```
[57]: %matplotlib inline
      from IPython.display import set_matplotlib_formats, display, HTML
      HTML('''<style>html, body{overflow-y: visible !important} .CodeMirror{min-
```

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

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

Installation

Install the OpenML developer version
'pip install openml' coming up (october 2017) pip install git+https://github.com/renatopp/liac-arff@master pip install git+https://github.com/openml/openml-python.git@develop

Authentication

- You need to authenticate to upload datasets, experiments,... to the server
- 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 'YOURKEY' with your API key.

```
[28]: # Uncomment and set your OpenML key. Don't share your key with others.
      import openml as oml
      #oml.config.apikey = 'YOURKEY'
```

It all starts with data

Explore thousands of datasets, or share your own

List datasets

`datasets.list_datasets()` returns a dictionary with information about all OpenML datasets

[29]:
```
import openml as oml
openml_list = oml.datasets.list_datasets() # Returns a dict

# Show a nice table with some key data properties
import pandas as pd
datalist = pd.DataFrame.from_dict(openml_list, orient='index')
datalist = datalist[['did','name','NumberOfInstances',
                     'NumberOfFeatures','NumberOfClasses']]
print("First 10 of %s datasets..." % len(datalist))
datalist[:10]
```

```
First 10 of 127 datasets...
```

	did	name	NumberOfInstances	NumberOfFeatures	NumberOfClasses
1	1	anneal	898	39	5
2	2	kr-vs-kp	3196	37	2
3	3	letter	20000	17	26
4	4	balance-scale	625	5	3
5	5	mfeat-factors	2000	217	10
6	6	mfeat-fourier	2000	77	10
7	7	breast-w	699	10	2
8	8	mfeat-karhunen	2000	65	10
9	9	mfeat-morphological	2000	7	10
10	10	mfeat-pixel	2000	241	10

Exercise

- Find datasets with more than 10000 examples
- Find a dataset called 'eeg-eye-state'
- Find all datasets with more than 50 classes

[30]:
```
datalist[datalist.NumberOfInstances>10000
         ].sort(['NumberOfInstances'])[:20]
```

	did	name	NumberOfInstances	NumberOfFeatures	\
61	61	artificial-characters	10218	8	
71	71	har	10299	562	
51	51	jm1	10885	22	

```
18    18                pendigits              10992                 17
92    92         PhishingWebsites              11055                 31
70    70                gas-drift              13910                129
45    45           sylva_agnostic              14395                217
68    68            eeg-eye-state              14980                 15
115  115                      pol              15000                 49
48    48                 mozilla4              15545                  6
58    58           MagicTelescope              19020                 12
3      3                   letter              20000                 17
91    91   Amazon_employee_access              32769                 10
75    75                    nomao              34465                119
60    60   Click_prediction_small              39948                 12
62    62           bank-marketing              45211                 17
28    28              electricity              45312                  9
85    85    tamilnadu-electricity              45781                  4
89    89                    adult              48842                 15
56    56       KDDCup09_upselling              50000                231

     NumberOfClasses
61                10
71                 6
51                 2
18                10
92                 2
70                 6
45                 2
68                 2
115               11
48                 2
58                 2
3                 26
91                 2
75                 2
60                 2
62                 2
28                 2
85                20
89                 2
56                 2
```

[31]:
```
datalist.query('name == "eeg-eye-state"')
```

```
    did           name  NumberOfInstances  NumberOfFeatures  NumberOfClasses
68   68  eeg-eye-state              14980                15                2
```

[32]:
```
datalist.query('NumberOfClasses > 50')
```

```
     did                        name  NumberOfInstances  NumberOfFeatures  \
78    78   one-hundred-plants-margin               1600                65
79    79    one-hundred-plants-shape               1600                65
```

```
80    80  one-hundred-plants-texture              1599                 65
103  103                        kin8nm              8192                  9
104  104                     mbagrade                61                  3
105  105                    wisconsin               194                 33
109  109                   auto_price               159                 16
110  110                      autoMpg               398                  8
111  111                      cpu_act              8192                 22
114  114                          pbc               418                 19
116  116                    autoHorse               205                 26
117  117                       lowbwt               189                 10
118  118                  cholesterol               303                 14
120  120                    triazines               186                 61
121  121                    autoPrice               159                 16
124  124                        cloud               108                  7
127  127                      pharynx               195                 12

     NumberOfClasses
78               100
79               100
80               100
103             8191
104               57
105               94
109              145
110              129
111               56
114              399
116               60
117              133
118              152
120              102
121              145
124               94
127              177
```

Download datasets

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

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

      # Print a summary
      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/v1/download/1587924/eeg-eye-state.ARFF
**Author**: Oliver Roesler
**Source**: [UCI](https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State), Baden-
```

```
**Please cite**: [UCI](https://archive.ics.uci.edu/ml/citation_policy.html)

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

Get the actual data.
Returned as numpy array, with meta-info (e.g. target feature, feature names,...)

```
[34]: 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	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
8	4202.049805	4273.850098	4627.180176	4389.740234	0
9	4212.819824	4277.950195	4637.439941	4393.330078	0

Train machine learning models

Exercise: Train a scikit-learn model (e.g. `KNeighborsClassifier`) on the data manually

```
[35]: from sklearn import neighbors
      dataset = oml.datasets.get_dataset(1471)
      X, y = dataset.get_data(target=dataset.default_target_attribute)

[36]: 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 for meta-data to automatically preprocess the data - e.g. categorical features -> do feature encoding

```
[37]: 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')
```

Upload datasets

- Option 1: Via web upload form on www.openml.org
  - Requires conversion to ARFF data format
  - See the `csv2arff.py` script to convert CSV to ARFF
- Option 2: Upload programmatically:

```
[38]: # Switch to test serve to avoid uploading datasets to the main server by a
      # Uncomment or set to main server when everthing looks good
      oml.config.server = 'https://test.openml.org/api/v1'
      my_data = oml.datasets.OpenMLDataset(data_file='data/ram_price.arff', name
                                           version='1', tag='test', format='ARFF
                                           default_target_attribute='price', dat
      response = my_data.publish()
      print("Uploaded to http://test.openml.org/d/" + str(response.dataset_id))
```

```
Uploaded to http://test.openml.org/d/1266
```

```
[39]: # Return to main server
      oml.config.server = 'https://www.openml.org/api/v1'
```

Tasks: set your own goals

and invite others to work on the same problem

- Option 1: Via task creation form ion your dataset's page
  - Click `Define a new task`
  - Choose task type (e.g. classification)
  - Set required inputs
- Option 2: Programmatically
  - Will be available soon

Listing tasks

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

      mytasks = pd.DataFrame(task_list).transpose()
      mytasks = mytasks[['tid','did','name','task_type','estimation_procedure','
      #print(mytasks.columns)
      print("First 5 of %s tasks:" % len(mytasks))
      print(mytasks.head())
```

```
First 5 of 5000 tasks:
  tid did        name                  task_type     estimation_procedure  \
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
6   6   6      letter  Supervised Classification  10-fold Crossvalidation

   evaluation_measures
2  predictive_accuracy
3  predictive_accuracy
4  predictive_accuracy
5  predictive_accuracy
6  predictive_accuracy
```

Exercise

Search for the tasks on the 'eeg-eye-state' dataset

```
[48]: 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

```
[49]: from pprint import pprint
      task = oml.tasks.get_task(3954)
      pprint(vars(task))

{'class_labels': ['g', 'h'],
 'cost_matrix': None,
 'dataset_id': 1120,
 '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': 'predictive_accuracy',
 'split': None,
 'target_name': 'class:',
 'task_id': 3954,
 'task_type': 'Supervised Classification',
 'task_type_id': 1}
```

Runs: Easily explore models by running them on tasks

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

```
[43]: 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)
```

Share the run on the OpenML server

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

```
Uploaded to http://www.openml.org/r/7932216
```

It also works with pipelines

When you need to handle 'dirty' data, build pipelines to model then automatically

```
[59]: 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())
                ])
      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/7932221
```

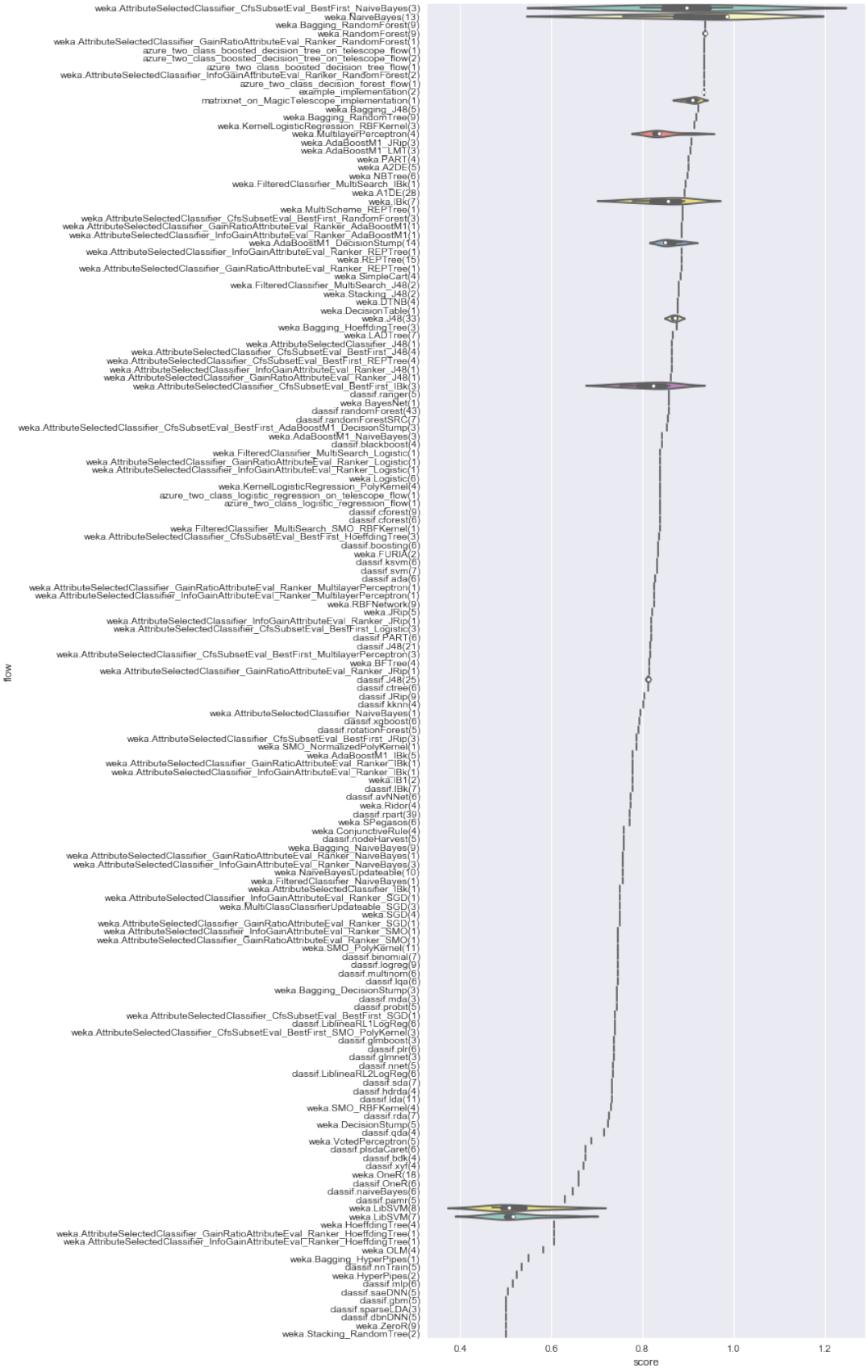
Download previous results

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

```
[60]: import seaborn as sns
      import pandas as pd
      from matplotlib import pyplot
      from openml import evaluations

      # Get the list of runs for task 3954
      evaluations = oml.evaluations.list_evaluations(task=[3954], function='area
      print(len(evaluations))
      # Download the tasks and plot the scores
      scores = []
      for id, e in evaluations.items():
          scores.append({"flow":e.flow_name, "score":e.value})

      sorted_score = sorted(scores, key=lambda x: -x["score"])
      fig, ax = pyplot.subplots(figsize=(8, 25))
      sns.violinplot(ax=ax, x="score", y="flow", data=pd.DataFrame(sorted_score)
```


weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_NaiveBayes(3)
weka.NaiveBayes(13)
weka.Bagging_RandomForest(9)
weka.RandomForest(9)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_RandomForest(1)
azure_two_class_boosted_decision_tree_on_telescope_flow(1)
azure_two_class_boosted_decision_tree_on_telescope_flow(2)
azure_two_class_boosted_decision_tree_flow(1)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_RandomForest(2)
azure_two_class_decision_forest_flow(1)
example_implementation(2)
matrixnet_on_MagicTelescope_implementation(1)
weka.Bagging_J48(5)
weka.Bagging_RandomTree(9)
weka.KernelLogisticRegression_RBFKernel(3)
weka.MultilayerPerceptron(4)
weka.AdaBoostM1_JRip(3)
weka.AdaBoostM1_LMT(3)
weka.PART(4)
weka.A2DE(5)
weka.NBTree(6)
weka.FilteredClassifier_MultiSearch_IBk(1)
weka.A1DE(28)
weka.IBk(7)
weka.MultiScheme_REPTree(1)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_RandomForest(3)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_AdaBoostM1(1)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_AdaBoostM1(1)
weka.AdaBoostM1_DecisionStump(14)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_REPTree(1)
weka.REPTree(15)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_REPTree(1)
weka.SimpleCart(4)
weka.FilteredClassifier_MultiSearch_J48(2)
weka.Stacking_J48(2)
weka.DTNB(4)
weka.DecisionTable(1)
weka.J48(33)
weka.Bagging_HoeffdingTree(3)
weka.LADTree(7)
weka.AttributeSelectedClassifier_J48(1)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_J48(4)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_REPTree(4)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_J48(1)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_J48(1)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_IBk(3)
dassif.ranger(5)
weka.BayesNet(1)
dassif.randomForest(43)
dassif.randomForestSRC(7)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_AdaBoostM1_DecisionStump(3)
weka.AdaBoostM1_NaiveBayes(3)
dassif.blackboost(4)
weka.FilteredClassifier_MultiSearch_Logistic(1)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_Logistic(1)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_Logistic(1)
weka.Logistic(6)
weka.KernelLogisticRegression_PolyKernel(4)
azure_two_class_logistic_regression_on_telescope_flow(1)
azure_two_class_logistic_regression_flow(1)
dassif.cforest(9)
dassif.cforest(6)
weka.FilteredClassifier_MultiSearch_SMO_RBFKernel(1)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_HoeffdingTree(3)
dassif.boosting(6)
weka.FURIA(2)
dassif.ksvm(6)
dassif.svm(7)
dassif.ada(6)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_MultilayerPerceptron(1)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_MultilayerPerceptron(1)
weka.RBFNetwork(9)
weka.JRip(5)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_JRip(1)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_Logistic(3)
dassif.PART(6)
dassif.J48(21)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_MultilayerPerceptron(3)
weka.BFTree(4)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_JRip(1)
dassif.J48(25)
dassif.ctree(6)
dassif.JRip(9)
dassif.kknn(4)
weka.AttributeSelectedClassifier_NaiveBayes(1)
dassif.xgboost(6)
dassif.rotationForest(5)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_JRip(3)
weka.SMO_NormalizedPolyKernel(1)
weka.AdaBoostM1_IBk(5)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_IBk(1)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_IBk(1)
weka.IB1(2)
dassif.IBk(7)
dassif.avNNet(6)
weka.Ridor(4)
dassif.rpart(39)
weka.SPegasos(6)
weka.ConjunctiveRule(4)
dassif.nodeHarvest(5)
weka.Bagging_NaiveBayes(9)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_NaiveBayes(1)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_NaiveBayes(3)
weka.NaiveBayesUpdateable(10)
weka.FilteredClassifier_NaiveBayes(1)
weka.AttributeSelectedClassifier_IBk(1)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_SGD(1)
weka.MultiClassClassifierUpdateable_SGD(3)
weka.SGD(4)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_SGD(1)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_SMO(1)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_SMO(1)
weka.SMO_PolyKernel(11)
dassif.binomial(7)
dassif.logreg(9)
dassif.multinom(6)
dassif.lqa(6)
weka.Bagging_DecisionStump(3)
dassif.mda(3)
dassif.probit(5)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_SGD(1)
dassif.LiblineaRL1LogReg(6)
weka.AttributeSelectedClassifier_CfsSubsetEval_BestFirst_SMO_PolyKernel(3)
dassif.glmboost(3)
dassif.plr(6)
dassif.glmnet(3)
dassif.nnet(5)
dassif.LiblineaRL2LogReg(6)
dassif.sda(7)
dassif.hdrda(4)
dassif.lda(11)
weka.SMO_RBFKernel(4)
dassif.rda(7)
weka.DecisionStump(5)
dassif.qda(4)
weka.VotedPerceptron(5)
dassif.plsdaCaret(6)
dassif.bdk(4)
dassif.xyf(4)
weka.OneR(18)
dassif.OneR(6)
dassif.naiveBayes(6)
dassif.pamr(5)
weka.LibSVM(8)
weka.LibSVM(7)
weka.HoeffdingTree(4)
weka.AttributeSelectedClassifier_GainRatioAttributeEval_Ranker_HoeffdingTree(1)
weka.AttributeSelectedClassifier_InfoGainAttributeEval_Ranker_HoeffdingTree(1)
weka.OLM(4)
weka.Bagging_HyperPipes(1)
dassif.nnTrain(5)
weka.HyperPipes(2)
dassif.mlp(6)
dassif.saeDNN(5)
dassif.gbm(5)
dassif.sparseLDA(3)
dassif.dbnDNN(5)
weka.ZeroR(9)
weka.Stacking_RandomTree(2)
flow
0.4
0.6
0.8
1.0
1.2
score

Easy benchmarking:

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

      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.
          except oml.exceptions.PyOpenMLError as err:
              print("OpenML: {0}".format(err))
```

```
kNN on eeg-eye-state: http://www.openml.org/r/7932218
kNN on volcanoes-a1: http://www.openml.org/r/7932219
```

A Challenge

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
- Your own uploaded dataset

```
[ ]:
```