Kickoff ExaDost, Paris 2023.09.13

Scaling up Scikit-Learn for Exascale

Scikit-learn:

The reference library for ML on tabular data

- Many reference algorithms (no deep learning)
- Tools designed to create ML Pipelines (preprocessing)
- Utilities to perform hyperparameter optimization
- Tools to evaluate the results

Scikit-learn:

The reference library for ML on tabular data

Current contraints

- Input are pandas or numpy arrays
- Parallelism is done using joblib
- No GPU acceleration (for now)

A zoom in the future development of Scikit-Learn

I - Parallelism in scikit-learn

II – The Array API

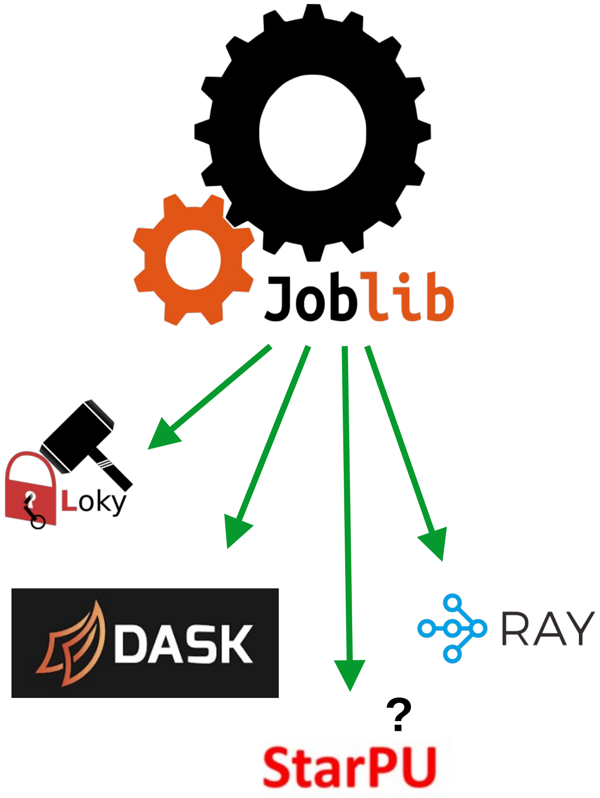
III - Exploring a plugin system in scikit-learn

A high level API with joblib:

```
>>> from math import sqrt
>>> [sqrt(i ** 2) for i in range(10)]
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

```
>>> from math import sqrt
>>> from joblib import Parallel, delayed
>>> Parallel(n_jobs=2)(delayed(sqrt)(i ** 2) for i in range(10))
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

```
>>> from joblib import parallel_config
>>> with parallel_config(backend='threading', n_jobs=2):
...     Parallel()(delayed(sqrt)(i ** 2) for i in range(10))
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```



> Embarassingly Parallel tasks, with pluggable backends

Low-level parallelism:

Calls to BLAS (through numpy) are parallelised

Low level parallelism using openMP

> Intrinsic parallelism, dependent on the data structure and the algorithm.

Challenge: sometime hard to make the different parallelism levels cohexist!

A zoom in the future development of Scikit-Learn

I - Parallelism in scikit-learn

II – The Array API

III - Exploring a plugin system in scikit-learn

Introduction to the array API standard:

```python
import numpy as np
data = np.arange(9).reshape(3, 3)
# with numpy, only get one output
# uses keyword `axis`
max_over_cols = np.max(data, axis=1)
```

Introduction to the array API standard:

```python
import numpy as np
data = np.arange(9).reshape(3, 3)
# with numpy, only get one output
# uses keyword `axis`
max_over_cols = np.max(data, axis=1)
```

```python
import torch
data = torch.arange(9).reshape(3, 3)
# with pytorch, `torch.max` on nd tensors
# returns a length 2 tuple
# `max` expects parameter `dim`
max_over_cols, _ = torch.max(data, dim=1)
```

Introduction to the array API standard:

```python
import numpy as np
data = np.arange(9).reshape(3, 3)
# with numpy, only get one output
# uses keyword `axis`
max_over_cols = np.max(data, axis=1)
```

```python
import torch
data = torch.arange(9).reshape(3, 3)
# with pytorch, `torch.max` on nd tensors
# returns a length 2 tuple
# `max` expects parameter `dim`
max_over_cols, _ = torch.max(data, dim=1)
```

Different array libraries can have different syntax for the same operation !

Introduction to the array API standard:

The Array API enables writing **agnostic single-source code** and interface it with all arrays !

```python
from array_api_compat import get_namespace
def max_over_cols_fn(data):
    xp = get_namespace(data)
    return xp.max(data, axis=1)
```

Introduction to the array API standard:

The Array API enables writing **agnostic single-source code** and interface it with all arrays !

```python
from array_api_compat import get_namespace
def max_over_cols_fn(data):
    xp = get_namespace(data)
    return xp.max(data, axis=1)
```

```python
import numpy as np
data = np.arange(9).reshape(3, 3)
max_over_cols = max_over_cols_fn(data)
```

```python
import torch
data = torch.arange(9).reshape(3, 3)
max_over_cols  = max_over_cols_fn(data)
```

Array API spec excerpt:

- Assumptions
- **API**
- Design topics & constraints
- Future API standard evolution
- API specification
  - Array object
  - Broadcasting
  - Constants
  - Creation Functions
  - Data Type Functions
  - Data Types
  - Element-wise Functions
  - Function and method signatures
  - Indexing
  - Indexing Functions
  - Linear Algebra Functions
  - Manipulation Functions
  - Searching Functions

A conforming implementation of the array API standard must provide and support the

Objects in API

`broadcast_arrays`(*arrays)	Broadcasts one or m
`broadcast_to`(x, /, shape)	Broadcasts an array
`concat`(arrays, /, *, axis=0)	Joins a sequence of
`expand_dims`(x, /, *, axis=0)	Expands the shape c (dimension) of size (
`flip`(x, /, *, axis=None)	Reverses the order c
`permute_dims`(x, /, axes)	Permutes the axes (
`reshape`(x, /, shape, *, copy=None)	Reshapes an array w
`roll`(x, /, shift, *, axis=None)	Rolls array elements
`squeeze`(x, /, axis)	Removes singleton c
`stack`(arrays, /, *, axis=0)	Joins a sequence of

https://data-apis.org/array-api/latest/API_specification/index.html

Compatible array libraries:
(done or ongoing)
(non-exhaustive)

NumPy
CPU

PyTorch
CPU
GPU AMD/INTEL/NVIDIA
OTHERS

dask
CPU
DISTRIBUTED

Dpctl + DPNP
CPU
GPU INTEL

CuPy
GPU NVIDIA

Toward Array API compliancy in scikit-learn:

When possible, use the Array Api rather than numpy calls !

```
636 -         np.square(X, out=X)
637 -         np.sum(X, axis=0, out=X[0])
638 -         total_var = (X[0] / N).sum()
639
640           self.explained_variance_ratio_ =
      self.explained_variance_ / total_var
641 -         self.singular_values_ = S.copy()  # Store the singular
      values.
642
643           if self.n_components_ < min(n_features, n_samples):
644 -             self.noise_variance_ = total_var -
      self.explained_variance_.sum()
```

```
659 +         X **= 2
660 +         total_var = xp.sum(xp.sum(X, axis=0) / N)

661
662           self.explained_variance_ratio_ =
      self.explained_variance_ / total_var
663 +         self.singular_values_ = xp.asarray(S, copy=True)  #
      Store the singular values.
664
665           if self.n_components_ < min(n_features, n_samples):
666 +             self.noise_variance_ = total_var -
      xp.sum(self.explained_variance_)
```

Excerpt of https://github.com/scikit-learn/scikit-learn/pull/26315
ENH Array API support for PCA

Towards Array API compliancy in scikit-learn:

- ✓ sklearn.lda.LDA
- ✓ sklearn.decomposition.PCA (next release)
- sklearn.preprocessing.MinMaxScaler (ongoing)
- Discussions and roadmaps #22352 #26024 #26083

Towards Array API compliancy in scikit-learn:

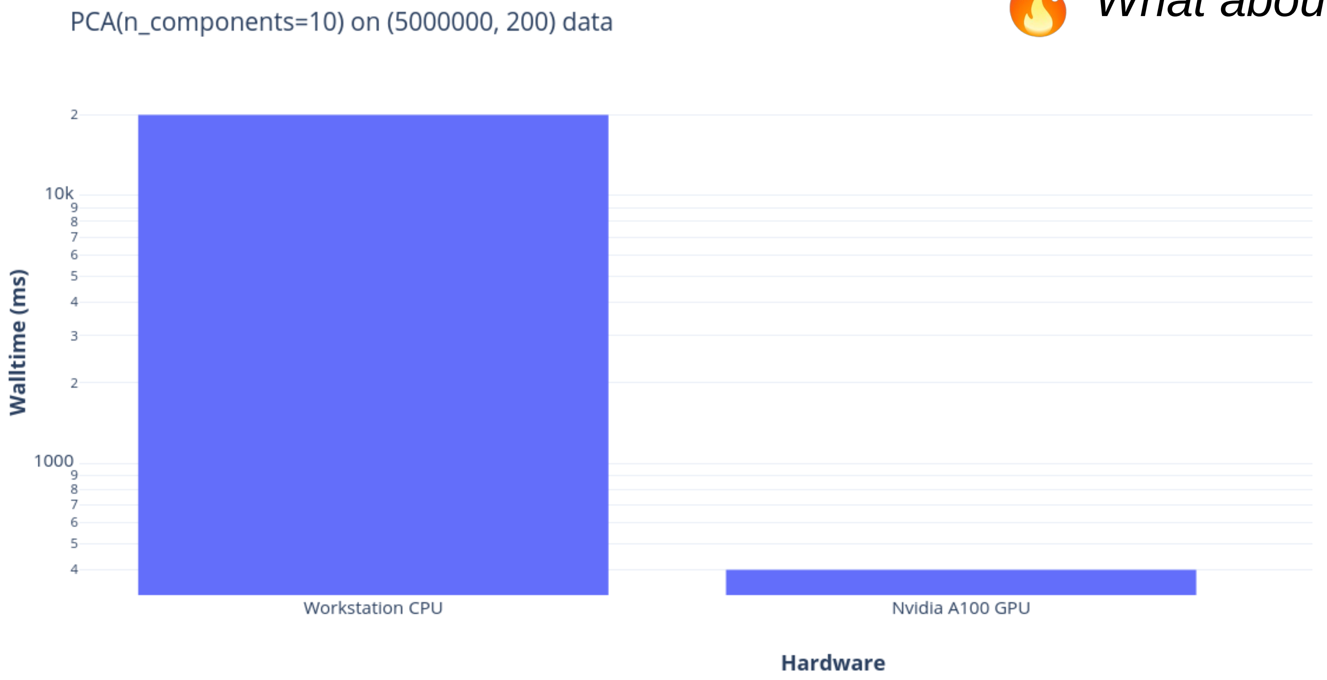
🔥 *How to use it ?* 🔥

```python
from sklearn import config_context
from sklearn.decomposition import PCA
pca_estimator = PCA(n_components=10)
with config_context(array_api_dispatch=True):
    pca_estimator.fit(X)
```

⚠️ Experimental features require explicit activation in scikit-learn config class

```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
with config_context(array_api_dispatch=True):
    lda.fit(X)
```

Towards Array API compliancy in scikit-learn:



What about performance ?

YES when CPU takes too long
40x speedup with pytorch on GPU

NB: pytorch backend for intel GPUs currently lacks optimization for QR factorization, so Intel GPUs are not included in the figure

Towards Array API compliancy in scikit-learn:

🔥 *Will all estimators support all compatible array libraries ?* 🔥

Likely not :-(

Current limitations of the Array API approach:

- ❌ Does not cover all functions for all libraries (torch.topk, np.argpartition, RNG…)
- ❌ Lacks synchronization functions for libraries that evaluate lazily (dask, jax...)
- ❌ Not all algorithms can be implemented (efficiently) with high level array operations !

(random forest, histogram gradient boosting trees,…)

Possible drawbacks for the Array API:

Each call to a high-level array function writes the result into memory.

Example of pairwise_distance + argmin common pattern:

```
import numpy as np
from scipy.spatial.distance import cdist

# create some data
rng = np.random.default_rng(123)
data = rng.random((500000, 200))
query = rng.random((1, 200))

# writes all pairwise distances to memory
pairwise_distances = cdist(query, data)

closest = np.argmin(pairwise_distances)
```

A zoom in the future development of Scikit-Learn

I - Parallelism in scikit-learn

II – The Array API

III - Exploring a plugin system in scikit-learn

Exploring a plugin system in scikit-learn:

> Preview of user workflow:

```
$
$ pip install scikit-learn some-compute-plugin
```

Exploring a plugin system in scikit-learn:

> Preview of user workflow:

```
$
$ pip install scikit-learn some-compute-plugin
```

```python
from sklearn import config_context
from sklearn.cluster import KMeans
kmeans_estimator = KMeans()
with config_context(engine_provider="some_compute_plugin"):
    y_pred = kmeans_estimator.fit_predict(X)
    y_transform = kmeans_estimator.transform(X)
```

Exploring a plugin system in scikit-learn:

> Example for plugin developers: register a custom engine class

```python
class MyKMeansEngine:

    def __init__(self, estimator):
        self.estimator = estimator

    def prepare_fit(self, X, y=None, sample_weight=None):
        "TODO: insert custom engine implementation"

    def kmeans_single(self, X, sample_weight, centers_init):
        "TODO: insert custom engine implementation"

    def prepare_prediction(self, X, sample_weight):
        "TODO: insert custom engine implementation"

    def get_labels(self, X, sample_weight):
        "TODO: insert custom engine implementation"
```

> Test against native scikit-learn unit tests

```
~# pytest --sklearn-engine-provider some-compute-plugin --pyargs sklearn.cluster.tests.test_k_means
```

Exploring a plugin system in scikit-learn:

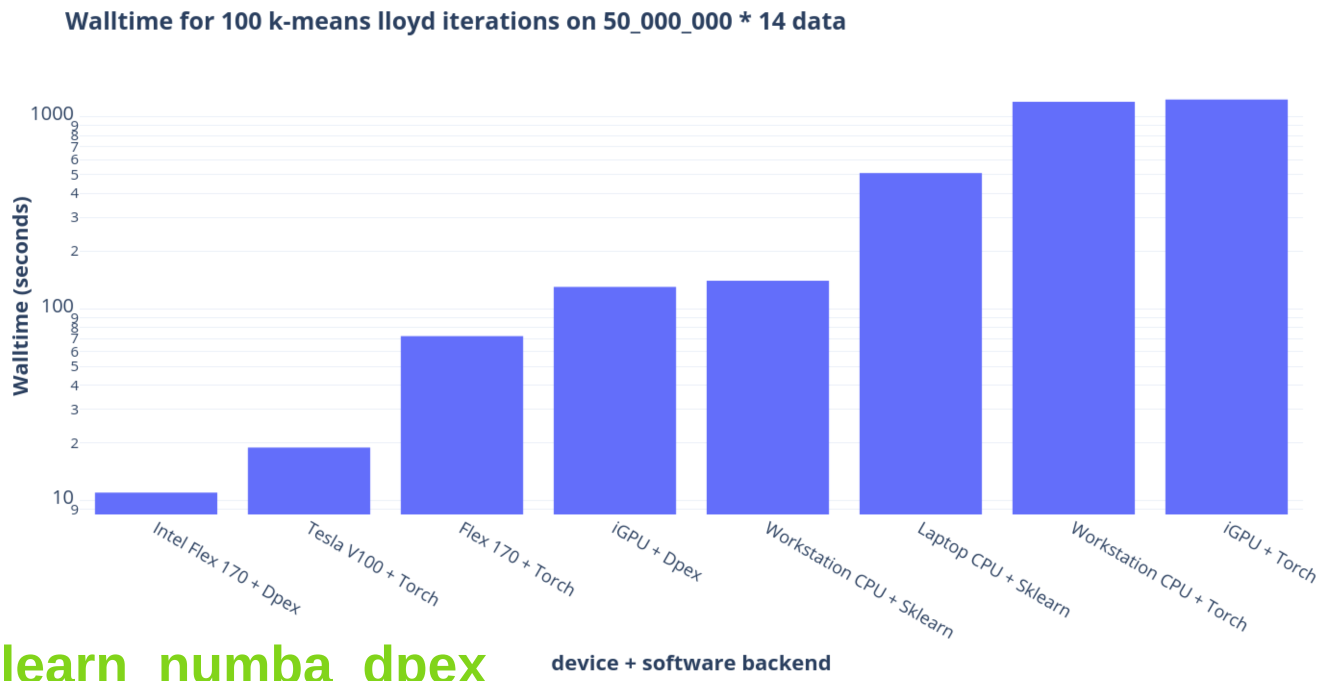
Foundational issue https://github.com/scikit-learn/scikit-learn/issues/22438

Pull request https://github.com/scikit-learn/scikit-learn/pull/25535

Development branch https://github.com/scikit-learn/scikit-learn/tree/feature/engine-api

Example plugin for KMeans on intel GPUs
https://github.com/soda-inria/sklearn-numba-dpex

A plugin for k-means with the oneAPI oneAPI toolchain:



k-means is <u>one order of magnitude faster</u> on GPU, more significantly so when optimized with a low-level implementation

NB: cuml implementation of k-means for Nvidia GPUs could not be included in the figure because it could not scale to this amount of data memory-wise.

sklearn_numba_dpex

https://github.com/soda-inria/sklearn-numba-dpex

ROADMAP in ExaDost:

- 1 Engineer position to work on scikit-learn and joblib (MIND)
  - Work on the lazy Array API
  - Work on online computations with scikit-learn (partial_fit),
  - Work on improving nested parallelism handeling.
- 1 PhD position with H. Hendrikx, (Thot)
  - Improve the distributed algorithms in scikit-learn