Scikit-learn
Machine learning for the small and the many

Gaël Varoquaux

Inria

Scikit-learn
Machine learning in Python

In this meeting, I represent low performance computing
Scikit-learn
Machine learning for the small and the many

Gaël Varoquaux

What I do: bridging psychology to neuroscience via machine learning on brain images
1. Scikit-learn

2. Statistical algorithms

3. Scaling up / scaling out?
1 Scikit-learn

Goals and tradeoff
Scikit-learn’s vision: Machine learning for everyone

Outreach
across scientific fields, applications, communities

Enabling foster innovation
Scikit-learn’s vision: Machine learning for everyone

Outreach across scientific fields, applications, communities

Enabling foster innovation

Minimal prerequisites & assumptions
1. scikit-learn user base

350,000 returning users

- 50% Windows
- 30% Mac
- 20% Linux

5,000 citations

- 63% industry
- 34% academia
- 3% other
Python
- High-level language, for users and developers
- General-purpose: suitable for any application
- Excellent interactive use
Python

- High-level language, for users and developers
- General-purpose: suitable for any application
- Excellent interactive use

Slow $\Rightarrow$ compiled code as a backend

Python’s primitive virtual machine makes it easy
Python
- High-level language, for users and developers
- General-purpose: suitable for any application
- Excellent interactive use

Scipy
- Vibrant scientific stack
- numpy arrays = wrappers on C pointers
- pandas for columnar data
- scikit-image for images
A Python library

Users like Python

Web searches:

Google trends

- scikit-learn
- "machine learning" python
- "machine learning" r
- "machine learning" java
A Python library

And developers like Python

Number of contributors active in a week

G Varoquaux
1 A Python library

And developpers like Python

⇒ Huge set of features

(≈ 160 different statistical models)
API: simplify, but do not dumb down

Universal estimator interface

```python
from sklearn import svm
classifier = svm.SVC()
classifier.fit(X_train, Y_train)
Y_test = classifier.predict(X_test)
# or
X_red = classifier.transform(X_test)
```
API: simplify, but do not dumb down

Universal estimator interface

```python
from sklearn import svm
classifier = svm.SVC()
classifier.fit(X_train, Y_train)
Y_test = classifier.predict(X_test)
# or
X_red = classifier.transform(X_test)
```

classifier often has hyperparameters
Finding good defaults is crucial, and hard
API: simplify, but do not dumb down

Universal estimator interface

```python
from sklearn import svm
classifier = svm.SVC()
classifier.fit(X_train, Y_train)
Y_test = classifier.predict(X_test)
# or
X_red = classifier.transform(X_test)
```

classifier often has hyperparameters
Finding good defaults is crucial, and hard

A lot of effort on the documentation
Example-driven development
1 Tradeoffs

- Algorithms and models with good failure mode
  Avoid parameters hard to set or fragile convergence
  Statistical computing = ill-posed & data-dependent

- Little or no dependencies
  Easy build everywhere

- All compiled code generated from Cython
  High-level languages give features (Spark)
  Low-level gives speed (eg cache-friendly code)
2 Statistical algorithms

Fast algorithms accept statistical error
2 Statistical algorithms

Fast algorithms accept statistical error

Models most used in scikit-learn:

1. Logistic regression, SVM
2. Random forests
3. PCA
4. Kmeans
5. Naive Bayes
6. Nearest neighbor
“Big” data

Many samples

or

Many features

Web behavior data

Cheap sensors (cameras)

Medical patients

Scientific experiments
Many features

Coordinate descent

Iteratively optimize \( w.r.t. \ w_j \) separately

It works because:

Features are redundant

Sparse models can guess which \( w_j \) are zero

Progress = better selection of features
2 Linear models

\[ \min_w \sum_i l(y_i, x_i w) \]

**Many features** Coordinate descent
Iteratively optimize w.r.t. \(w_j\) separately

**Many samples** Stochastic gradient descent

\[ \min_w \mathbb{E}[l(y, x w)] \]

Gradient descent:
\[ w \leftarrow w + \alpha \nabla_w l \]

Stochastic gradient descent
\[ w \leftarrow w + \alpha \mathbb{E}[\nabla_w l] \]
Use a cheap estimate of \(\mathbb{E}[\nabla_w l]\) (e.g. subsampling)

Progress = second order schemes
Linear models

\[ \min_w \sum_i l(y_i, x_i w) \]

Many features
Coordinate descent
Iteratively optimize w.r.t. \( w_j \) separately

Many samples
Stochastic gradient descent

\[ \min_w \mathbb{E}[l(y, x w)] \]

Gradient descent: \( w \leftarrow w + \alpha \nabla_w l \)

Stochastic gradient descent \( w \leftarrow w + \alpha \mathbb{E}[\nabla_w l] \)

Use a cheap estimate of \( \mathbb{E}[\nabla_w l] \) (e.g. subsampling)

Data-access locality
Linear models

\[ \min_w \sum_i l(y_i, x_i w) \]

Deep learning

- Composition of linear models
- Optimized jointly (non-convex)
- With stochastic gradient descent

Many features

Coordinate descent

Iteratively optimize \( w \) separately

Many samples

Stochastic gradient descent

\[ \min_w \mathbb{E}[l(y, x w)] \]

Gradient descent:

\[ w \leftarrow w + \alpha \nabla_w l \]

Stochastic gradient descent

\[ w \leftarrow w + \alpha \mathbb{E}[\nabla_w l] \]

Use a cheap estimate of \( \mathbb{E}[\nabla_w l] \) (e.g. subsampling)

Data-access locality
Trees & (random) forests

(on subsets of the data)

- Compute simple bi-variate statistics
- Split data accordingly

**Speed ups**

- Share computing between trees or precompute
- Cache friendly access ⇒ optimize traversal order
- Approximate histograms / statistics

LightGBM, XGBoost
Truncated SVD (singular value decomposition)

\[ X = U s V^T \]
PCA: principal component analysis

Truncated SVD (singular value decomposition)

\[ X = U \, s \, V^T \]

Randomized linear algebra

for \( i \) in \([1, \ldots k]\):

\[ \tilde{X} = \text{random projection}(X) \]

\[ \tilde{U}_i, \tilde{s}_i, \tilde{V}_i^T = \text{SVD}(\tilde{X}) \]

\[ V_{\text{red}}, R = \text{QR}([\tilde{V}_1, \ldots, \tilde{V}_k]) \]

\[ X_{\text{red}} = V_{\text{red}}^T \, X \]

\[ U' \, s' \, V'^T = \text{SVD}(X_{\text{red}}) \]

\[ V^T = V'^T \, V_{\text{red}}^T \]

\[ \rightarrow 20x \text{ speed ups} \]

# e.g. subsampling

[Halko... 2011]
Truncated SVD (singular value decomposition)

\[ X = U s V^T \]

Randomized linear algebra

\[
\text{for } i \text{ in } [1, \ldots k]: \\
\tilde{X} = \text{random projection}(X) \\
\tilde{U}_i, \tilde{s}_i, \tilde{V}_i^T = \text{SVD}(\tilde{X}) \\
V_{\text{red}}, R = \text{QR}([\tilde{V}_1, \ldots, \tilde{V}_k]) \\
X_{\text{red}} = V_{\text{red}}^T X \\
U' s' V'^T = \text{SVD}(X_{\text{red}}) \\
V^T = V'^T V_{\text{red}}^T
\]

\[ \tilde{X} \text{ summarize well the data} \]

Each SVD is on local data

[Halko... 2011]
Factorization of dense matrices $\sim 200\,000 \times 2\,000\,000$

$$\min_{U,V} \|X - UV^T\|_2 + \|V\|_1$$

Data matrix $X$
2 Stochastic factorization of huge matrices

Factorization of dense matrices $\sim 200\ 000 \times 2\ 000\ 000$

$$\min_{U,V} \|X - UV^T\|_2 + \|V\|_1$$
Stochastic factorization of huge matrices

Factorization of dense matrices $\sim 200\,000 \times 2\,000\,000$

- Data access
- Code computation
- Dictionary update

Alternating minimization

Online matrix factorization

Stream columns

[Mairal... 2010]

out of core, huge speed ups
Stochastic factorization of huge matrices

Factorization of **dense** matrices \( \sim 200\,000 \times 2\,000\,000 \)

- Data access
- Code computation
- Dictionary update

**Data matrix**

Alternating minimization

Online matrix factorization

New subsampling algorithm

Stream columns

Subsample rows

- Seen at \( t \)
- Seen at \( t+1 \)
- Unseen at \( t \)

10X speed ups, or more

[Mensch... 2017]
Scaling up / scaling out?
Dataflow is key to scale

Array computing

Data parallel

Streaming

- Parallel computing
- Data + code transfer
- Out-of-memory persistence

These patterns can yield horrible code

G Varoquaux
Parallel-computing engine: joblib

```python
sklearn.Estimator(n_jobs=2)
```

**Under the hood: joblib**

- Parallel for loops
- Concurrency is hard

Queues are the central abstraction
Parallel-computing engine: joblib

Under the hood:

`joblib` Parallel for loops concurrency is hard

```
from joblib import Parallel, parallel_backend

with parallel_backend('dask.distributed', scheduler='HOST:PORT'):
    # normal Joblib code
```

Andreas Mueller @t3k@it · Feb 14

Just a quick reminder what sklearn random forests look like on EC2. want?
Parallel-computing engine: joblib

```
sklearn.Estimator(n_jobs=2)
```

**Under the hood:** joblib

Parallel for loops concurrency is hard

**New:** distributed computing backends:

Yarn, dask.distributed, IPython.parallel

```
import distributed.joblib
from joblib import Parallel, parallel_backend
with parallel_backend('dask.distributed', scheduler_host='HOST:PORT'):
    # normal Joblib code
```
Parallel-computing engine: joblib

```
sklearn.Estimator(n_jobs=2)
```

**Under the hood**: joblib

Parallel for loops concurrency is hard

**New**: distributed computing backends:

- Yarn, dask.distributed, IPython.parallel

```
import distributed.joblib
from joblib import Parallel, parallel_backend
with parallel_backend('dask.distributed', scheduler_host='HOST:PORT'):
    # normal Joblib code
```

Middleware to plug in distributed infrastructures

G Varoquaux
Moving data around is costly
Distributed data flow and storage

Moving data around is costly

Why databases and not files?
- Maintain integrity themselves
- Know how to do data replication & distribution
- Fast lookup via indexes
- Not bound by POSIX FS specs

Very big data calls for coupling a database to a computing engine
Distributed data flow and storage

Why databases and not files?
- Maintain integrity themselves
- Know how to do data replication & distribution
- Fast lookup via indexes
- Not bound by POSIX FS specs

Very big data calls for coupling a database to a computing engine

G Varoquaux
3 joblib.Memory as a storage pool

A caching / function memoizing system
Stores results of function executions

S3/HDFS/cloud backend:
```
joblib.Memory('uri', backend='s3')
```

https://github.com/joblib/joblib/pull/397
A caching / function memoizing system
Stores results of function executions

Out-of-memory computing

```python
>>> result = mem.cache(g).call_and_shelve(a)
```

```python
>>> result
MemorizedResult(cachedir="...", func="g", argument_hash="...")
```

```python
>>> c = result.get()
```
3 joblib.Memory as a storage pool

- A caching / function memoizing system
  Stores results of function executions

- Out-of-memory computing
  ```python
  >>>> result = mem.cache(g).call_and_shelve(a)
  >>>> result
  MemorizedResult(cachedir="...", func="g", argument_hash="...")
  >>>> c = result.get()
  ```

- S3/HDFS/cloud backend:
  ```python
  joblib.Memory(’uri’, backend=’s3’)
  ```
  https://github.com/joblib/joblib/joblib/pull/397
Challenges and dreams

- High-level constructs for distributed computation & data exchange
  MPI feels too low level and without data concepts

**Goal:** reusable algorithms from laptops to datacenters
  Capturing data access patterns is the missing piece
Challenges and dreams

- High-level constructs for distributed computation & data exchange
  MPI feels too low level and without data concepts

**Goal:** reusable algorithms from laptops to datacenters
Capturing data access patterns is the missing piece

- Dask project:
  - Limit to purely-functional code
  - Lazy computation / compilation
  - Build a data flow + execution graph

Also: deep-learning engines, for GPUs
Lessons from scikit-learn
Small-computer machine-learning trying to scale

Python gets us very far
- Enables focusing on algorithmic optimization
- Great to grow a community
- Can easily drop to compiled code
Lessons from scikit-learn
Small-computer machine-learning trying to scale

Python gets us very far

Statistical algorithmics

- Algorithms operate on expectancies
  - Stochastic Gradient Descent
  - Random projections
- Can bring data locality
Lessons from scikit-learn
Small-computer machine-learning trying to scale

Python gets us very far

Statistical algorithmics

Distributed data computing
- Data access is central
- Must be optimized for algorithm
- File system and memory no longer suffice
Lessons from scikit-learn
Small-computer machine-learning trying to scale

Python gets us very far

Statistical algorithmics

Distributed data computing

If you know what your doing, you can scale scikit-learn

The challenge is to make this easy and generic
