	True	no
8	sunny	69	70	True	yes
9	rainy	75	80	True	yes
10	sunny	75	70	True	yes
11	overcast	72	90	True	yes
12	overcast	81	75	True	yes

Create and publish

In [20]:
```
from openml.datasets.functions import create_dataset
weather_dataset = create_dataset(name="Weather", description=description,
    creator='I. H. Witten, E. Frank, M. A. Hall, and ITPro', contributor=None,
    collection_date='01-01-2011', language='English', licence=None,
    default_target_attribute='play', row_id_attribute=None,
    ignore_attribute=None, citation=citation, attributes='auto',
    data=df, version_label='example',
)
data_id = weather_dataset.publish()
print("Uploaded to https://test.openml.org/d/" + str(data_id))
```

```
Uploaded to https://test.openml.org/d/41521
```

Dataset is a numpy array

This requires a bit more work to prepare the data and attributes.

In [22]:

```
from openml.datasets.functions import create_dataset
data = np.concatenate((X, y.reshape((-1, 1))), axis=1)
attributes = [
    (attribute_name, 'REAL') for attribute_name in attribute_names
] + [('class', 'INTEGER')]
my_data = create_dataset(data=data, attributes=attributes, name=name,
    description=description, licence='CC0', default_target_attribute='class',
    creator=creator, contributor=None, collection_date='09-01-2012',
    language='English', ignore_attribute=None, citation=citation)

data_id = my_data.publish()
print("Uploaded to https://test.openml.org/d/" + str(data_id))
```

```
Uploaded to https://test.openml.org/d/6679
```

Dataset is an ARFF file

Only ARFF for now. CSV and DataPackage support in progress.

In [23]:
```
dataset = oml.datasets.OpenMLDataset(data_file='dataset.arff', name='test_dataset'
, description='Test dataset',
                                     format='ARFF', licence='Public', default_targ
et_attribute='class')
data_id = dataset.publish()
print("Uploaded to https://test.openml.org/d/" + str(data_id))
```

```
Uploaded to https://test.openml.org/d/6680
```

Tasks

Tasks define the exact machine learning problem that you want to solve, in a machine-readable way. They help you to build correct and useful models, and they allow the OpenML server to evaluate all shared models objectively.

- Task type (classification, regression, clustering,...)
- Which are the input variables?
- For predictive tasks, which are the target variables?
- How should the model be evaluated, e.g. train-test splits
- Any task-specific aspects that need to be known beforehand

Task types

OpenML supports several task types. The main types and their IDs are:

In [26]:
```
tasktypes
```

Out[26]:

	Task type name
1	Classification
2	Regression
3	Learning curves
4	Data stream classification
5	Clustering

Exploring tasks

`datasets.list_tasks()` returns a dict with all tasks.

In [27]:
```
task_dict = oml.tasks.list_tasks(task_type_id=1) # Get classification tasks
task_list = pd.DataFrame.from_dict(task_dict, orient='index') # dataframe
print("First 10 of %s tasks..." % len(data_list))
task_list[columns][:10]
```

```
First 10 of 2588 tasks...
```

Out[27]:

	name	task_type	estimation_procedure	evaluation_measures	ta
2	anneal	Supervised Classification	10-fold Crossvalidation	predictive_accuracy	cla
3	kr-vs-kp	Supervised Classification	10-fold Crossvalidation	NaN	cla
4	labor	Supervised Classification	10-fold Crossvalidation	predictive_accuracy	cla
5	arrhythmia	Supervised Classification	10-fold Crossvalidation	predictive_accuracy	cla
6	letter	Supervised Classification	10-fold Crossvalidation	NaN	cla
7	audiology	Supervised Classification	10-fold Crossvalidation	predictive_accuracy	cla
8	liver-disorders	Supervised Classification	10-fold Crossvalidation	predictive_accuracy	se
9	autos	Supervised Classification	10-fold Crossvalidation	predictive_accuracy	sy

You can filter tasks by:

- task type id
- task tags
- properties of the underlying dataset: data_tag, status, data_id, data_name, number_instances, number_features, number_classes, ...

Classification tasks on dataset *Bioresponse*

In [28]:
```
task_dict = oml.tasks.list_tasks(task_type_id=1, data_name='Bioresponse')
pd.DataFrame.from_dict(task_dict, orient='index')[columns]
```

Out[28]:

	name	task_type	estimation_procedure	evaluation_measure:
9910	Bioresponse	Supervised Classification	10-fold Crossvalidation	NaN
14966	Bioresponse	Supervised Classification	10-fold Crossvalidation	NaN
75156	Bioresponse	Supervised Classification	33% Holdout set	predictive_accuracy
145677	Bioresponse	Supervised Classification	10-fold Crossvalidation	area_under_roc_curv
167195	Bioresponse	Supervised Classification	33% Holdout set	NaN
168837	Bioresponse	Supervised Classification	10 times 10-fold Crossvalidation	NaN

Download tasks

`tasks.get_task(task_id)` returns an `OpenMLTask` object with the underlying dataset and all task-specific meta-data.

Tasks are basically wrappers around a dataset with information on how to analyse it.

In [29]:

```
task = oml.tasks.get_task(145677)
print("Target:", task.target_name, ", Classes:", task.class_labels)
pprint(task.estimation_procedure)
```

```
Target: target , Classes: ['0', '1']
{'data_splits_url': 'https://www.openml.org/api_splits/get/145677/Task_145677_s
plits.arff',
 'parameters': {'number_folds': '10',
                'number_repeats': '1',
                'percentage': '',
                'stratified_sampling': 'true'},
 'type': 'crossvalidation'}
```

`OpenMLTask.get_dataset()` returns the underlying data

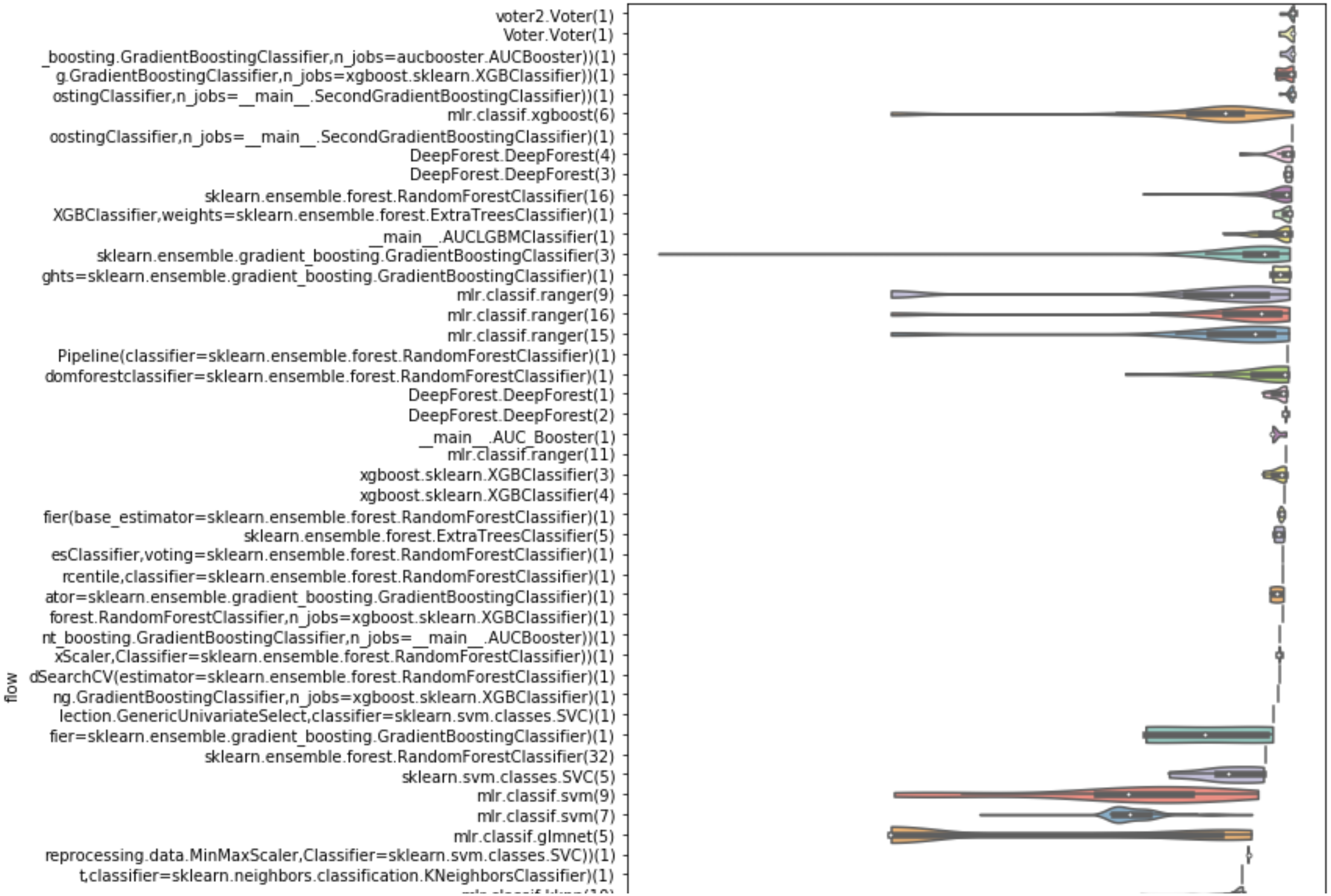
In [30]:
```
bio_data = task.get_dataset()
X, y = bio_data.get_data(target=bio_data.default_target_attribute)
bio_df = pd.DataFrame(X[:,0:100]) # select first 100 of 1776 features
bio_df['bioresponse'] = y
plt.figure(figsize=(25,5))
pd.plotting.parallel_coordinates(bio_df, 'bioresponse', axvlines=False, color=('#F
FE888', '#FF9999'));
```

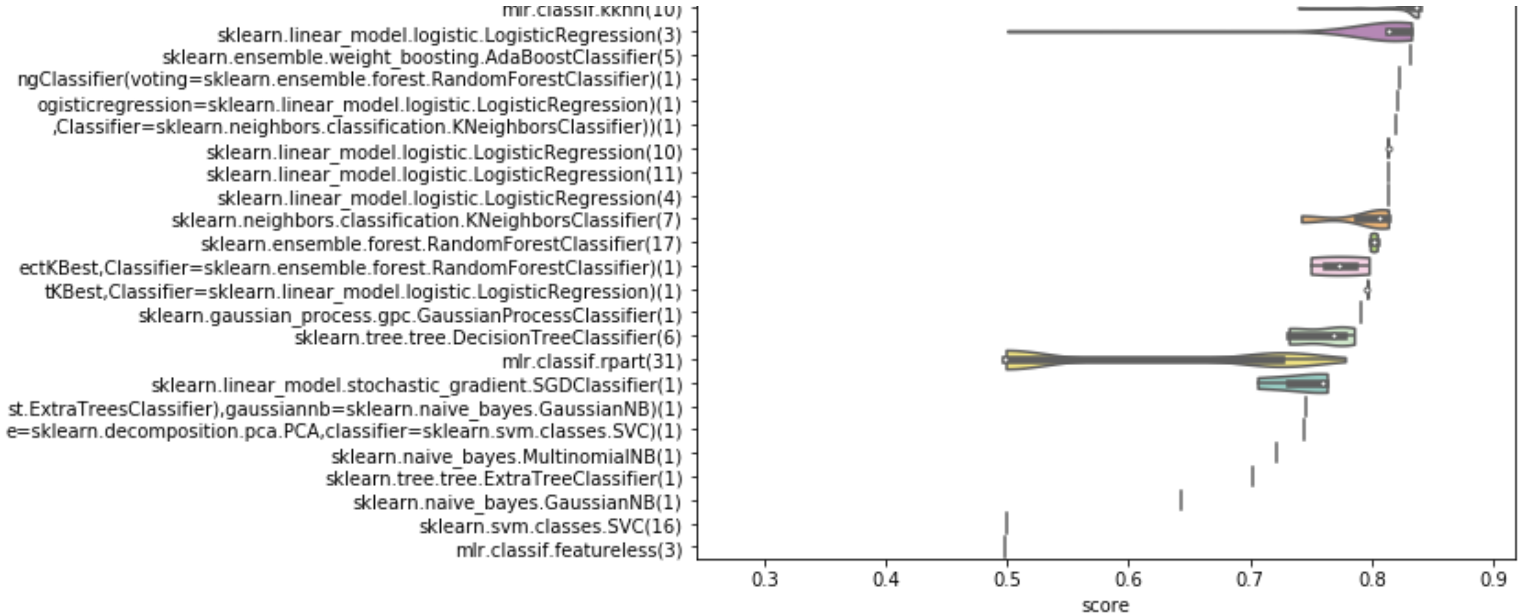
Download existing task results

That data looks quite complex. Which are the best known techniques to model it?

With `list_evaluations` you can download all existing runs (by anyone) on any task. You do need to choose an evaluation measure you are interested in, e.g. accuracy or area under the ROC curve.

In [31]:
```
evals = oml.evaluations.list_evaluations(task=[145677], function='area_under_roc_curve')
scores = sorted([{"flow":e.flow_name[-70:], "score":e.value} for id, e in evals.items()], key=lambda x: -x["score"])
plt.figure(figsize=(8, 16))
sns.violinplot(x="score", y="flow", data=pd.DataFrame(scores), scale="width", cut=0, palette="Set3");
```




mlr.classif.kknn(10)
sklearn.linear_model.logistic.LogisticRegression(3)
sklearn.ensemble.weight_boosting.AdaBoostClassifier(5)
ngClassifier(voting=sklearn.ensemble.forest.RandomForestClassifier)(1)
ogisticregression=sklearn.linear_model.logistic.LogisticRegression)(1)
,Classifier=sklearn.neighbors.classification.KNeighborsClassifier))(1)
sklearn.linear_model.logistic.LogisticRegression(10)
sklearn.linear_model.logistic.LogisticRegression(11)
sklearn.linear_model.logistic.LogisticRegression(4)
sklearn.neighbors.classification.KNeighborsClassifier(7)
sklearn.ensemble.forest.RandomForestClassifier(17)
ectKBest,Classifier=sklearn.ensemble.forest.RandomForestClassifier)(1)
tKBest,Classifier=sklearn.linear_model.logistic.LogisticRegression)(1)
sklearn.gaussian_process.gpc.GaussianProcessClassifier(1)
sklearn.tree.tree.DecisionTreeClassifier(6)
mlr.classif.rpart(31)
sklearn.linear_model.stochastic_gradient.SGDClassifier(1)
st.ExtraTreesClassifier),gaussiannb=sklearn.naive_bayes.GaussianNB)(1)
e=sklearn.decomposition.pca.PCA,classifier=sklearn.svm.classes.SVC)(1)
sklearn.naive_bayes.MultinomialNB(1)
sklearn.tree.tree.ExtraTreeClassifier(1)
sklearn.naive_bayes.GaussianNB(1)
sklearn.svm.classes.SVC(16)
mlr.classif.featureless(3)
0.3
0.4
0.5
0.6
0.7
0.8
0.9
score

Creating new tasks

Creating new tasks is not yet available in the current release of the Python API.

Right now, you need to use another API (e.g. REST or Java) or the OpenML website.

Flows

OpenML Flows represent (almost) arbitrary *pipelines* (or *workflows*) of operations to build machine learning models

- Which operations, in what order
- Data cleaning, preprocesssing, model building
- Exact implementations and versions used
- All necessary information to reproduce the models

Discover flows

Discover useful flows by how well they do on a given task (or across many tasks)

In [32]:

```
evals = oml.evaluations.list_evaluations(task=[145677], function='area_under_roc_c
urve')
scores = sorted([{"flow_id":e.flow_id, "flow":e.flow_name, "score":e.value} for id
, e in evals.items()], key=lambda x: -x["score"])
pd.DataFrame(scores)[0:30]
```

Out[32]:

	flow	flow_id	score
0	voter2.Voter(1)	5858	0.887468
1	voter2.Voter(1)	5858	0.887404
2	voter2.Voter(1)	5858	0.887388
3	Voter.Voter(1)	5827	0.887354
4	voter2.Voter(1)	5858	0.887344
5	voter2.Voter(1)	5858	0.887228
6	voter2.Voter(1)	5858	0.887207
7	sklearn.pipeline.Pipeline(standardscaler=sklea...	5819	0.887189
8	sklearn.pipeline.Pipeline(standardscaler=sklea...	5819	0.887188
9	Voter.Voter(1)	5827	0.887178
10	voter2.Voter(1)	5858	0.887132
11	Voter.Voter(1)	5827	0.887065
12	sklearn.pipeline.Pipeline(standardscaler=sklea...	5773	0.887017
13	sklearn.pipeline.Pipeline(standardscaler=sklea...	5808	0.886996

Get and rebuild the pipeline

- Get the flow and check the dependencies

In [33]:

```
flow = oml.flows.get_flow(5819)
vars(flow)
```

Out[33]:

```
{'binary_format': None,
 'binary_md5': None,
 'binary_url': None,
 'class_name': 'sklearn.pipeline.Pipeline',
 'components': OrderedDict([('standardscaler',
               <openml.flows.flow.OpenMLFlow at 0x1a1c31fe80>),
              ('votingclassifier',
               <openml.flows.flow.OpenMLFlow at 0x1a1c31f4e0>)]),
 'custom_name': None,
 'dependencies': 'sklearn==0.18.1\nnumpy>=1.6.1\nscipy>=0.9',
 'description': 'Automatically created scikit-learn flow.',
 'external_version': 'aucbooster==1,sklearn==0.18.1',
 'flow_id': 5819,
 'language': 'English',
 'model': None,
 'name': 'sklearn.pipeline.Pipeline(standardscaler=sklearn.preprocessing.data.S
tandardScaler,votingclassifier=sklearn.ensemble.voting_classifier.VotingClassif
ier(voting=sklearn.ensemble.forest.RandomForestClassifier,weights=sklearn.ensem
ble.gradient_boosting.GradientBoostingClassifier,n_jobs=aucbooster.AUCBooste
r))',
 'parameters': OrderedDict([('steps',
               '[{"oml-python:serialized_object": "component_reference", "valu
e": {"key": "standardscaler", "step_name": "standardscaler"}}, {"oml-python:ser
ialized_object": "component_reference", "value": {"key": "votingclassifier", "s
tep_name": "votingclassifier"}}]')]),
 'parameters_meta_info': OrderedDict([('steps',
               OrderedDict([('description', None), ('data_type', None)]))]),
 'tags': ['Verified_Supervised_Classification'],
 'upload_date': '2017-03-16T22:17:46',
 'uploader': '2514',
 'version': '1'}
```

If your environment matches the dependencies, you can rebuild and reuse the pipeline

```
pipeline = oml.flows.flow_to_sklearn(flow)
```

Creating flows

`sklearn_to_flow(clf)` converts any scikit-learn estimator or pipeline to an OpenML flow.

In [34]:
```
from sklearn import ensemble

# Build any classifier or pipeline
clf = ensemble.RandomForestClassifier()

# Create an openml flow
flows = oml.flows.sklearn_to_flow(clf)
```

It also works with pipelines

When you need to handle 'dirty' data, build pipelines to clean and model them automatically

Note: pipelines that use the same estimator multiple times are not supported yet

In [35]:

```
from sklearn import pipeline, ensemble, preprocessing
task = oml.tasks.get_task(59)
pipe = pipeline.Pipeline(steps=[
            ('Imputer', preprocessing.Imputer(strategy='median')),
            ('OneHotEncoder', preprocessing.OneHotEncoder(sparse=False, handle_unk
nown='ignore')),
            ('Classifier', ensemble.RandomForestClassifier())
           ])
flow = oml.flows.sklearn_to_flow(pipe)
```

Runs

- When you run a flow on a task, this returns an OpenML Run
- A run will contain
  - The exact pipeline and task
  - The exact hyperparameter settings
  - Task-specific results, e.g. predictions
  - Evaluations of the produced models
  - Optionally, the models themselves
- After publishing, OpenML will add server-side evaluations and other meta-data

Discover runs

`list_runs(options)` returns all runs

- Filter by run id, task, flow, uploader
- display_errors: whether to return failed runs

In [49]:
```
myruns = oml.runs.list_runs(task=[14951],size=10000)
run_list = pd.DataFrame.from_dict(myruns, orient='index') # dataframe
print("First 10 of %s tasks..." % len(run_list))
run_list[:10]
```

```
First 10 of 10000 tasks...
```

Out[49]:

	run_id	task_id	setup_id	flow_id	uploader
544246	544246	14951	2961	2390	2
544272	544272	14951	2963	2393	869
544274	544274	14951	5538	3401	287
544316	544316	14951	3526	2629	869
544318	544318	14951	3526	2629	869
544325	544325	14951	5539	2277	287
544514	544514	14951	5540	3404	2
545897	545897	14951	3526	2629	869
545898	545898	14951	5554	2629	869
545899	545899	14951	5555	2629	869

Download runs

`get_run(id)` returns an `OpenMLRun` object with all details

In [60]:
```
run = oml.runs.get_run(id)
vars(run)
```

Out[60]:
```
{'data_content': None,
 'dataset_id': 1471,
 'error_message': None,
 'evaluations': OrderedDict(),
 'flow': None,
 'flow_id': 9651,
 'flow_name': 'sklearn.neighbors.classification.KNeighborsClassifier(36)',
 'fold_evaluations': OrderedDict(),
 'model': None,
 'output_files': OrderedDict([('description', 21230611),
              ('predictions', 21230612)]),
 'parameter_settings': [OrderedDict([('oml:name', 'algorithm'),
               ('oml:value', '"auto"'),
               ('oml:component', '9651')]),
  OrderedDict([('oml:name', 'leaf_size'),
               ('oml:value', '30'),
               ('oml:component', '9651')]),
  OrderedDict([('oml:name', 'metric'),
               ('oml:value', '"minkowski"'),
               ('oml:component', '9651')]),
  OrderedDict([('oml:name', 'metric_params'),
               ('oml:value', 'null'),
               ('oml:component', '9651')]),
  OrderedDict([('oml:name', 'n_jobs'),
               ('oml:value', 'null'),
               ('oml:component', '9651')]),
  OrderedDict([('oml:name', 'n_neighbors'),
               ('oml:value', '5'),
               ('oml:component', '9651')]),
  OrderedDict([('oml:name', 'p'),
               ('oml:value', '2'),
               ('oml:component', '9651')]),
  OrderedDict([('oml:name', 'weights'),

               ('oml:value', '"uniform"'),
```

Build, evaluate, and upload runs

A completely self-contained experiments in 5 lines of code:

- Download the task (a wrapper around the data also including evaluation details, e.g. train/test splits)
- Create any scikit-learn classifier (or pipeline)
- Convert the pipeline to an OpenML 'flow'
- Run the flow on the task
    - `run_flow_on_task(flow, task)`: for every OpenML flow
    - `run_model_on_task(model, task)`: shorthand for sklearn pipelines/estimators
- Publish (upload) if you want

In [42]:

```python
from sklearn import ensemble

# Get a task
task = oml.tasks.get_task(3954)

# Build any classifier or pipeline
clf = ensemble.RandomForestClassifier()

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

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

Out[42]:

```
[run id: 10155006, task id: 3954, flow id: 9154, flow name: sklearn.ensemble.fo
rest.Ra...]
```

Share the run on the OpenML server

In [39]:
```
myrun = run.publish()
print("Uploaded to http://www.openml.org/r/" + str(myrun.run_id))
```

```
Uploaded to http://www.openml.org/r/10155005
```

A complete experiment

In [37]:

```
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, handle_unk
nown='ignore')),
            ('Classifier', ensemble.RandomForestClassifier())
           ])
flow = oml.flows.sklearn_to_flow(pipe)

run = oml.runs.run_flow_on_task(task, flow)
myrun = run.publish()
print("Uploaded to http://www.openml.org/r/" + str(myrun.run_id))
```

```
Uploaded to http://www.openml.org/r/10155004
```

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

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

In [38]:

```
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, F
alse, True, True, True, True, True, True, True, False]
```

Out[38]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=11811, verbose=0,
            warm_start=False)
```

Benchmarking

In [64]:
```
import openml as oml
from sklearn import neighbors, linear_model

for task_id in [14951,10103]:
    task = oml.tasks.get_task(task_id)
    data = oml.datasets.get_dataset(task.dataset_id)
    clf = neighbors.KNeighborsClassifier(n_neighbors=5)
    flow = oml.flows.sklearn_to_flow(clf)

    try:
        run = oml.runs.run_flow_on_task(task, flow)
        myrun = run.publish()
        print("kNN on %s: http://www.openml.org/r/%d" % (data.name, myrun.run_id))
    except oml.exceptions.PyOpenMLError as err:
        print("OpenML: {0}".format(err))
```

```
OpenML: Run already exists in server. Run id(s): {10154944}
OpenML: Run already exists in server. Run id(s): {10154945}
```

Benchmarking suites

- Curated collections of tasks for benchmarking
- Run any model or pipeline on all tasks
- Frictionless evaluation and sharing

In [65]:

```
benchmark_suite = oml.study.get_study('OpenML-CC18','tasks')
clf = sklearn.linear_model.LogisticRegression()
for task_id in benchmark_suite.tasks[0:2]: # take small subset for this example
    run = runs.run_model_on_task(clf, tasks.get_task(task_id))
    score = run.get_metric_fn(sklearn.metrics.accuracy_score)
    print('Data set: %s; Accuracy: %0.2f' % (task.get_dataset().name,score.mean
()))
    # run.publish()
```

```
Data set: volcanoes-a1; Accuracy: 0.96
Data set: volcanoes-a1; Accuracy: 0.72
```

Further information

That's it. You are now an expert in using OpenML. In case you have further questions:

OpenML Documentation (https://docs.openml.org)
Python API Documentation and examples (https://openml.github.io/openml-python)