High-Performance Machine Learning Using Scikit-Learn

Qiyang Hu
UCLA Office of Advanced Research Computing

Aug 4th, 2023

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

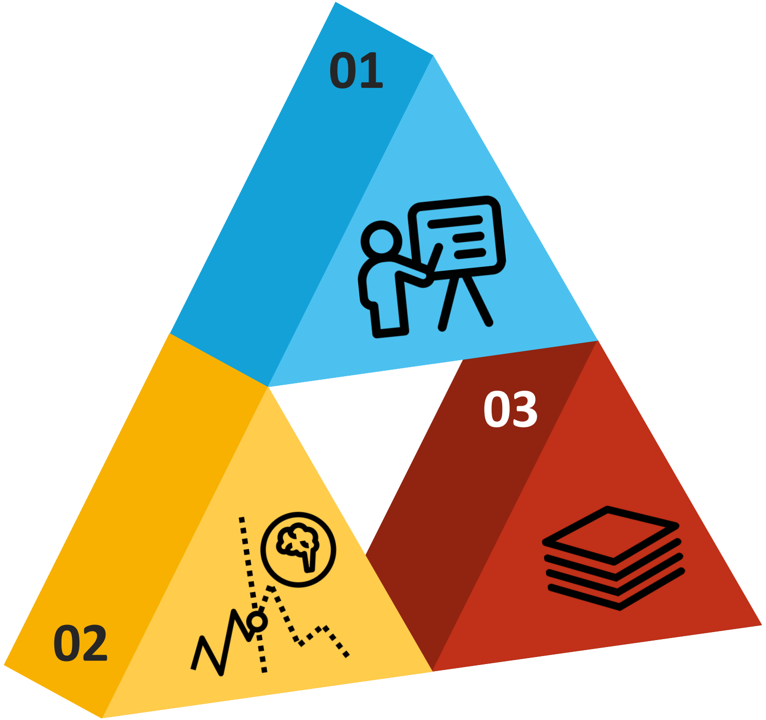
Number of training instances goes through the training processed in unit time

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

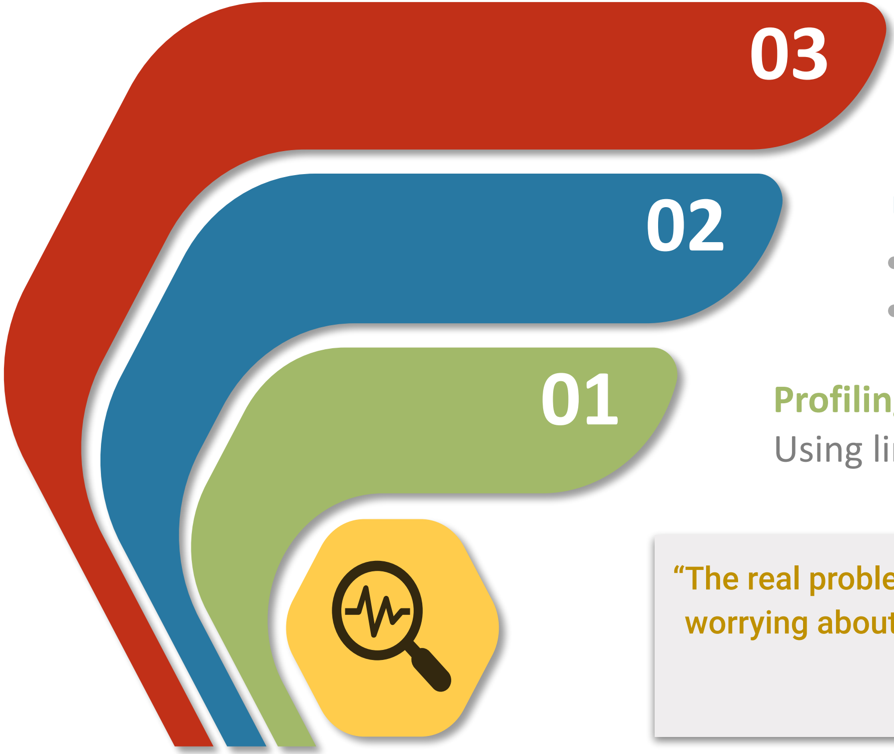
- Algorithm complexity
  - Hyperparameters
  - Optimized libraries
  - Multi-core Parallelism
  - Out-of-core learning
- Data complexity
  - Feature selection
  - Feature extraction
  - Feature transformation
- For prediction:
  - Validation overhead
  - Model compression
  - Model reshaping

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



Writing Cython wrappers

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

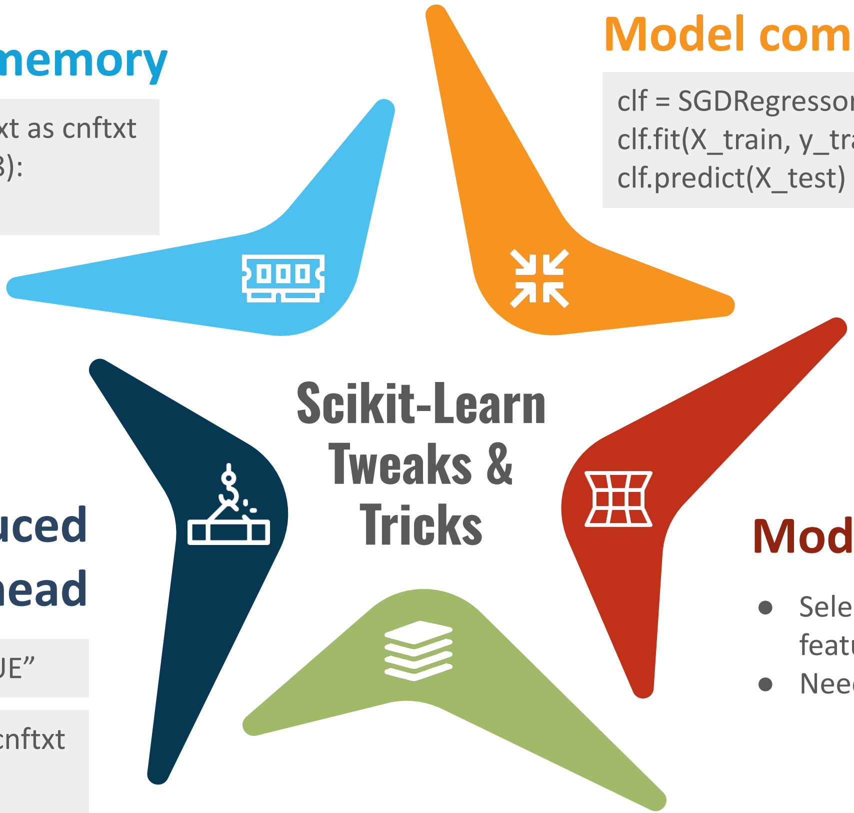
Using Numpy and Scipy

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

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

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

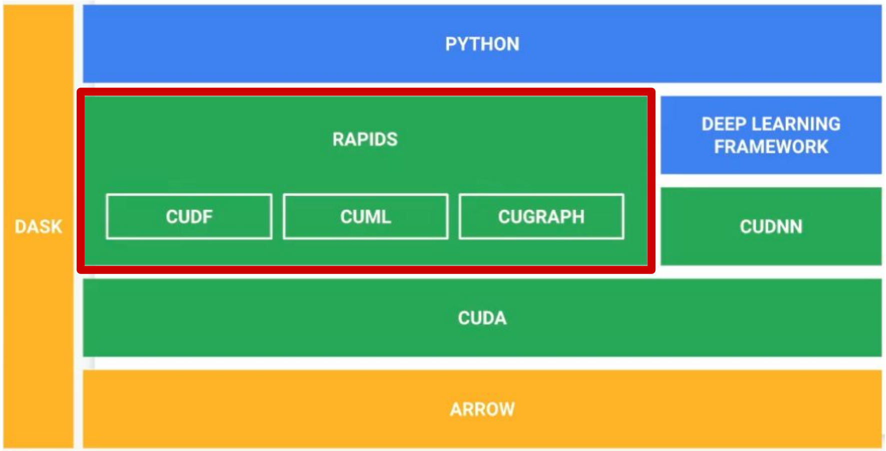
- Speeding up long-running jobs:
  - Specific optimizations for Numpy arrays
  - Caching and lazy evaluation to avoid computing repeatedly
  - No need to change the code or control flow
  - Various parallel backends: `locky`, `multiprocessing`, `threading`, `dask`, `ray`, …
- Tight and powerful integration 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, ...



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()
```

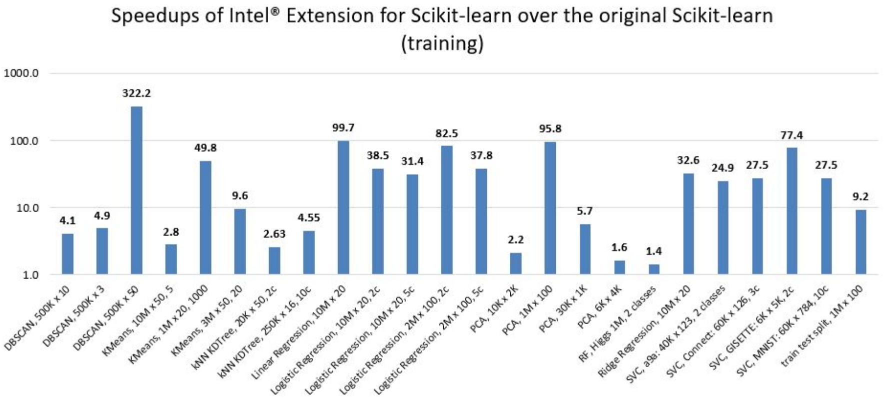
    - **Intel** CPU/GPU optimizations patching

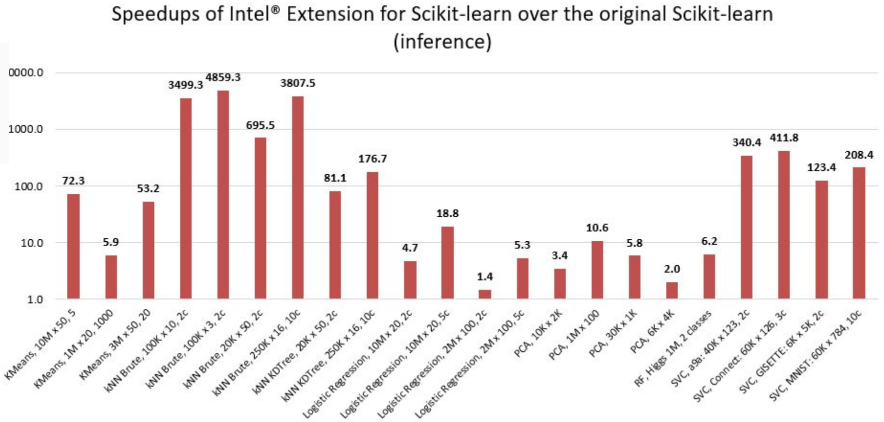
```
from sklearnex import patch_sklearn, config_context
patch_sklearn()
with config_context(target_offload="gpu:0"):
```

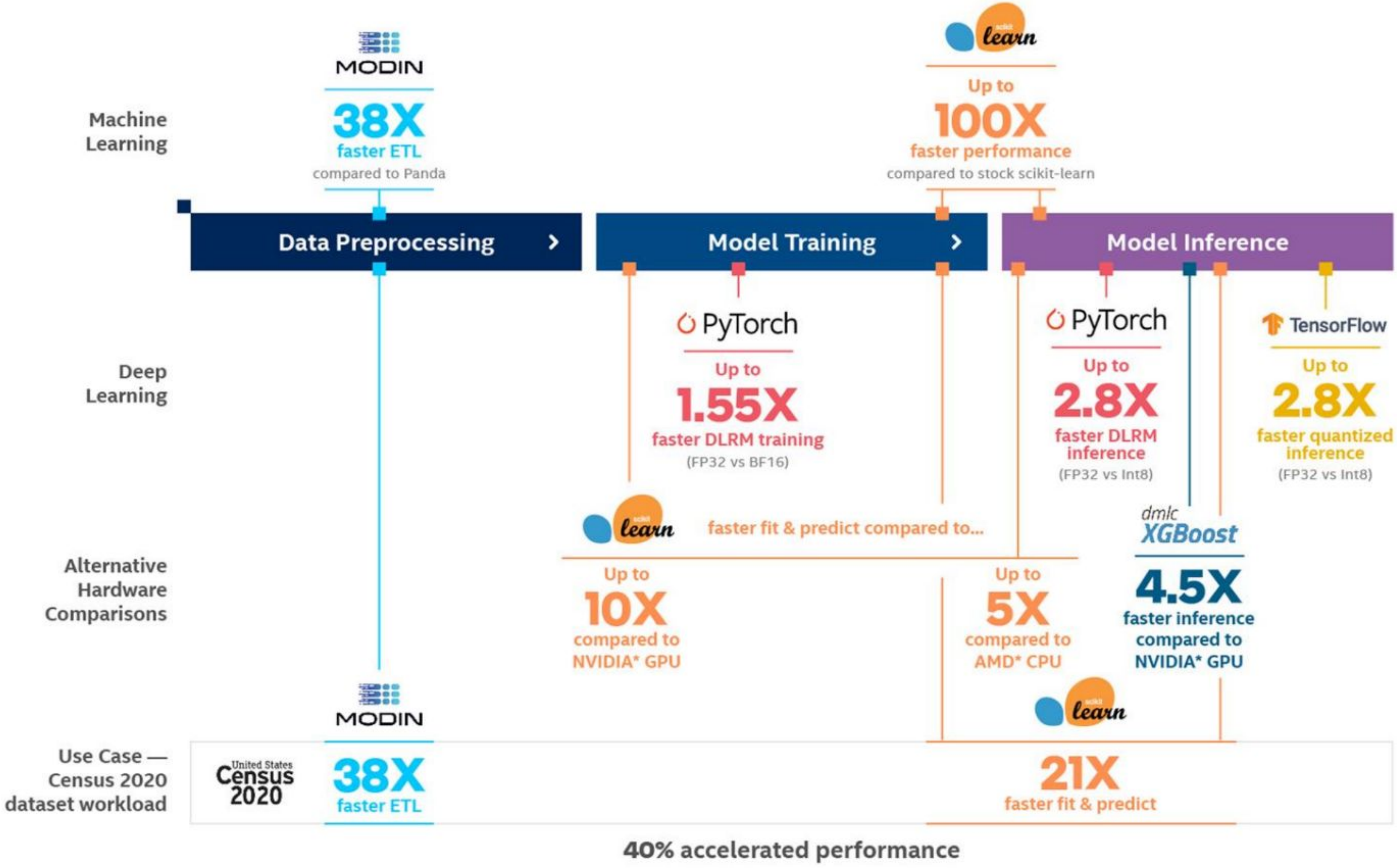
  - Installation:

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

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







MODIN
38X
faster ETL
compared to Panda
scikit learn
Up to
100X
faster performance
compared to stock scikit-learn
Machine Learning
Data Preprocessing
Model Training
Model Inference
Deep Learning
PyTorch
Up to
1.55X
faster DLRM training
(FP32 vs BF16)
PyTorch
Up to
2.8X
faster DLRM inference
(FP32 vs Int8)
TensorFlow
Up to
2.8X
faster quantized inference
(FP32 vs Int8)
Alternative Hardware Comparisons
scikit learn
faster fit & predict compared to...
Up to
10X
compared to NVIDIA* GPU
Up to
5X
compared to AMD* CPU
dmlc XGBoost
4.5X
faster inference compared to NVIDIA* GPU
Use Case — Census 2020 dataset workload
United States Census 2020
MODIN
38X
faster ETL
scikit learn
21X
faster fit & predict
40% accelerated performance

Qiyang Hu

High-performance ML using accelerators (cont'd)

- Option 3: JAX
  - A numpy library re-compiled for high-performance numerical computing
    - using XLA & JIT, on GPUs and TPUs
  - Key features
    - Automatic differentiation:
      - Forward and reverse mode AD of arbitrary numerical functions
      - `grad`, `hessian`, `jacfwd`, `jacrev`
    - Vectorization:
      - SIMD programming via automatic vectorization
      - `vmap`, `pmap`
    - JIT-compilation:
      - XLA is used to just-in-time(JIT)-compile and execute JAX programs
      - faster CPU code and transparent GPU and Cloud TPU acceleration
  - `sklearn-jax-kernel` is in an early stage.

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
  - 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, <u>not all available upfront</u>.
- 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

	class name		nr of documents train	test	mean number of words in train set
1:	earn	:	2877	1087	104.4
2:	acq	:	1650	719	150.1
3:	money-fx	:	538	179	219.0
4:	grain	:	433	149	223.6

- 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, sklearnex, vaex.ml if needed.