Scikit-learn

Machine learning for the small and the many

Gaël Varoquaux

machine learning in Python

In this meeting, I represent low performance computing

Scikit-learn

Machine learning for the small and the many

Gaël Varoquaux

Inria

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

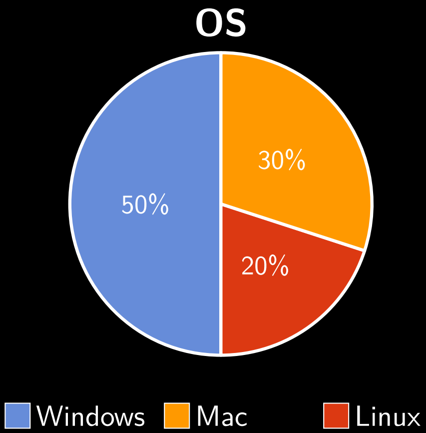
Outreach
across scientific fields,
applications, communities

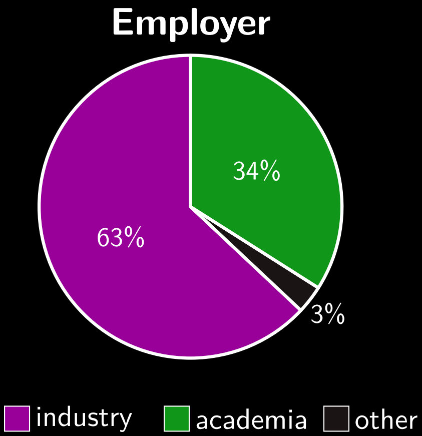
Enabling
foster innovation

Minimal prerequisites & assumptions

350 000 returning users

5 000 citations





1 A Python library

Python

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

1 A Python library

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

1 A Python library

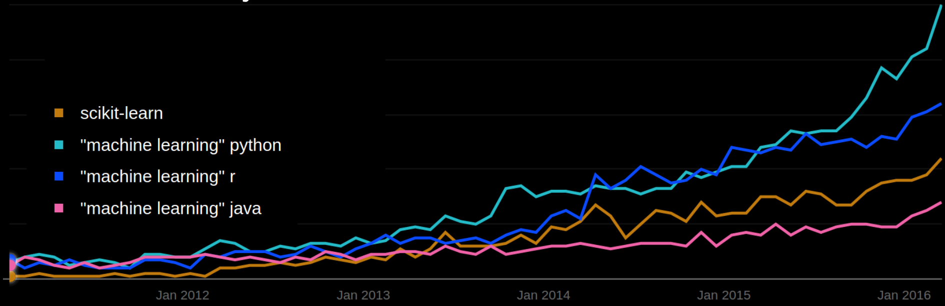
Python

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

Scipy

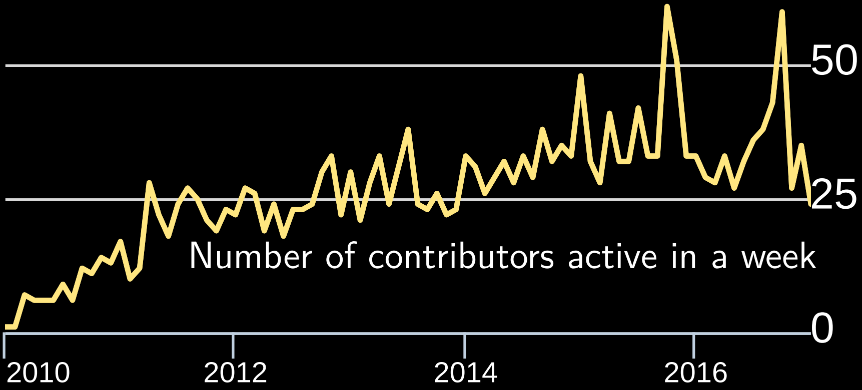
- Vibrant scientific stack
- `numpy` arrays = wrappers on C pointers
- `pandas` for columnar data
- `scikit-image` for images

Users like Python

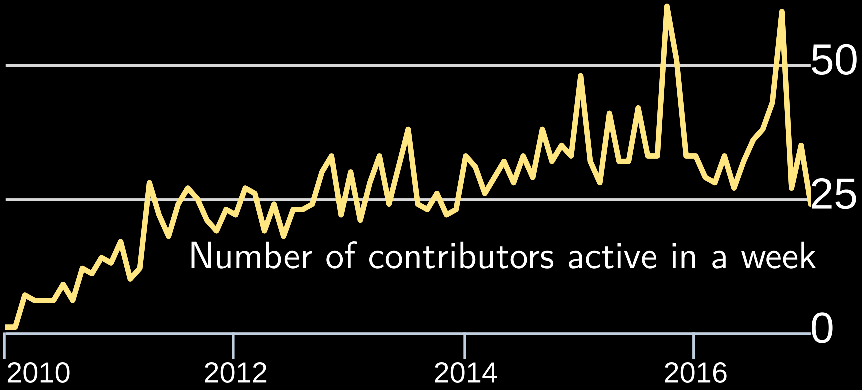


Web searches: Google trends

And developpers like Python



And developpers like Python



⇒ Huge set of features
($\sim$ 160 different statistical models)

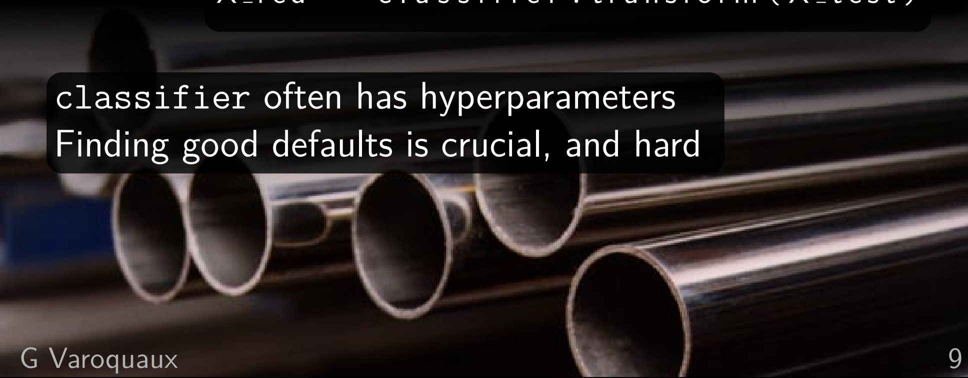
1 API: simplify, but do not dumb down

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)
```

Universal estimator interface

```
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

1 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

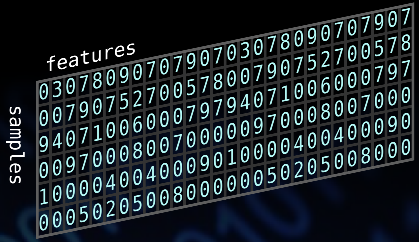
Fast algorithms accept statistical error

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

$$\min_{\mathbf{w}} \sum_i l(y_i, \mathbf{x}_i\, \mathbf{w})$$

Many features Coordinate descent

Iteratively optimize *w.r.t.* $\mathbf{w}_j$ separately

It works because:

Features are redundant

Sparse models can guess which $\mathbf{w}_j$ are zero

Progress = better selection of features

$$\min_{\mathbf{w}} \sum_i l(y_i, \mathbf{x}_i\, \mathbf{w})$$

Many features Coordinate descent

Iteratively optimize *w.r.t.* $\mathbf{w}_j$ separately

Many samples Stochastic gradient descent

$$\min_{\mathbf{w}} \mathbb{E}[l(y, \mathbf{x}\, \mathbf{w})]$$

Gradient descent: $\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla_{\mathbf{w}} l$

Stochastic gradient descent $\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbb{E}[\nabla_{\mathbf{w}} l]$

Use a cheap estimate of $\mathbb{E}[\nabla_{\mathbf{w}} l]$ (*e.g.* subsampling)

Progress = second order schemes

2 Linear models

$$\min_{\mathbf{w}} \sum_i l(y_i, \mathbf{x}_i\, \mathbf{w})$$

Many features Coordinate descent

Iteratively optimize *w.r.t.* $\mathbf{w}_j$ separately

Many samples Stochastic gradient descent

$$\min_{\mathbf{w}} \mathbb{E}[l(y, \mathbf{x}\, \mathbf{w})]$$

Gradient descent: $\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla_{\mathbf{w}} l$

Stochastic gradient descent $\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbb{E}[\nabla_{\mathbf{w}} l]$

Use a cheap estimate of $\mathbb{E}[\nabla_{\mathbf{w}} l]$ (*e.g.* subsampling)

Data-access locality

$$\min_{\mathbf{w}} \sum_i l(y_i, \mathbf{x}_i\, \mathbf{w})$$

Many fea

Deep learning

- Composition of linear models
- optimized jointly (non-convex)
- with stochastic gradient descent

parately

Many samples Stochastic gradient descent

$$\min_{\mathbf{w}} \mathbb{E}[l(y, \mathbf{x}\, \mathbf{w})]$$

Gradient descent: $\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla_{\mathbf{w}} l$

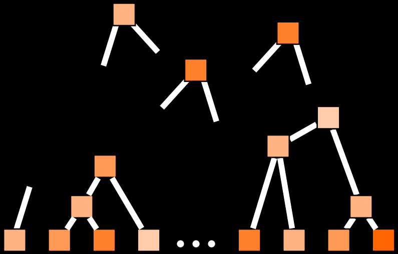
Stochastic gradient descent $\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbb{E}[\nabla_{\mathbf{w}} l]$

Use a cheap estimate of $\mathbb{E}[\nabla_{\mathbf{w}} l]$ (*e.g.* subsampling)

Data-access locality

(on subsets of the data)

- Compute simple bi-variate statistics
- Split data accordingly



Speed ups

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

`LightGBM, XGBoost`

Truncated SVD (singular value decomposition)

$$\mathbf{X} = \mathbf{U}\,\mathbf{s}\,\mathbf{V}^{\mathsf{T}}$$

Truncated SVD (singular value decomposition)

$$\mathbf{X} = \mathbf{U}\,\mathbf{s}\,\mathbf{V}^{\mathsf{T}}$$

Randomized linear algebra → 20x speed ups

`for` i `in` $[1, \dots k]$:

$\tilde{\mathbf{X}}$ =random_projection($\mathbf{X}$) # *e.g.* subsampling

$\tilde{\mathbf{U}}_{\mathbf{i}}, \tilde{\mathbf{s}}_i, \tilde{\mathbf{V}}_i^{\mathsf{T}} = \mathrm{SVD}(\tilde{\mathbf{X}})$

$\mathbf{V}_{\mathrm{red}}, \mathbf{R} = \mathrm{QR}([\tilde{\mathbf{V}}_1, \dots, \tilde{\mathbf{V}}_k])$

$\mathbf{X}_{\mathrm{red}} = \mathbf{V}_{\mathrm{red}}^{\mathsf{T}}\mathbf{X}$

$\mathbf{U}'\,\mathbf{s}'\,\mathbf{V}'^{\mathsf{T}} = \mathrm{SVD}(\mathbf{X}_{\mathrm{red}})$

$\mathbf{V}^{\mathsf{T}} = \mathbf{V}'^{\mathsf{T}}\mathbf{V}_{\mathrm{red}}^{\mathsf{T}}$

Truncated SVD (singular value decomposition)

$$\mathbf{X} = \mathbf{U}\,\mathbf{s}\,\mathbf{V}^{\mathsf{T}}$$

Randomized linear algebra → 20x speed ups

for i in $[1, \ldots k]$:

$\tilde{\mathbf{X}}$ =random_projection($\mathbf{X}$) # *e.g.* subsampling

$\tilde{\mathbf{U}}_{\mathbf{i}}, \tilde{\mathbf{s}}_i, \tilde{\mathbf{V}}_i^{\mathsf{T}} = \mathrm{SVD}(\tilde{\mathbf{X}})$

$\mathbf{V}_{\text{red}}, \mathbf{R} = \mathrm{QR}([\tilde{\mathbf{V}}_1, \ldots, \tilde{\mathbf{V}}_k])$

$\mathbf{X}_{\text{red}} = \mathbf{V}_{\text{red}}^{\mathsf{T}}\mathbf{X}$

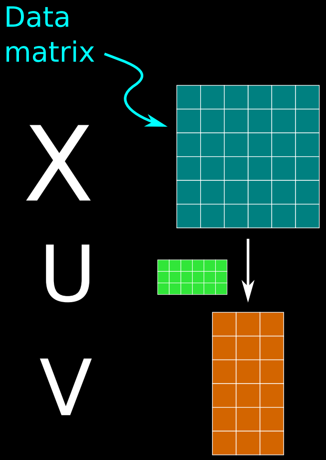
$\mathbf{U}'\,\mathbf{s}'\,\mathbf{V}'^{\mathsf{T}} = \mathrm{SVD}(\mathbf{X}_{\text{red}})$

$\mathbf{V}^{\mathsf{T}} = \mathbf{V}'^{\mathsf{T}}\mathbf{V}_{\text{red}}^{\mathsf{T}}$

$\tilde{\mathbf{X}}$ summarize well the data
Each SVD is on local data

[Halko... 2011]

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

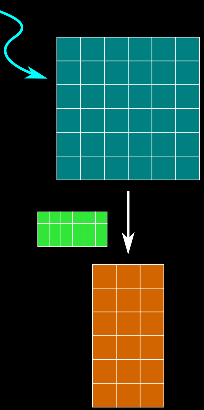


$$\min_{\mathbf{U},\mathbf{V}} \|\mathbf{X} - \mathbf{U}\mathbf{V}^\mathsf{T}\|_2 + \|\mathbf{V}\|_1$$

2 Stochastic factorization of huge matrices

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

Data matrix

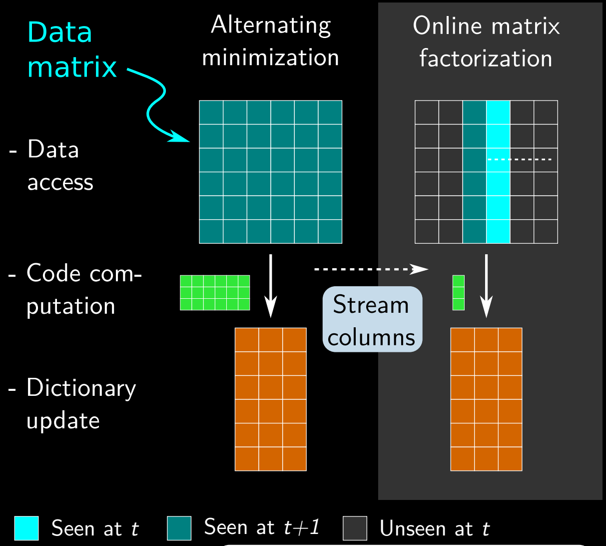


- Data access
- Code computation
- Dictionary update

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

2 Stochastic factorization of huge matrices

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

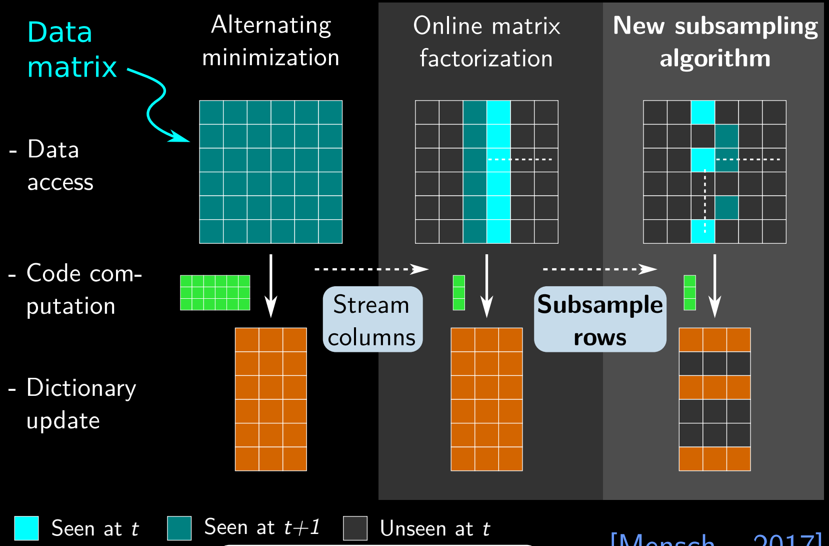


out of core, huge speed ups

[Mairal... 2010]

2 Stochastic factorization of huge matrices

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



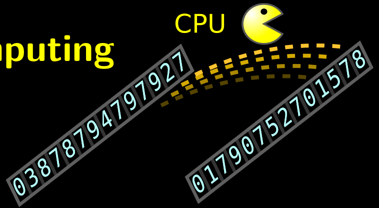
10X speed ups, or more

[Mensch... 2017]

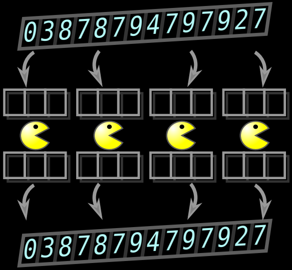
3 Scaling up / scaling out?

3 Dataflow is key to scale

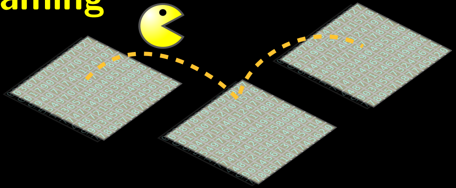
Array computing



Data parallel



Streaming



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

These patterns can yield horrible code

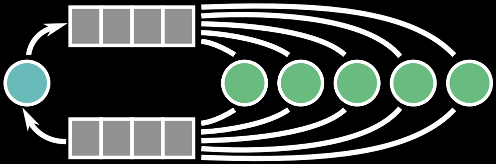
`sklearn.Estimator(n_jobs=2)`

Under the hood: `joblib`

Parallel for loops

concurrency is hard

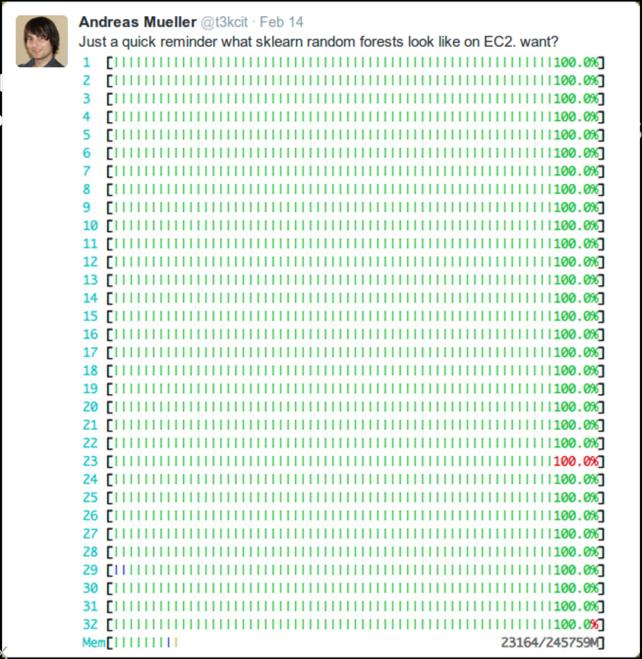
Queues are the central abstraction



Under

P

hard


Andreas Mueller @t3kcit · Feb 14
Just a quick reminder what sklearn random forests look like on EC2. want?
Mem[
23164/245759M]

`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
```

`sklearn.Estimator(n_jobs=2)`

Under the hood: `joblib`

Parallel for loops

concurrency is hard

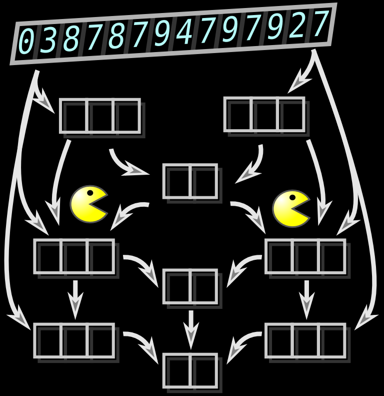
New: distributed computing backends:

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

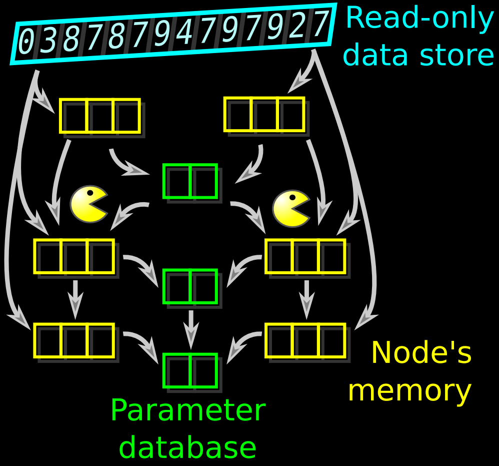
Moving data around
is costly



3 Distributed data flow and storage

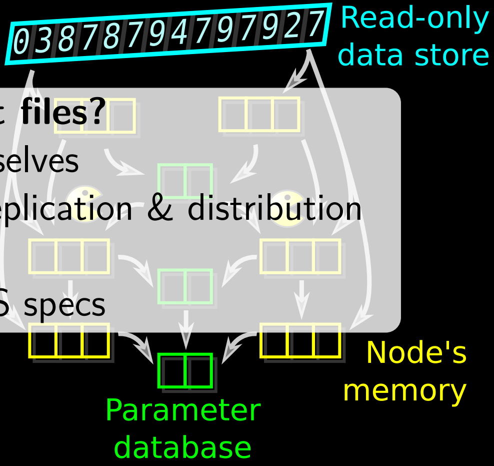
Moving data around

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

3 joblib.Memory as a storage pool

- A caching / function memoizing system

  Stores results of function executions

3 joblib.Memory as a storage pool

- A caching / function memoizing system

  Stores results of function executions

- Out-of-memory computing

```
>>> result = mem.cache(g).call_and_shelve(a)
>>> result
MemorizedResult(cachedir="...", func="g", argument_hash="...")
>>> c = result.get()
```

3 joblib.Memory as a storage pool

- A caching / function memoizing system

  Stores results of function executions

- Out-of-memory computing

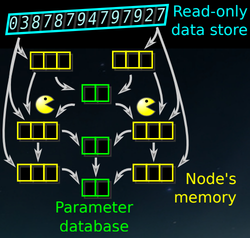
```
>>> result = mem.cache(g).call_and_shelve(a)
>>> result
MemorizedResult(cachedir="...", func="g", argument_hash="...")
>>> c = result.get()
```

- S3/HDFS/cloud backend:

  `joblib.Memory('uri', backend='s3')`

  https://github.com/joblib/joblib/pull/397

Challenges and dreams



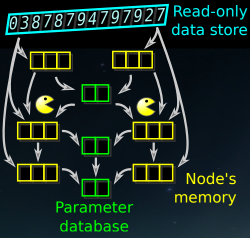
- High-level constructs for distributed computation & data exchange

  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

@GaelVaroquaux

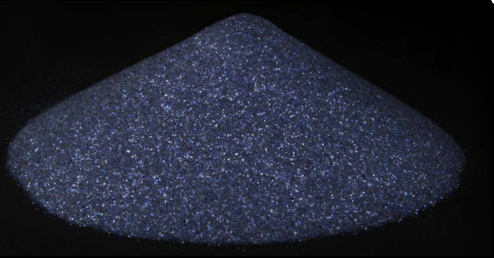
Lessons from scikit-learn

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 acces is central
- Must be optimized for algorithm
- File system and memory no longer suffice

@GaelVaroquaux

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

@GaelVaroquaux

4 References I

N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53, 2011. ISSN 0036-1445. doi: 10.1137/090771806. URL http://dx.doi.org/10.1137/090771806.

J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11:19, 2010.

A. Mensch, J. Mairal, B. Thirion, and G. Varoquaux. Stochastic subsampling for factorizing huge matrices. *Arxiv preprint*, 2017.