High-Performance Machine Learning Using Scikit-Learn

Qiyang Hu
UCLA Office of Advanced Research Computing
May 16th, 2025

Outline

- Learning Scikit-learn basics
  - High-level overview of scikit-learn libraries
  - Practical usage of scikit-learn
    - Deep learning and scikit-learn
    - Scikit-learn extension libraries
- High-performance machine learning using scikit-learn-ish tools
  - Overview of performance issues in machine learning
  - General performance tips and tricks in scikit-learn
  - Making the computation faster
  - Processing large datasets

Outline

- Learning Scikit-learn basics
  - High-level overview of scikit-learn libraries
  - Practical usage of scikit-learn
    - Deep learning and scikit-learn
    - Scikit-learn extension libraries

- High-performance machine learning using scikit-learn-*ish tools*
  - Overview of performance issues in machine learning
  - General performance tips and tricks in scikit-learn
  - Making the computation faster
  - Processing large datasets

Measures of Machine Learning Performance

Training Throughput

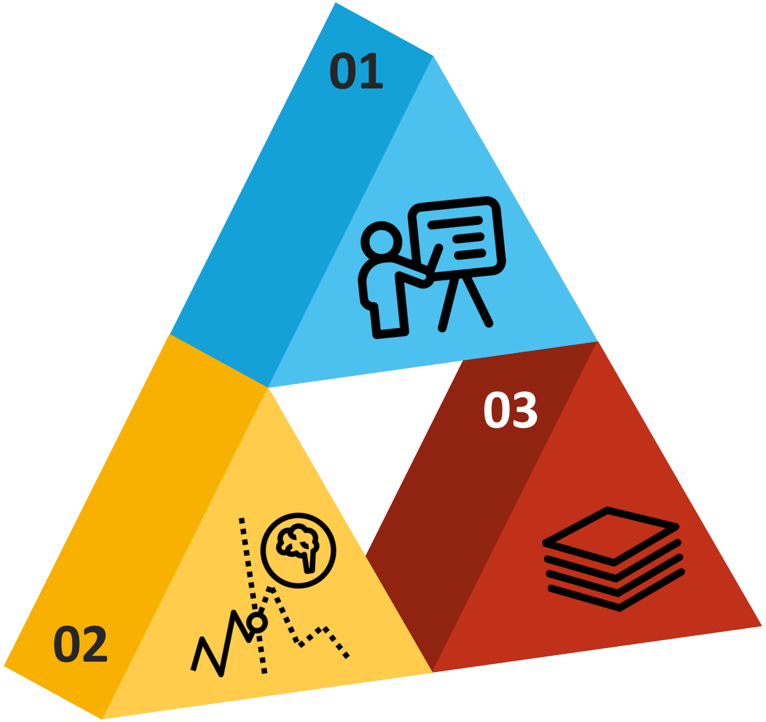
Number of training instances goes through the training processed in unit time

Prediction Latency

Time to make a single prediction taken by a deployed model

Prediction Throughput

Number of predictions made in unit time



Factors Affecting Machine Learning Performance

DATA

- Number of Instances
  - For training
  - For prediction
  - Upfront availability
- Features
  - Number of features
  - Importance of features
  - Feature representation

MODEL

- Algorithm complexity
  - Hyperparameters
  - Optimized libraries
  - Multi-core Parallelism
  - Out-of-core learning
- Data complexity
  - Feature selection
  - Feature extraction
  - Feature transformation
- For prediction:
  - Validation overhead
  - Model compression
  - Model reshaping

Outline

- Learning Scikit-learn basics
    - High-level overview of scikit-learn libraries
    - Practical usage of scikit-learn
        - Deep learning and scikit-learn
        - Scikit-learn extension libraries

- High-performance machine learning using scikit-learn-*ish tools*
    - Overview of performance issues in machine learning
    - General performance tips and tricks in scikit-learn
    - Making the computation faster
    - Processing large datasets

Basic performance tips for scikit-learn projects

03 Writing Cython wrappers

- e.g. liblinear, libsvm in LogReg and SVM models.
- Using OpenMP functions through Cython

02 Using Numpy and Scipy

- Replacing nested for loops
- Performance tips for numpy and scipy

01 Profiling your Python code

Using line_profiler, memory_profiler

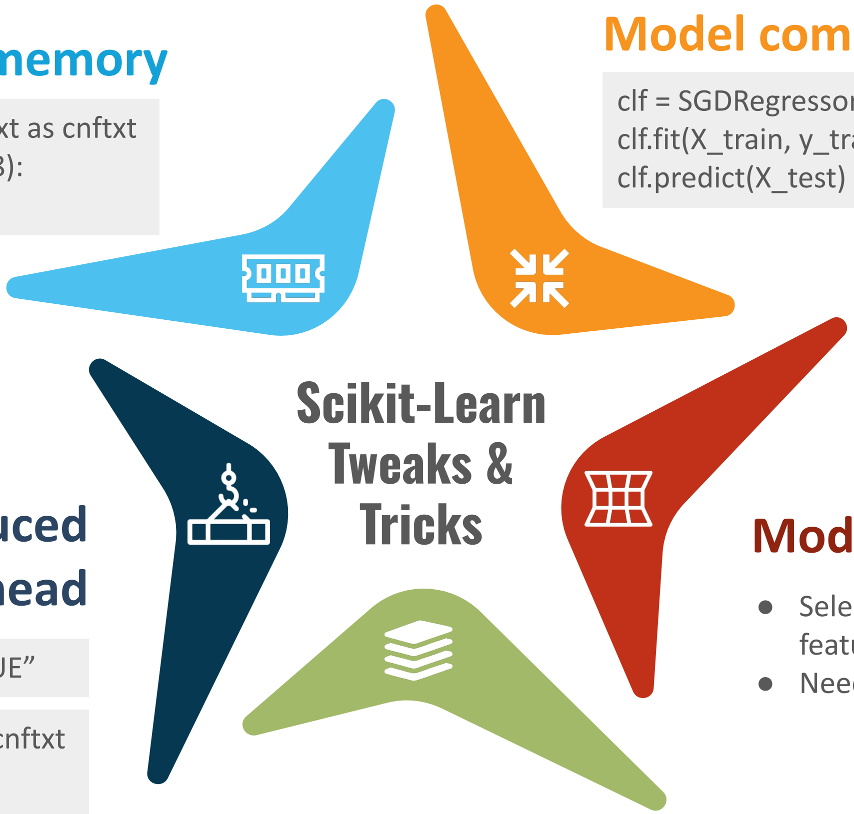
> "The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times…"
>
> —— Donald Knuth in "TAOCP"

Limiting working memory

```
from sklearn import config_context as cnftxt
with cnftxt(working_memory=128):
    pass
```

Model compression

```
clf = SGDRegressor(penalty='elasticnet', l1_ratio=0.25)
clf.fit(X_train, y_train).sparsify()
clf.predict(X_test)
```



Configuring for reduced validation overhead

```
export SKLEARN_ASSUME_FINITE=”TRUE”
```

```
from sklearn import config_context as cnftxt
with cnftxt(assume_finite=True):
    pass
```

Model reshaping

- Selecting only a portion of the available features to fit a model
- Needs to be performed manually

Prediction in bulk mode

Doing predictions many instances at the same time

Qiyang Hu

Outline

- Learning Scikit-learn basics
  - High-level overview of scikit-learn libraries
  - Practical usage of scikit-learn
    - Deep learning and scikit-learn
    - Scikit-learn extension libraries

- High-performance machine learning using scikit-learn-*ish tools*
  - Overview of performance issues in machine learning
  - General performance tips and tricks in scikit-learn
  - Making the computation faster
  - Processing large datasets

Multicore Parallelism and Optimizations for Scikit-Learn

Qiyang Hu

Joblib: lightweight pipelining tools in Python

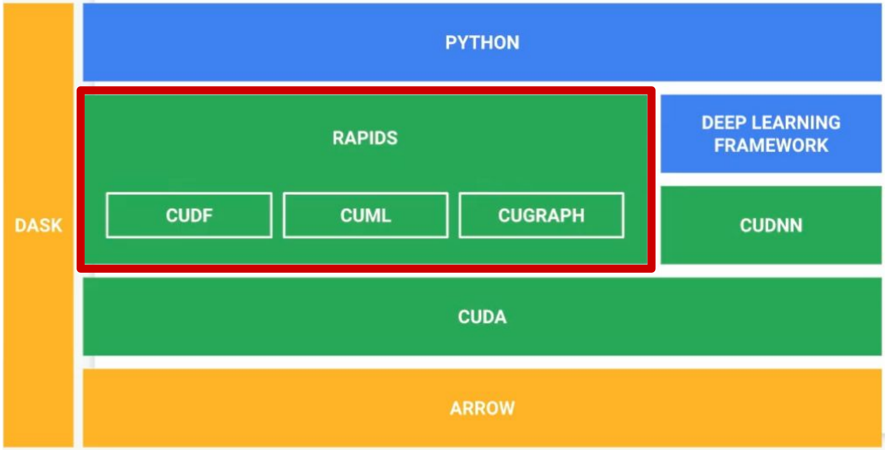
- Speeding up long-running jobs:
  - Specific optimizations for Numpy arrays
  - Caching and lazy evaluation to avoid computing repeatedly
  - No need to change the code or control flow
  - Various parallel backends: `locky`, `multiprocessing`, `threading`, `dask`, `ray`, …
- Tight and powerful integration with Scikit-learn
  - Already used in some sklearn classes (ElasticNet, SGDClassifier, …)
  - Simply tweak `n_jobs`: `n_jobs=-1` to use all available cores
  - Typical tasks: cross-validation, grid search, multi-label prediction, ensemble learning
- Distributed Scikit-Learn using Dask as backend

```
from dask.distributed import Client
import joblib

client = Client(processes=False)             # create local cluster
# client = Client("scheduler-address:8786")  # or connect to remote cluster

with joblib.parallel_backend('dask'):
    # Your scikit-learn code
```

bit.ly/hpskl_01

High-performance ML using accelerators

- Scikit-learn alone doesn't have support to GPU or TPU.
- Option 1: RAPIDS software libraries from nvidia
  - Exactly same APIs
  - Kernels rewritten by CUDA
    - *Numpy -> CuPy*
    - pandas -> cuDF
    - **scikit-learn -> cuML**
    - networkx -> cuGraph
  - Dask + RAPIDS + blazingSQL, RAPIDS + Spark, xgboost, ...



CUML

Accelerated Machine Learning with a scikit-learn API

CPU

Scikit-learn

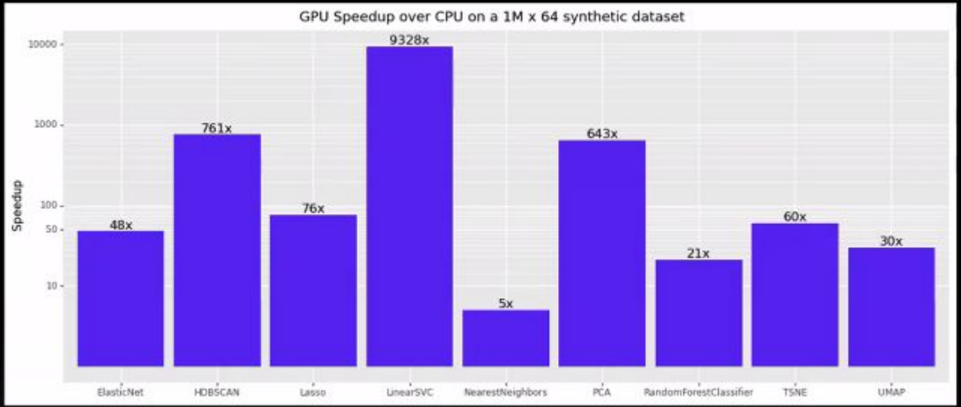
```
>>> from sklearn.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

GPU

cuML

```
>>> from cuml.ensemble import
RandomForestClassifier
>>> clf = RandomForestClassifier()
>>> clf.fit(x, y)
```

50+ GPU-Accelerated Algorithms & Growing



Time Series	Classification	Regression	Clustering	Preprocessing
Cross Validation	Tree Models	Dimensionality Reduction	Explainability	

A100 GPU vs. AMD EPYC 7642 (96 logical cores)
cuML 23.04, scikit-learn 1.2.2, umap-learn 0.5.3

High-performance ML using accelerators (cont’d)

- Option 2: using scikit-learn-intelex
  - Higher-level APIs to daal4py
  - Full conformance with all skearn apis/algos.
  - Acceleration without code change:
    - **Intel** CPU optimization patching

```
from sklearnex import patch_sklearn
patch_sklearn()
```

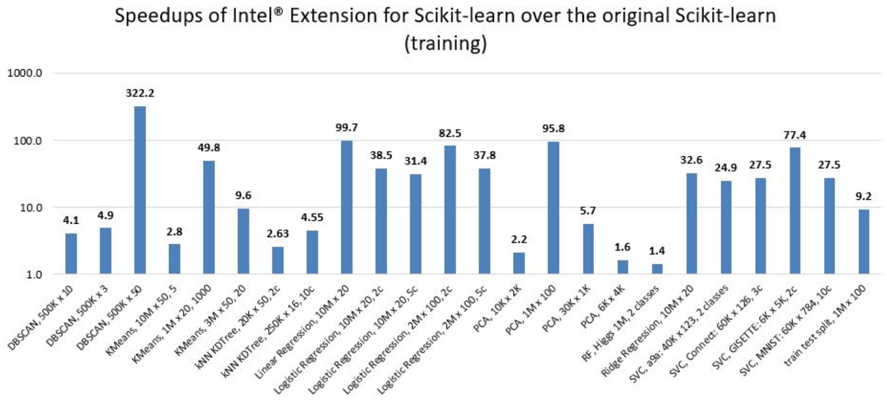
    - **Intel** CPU/GPU optimizations patching

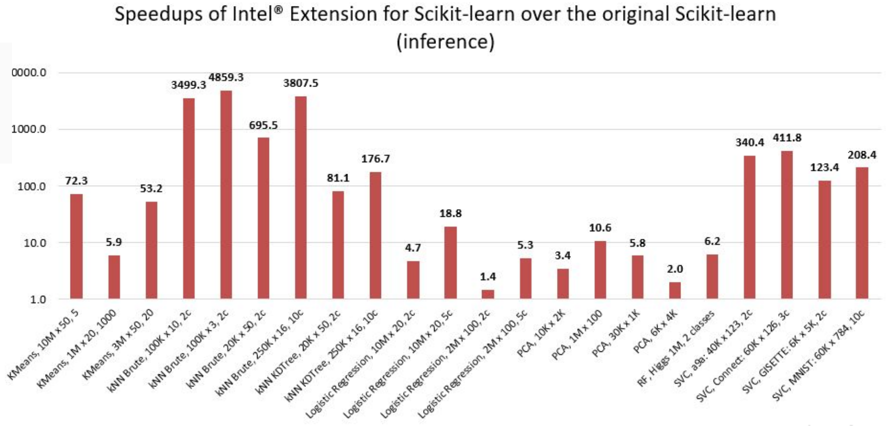
```
from sklearnex import patch_sklearn, config_context
patch_sklearn()
with config_context(target_offload="gpu:0"):
```

  - Installation:

```
pip install scikit-learn-intelex
```

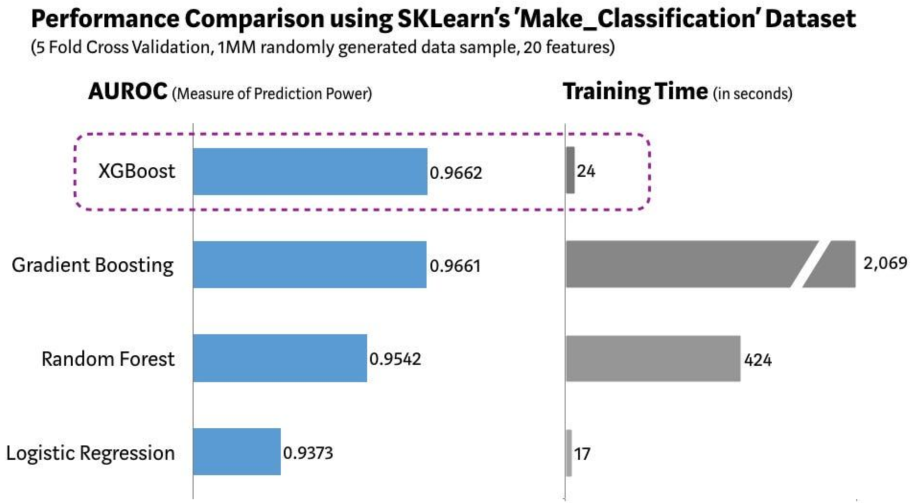
```
pip install dpcpp-cpp-rt
```





High-performance ML using accelerators (cont'd)

- Option 3: XGBoost (eXtreme Gradient Boosting)
  - A specialized ML algorithm & library
    - Processing very big tabular data
    - High accuracy prediction results
    - Very fast!
      - Using 2nd order Taylor approx.
      - Support GPU acceleration
      - Support distributed computation
  - Gold standard for tabular data modeling
    - 70% of winning solutions in Kaggle
    - Widely used in industry
- Siblings within the GBDT family
  - LightGBM: Histogram-based splitting, leaf-wise growth
  - CatBoost: Native categorical feature handling



Source

ACCELERATED XGBOOST

"XGBoost is All You Need" – Bojan Tunguz, 4x Kaggle Grandmaster

CPU

XGBoost

```
>>> from xgboost import XGBClassifier
>>> clf = XGBClassifier()
>>> clf.fit(x, y)
```

GPU

XGBoost

```
>>> from xgboost import XGBClassifier
>>> clf =
XGBClassifier(tree_method="gpu_hist")
>>> clf.fit(x, y)
```

Up to 20x Speedups

- One line of code change to unlock up to 20x speedups with GPUs
- Scalable to the world's largest datasets with Dask and PySpark
- Built-in SHAP support for model explainability
- Deployable with Triton for lighting-fast inference in production
- RAPIDS helps maintain the XGBoost project

RAPIDS ❤ dmlc XGBoost

bit.ly/hpskl_02

Outline

- Learning Scikit-learn basics
  - High-level overview of scikit-learn libraries
  - Practical usage of scikit-learn
    - Deep learning and scikit-learn
    - Scikit-learn extension libraries
- High-performance machine learning using scikit-learn-*ish tools*
  - Overview of performance issues in machine learning
  - General performance tips and tricks in scikit-learn
  - Making the computation faster
  - Processing large datasets

Training on Large Datasets
(*where* data is available upfront for processing)

- Challenges for large datasets
  - Most scikit-learn estimators works for in-memory arrays
  - Needs different dataframes
  - Needs to implement different algorithms
- Dask for Machine Learning
  - Using dask array, dask dataframe
- Machine Learning with vaex.ml
  - Wrappers to scikit-learn, xgboost, LightGBM and CatBoost.
  - Predictive models not implemented yet.
  - Implemented standard data transformers and KMeans algorithms
  - Using Vaex dataframe

bit.ly/hpskl_03

Data in Out-of-core Learning

- Batch vs. Streaming
  - Batch: stored on disk or database, available upfront for processing
  - Streaming: all data streams at different time points, not all available upfront.
- Streaming plays a pivotal role in big data processing
  - Streaming architectures: Lambda vs. Kappa
  - Streaming technologies: Kafka, Flink, Storm, Cloud-based solutions on AWS, GCP, Azure
- Training-Serving Skew
  - Performance difference between training+testing and production
  - Possible causes:
    - Sourcing data from different pipelines for training and prediction
    - Processed in an ad-hoc manner with many short cuts
    - A feedback loop between your model and your algorithm.
  - Solution: to process both training and prediction data as part of the same pipeline

Out-of-core learning using scikit-learn

- Feature extraction for a subset of data
  - Stateful feature extractor to build a “hash table”:
    - Must know the complete feature set known in advance
    - In-memory mapping from the string tokens to the feature indices
  - Stateless feature extractor using hashing tricks:
    - Preprocessing vectorizers having no ‘fit’
    - It creates reduced dimensionality hashes of data

- Incremental learning algorithm
  - Learning without seeing all instances at once
  - Scikit-learn supports through **partial_fit** API
    - Classifiers: Naive Bayes, Perceptron, SGD, Passive-aggressive classifiers
    - Regressors: SGD, Perceptron, Passive-aggressive regressors
    - Clustering: Mini-batch K-means, Birch
    - Feature extraction: Dictionary learning, PCA, Latent Dirichlet Allocation
    - Preprocessing: Standard, MinMax, MaxAbs scalars
  - Mini-batch sizes may or may not influence results.

Out-Of-Core Scikit-Learn Demo

- Classification of text documents
  - Using Reuters-21578 benchmark dataset:
    - multi-class (90), multi-label
    - 7769 training documents and 3019 testing documents
    - provided by the UCI ML repository
  - Binary classification between the "acq" class and all the others
    - "acq" was chosen as it is more or less evenly distributed in the Reuters # files

	nr of documents		mean number of
class name	train	test	words in train set
1: earn	: 2877	1087	104.4
2: acq	: 1650	719	150.1
3: money-fx	: 538	179	219.0

- Scikit-learn
  - Data streaming with batches
  - Create the vectorizer
  - Classifiers with partial_fit

```
data_stream = stream_reuters_documents()
get_minibatch(data_stream, n_test_documents)

vectorizer = HashingVectorizer(decode_error='ignore',
                               n_features=2 ** 18,
                               alternate_sign=False)
X_train = vectorizer.transform(X_train_text)

cls.partial_fit(X_train, y_train, classes=all_classes)
```

bit.ly/hpskl_04

Key Takeaways

- Learning Scikit-learn means:
  - Understanding the machine learning workflows
  - Learning its API conventions
  - Familiarizing its documentations
  - Inspecting the example codes
  - Considering the scikit-learn extensive libraries
- Making scikit-learn more productive means:
  - Making sure everything work fine without performance optimization
  - Carefully profiling the code to address the performance bottleneck
  - Using Joblib with n_jobs
  - Trying RAPIDS, XGBoost if you can.

[Post-workshop Survey]