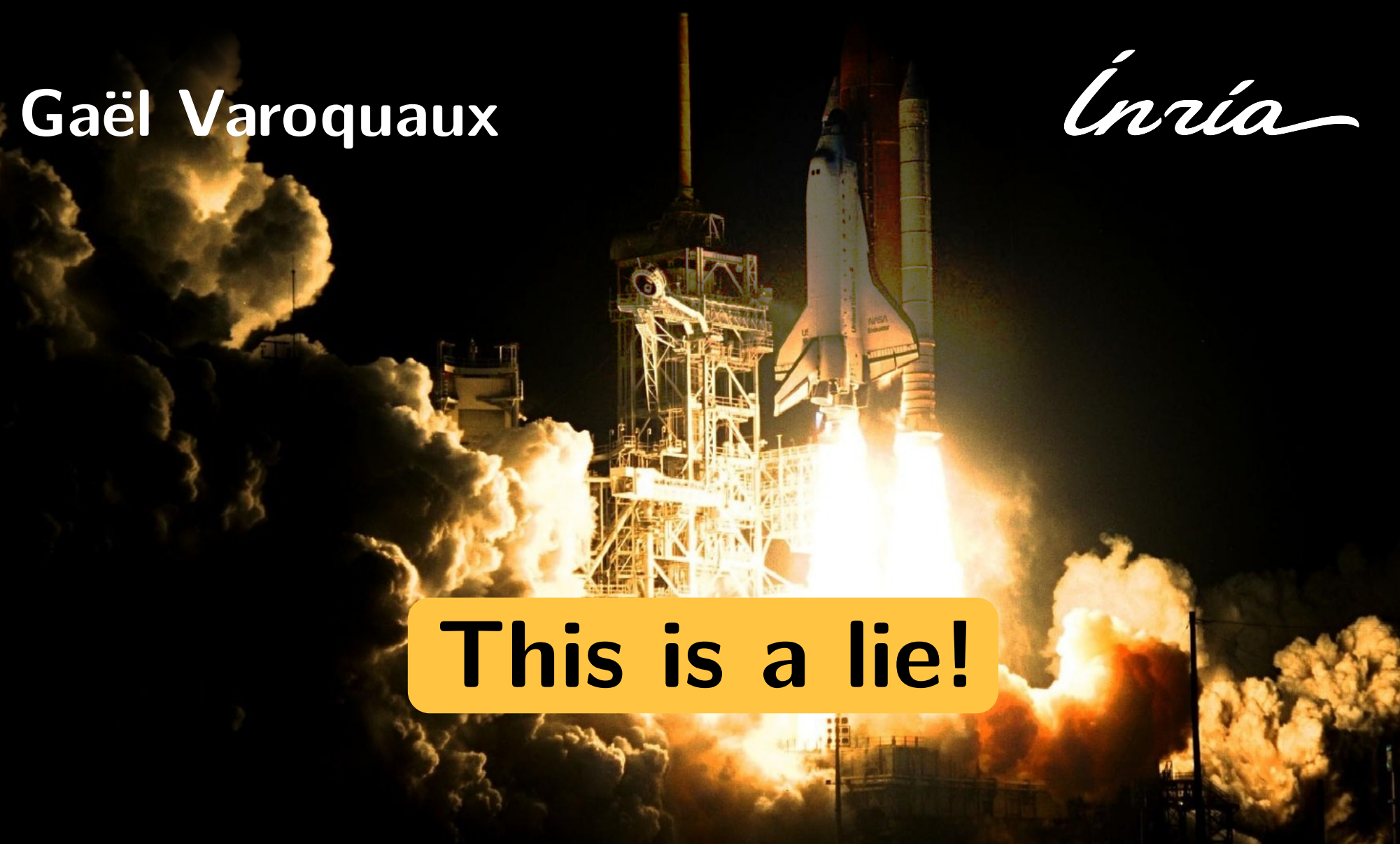
Simple big data, in Python

Gaël Varoquaux

Simple big data, in Python

Gaël Varoquaux

Inria



This is a lie!

Please allow me to introduce myself

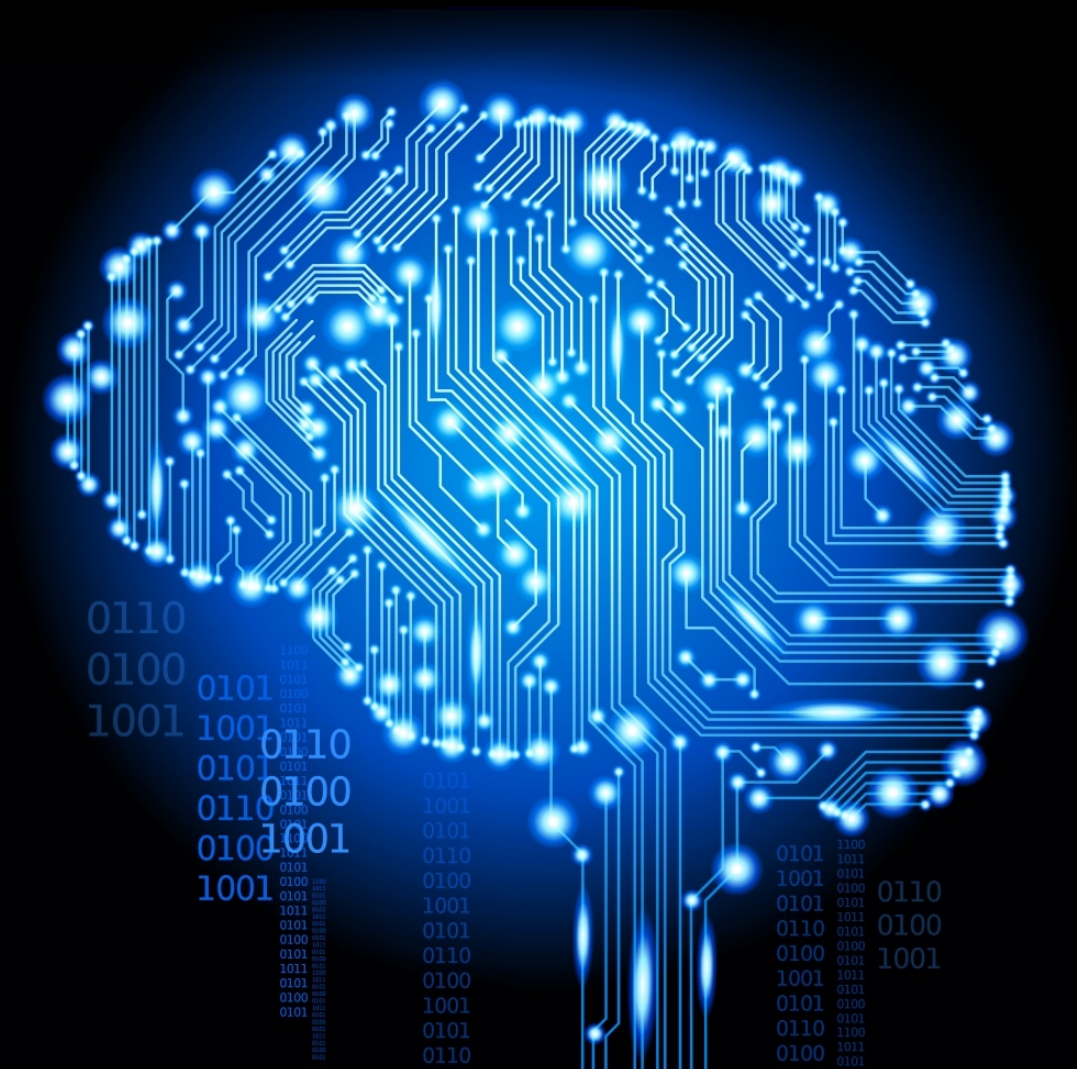
I'm a man of wealth and taste

I've been around for a long, long year

- Physicist gone bad
  - Neuroscience, Machine learning
- Worked in a software startup
  - Enthought: scientific computing consulting in Python
- Coder (done my share of mistake)
  - Mayavi, scikit-learn, joblib...
- Scipy community
  - Chair of scipy and EuroScipy conferences
- Researcher (PI) at INRIA

1. Machine learning in 2 words
2. Scikit-learn
3. Big data on a budget
4. The bigger picture: a community

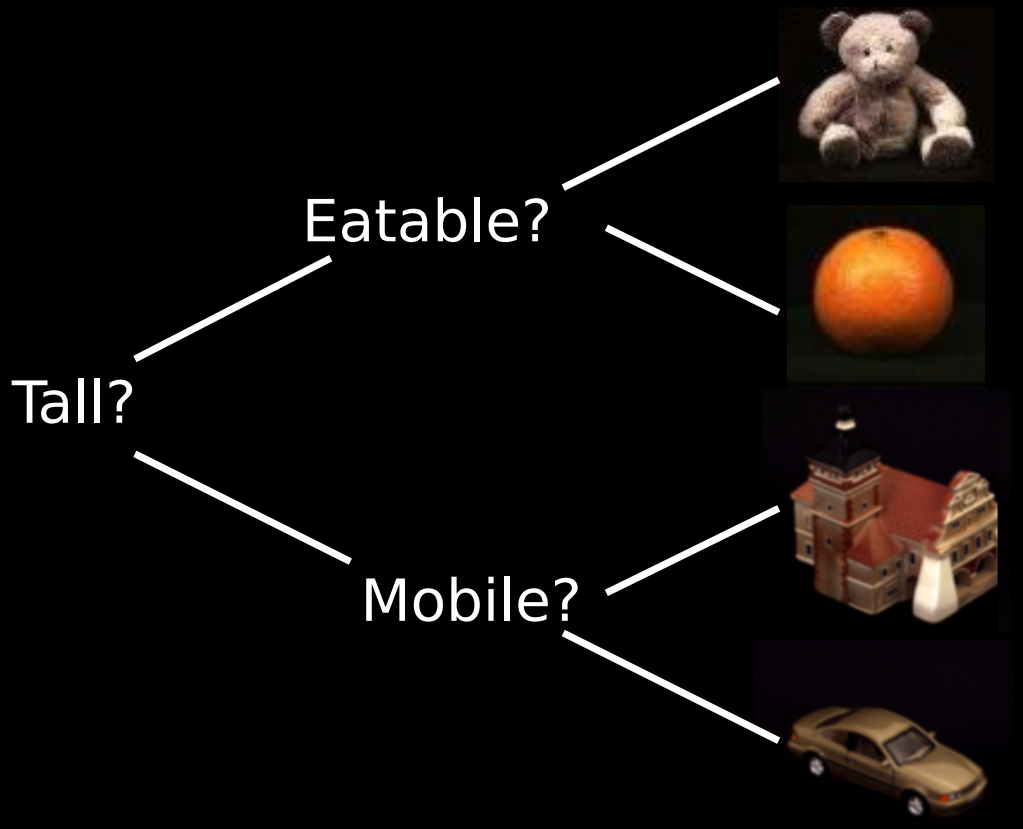
1 Machine learning in 2 words



Artificial Intelligence

The 80s

- Building decision rules

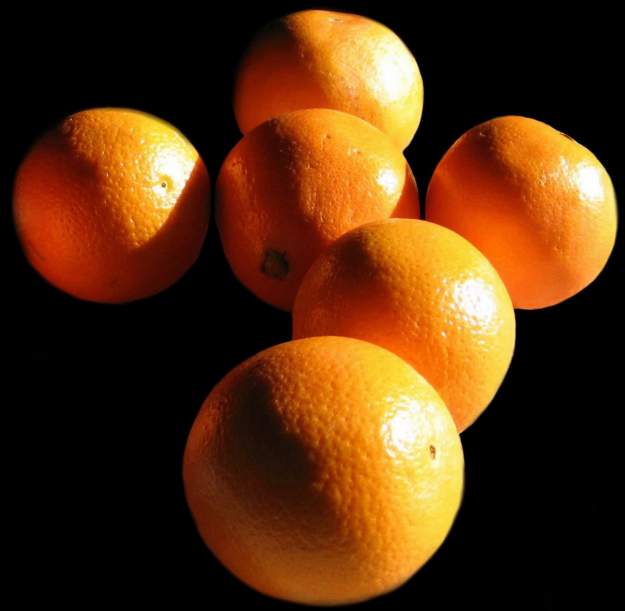


Artificial Intelligence The 80s

- Building decision rules

Machine learning The 90s

- Learn these from observations



Artificial Intelligence The 80s

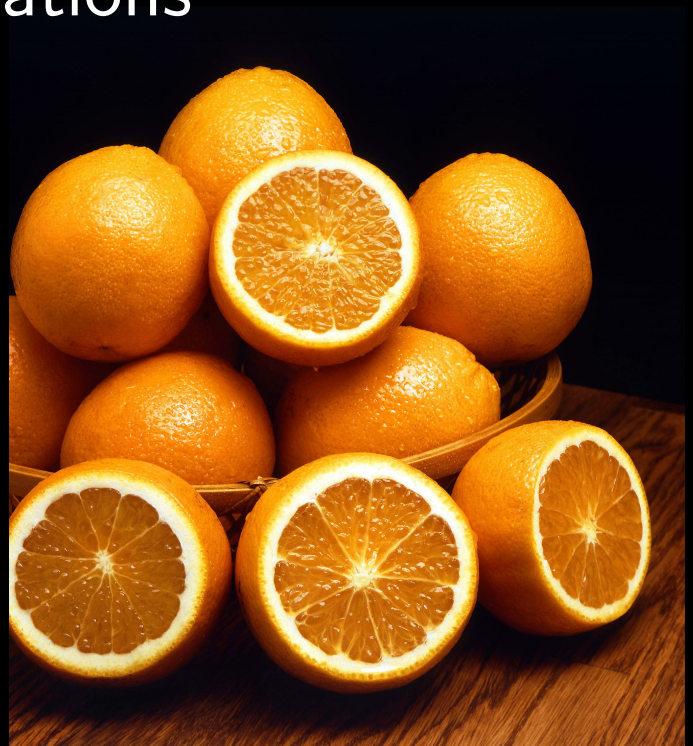
- Building decision rules

Machine learning The 90s

- Learn these from observations

Statistical learning 2000s

- Model the noise in the observations



Artificial Intelligence The 80s

- Building decision rules

Machine learning The 90s

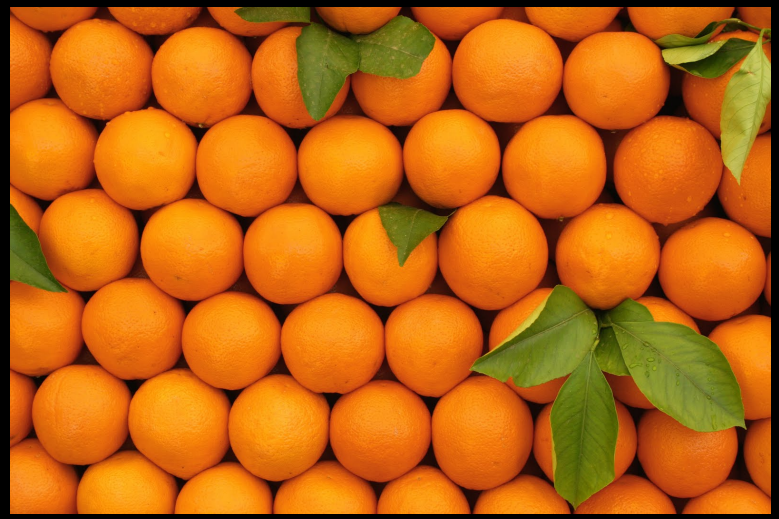
- Learn these from observations

Statistical learning 2000s

- Model the noise in the observations

Big data today

- Many observations, simple rules



Artificial Intelligence The 80s

- Building decision rules

Machine learning The 90s

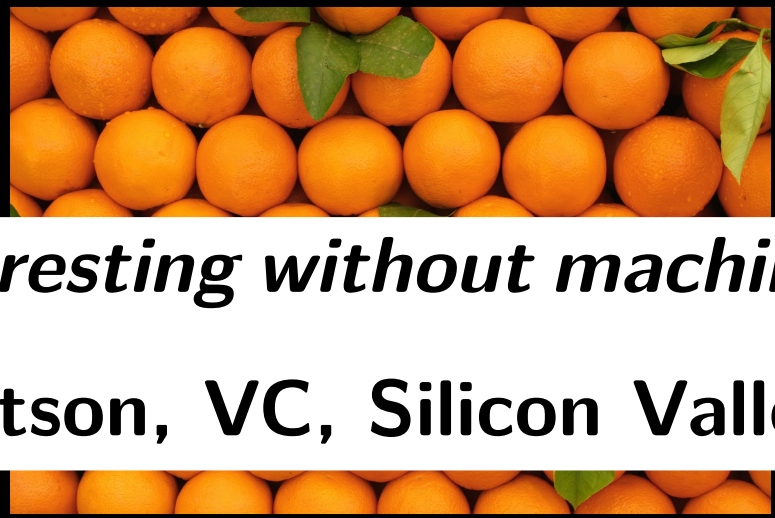
- Learn these from observations

Statistical learning 2000s

- Model the noise in the observations

Big data today

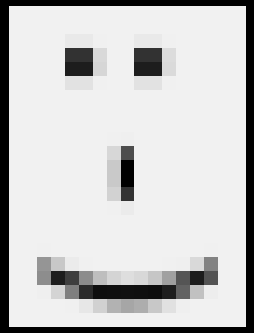
- Many observations, simple rules



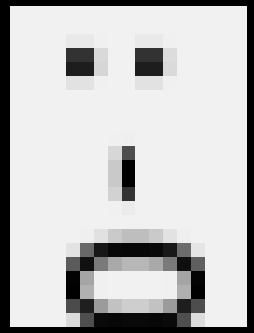
"Big data isn't actually interesting without machine learning"

Steve Jurvetson, VC, Silicon Valley

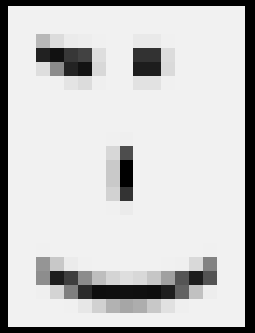
Example: face recognition



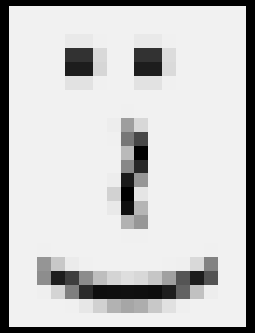
Andrew



Bill

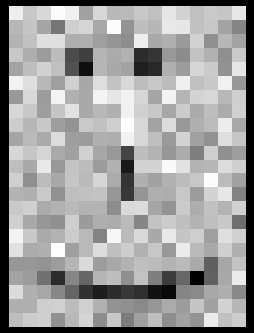


Charles

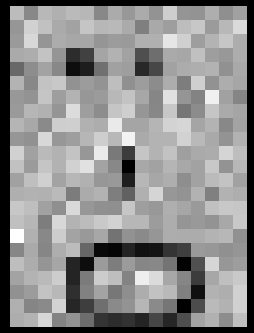


Dave

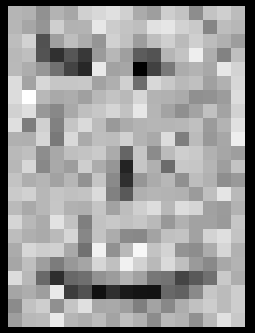
Example: face recognition



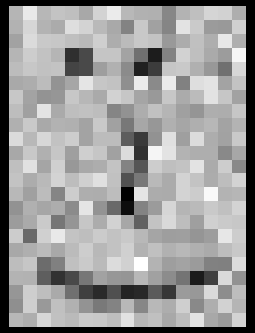
Andrew



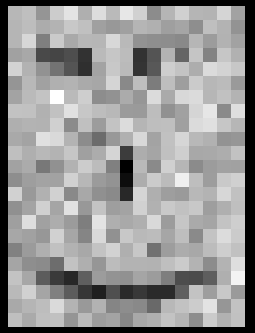
Bill



Charles



Dave



?

1 A simple method

1. Store all the known (noisy) images and the names that go with them.

2. From a new (noisy) images, find the image that is most similar.

"Nearest neighbor" method

1 Store all the known (noisy) images and the names that go with them.

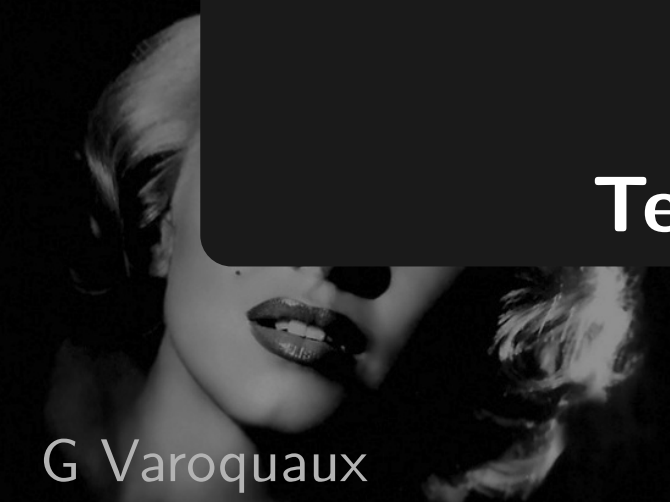
2 From a new (noisy) images, find the image that is most similar.

"Nearest neighbor" method

How many errors on already-known images?

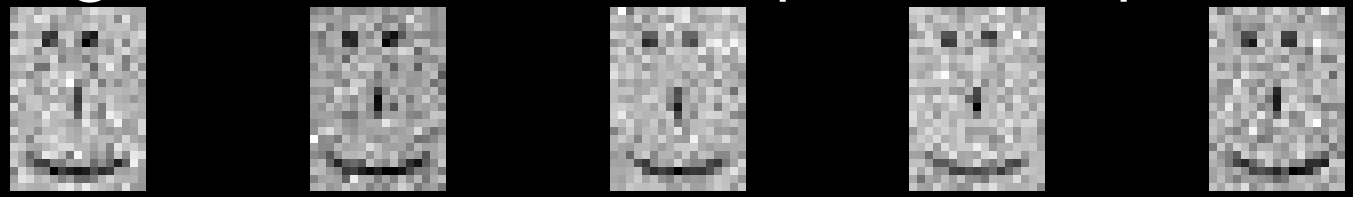
... 0: no erreurs

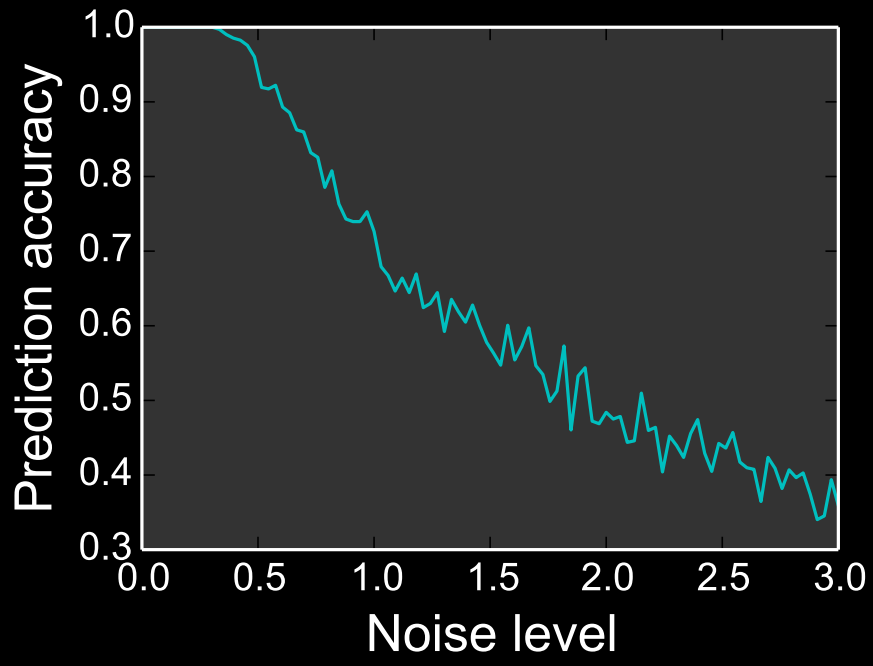
Test data $\neq$ Train data

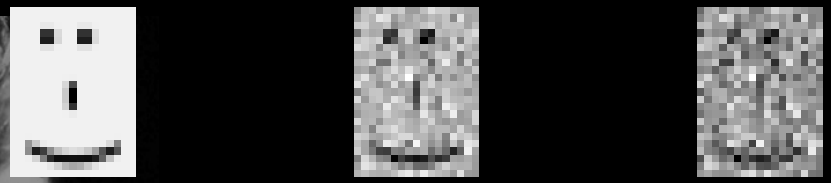


1 1st problem: noise

Signal unrelated to the prediction problem

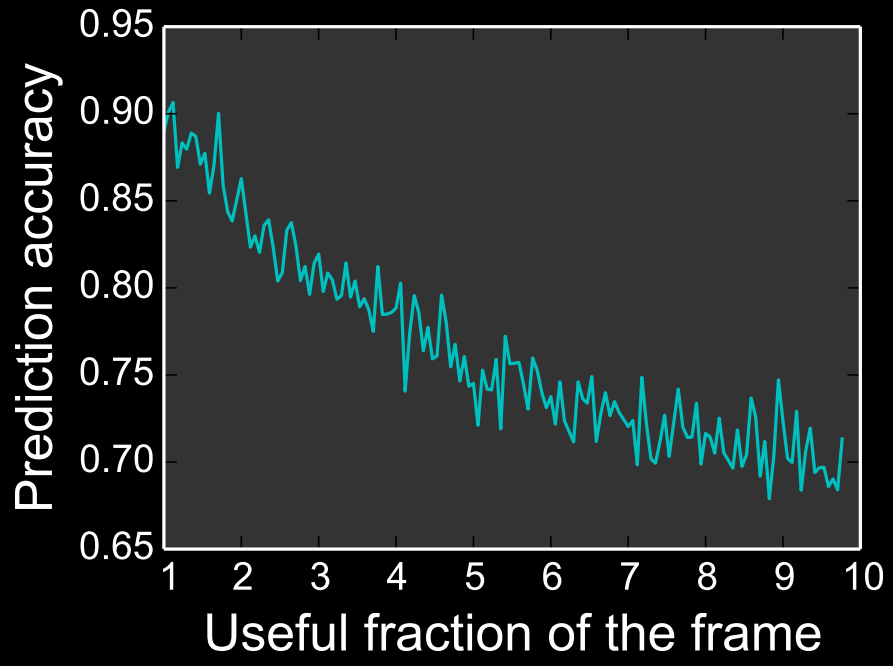


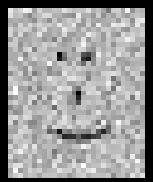


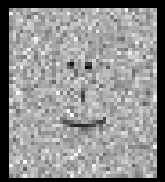


2nd problem: number of variables

Finding a needle in a haystack





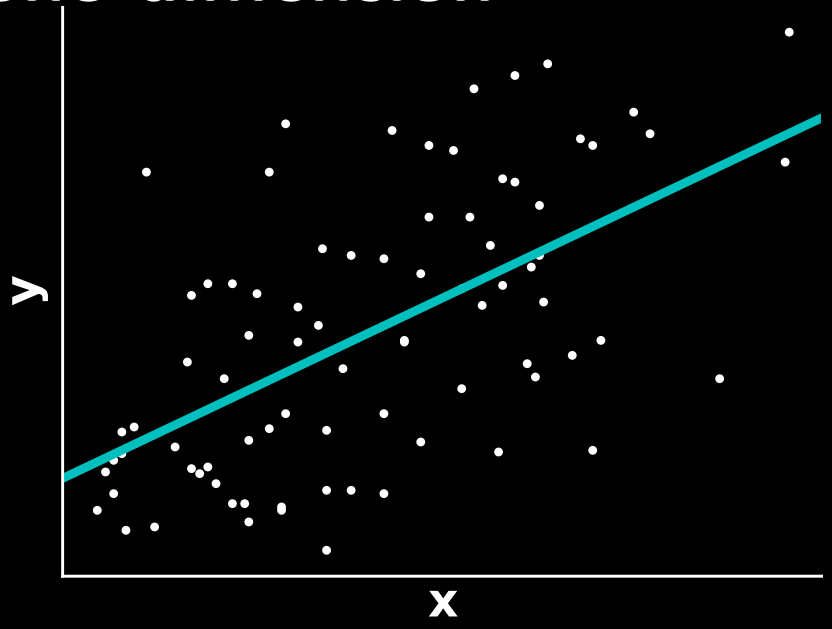


Example: face recognition

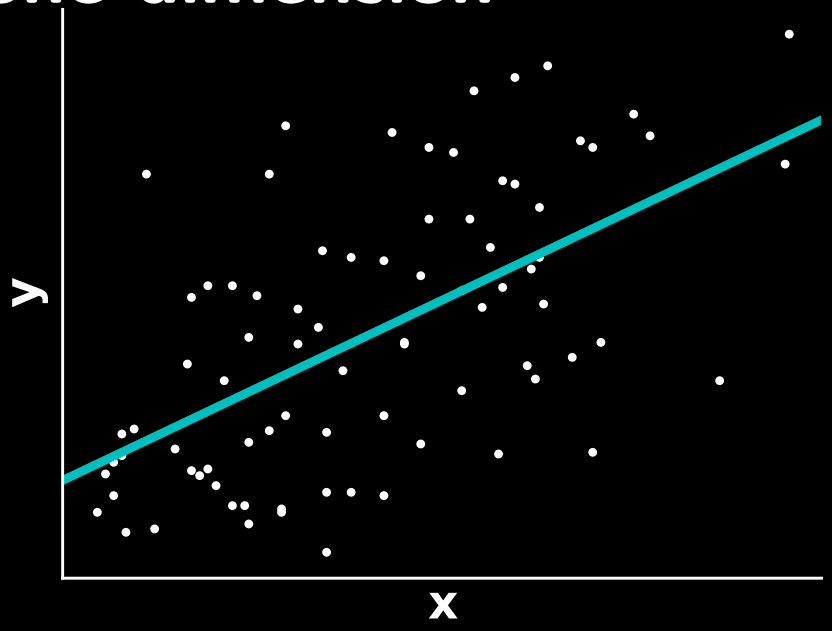
- **Learning from numerical descriptors**
- **Difficulties:** *i)* **noise,**
  ii) **number of features**
- ***"supervised"* task: known labels**
  ***"unsupervised"* task: unknown labels**

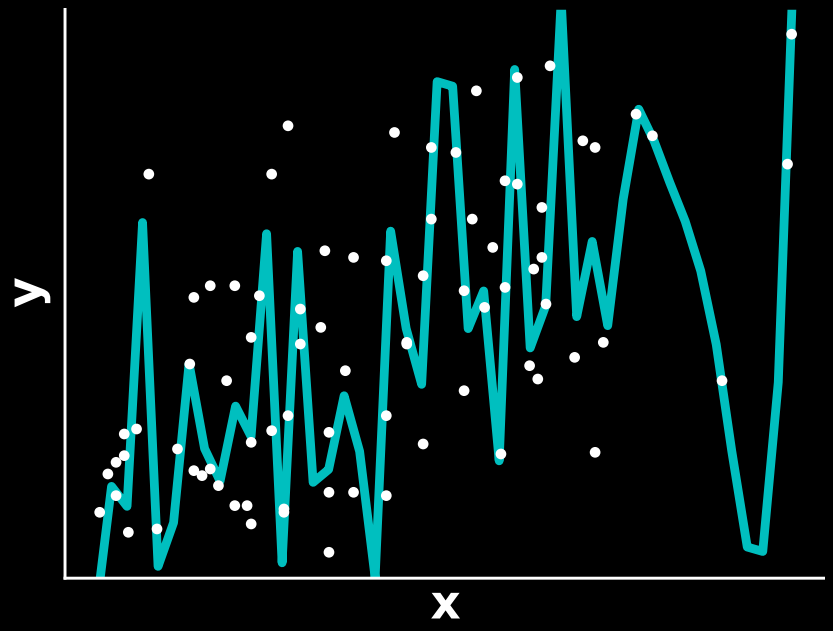
?

A single descriptor: one dimension



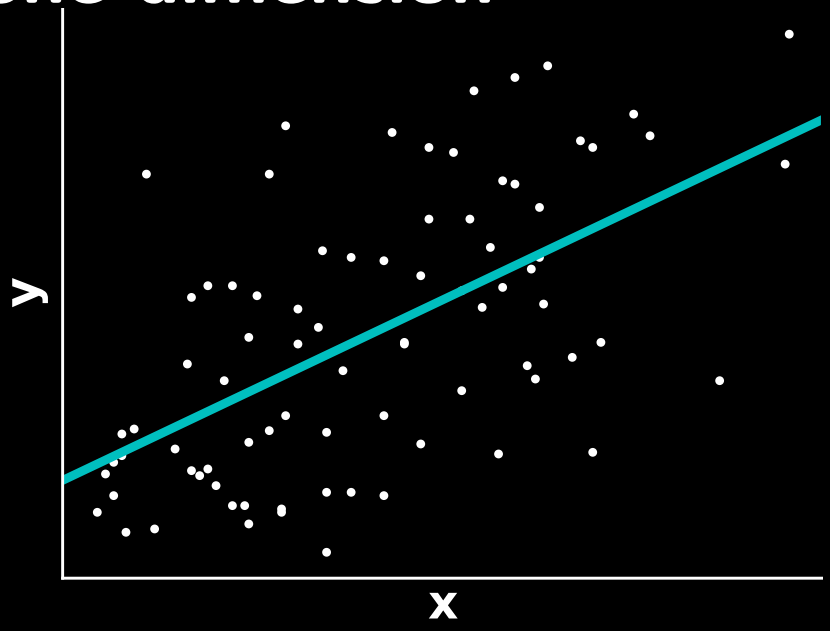
A single descriptor: one dimension

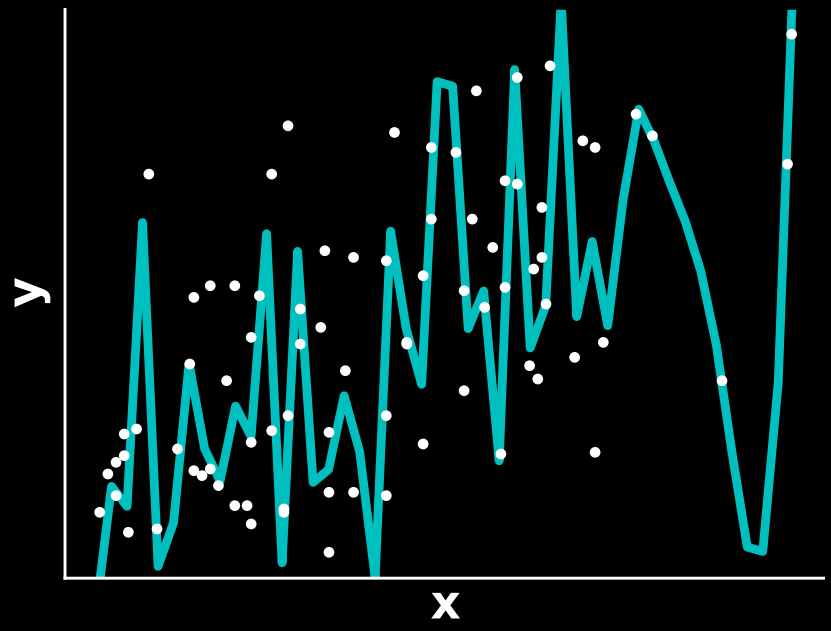




Which model to prefer?

A single descriptor: one dimension

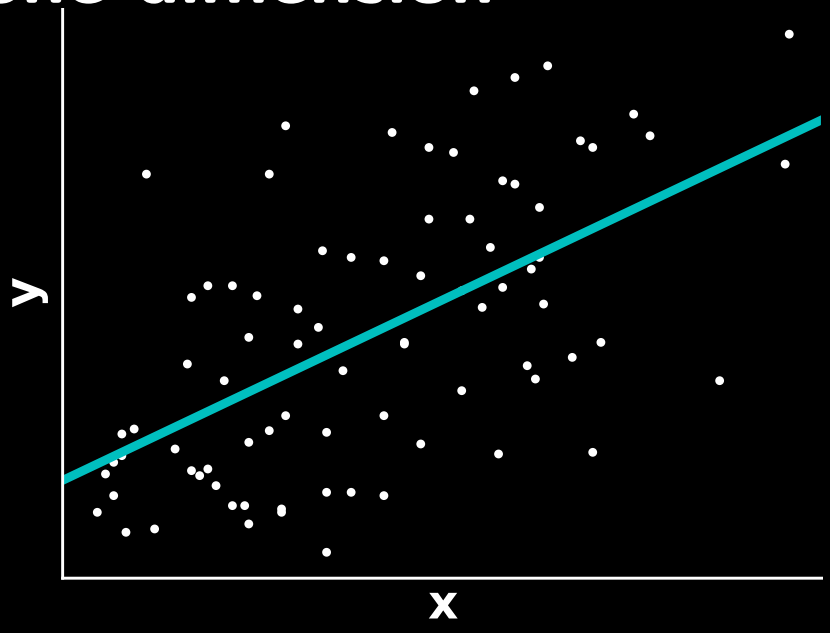


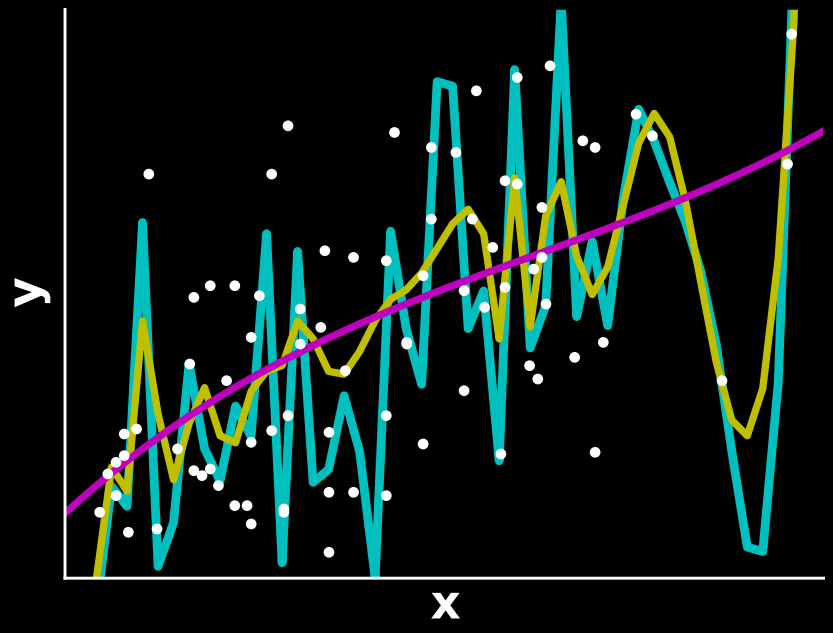


Problem of *"over-fitting"*

- Minimizing error is not always the best strategy (learning noise)
- Test data $\neq$ train data

A single descriptor: one dimension



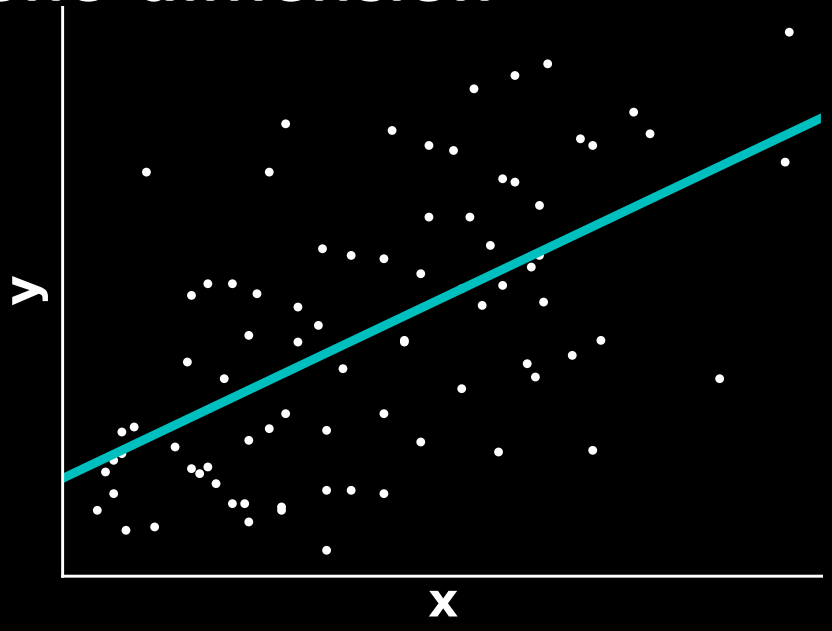


Prefer simple models

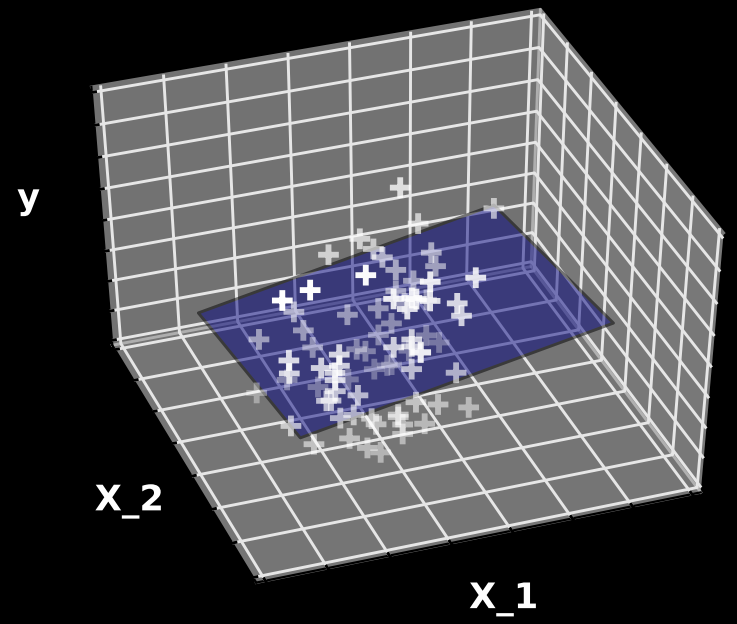
= concept of *"regularization"*

Balance the number of parameters to learn with the amount of data

A single descriptor: one dimension

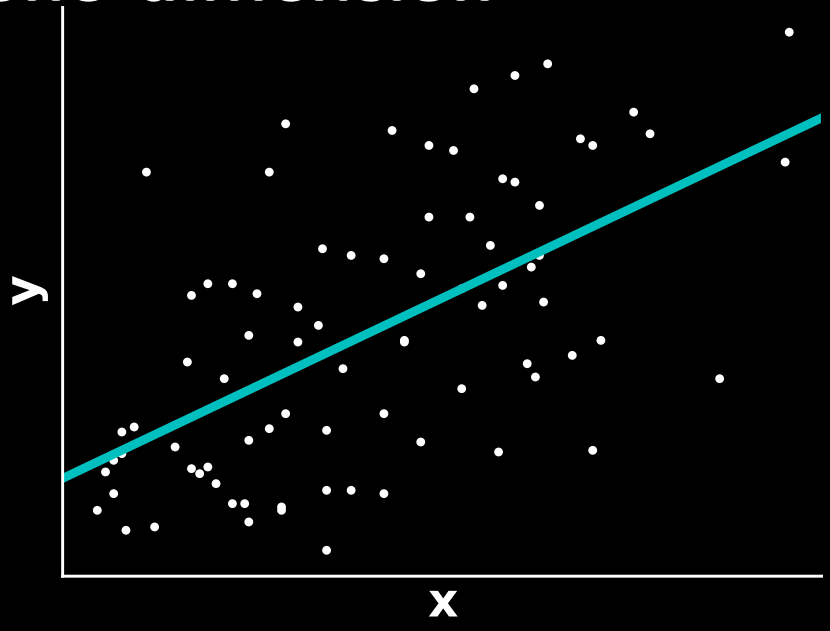


Two descriptors: 2 dimensions

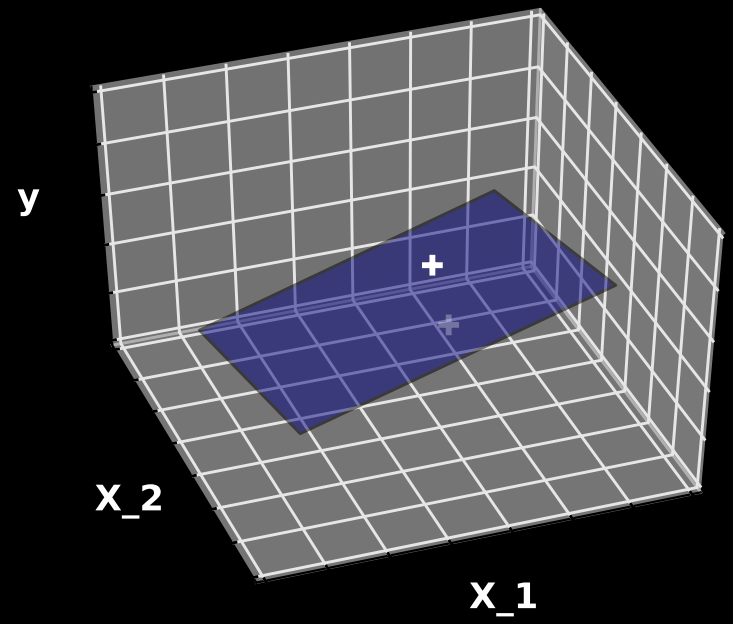


More parameters

A single descriptor: one dimension



Two descriptors: 2 dimensions

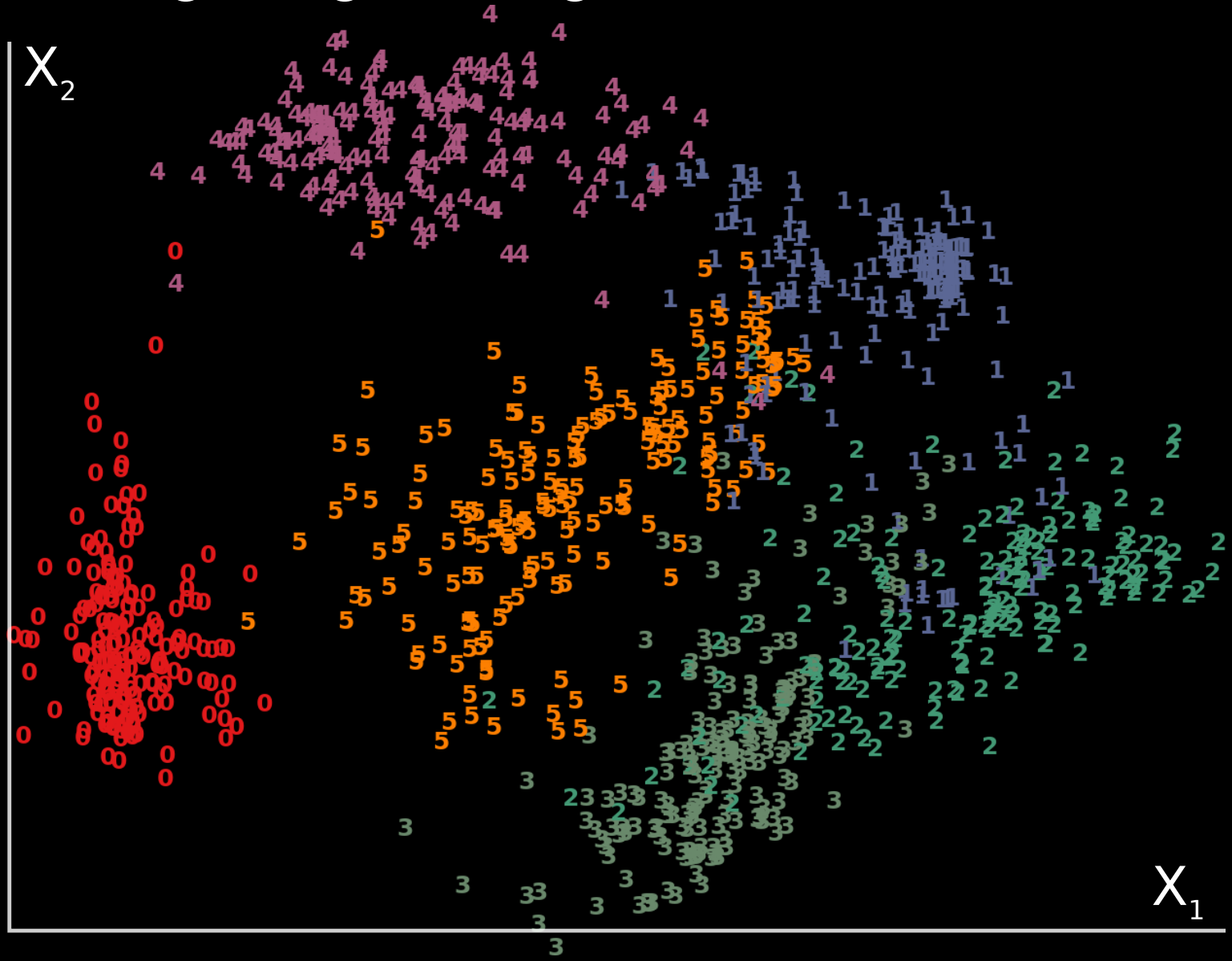


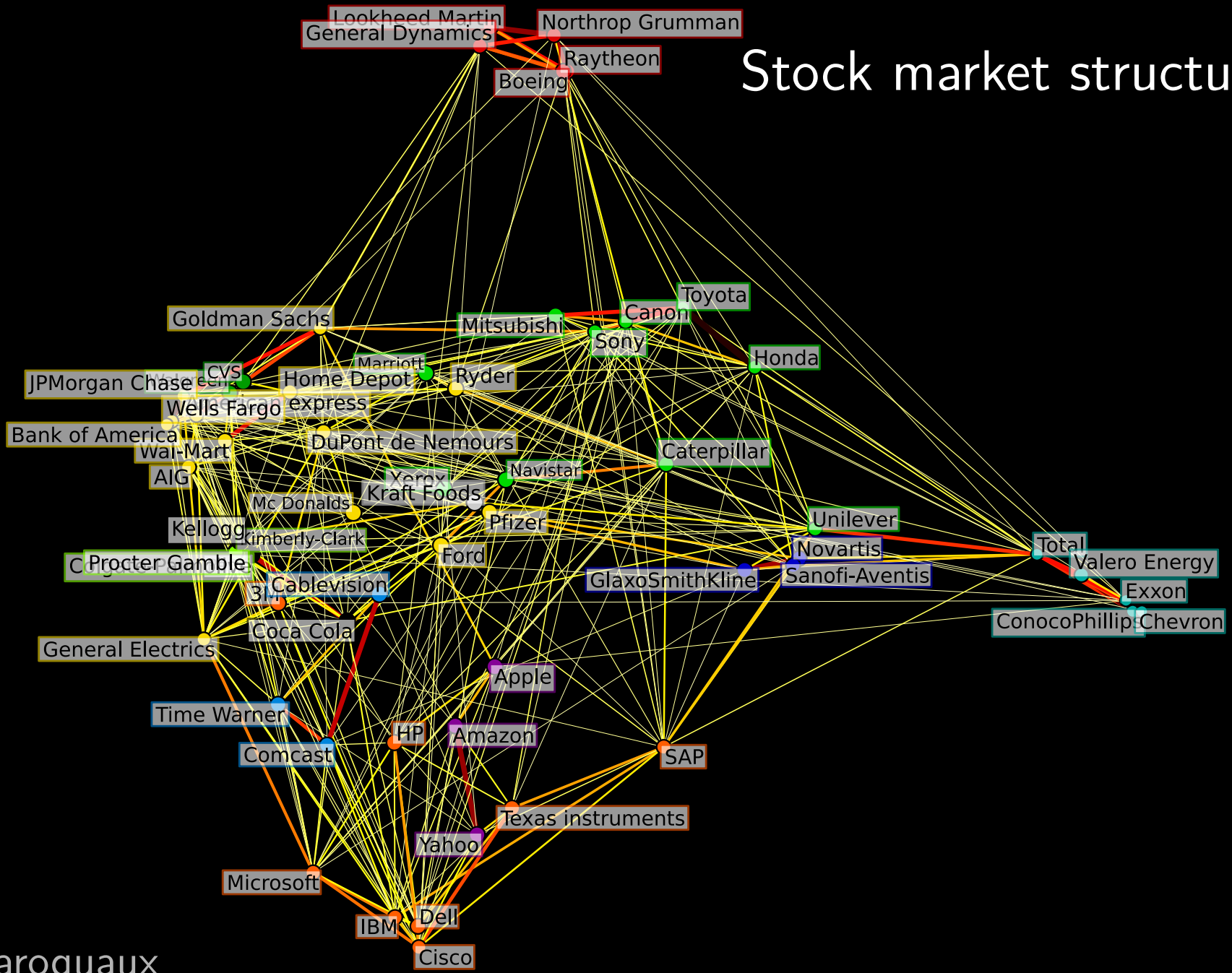
More parameters

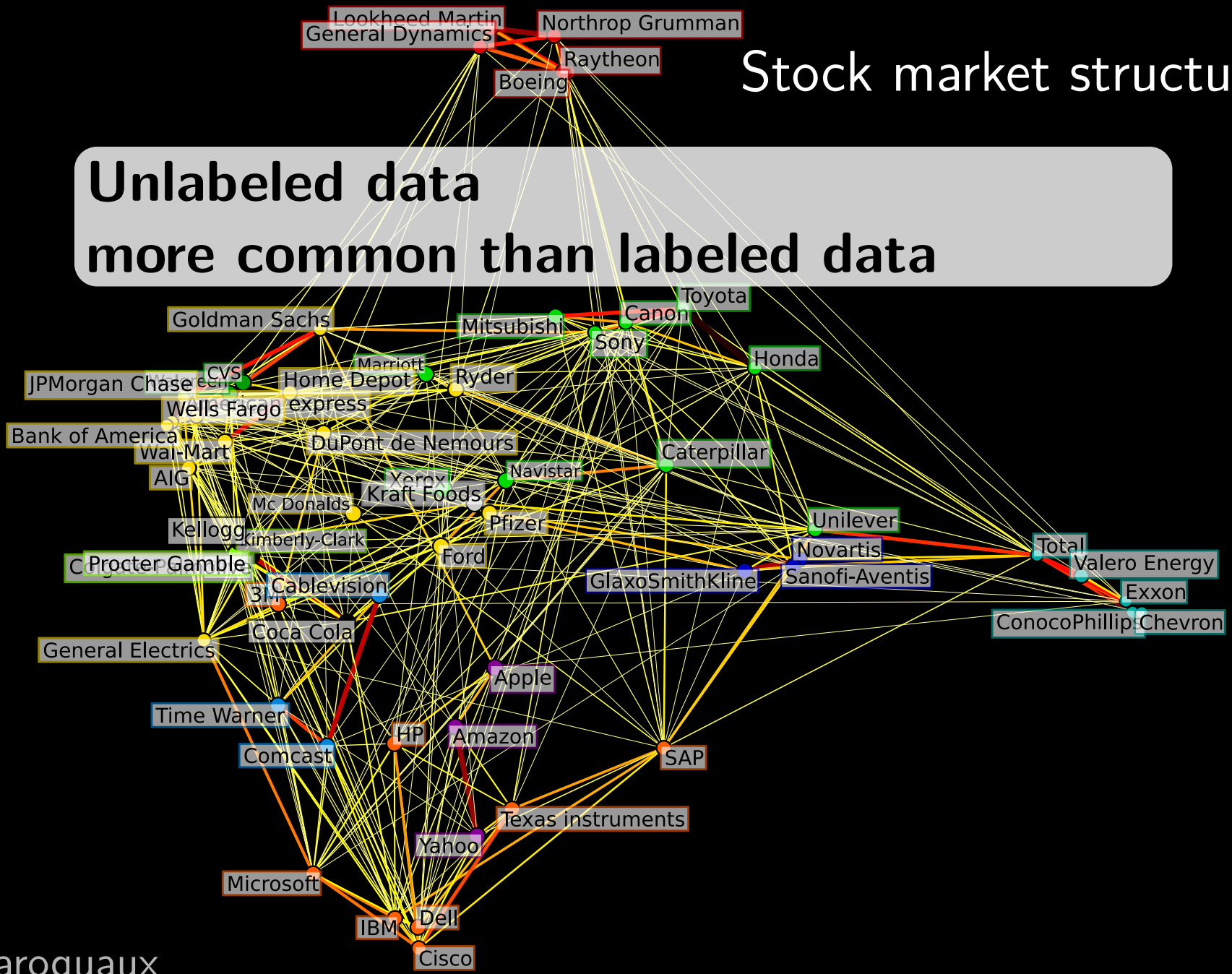
$\Rightarrow$ need more data

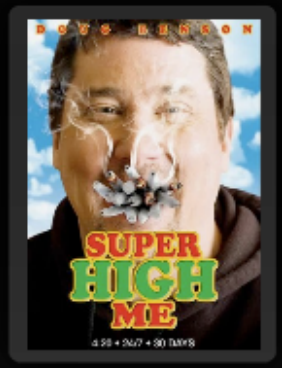
"curse of dimensionality"

Predicting categories, e.g. numbers

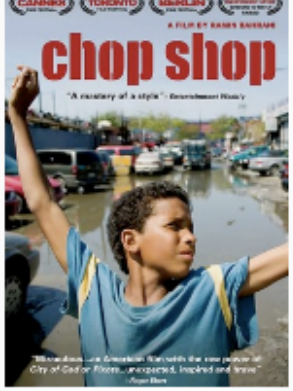



Stock market structure
Lookheed Martin
Northrop Grumman
General Dynamics
Raytheon
Boeing
Toyota
Canon
Goldman Sachs
Mitsubishi
Sony
Honda
Marriott
CVS
Home Depot
Ryder
JPMorgan Chase
express
Wells Fargo
Bank of America
DuPont de Nemours
Caterpillar
Wal-Mart
AIG
Navistar
Kraft Foods
Mc Donalds
Pfizer
Unilever
Kellogg
Kimberly-Clark
Total
Novartis
Procter Gamble
Ford
Valero Energy
GlaxoSmithKline
Sanofi-Aventis
Cablevision
Exxon
3M
ConocoPhillip
Chevron
Coca Cola
General Electrics
Apple
Time Warner
HP
Amazon
Comcast
SAP
Texas instruments
Yahoo
Microsoft
Dell
IBM
Cisco


Stock market structure
Unlabeled data
more common than labeled data
Lookheed Martin
Northrop Grumman
General Dynamics
Raytheon
Boeing
Toyota
Canon
Goldman Sachs
Mitsubishi
Sony
Honda
Marriott
CVS
Home Depot
Ryder
JPMorgan Chase
express
Wells Fargo
Bank of America
DuPont de Nemours
Wal-Mart
Caterpillar
AIG
Navistar
Kraft Foods
Mc Donalds
Pfizer
Unilever
Kellogg
Kimberly-Clark
Total
Novartis
Procter Gamble
Ford
Valero Energy
Cablevision
GlaxoSmithKline
Sanofi-Aventis
Exxon
3M
ConocoPhillip
Chevron
Coca Cola
General Electrics
Apple
Time Warner
HP
Amazon
Comcast
SAP
Texas instruments
Yahoo
Microsoft
Dell
IBM
Cisco



★★★☆☆
Super High Me



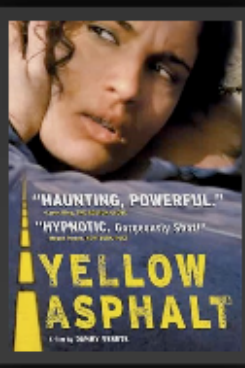
★★★☆☆
Chop Shop

★★½☆☆
Romance &
Cigarettes

★★★★☆
Scarface

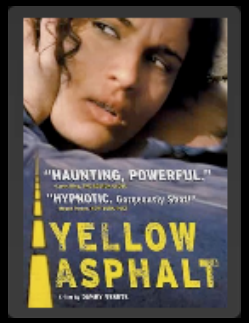
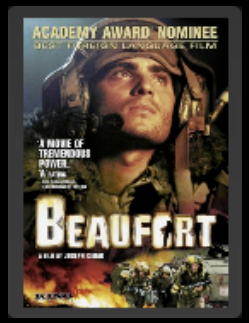
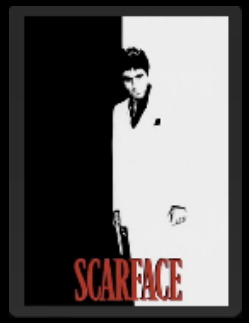
★★★½☆
Donnie Brasco

★★★☆☆
Faithless



★★★☆☆
Yellow Asphalt

★★★☆☆
Beaufort

					
Andrew	⋆	⋆⋆⋆			
Bill			⋆⋆		⋆⋆⋆
Charles	⋆⋆⋆			⋆⋆	
Dave		⋆⋆⋆			⋆
Edie			⋆⋆	⋆⋆⋆	

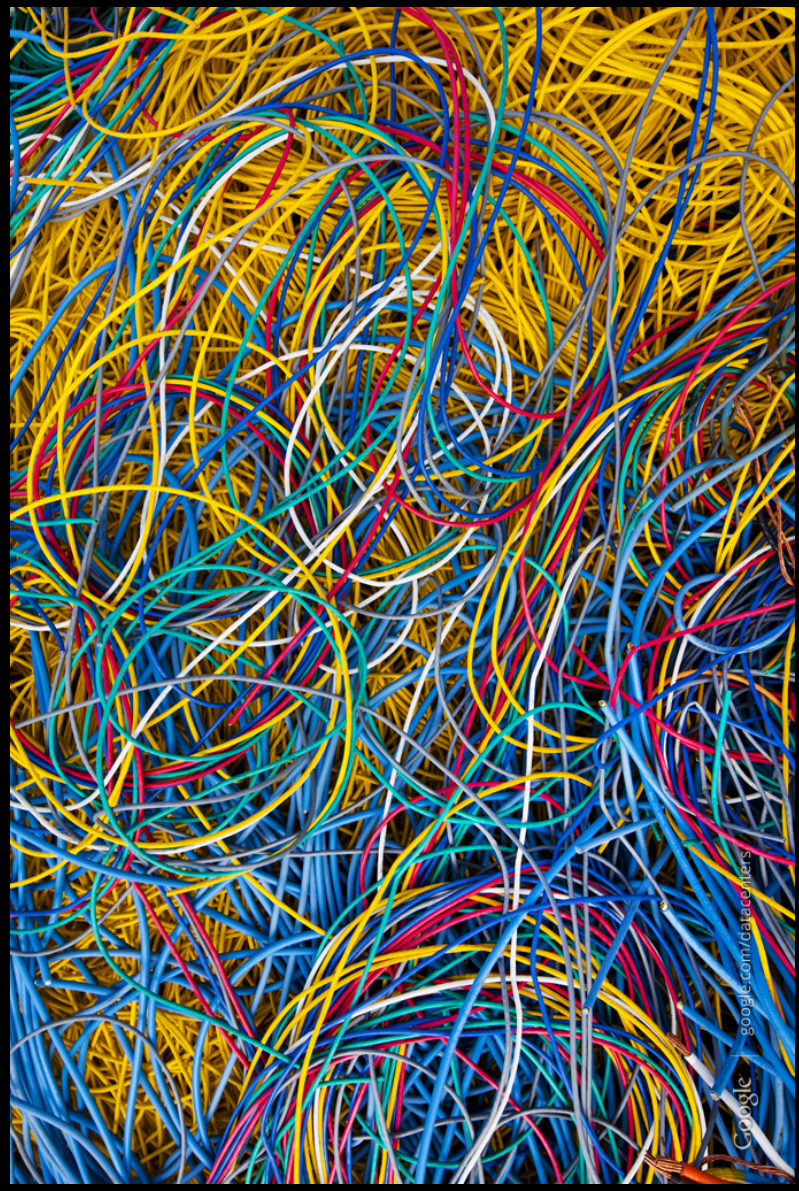
Little overlap between users

Challenges

- Statistics
- Computational

1 Petty day-to-day technicalities

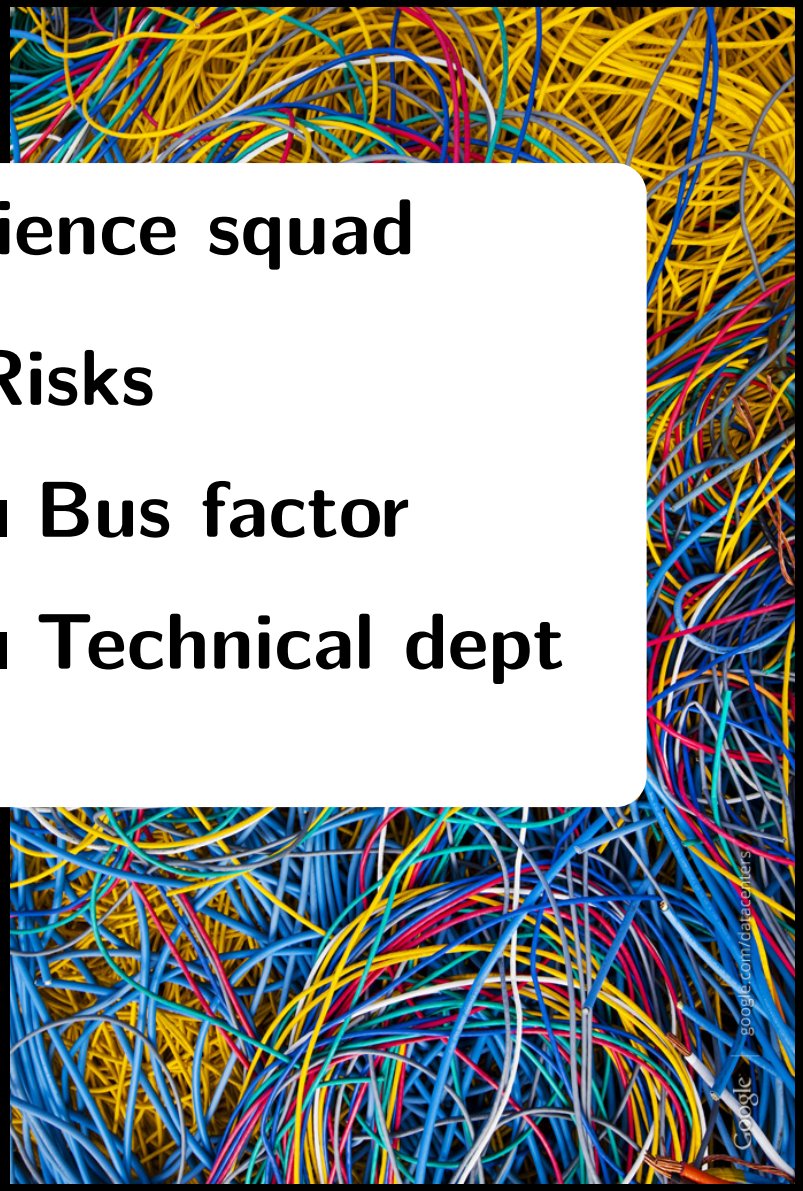
- Buggy code
- Slow code
- Lead data scientist leaves
- New intern to train
- I don't understand the code I have written a year ago



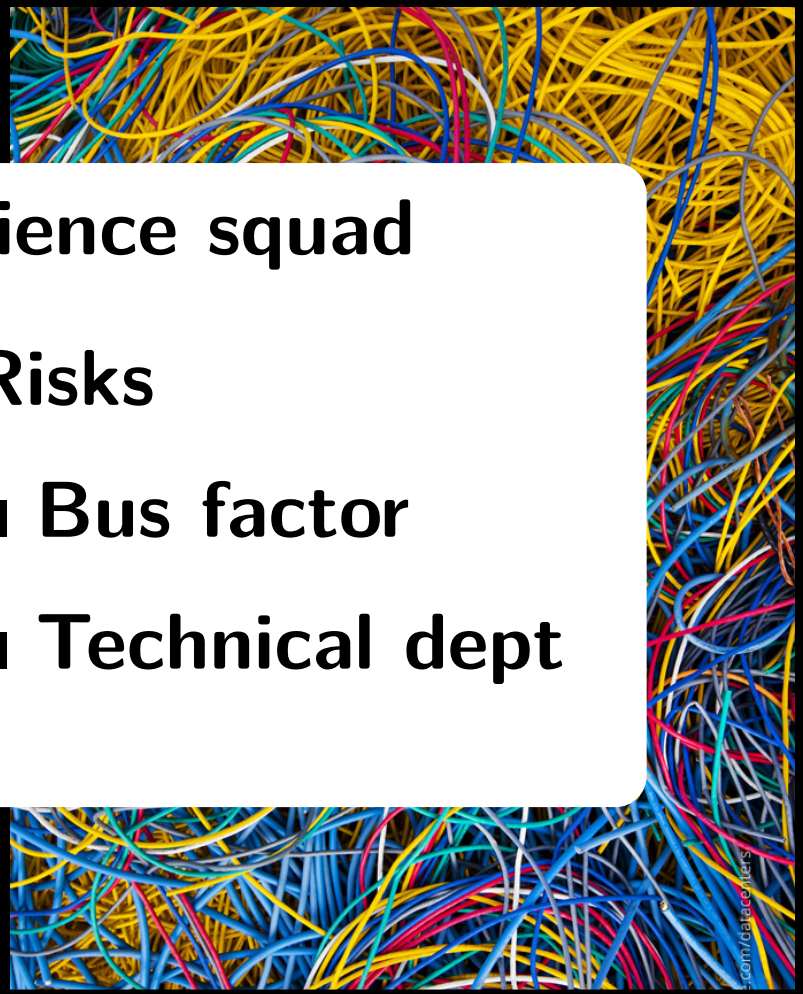
- Bu
- Sl
- Le
- Ne
- I don't understand the code I have written a year ago

An in-house data science squad

Difficulties	Risks
■ Recruitment	■ Bus factor
■ Limited resources (people & hardware)	■ Technical dept



- Bu
- Sl
- Le
- Ne
- I don't understand the code I have written a year ago



An in-house data science squad

Difficulties	**Risks**
■ **Recruitment**	■ **Bus factor**
■ **Limited resources (people & hardware)**	■ **Technical dept**

We need big data (and machine learning)
on a tight budget

2 Scikit-learn

Machine learning without learning the machinery

©Theodore W. Gray

Python, what else?

- Interactive language
- Easy to read / write
- General purpose

Python, what else?

The scientific Python stack

- numpy arrays
- pandas

...

It's about plugin things together

2 scikit-learn vision

Machine learning for all

No specific application domain

No requirements in machine learning

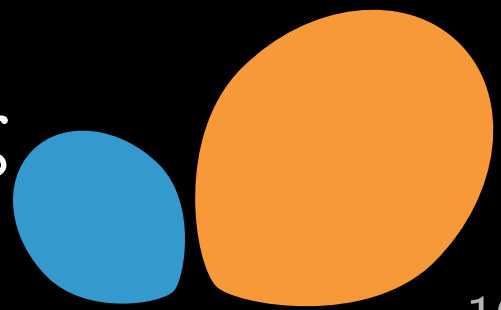
High-quality software library

Interfaces designed for users

Community-driven development

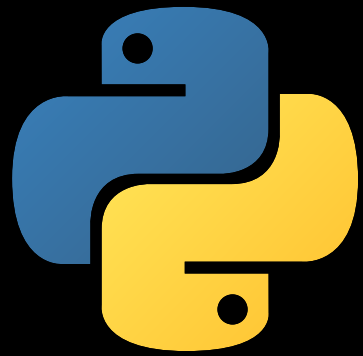
BSD licensed, very diverse contributors

`http://scikit-learn.org`



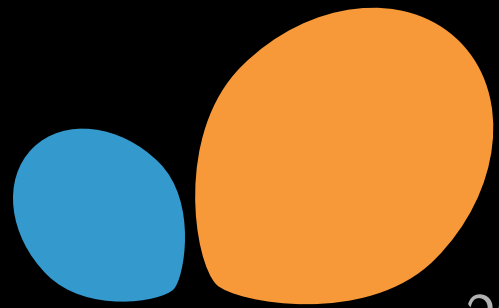
A library, not a program

- More expressive and flexible
- Easy to include in an ecosystem



As easy as py

```
from sklearn import svm
classifier = svm.SVC()
classifier.fit(X_train, Y_train)
Y_test = classifier.predict(X_test)
```



2 Very rich feature set

Supervised learning

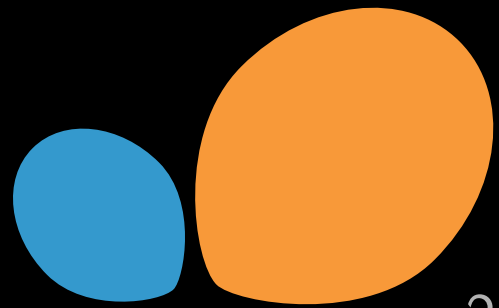
- Decision trees (Random-Forest, Boosted Tree)
- Linear models
- SVM

Unsupervised Learning

- Clustering
- Dictionary learning
- Outlier detection

Model selection

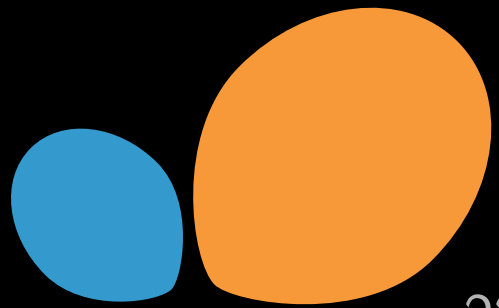
- Built in cross-validation
- Parameter optimization



2 Computational performance

	scikit-learn	mlpy	pybrain	pymvpa	mdp	shogun
SVM	**5.2**	9.47	17.5	11.52	40.48	5.63
LARS	**1.17**	105.3	-	37.35	-	-
Elastic Net	**0.52**	73.7	-	1.44	-	-
kNN	0.57	1.41	-	**0.56**	0.58	1.36
PCA	**0.18**	-	-	8.93	0.47	0.33
k-Means	1.34	0.79	∞	-	35.75	**0.68**

- Algorithmic optimizations
- Minimizing data copies



2 Computational performance

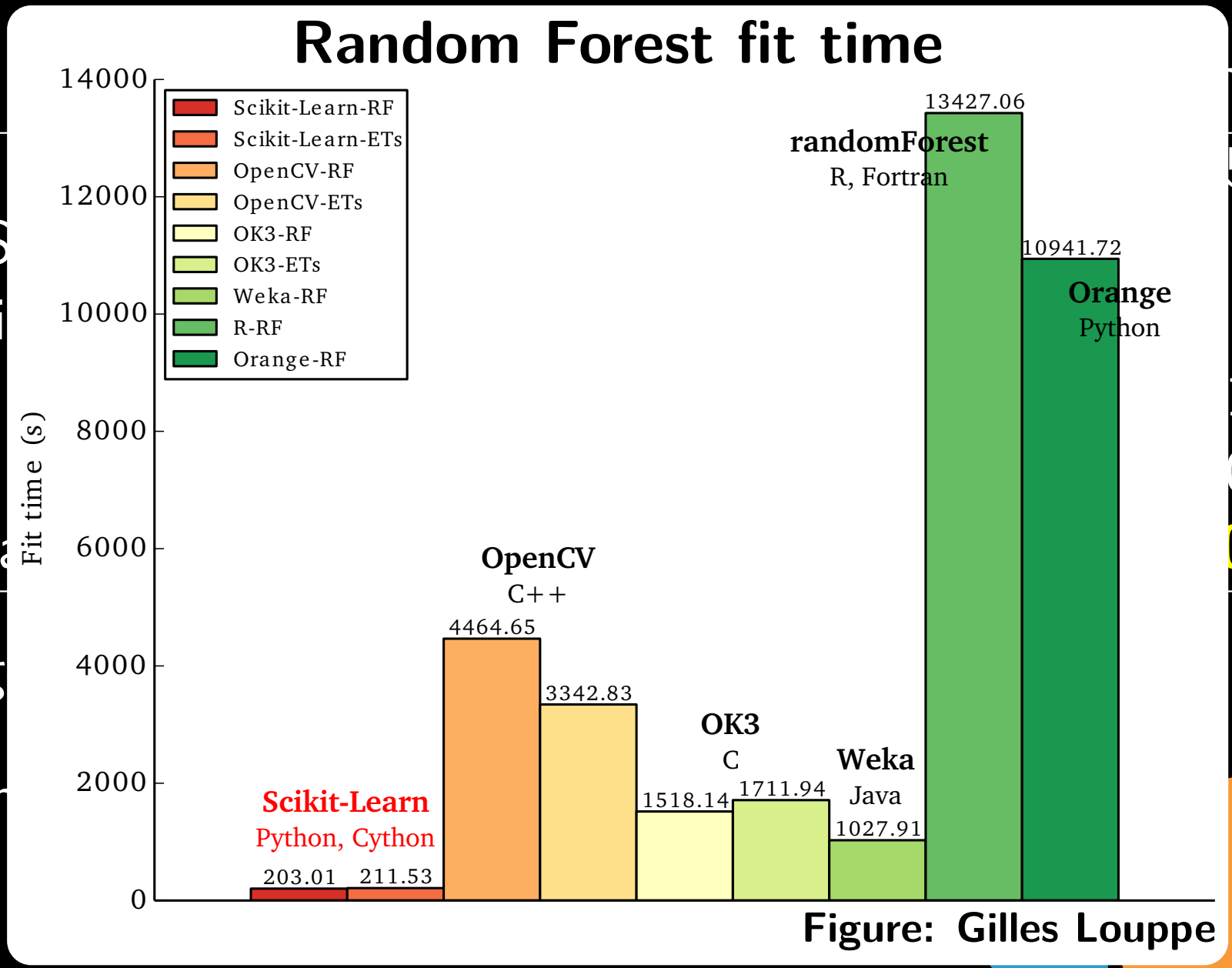


Figure: Gilles Louppe

3 Big data on a budget

3 Bigish data on a budget

“Big data”:

- Petabytes…
- Distributed storage
- Computing cluster

Mere mortals:

- Gigabytes…
- Python programming
- Off-the-self computers

WE ARE THE 99%

3 Big data on a budget

Bigish

"Big data":

- Petabytes
- Distributed storage
- Computing cluster

Simple data processing patterns

Modest needs:

- Gigabytes...
- Python programming
- Off-the-self computers

WE ARE THE 99%

"Big data", but big how?

2 scenarios:

- Many observations –samples

  e.g. twitter

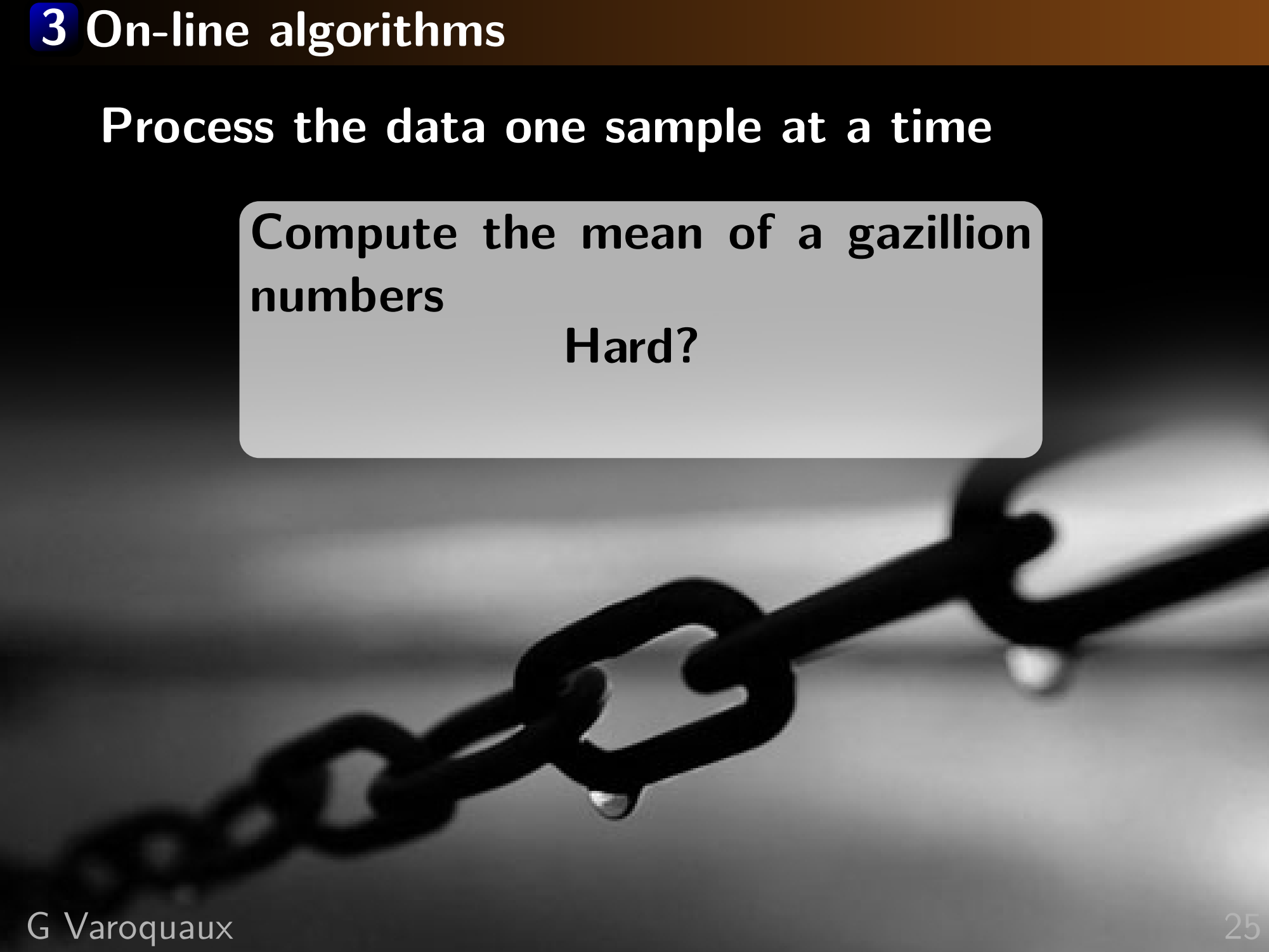
- Many descriptors per observation –features

  e.g. brain scans

Process the data one sample at a time

Compute the mean of a gazillion numbers

Hard?

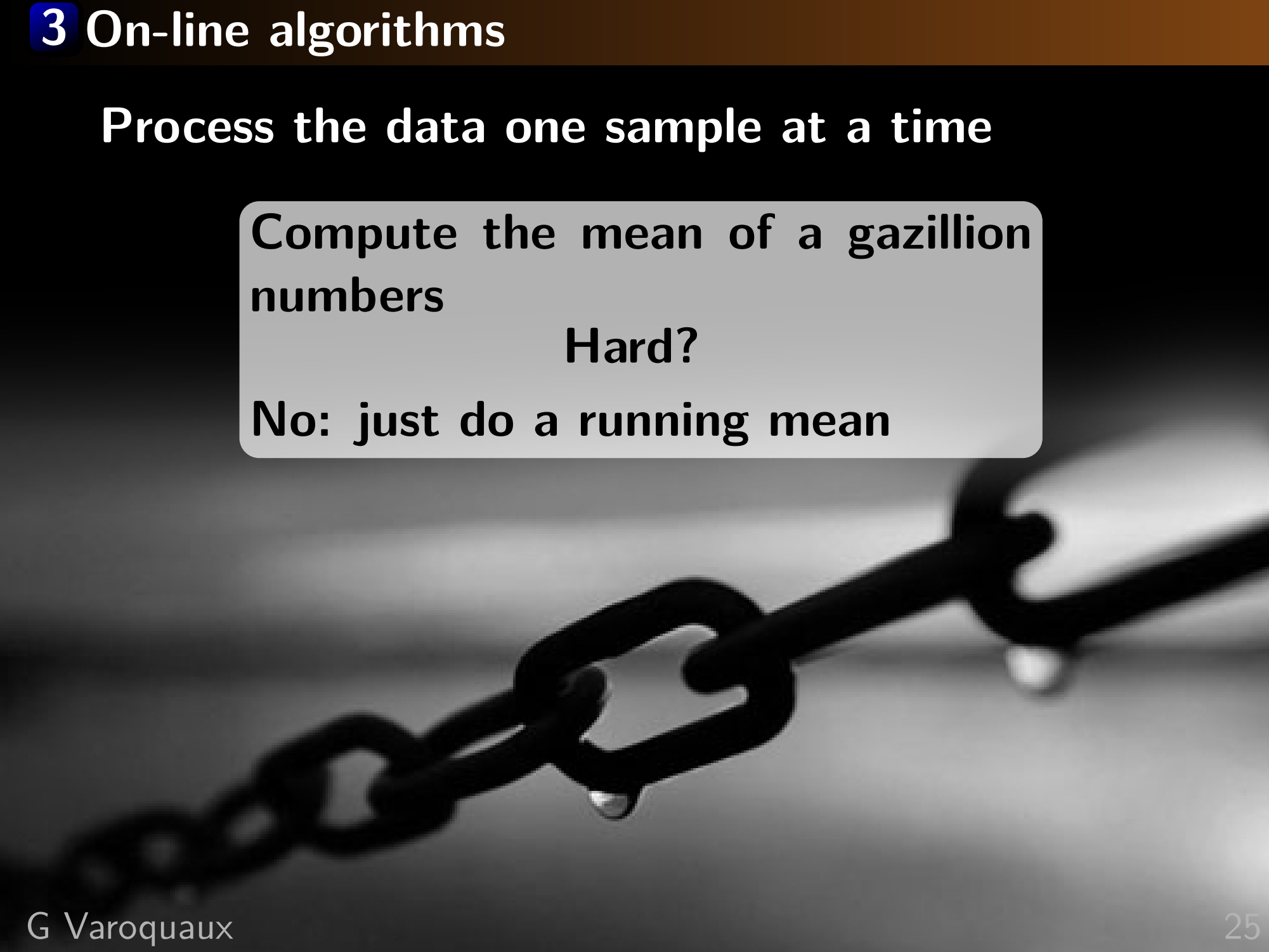


Process the data one sample at a time

Compute the mean of a gazillion numbers

Hard?

No: just do a running mean



3 On-line algorithms

- Converges to expectations

- **Mini-batch** = bunch observations for vectorization

Example: K-Means clustering

X = np.random.normal(size=(10 000, 200))

scipy.cluster.vq.	sklearn.cluster.
kmeans(X, 10, iter=2)	MiniBatchKMeans(n_clusters=10, n_init=2).fit(X)
11.33 s	**0.62 s**

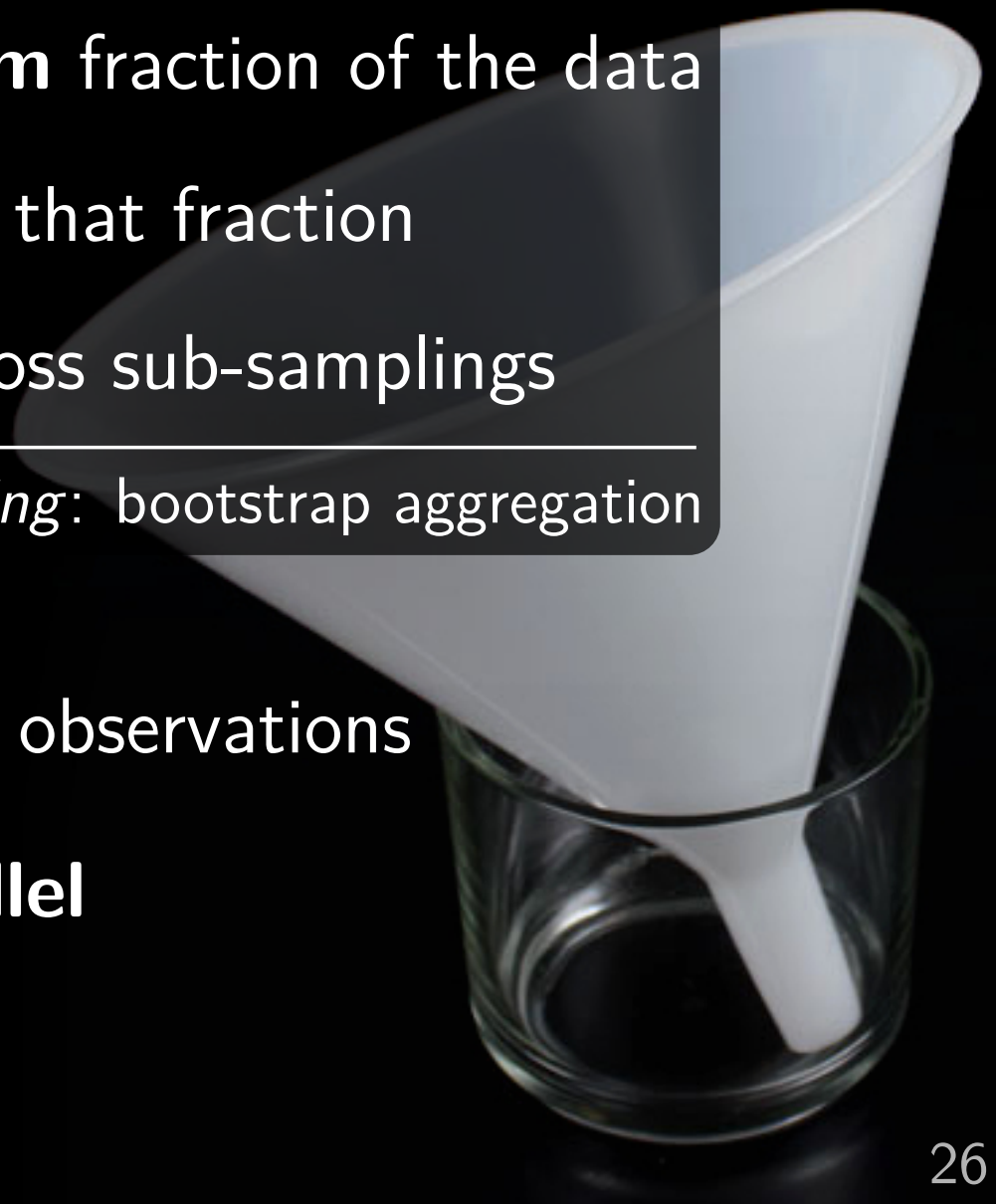
3 On-the-fly data reduction

- Big data is often I/O bound
- Layer memory access
  - CPU caches
  - RAM
  - Local disks
  - Distant storage
- Less data also means less work

Dropping data

1. `loop:` take a **random** fraction of the data
2. run algorithm on that fraction
3. aggregate results across sub-samplings

Looks like bagging: bootstrap aggregation



Exploits redundancy across observations

Run the loop in parallel

3 On-the-fly data reduction

Random projections (will average features)

`sklearn.random_projection`

random linear combinations of the features

Fast clustering of features

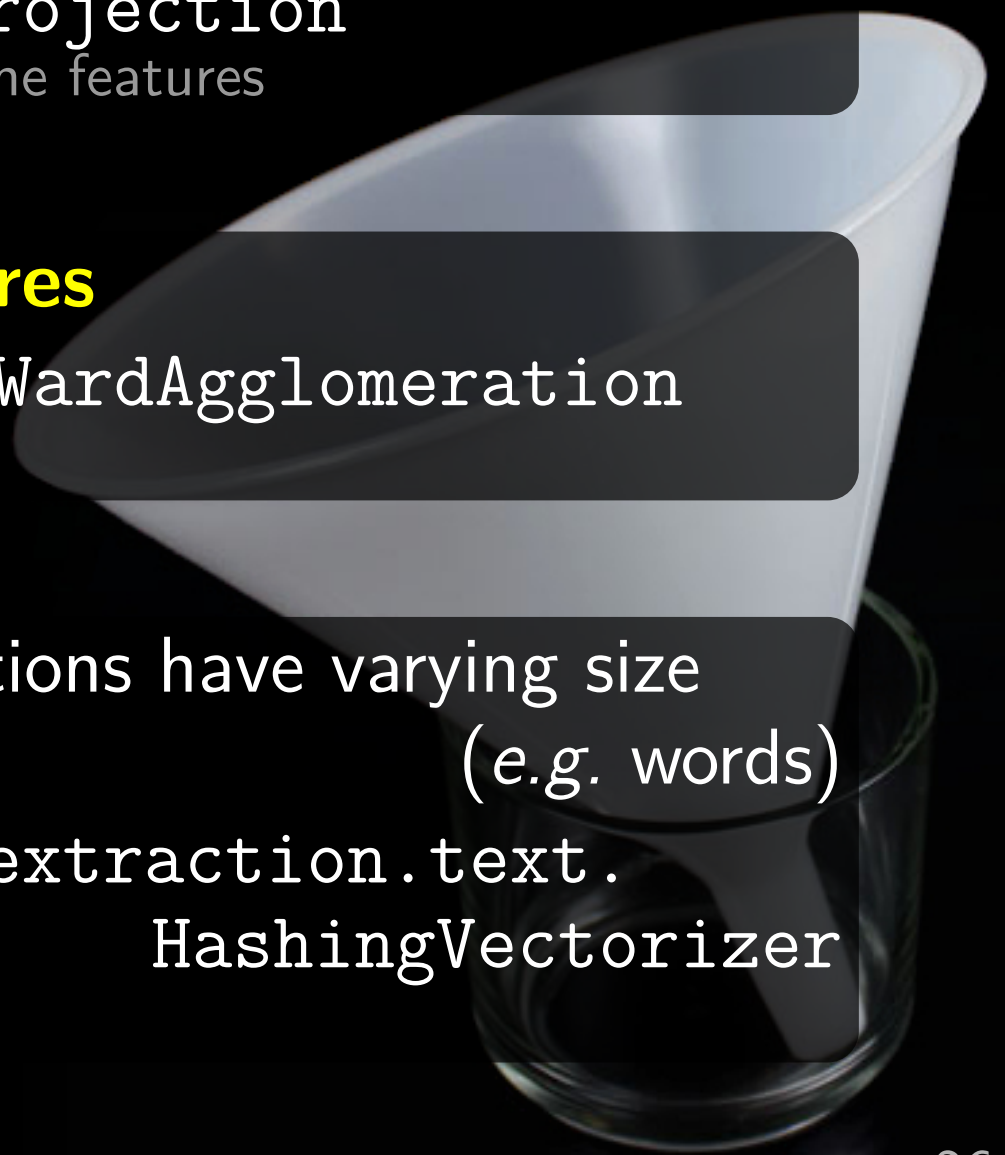
`sklearn.cluster.WardAgglomeration`

on images: super-pixel strategy

Hashing when observations have varying size (*e.g.* words)

`sklearn.feature_extraction.text.HashingVectorizer`

stateless: can be used in parallel



Example: randomized SVD

Random projection

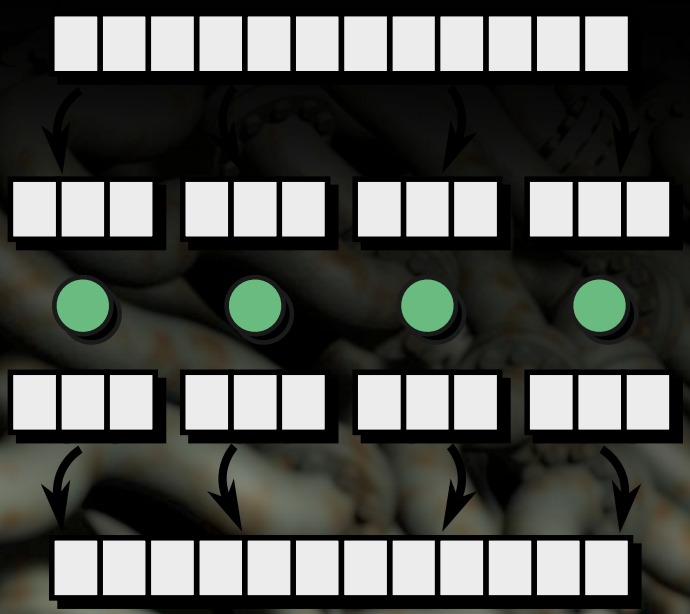
`sklearn.utils.extmath.randomized_svd`

```
X = np.random.normal(size=(50000, 200))
%timeit lapack = linalg.svd(X, full_matrices=False)
  1 loops, best of 3: 6.09 s per loop
%timeit arpack=splinalg.svds(X, 10)
  1 loops, best of 3: 2.49 s per loop
%timeit randomized = randomized_svd(X, 10)
  1 loops, best of 3: 303 ms per loop

linalg.norm(lapack[0][:, :10] - arpack[0]) / 2000
  0.0022360679774997738
linalg.norm(lapack[0][:, :10] - randomized[0]) / 2000
  0.0022121161221386925
```

I tackle only embarrassingly parallel problem

Life is too short to worry about deadlocks



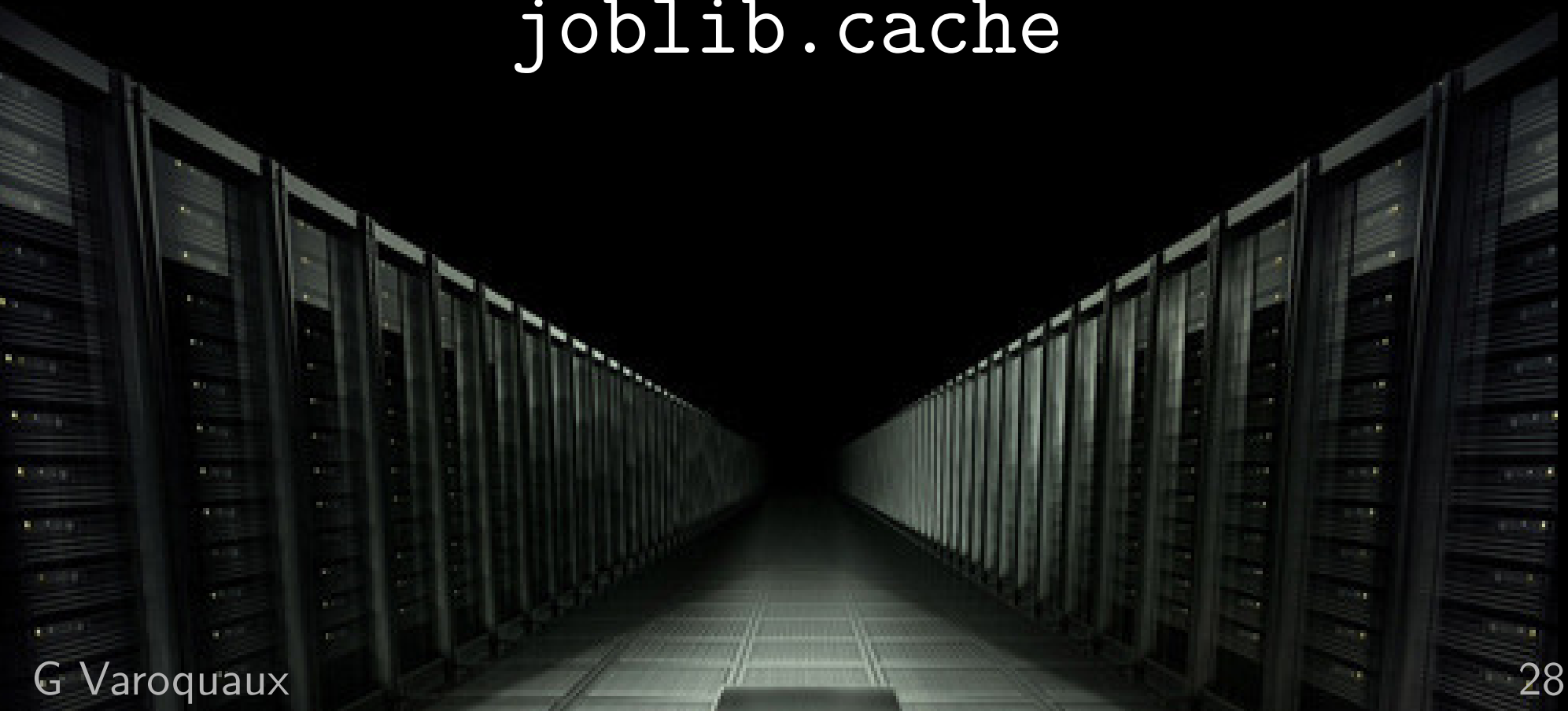
- Stratification to follow the statistical dependencies and the data storage structure
- Batch size scaled by the relevent cache pool
  - Too fine $\Rightarrow$ overhead
  - Too coarse $\Rightarrow$ memory shortage

`joblib.Parallel`

3 Caching

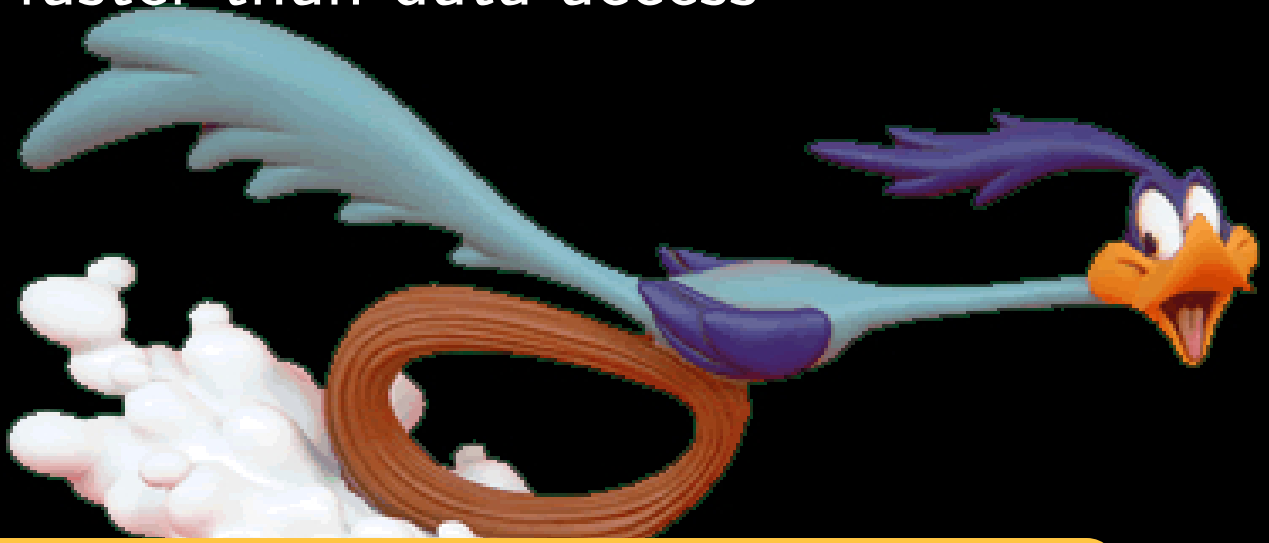
- Minimizing data-access latency
- Never computing twice the same thing

`joblib.cache`



3 Fast access to data

- Stored representation consistent with access patterns
- Compression to limit bandwidth usage
  - CPU are faster than data access



Data access speed is often more a limitation than raw processing power

`joblib.dump/joblib.load`

3 Biggish iron

Our new box: 15 k€

- 48 cores
- 384G RAM
- 70T storage

(SSD cache on RAID controller)

Gets our work done faster than our 800 CPU cluster

It's the access patterns!

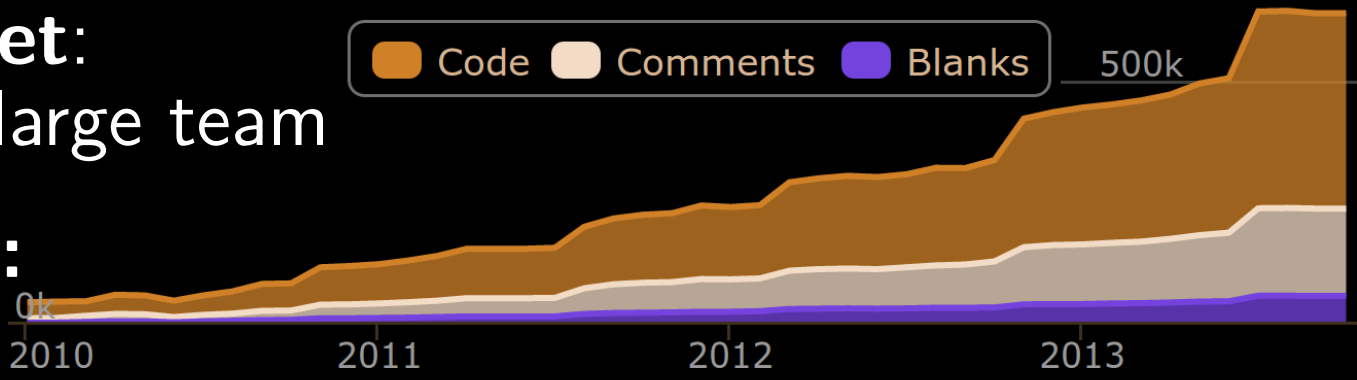
"Nobody ever got fired for using Hadoop on a cluster"
A. Rowstron *et al.*, HotCDP '12

4 The bigger picture: a community

4 Community-based development in scikit-learn

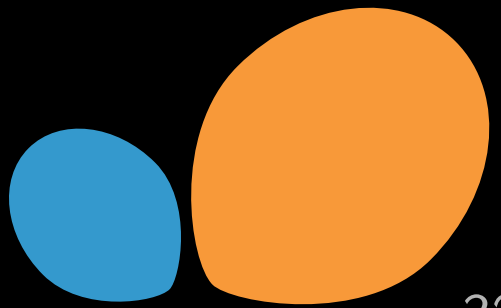
Huge feature set:
benefits of a large team

Project growth:



- More than 200 contributors
- $\sim$ 12 core contributors
- 1 full-time INRIA programmer from the start

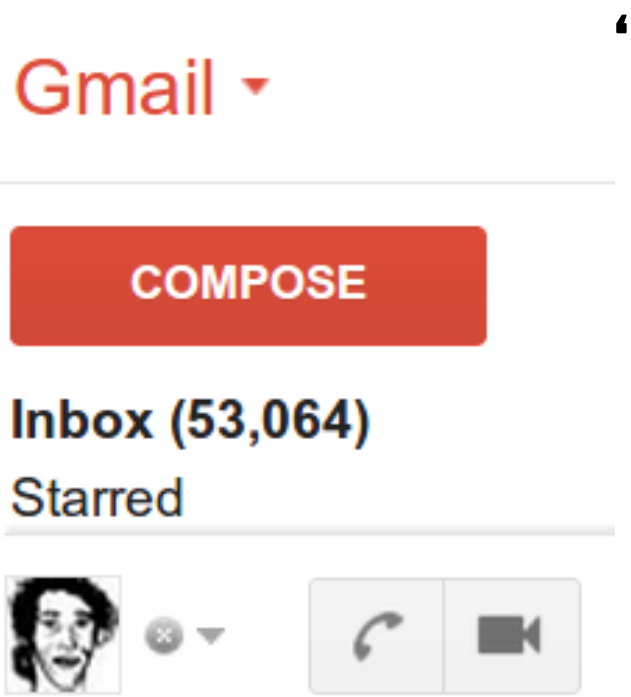
Estimated cost of development: **$ 6 millions**
COCOMO model,
`http://www.ohloh.net/p/scikit-learn`



Code maintenance too expensive to be alone

scikit-learn $\sim$ 300 email/month nipy $\sim$ 45 email/month

joblib $\sim$ 45 email/month mayavi $\sim$ 30 email/month



"Hey Gael, I take it you're too busy. That's okay, I spent a day trying to install XXX and I think I'll succeed myself. Next time though please don't ignore my emails, I really don't like it. You can say, 'sorry, I have no time to help you.' Just don't ignore."

Code maintenance too expensive to be alone

scikit-learn $\sim$ 300 email/month nipy $\sim$ 45 email/month

joblib $\sim$ 45 email/month mayavi $\sim$ 30 email/month

Your "benefits" come from a fraction of the code

- Data loading? Maybe?
- Standard algorithms? Nah

Share the common code...

...to avoid dying under code

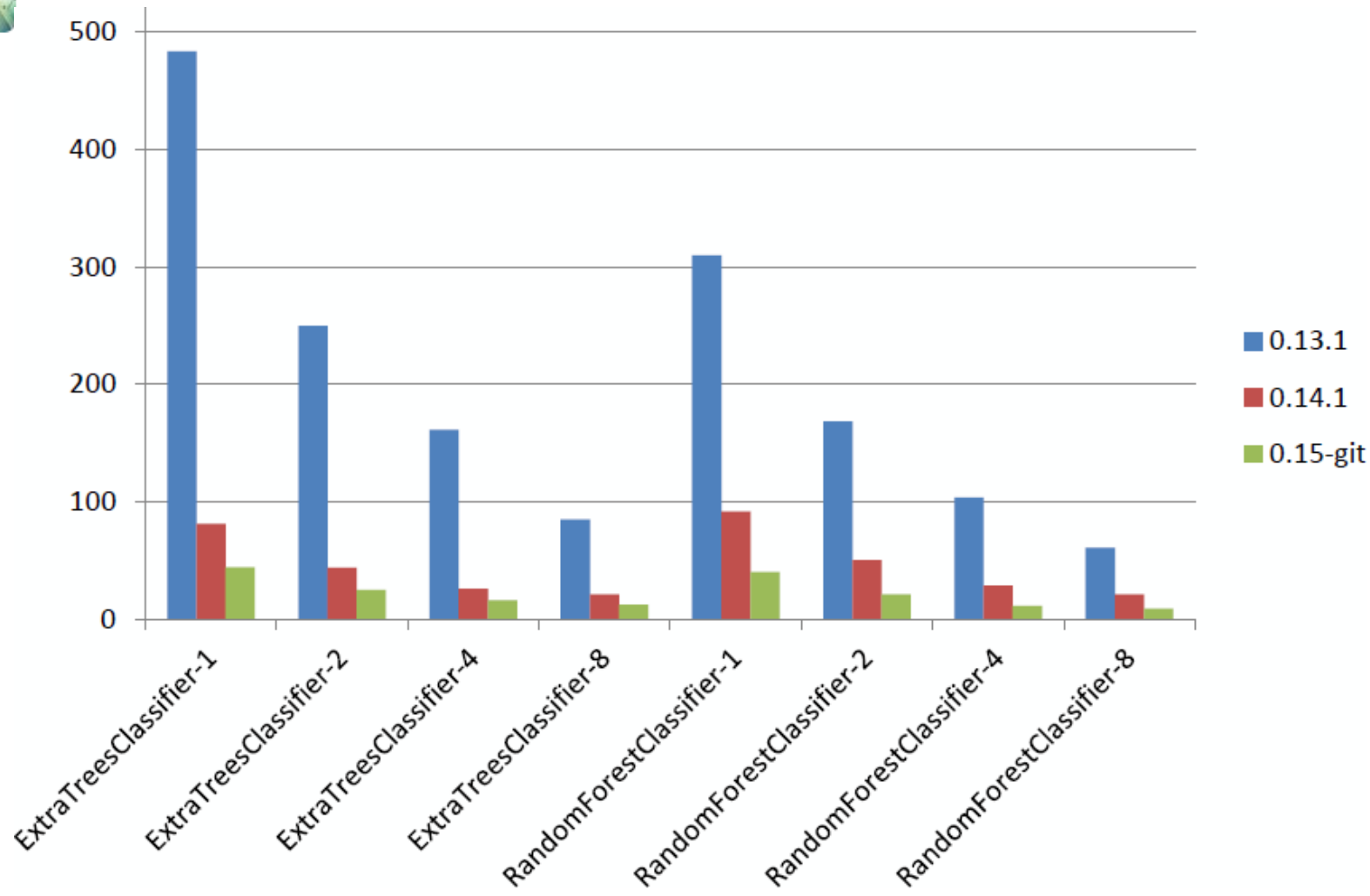
Code becomes less precious with time

And somebody might contribute features

4 Many eyes makes code fast

Gilles Louppe @glouppe · Feb 18

Speed improvement from 0.13 to 0.15-git of Random Forests in Scikit-Learn:



L. Buitinck, O. Grisel, A. Joly, G. Louppe, J. Nothman, P. Prettenhofer

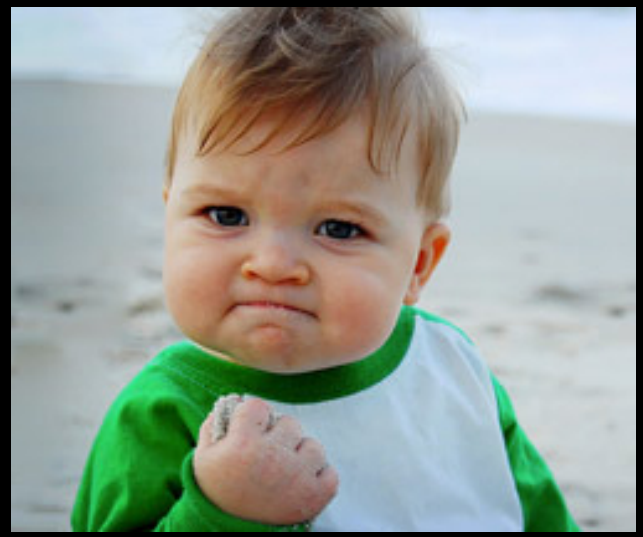
4 Communities increase the knowledge pool

Even if you don't do software,
you should worry about communities

- More experts in the package
  $\Rightarrow$ Easier recruitement
- The future is open
  $\Rightarrow$ Enhancements are possible

Meetups, conferences,
are where new ideas are born

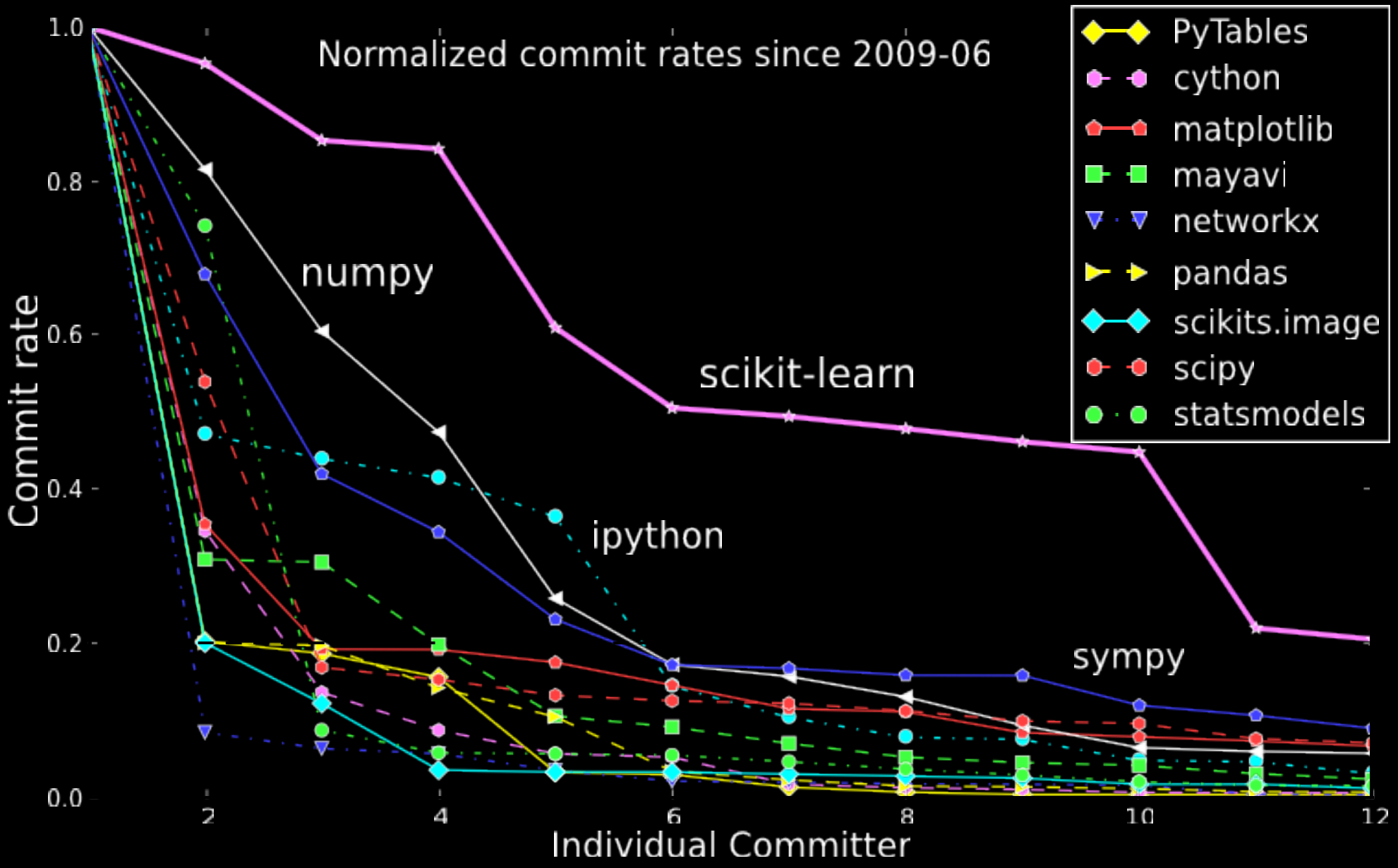
4 6 steps to a community-driven project



1. Focus on **quality**
2. Build great **docs and examples**
3. Use **github**
4. Limit the technicality of your codebase
5. Releasing and packaging matter
6. Focus on your contributors, give them credit, decision power

http://www.slideshare.net/GaelVaroquaux/scikit-learn-dveloppement-communautaire

4 Core project contributors



Credit: Fernando Perez, Gist 5843625

4 The tragedy of the commons

Individuals, acting independently and rationally according to each one's self-interest, behave contrary to the whole group's long-term best interests by depleting some common resource.

Wikipedia

Make it work, make it right, make it boring

Core projects (boring) taken for granted

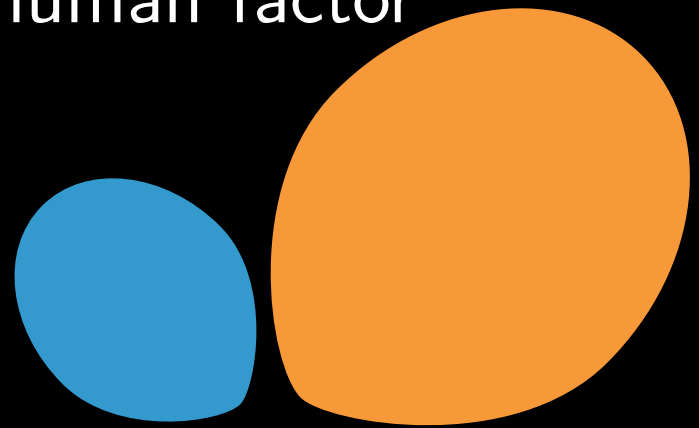
$\Rightarrow$ Hard to fund, less excitement

They need citation, in papers & on corporate web pages

Simple big data, in Python

Beyond the lie

- Machine learning gives value to (big) data
- Python + scikit-learn =
  - from interactive data processing (IPython notebook)
  - to crazy big problems (Python + spark)
- Big data will require you to understand the data-flow patterns (access, parallelism, statistics)
- Big ~~data~~ community addresses the human factor



@GaelVaroquaux