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

- `pip install openml`

In [ ]:

```
!pip install openml
```

Authentication

It is important to configure the Python connector with the proper API endpoint (usually good by default) and the proper API key. Find your API key after logging in on OpenML

In [ ]:

```
import openml

openml.config.server = 'https://test.openml.org/api/v1/'
openml.config.apikey = 'FILL_IN'

import warnings
warnings.simplefilter(action="ignore", category=DeprecationWarning)
```

Run model/flow on task

Running a scikit-learn model on a task is done using the function `run_model_on_task(...)` (see docs (https://openml.github.io/openml-python/master/generated/openml.runs.run_model_on_task.html#openml.runs.run_model_on_task)) or `run_flow_on_task(...)`. In particular, review the `avoid_duplicate_run` option (especially important for tutorials). The function `get_metric_fn` (doc (https://openml.github.io/openml-python/master/generated/openml.OpenMLRun.html#openml.OpenMLRun)) can be used to obtain metric scores before uploading.

- Use the function `run_model_on_task` to run your favorite scikit-learn classifier (e.g., a `Random Forest Classifier`) on the `diabetes` dataset. (Hint: there are several ways of obtaining a task from the diabetes dataset). Report the score.
- Use the function `run_flow_on_task` to run another scikit-learn classifier on the `diabetes` dataset. Report the score.

In [ ]:

```
import sklearn.ensemble

task_list = openml.tasks.list_tasks(data_name='diabetes', size=1)
task_id = list(task_list.keys())[0]
task = openml.tasks.get_task(task_id)
clf = sklearn.ensemble.RandomForestClassifier()

run = openml.runs.run_model_on_task(clf, task)
run = run.publish()
scores = run.get_metric_fn(sklearn.metrics.accuracy_score)
print('Uploaded with run id=%d; score=%s' % (run.run_id, scores.mean()))
```

In [ ]:

```
flow = openml.flows.sklearn_to_flow(clf)
run = openml.runs.run_flow_on_task(flow, task, avoid_duplicate_runs=False)
run = run.publish()
scores = run.get_metric_fn(sklearn.metrics.accuracy_score)
print('Uploaded with run id=%d; score=%s' % (run.run_id, scores.mean()))
```

Random Search and Grid Search

Scikit-learn natively supports Random Search and Grid Search procedures, to optimize the hyperparameters. These classifiers can natively be used using the openml connector. Read this article (https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html) to understand how these work.

- Run Random Search and Grid Search on a SVM from scikit-learn. Make sure to optimize at least 2 hyperparameters. What are the most important hyperparameters? What is the main difference between these two classifiers?

In [ ]:

```
import sklearn.model_selection
import sklearn.svm

param_dist = {
    'C': [0.0001, 0.001, 0.01, 0.1, 1],
    'gamma': [0.0001, 0.001, 0.01, 0.1, 1],
}
base = sklearn.svm.SVC()

clf = sklearn.model_selection.RandomizedSearchCV(
    base, param_distributions=param_dist, n_iter=10
)

run = openml.runs.run_model_on_task(clf, task, avoid_duplicate_runs=False)
scores = run.get_metric_fn(sklearn.metrics.accuracy_score)
print('score=%s' % (scores.mean()))
```

In [ ]:

```
import sklearn.model_selection
import sklearn.svm

param_dist = {
    'C': [0.0001, 0.001, 0.01, 0.1, 1],
    'gamma': [0.0001, 0.001, 0.01, 0.1, 1],
}
base = sklearn.svm.SVC()

clf = sklearn.model_selection.GridSearchCV(
    base, param_grid=param_dist
)

run = openml.runs.run_model_on_task(clf, task, avoid_duplicate_runs=False)
scores = run.get_metric_fn(sklearn.metrics.accuracy_score)
print('score=%s' % (scores.mean()))
```

ColumnTransformer and Pipelines

Note that we did the previos examples on the `diabetes` dataset. This is a particular nice dataset, as it only contains numeric features and no missing values. In many cases, we have to deal with complicated workflows. For example, the `credit-a` dataset mixes categorical and numeric features, and contains missing values.

- verify that our previously used classifier does not work on the `credit-a` dataset. What is the reason for this?
- review the following scikit-learn components:
  - ColumnTransformer (https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html)
  - Pipeline (https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html)
  - SimpleImputer (https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html)
  - OneHotEncoder (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)
  - StandardScaler (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)
  - Remember that in order to make a flow compatible with OpenML, there can be no duplicate polymorph classifiers
- create a generat classifier that runs on the `credit-a` (or in general each) dataset. Note that the function `get_features_by_type` from the OpenMLDataset (https://openml.github.io/openml-python/master/generated/openml.OpenMLDataset.html#openml.OpenMLDataset) object can prove useful.

In [ ]:

```
task_list = openml.tasks.list_tasks(data_name='credit-a', size=1)
task_id = list(task_list.keys())[0]
task = openml.tasks.get_task(task_id)
clf = sklearn.ensemble.RandomForestClassifier()

try:
  run = openml.runs.run_model_on_task(clf, task)
  # Note that this is supposed to throw an error
except ValueError as e:
  print('Found error: %s' % e)
```

In [ ]:

```
import sklearn.preprocessing
import sklearn.pipeline
import sklearn.feature_selection
import sklearn.compose
import sklearn.impute


nominal_indices = task.get_dataset().get_features_by_type('nominal', [task.targe
t_name])
numeric_indices = task.get_dataset().get_features_by_type('numeric', [task.targe
t_name])

numeric_transformer = sklearn.pipeline.make_pipeline(
      sklearn.preprocessing.Imputer(),
      sklearn.preprocessing.StandardScaler())

# note that the dataset is encoded numerically, hence we can only impute
# numeric values, even for the categorical columns.
categorical_transformer = sklearn.pipeline.make_pipeline(
      sklearn.impute.SimpleImputer(strategy='constant', fill_value=-1),
      sklearn.preprocessing.OneHotEncoder(handle_unknown='ignore'))

transformer = sklearn.compose.ColumnTransformer(
      transformers=[
          ('numeric', numeric_transformer, numeric_indices),
          ('nominal', categorical_transformer, nominal_indices)],
      remainder='passthrough')

clf = sklearn.pipeline.make_pipeline(transformer,
                                     sklearn.feature_selection.VarianceThreshold
(),
                                     sklearn.ensemble.RandomForestClassifier())

run = openml.runs.run_model_on_task(clf, task, avoid_duplicate_runs=False)
scores = run.get_metric_fn(sklearn.metrics.accuracy_score)
print('score=%s' % (scores.mean()))
```

Pipelines, Columntransformers and Random Search

Combine Pipelines, ColumnTransformers and RandomSearchCV to work on any dataset (in particular the `credit-a` dataset). Note that the parameter distribution parameter needs to be adjusted.

In [ ]:

```
import sklearn.preprocessing
import sklearn.pipeline
import sklearn.feature_selection
import sklearn.compose
import sklearn.impute


nominal_indices = task.get_dataset().get_features_by_type('nominal', [task.targe
t_name])
numeric_indices = task.get_dataset().get_features_by_type('numeric', [task.targe
t_name])

numeric_transformer = sklearn.pipeline.make_pipeline(
      sklearn.preprocessing.Imputer(),
      sklearn.preprocessing.StandardScaler())

# note that the dataset is encoded numerically, hence we can only impute
# numeric values, even for the categorical columns.
categorical_transformer = sklearn.pipeline.make_pipeline(
      sklearn.impute.SimpleImputer(strategy='constant', fill_value=-1),
      sklearn.preprocessing.OneHotEncoder(handle_unknown='ignore'))

transformer = sklearn.compose.ColumnTransformer(
      transformers=[
          ('numeric', numeric_transformer, numeric_indices),
          ('nominal', categorical_transformer, nominal_indices)],
      remainder='passthrough')


param_dist = {
    'svc__C': [0.0001, 0.001, 0.01, 0.1, 1],
    'svc__gamma': [0.0001, 0.001, 0.01, 0.1, 1],
}
base = sklearn.svm.SVC()

clf = sklearn.pipeline.make_pipeline(transformer,
                                     sklearn.feature_selection.VarianceThreshold
(),
                                     base)

search = sklearn.model_selection.RandomizedSearchCV(
    clf, param_distributions=param_dist, n_iter=10
)

run = openml.runs.run_model_on_task(search, task, avoid_duplicate_runs=False)
scores = run.get_metric_fn(sklearn.metrics.accuracy_score)
run = run.publish()

print(run.run_id)
```