Writing code for science and data

Gaël
Varoquaux

Inria

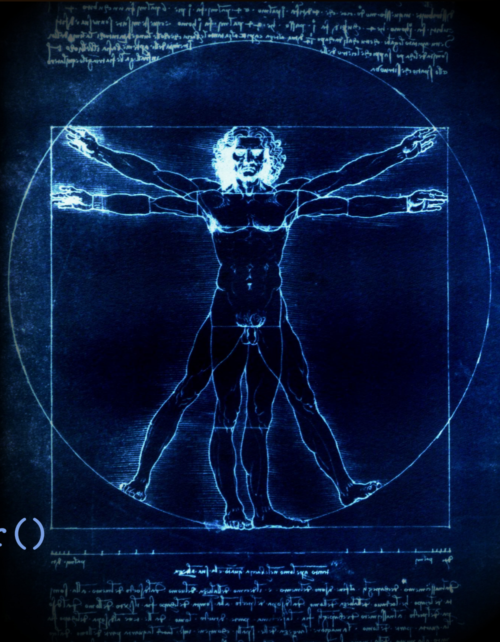
```
import science
science.discover()
```

Writing code for science and data

Gaël
Varoquaux

```
import science
science.discover()

import data_science
data_science.discover()
```




I am a “scientist”
quantum physics PhD
Active member of the scipy ecosystem since early 2000s
before scipy was cool
before pydata existed

I am now interested in cognitive neuroscience

linking psychology and neuroscience (neural implementations)

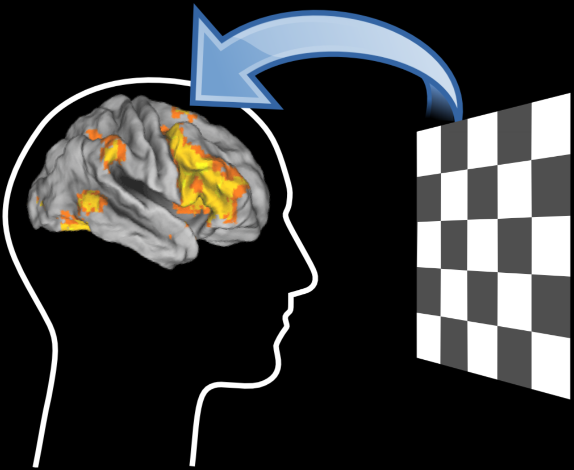
Connect neural activity to thoughts and cognition

Machine learning for cognitive neuroimaging

Brain imaging

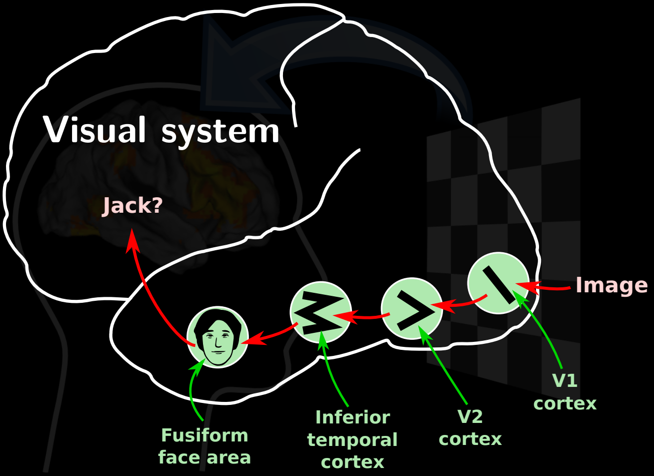
Learn a bilateral link between brain activity and cognitive function

Machine learning for cognitive neuroimaging



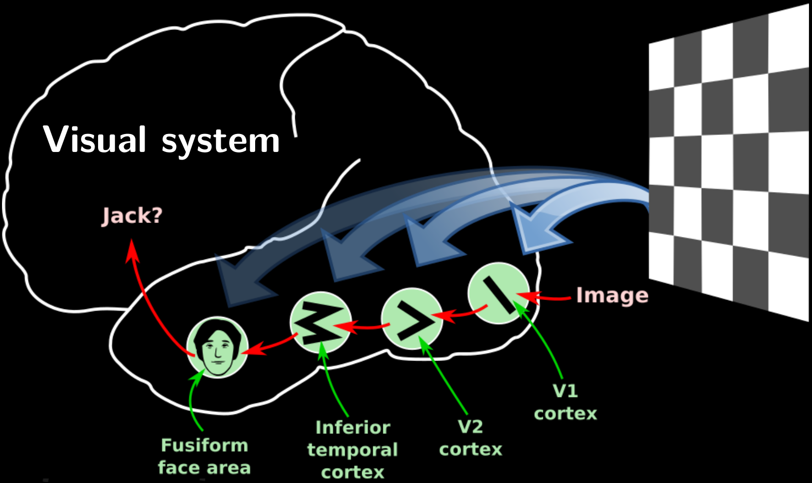
Predicting neural response from stimuli

Machine learning for cognitive neuroimaging



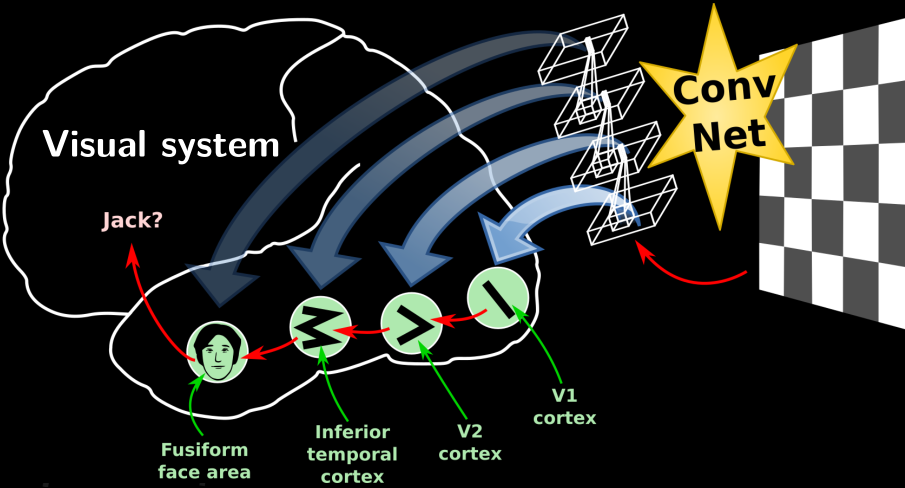
Predicting neural response from stimuli

Machine learning for cognitive neuroimaging



Predicting neural response from stimuli

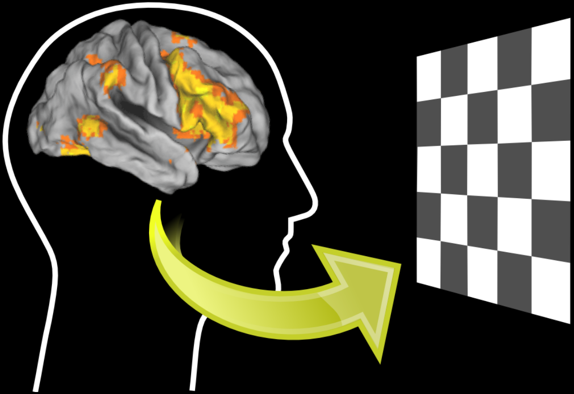
Machine learning for cognitive neuroimaging



Predicting neural response from stimuli

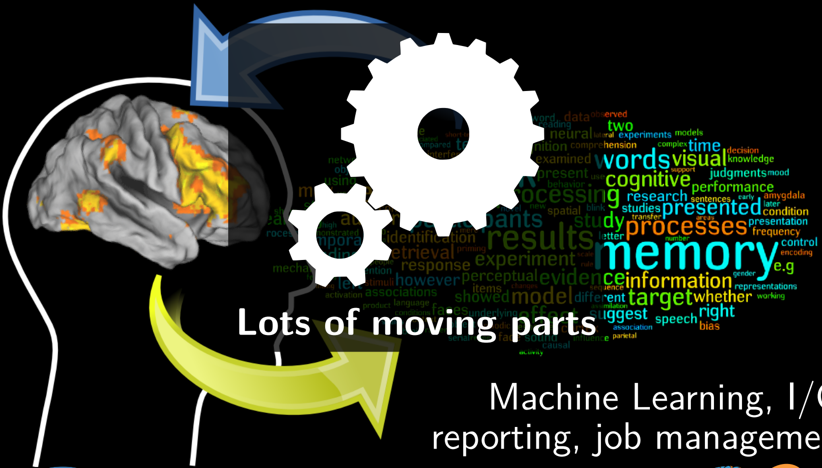
Convolutional networks map well to human visual system

Machine learning for cognitive neuroimaging



"Brain reading": **decoding**

Machine learning for cognitive neuroimaging



Lots of moving parts

Machine Learning, I/O, reporting, job management

scikit-learn

nilearn: neuroimaging

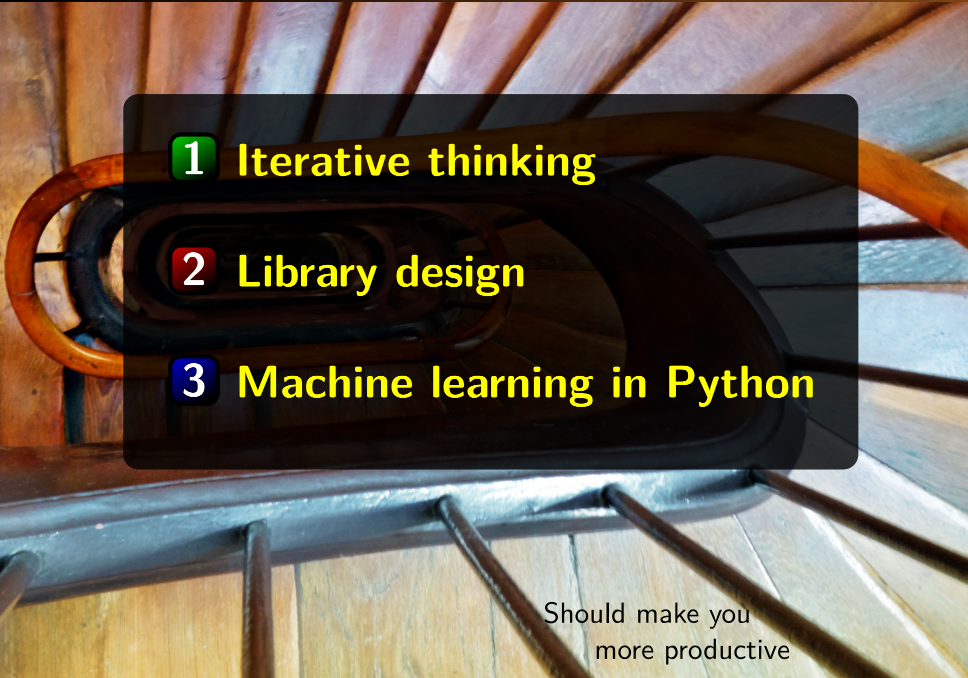
Machine learning for cognitive neuroimaging

Software: make it work, make it right, make it boring

scikit-learn

nilearn:
neuroimaging

Writing code for science and data



1. **Iterative thinking**
2. **Library design**
3. **Machine learning in Python**

Should make you
more productive

1 Iterative thinking

1 **Our workflow**: (data) science with computers

Work based on intuition
and experimentation

Conjecture

Experiment

⇒ Interactive & framework-less

Yet **needs consolidation keeping flexibility**

1 Reproducibility challenge in this iterative workflow

Reproducibility

New analysis

coming to the same conclusion

Enables verification / falsification

Also relevant for data science:
Operational recommendations can be questioned

Akin to challenges in sys-admin:
Try rebuilding a server after disk loss

1 Reproducibility challenge in this iterative workflow

Reproducibility

New analysis

coming to the same conclusion

Enables verification / falsification

Impediments

- Missing steps / files
- Libraries have changed
- Non portable code
- Statistical / numerical instabilities
- No one knows where the info is

1 Reproducibility challenge in this iterative workflow

Reproducibility

New analysis

coming to the same conclusion

Enables verification / falsification

Impediments

- Missing steps / files
- Libraries have changed
- Non portable code
- Statistical / numerical instabilities
- No one knows where the info is

- Technical
- Human

Code quality matters **Manual steps are evil**

1 Reproducibility challenge in this iterative workflow

Reproducibility

New analysis

coming to the same conclusion

Enables verification / falsification

Frozen food

Reusability

Applying the approach to a new problem

Being able to understand, modify,
run in new settings

Let us enable reusability

1 A design pattern in computational experiments

MVC pattern from Wikipedia:

Model	View	Controller
Manages the data and rules of the application	Output representation Possibly several views	Accepts input and converts it to commands for model and view

Photo-editing software

Filters	Canvas	Tool palettes

Typical web application

Database	Web-pages	URLs

1 A design pattern in computational experiments

MVC pattern from Wikipedia:

Model
Manages the data and rules of the application

View
Output representation
Possibly several views

Controller
Accepts input and converts it to commands
for model and view

For science and data:

Numerical, data-processing, & experimental logic

Results, as files.
Data & plots

Imperative **API**
Avoid input as files: not expressive

Module
with functions

Post-processing script
CSV & data files

Script
⇒ `for loops`

1 A design pattern in computational experiments

A recipe

- 3 types of files:
  - modules
  - command scripts
  - post-processing scripts
- CSVs & intermediate data files

  Separate computation from analysis / plotting
- Code and text (and data) $\Rightarrow$ **version control**

Numerical, data-processing, & experimental logic

Results, as files. Data & plots

Imperative **API**
Avoid input as files: not expressive

Module
with functions

Post-processing script
CSV & data files

Script
$\Rightarrow$ `for loops`

1 A design pattern in computational experiments

A recipe

- 3 types of files:
  - modules
  - command scripts
  - post-processing scripts
- CSVs & intermediate data files

  Separate computation from analysis / plotting
- Code and text (and data) ⇒ **version control**

Goals:
- Decouple steps
- Reuse code
- Mitigate compute time

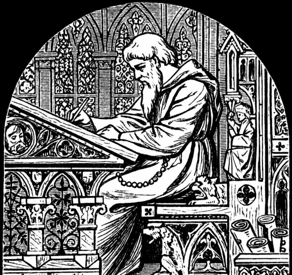
Module
with functions

Post-processing script
CSV & data files

Script
⇒ `for loops`

progressive consolidation

- Start with a script playing to understand the problem



- Start with a script playing to understand the problem
- Identify blocks/operations $\Rightarrow$ move to a function

Use functions

Obstacle: local *scope*

requires identifying input and output variables

That's a good thing

Interactive debugging / understanding

inside a function: `%debug` in IPython

Functions are the basic reusable abstraction

1 How I work — progressive consolidation

- Start with a script — playing to understand the problem
- Identify blocks/operations $\Rightarrow$ move to a function
- As they stabilize, move to a module

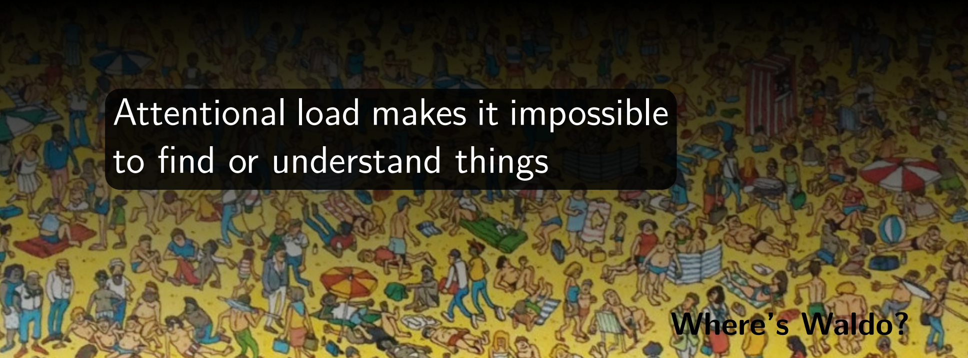
Modules

- enable sharing between experiments
  $\Rightarrow$ avoid 1000 lines scripts + commented code
- enable testing
  Fast experiments as tests
  $\Rightarrow$ gives confidence, hence refactorings

1 How I work — progressive consolidation

- Start with a script — playing to understand the problem
- Identify blocks/operations $\Rightarrow$ move to a function
- As they stabilize, move to a module
- Clean: delete code & files — you have version control

Attentional load makes it impossible to find or understand things



Where's Waldo?

progressive consolidation

- Start with a script — playing to understand the problem
- Identify blocks/operations $\Rightarrow$ move to a function
- As they stabilize, move to a module
- Clean: delete code & files — you have version control

Why is it hard?

Long compute times
make us unadventurous

Know your tools

- Refactoring editor
- Version control

1 joblib.Memory

The memoize pattern

```
mem = joblib.Memory(cachedir='.')
g = mem.cache(f)
b = g(a)    # computes a using f
c = g(a)    # retrieves results from store
```

For scientific and data computing

- a & b can be big
- a & b arbitrary objects — no change in workflow
- Results stored on disk
- Cache flushed when `f` changes — safe caching

1 joblib.Memory

The memoize pattern

```
mem = joblib.Memory(cachedir='.')
g = mem.cache(f)
b = g(a)    # computes a using f
c = g(a)    # retrieves results from store
```

For scientific and data computing

Fits in experimentation loop
Helps decrease re-run times

Black-boxy, persistence only implicit
Discourages function refactoring (avoid recomputing)

tip: cache functions inside functions

Using software-engineering best practices

1 The ladder of code quality

- Use pyflakes in your editor — seriously
- Coding convention, good naming
- Version control — Use git + github
- Code review
- Unit testing

  If it's not tested, it's broken or soon will be
- Make a package

  controlled dependencies and compilation

...

1 The ladder of code quality

Increasing cost

- Use pyflakes in your editor — seriously
- Coding convention, good naming
- Version control — Use git + github
- Code review
- Unit testing
  If it's not tested, it's broken or soon will be
- Make a package
  controlled dependencies and compilation
- ...

Avoid premature software engineering

1 The ladder of code quality

- Use pyflakes in your editor
- C
- V
- C
- Unit testing
- Make a package

controlled dependencies and compilation

...

Increasing cost

Over versus under engineering

Our goal is generating insights

- Experimentation to develop intuitions
  $\Rightarrow$ new ideas
- As the path becomes clear: consolidation

Heavy engineering too early freezes bad ideas

Avoid premature software engineering

1 Libraries

Increasing cost

- Use pyflakes in your editor
- Coding convention, good naming
- Version control
- Code review
- Unit testing
- Make a package

controlled dependencies and compilation

...

A library

2 Library design

If doing research is like crossing oceans
doing software is like building briges

2 Principles of API design for SciPy / PyData stack

- Be a library

- Functions trump classes

- Shallow objects, understandable by their “surface”:
  - interface (set of methods)
  - attributes

  No too many

- Universal data objects for inputs & output:
  dicts, numpy arrays, pandas dataframe

- Few kinds of “action” objects,
  defined by their function

Building on solid foundations

Plug components together for an application

3D plotting + statistics $\rightsquigarrow$ Neuroimaging

How do we ensure correctness?

Testing If it ain't tested, it's broken

Building on solid foundations

Plug components together for an application

3D plotting + statistics $\rightsquigarrow$ Neuroimaging

How do we ensure correctness?

Testing If it ain't tested, it's broken

establishes correctness enables refactoring

2 **Testing**: what we've learned in scikit-learn

- Testing basic mathematical properties
  eg a minimizer decreases cost function
  or symmetries, or special cases

  Tests should run very fast

2 **Testing**: what we've learned in scikit-learn

- Testing basic mathematical properties

- Make everything perfectly reproducible.

  Never use the global generator `np.random` in tests

  it creates side effects

  Generators as optional inputs to functions:

```
def f(x, random_state=None):
  if random_state is None:
    random_state = np.random.RandomState()
   noise = random_state.randn()
```

2 **Testing**: what we've learned in scikit-learn

- Testing basic mathematical properties

- Make everything perfectly reproducible.

- Test interface specification: "auto" tests
  - Reproducibility on simple data
  - Multiple data types
  - Proper errors on bad input
  - Objects respect interface

2 **Testing**: what we've learned in scikit-learn

- Testing basic mathematical properties
- Make everything perfectly reproducible.
- Test interface specification: "auto" tests
- Add a test each time there is a bug

3 Machine learning in Python

scikit-learn

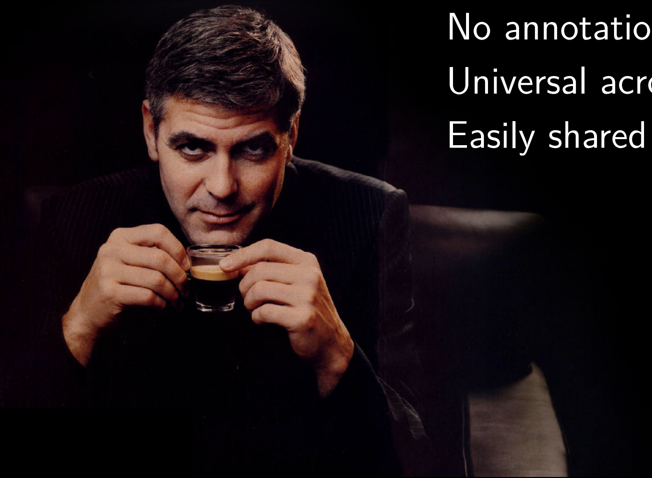
3 My stack for data science

Python, what else?

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

3 My stack for data science

The scientific Python stack



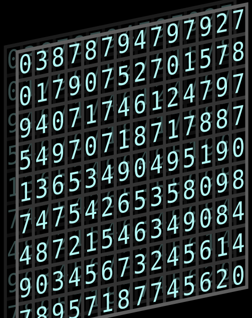
numpy arrays

Mostly a `float**`

No annotation / structure ☹

Universal across applications ☺

Easily shared across languages



3 My stack for data science

The scientific Python stack

numpy arrays

Connecting to

- pandas

  Columnar data

- scikit-image

  Images

- scipy

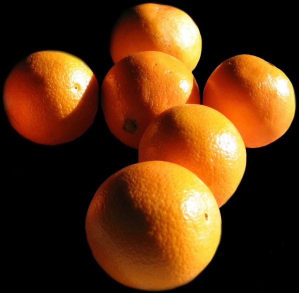
  Numerics, signal processing

...

3 Machine learning in a nutshell

Machine learning is about making predictions from data

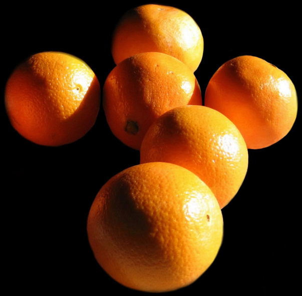
e.g. learning to distinguish apples from oranges



3 Machine learning in a nutshell

Machine learning is about making predictions from data

e.g. learning to distinguish apples from oranges



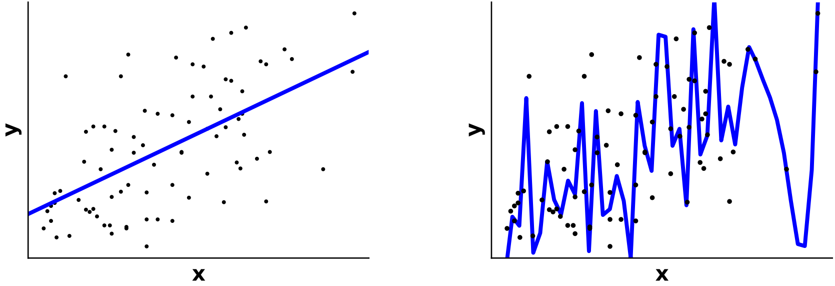
Prediction is very difficult, especially about the future. *Niels Bohr*

Learn as much as possible from the data

but not too much

3 Machine learning in a nutshell

Machine learning is about making predictions from data



Which model do you prefer?

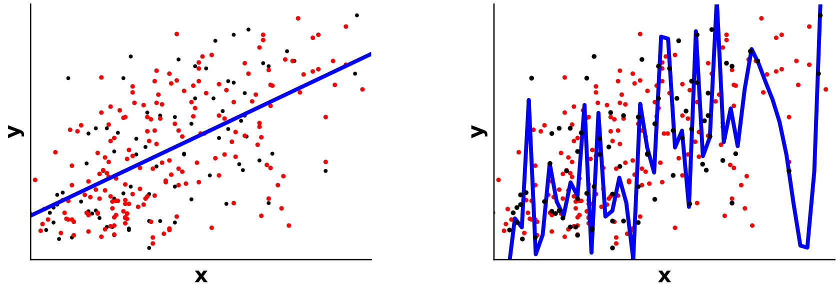
Prediction is very difficult, especially about the future. *Niels Bohr*

Learn as much as possible from the data

but not too much

3 Machine learning in a nutshell

Machine learning is about making predictions from data



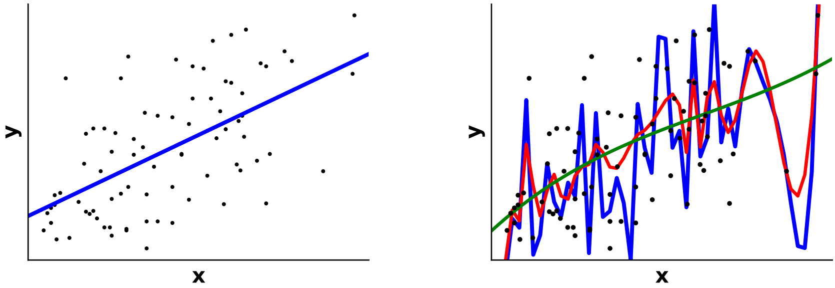
Minimizing train error $\neq$ generalization : *overfit*

Prediction is very difficult, especially about the future. *Niels Bohr*

Learn as much as possible from the data
but not too much

3 Machine learning in a nutshell

Machine learning is about making predictions from data



Adapting model complexity to data – *regularization*

Prediction is very difficult, especially about the future. *Niels Bohr*

Learn as much as possible from the data

but not too much

3 Machine learning without learning the machinery

scikit learn

machine learning in Python

A library, not a program

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

let's disrupt something new

scikit learn

machine learning in Python

3 Machine learning without learning the machinery

A library, not a program

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

let's disrupt something new

As easy as py

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

machine learning in Python

3 Show me your data: the samples × features matrix

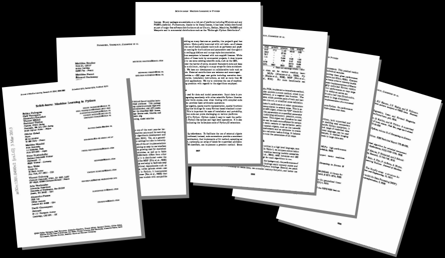
Data input: a 2D numerical array

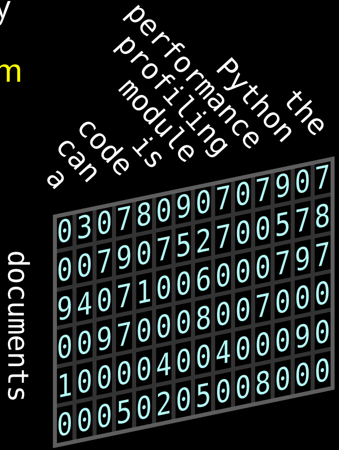
Requires transforming your problem

Data input: a 2D numerical array

Requires transforming your problem

With text documents:





`sklearn.feature_extraction.text.TfIdfVectorizer`

“Big” data

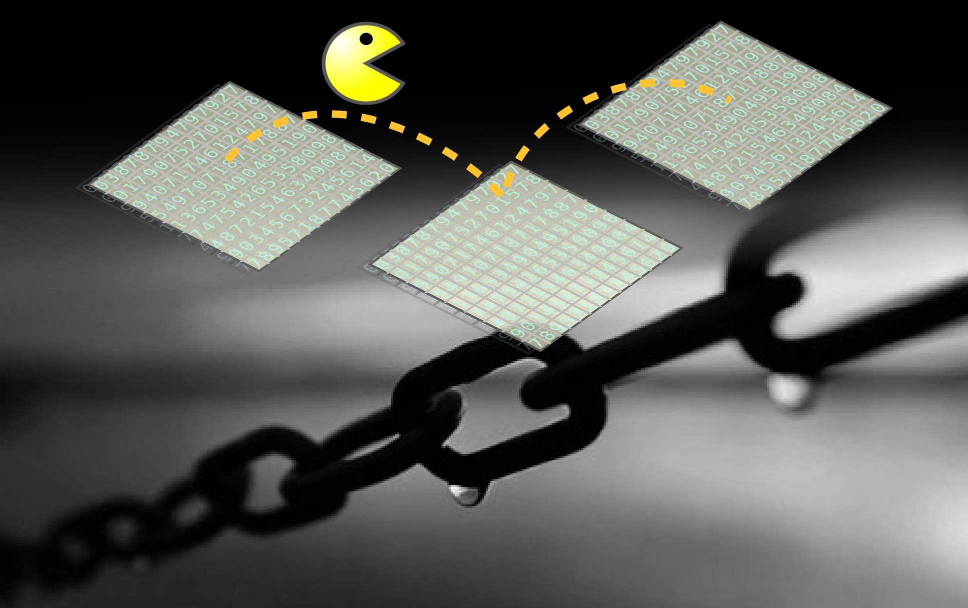
Engineering efficient processing pipelines

Many samples or **Many features**

See also: http://www.slideshare.net/GaelVaroquaux/processing-biggish-data-on-commodity-hardware-simple-python-patterns

3 Many samples: on-line algorithms

```
estimator.partial_fit(X_train, Y_train)
```



3 Many samples: on-line algorithms

```
estimator.partial_fit(X_train, Y_train)
```

Supervised models: predicting

```
sklearn.naive_bayes...
sklearn.linear_model.SGDRegressor
sklearn.linear_model.SGDClassifier
```

Clustering: grouping samples

```
sklearn.cluster.MiniBatchKMeans
sklearn.cluster.Birch
```

Linear decompositions: finding new representations

```
sklearn.decompositions.IncrementalPCA
sklearn.decompositions.MiniBatchDictionaryLearning
sklearn.decompositions.LatentDirichletAllocation
```

3 Many features: on-the-fly data reduction

⇒ Reduce the data as it is loaded

```
X_small =
    estimator.transform(X_big, y)
```

3 Many features: on-the-fly data reduction

Random projections (will average features)

`sklearn.random_projection`

random linear combinations of the features

Fast clustering of features

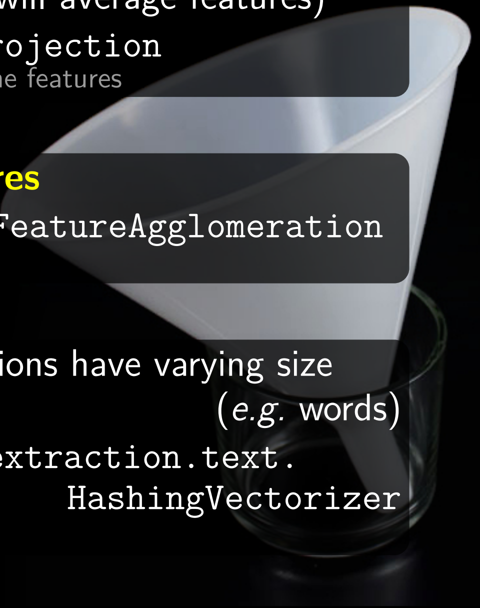
`sklearn.cluster.FeatureAgglomeration`

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



More gems in scikit-learn

SAG:

```
linear_model.LogisticRegression(solver=’sag’)
```

Fast linear model on biggish data

More gems in scikit-learn

SAG:

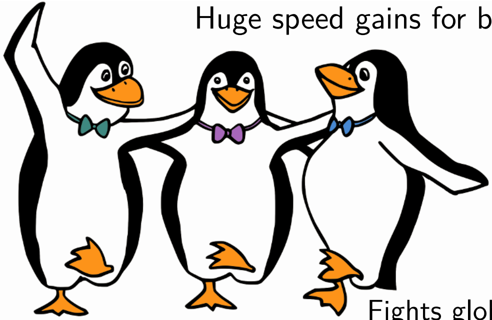
`linear_model.LogisticRegression(solver='sag')`

Fast linear model on biggish data

PCA == RandomizedPCA: (0.18)

Heuristic to switch PCA to random linear algebra

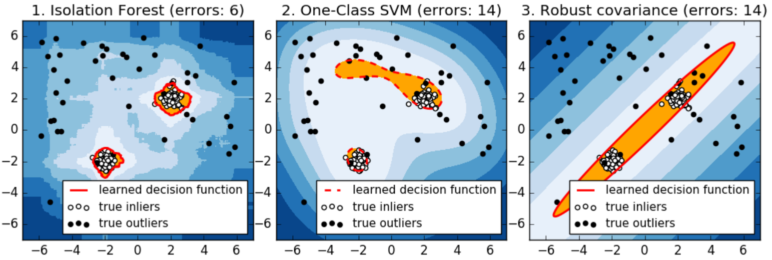
Huge speed gains for biggish data



Fights global warming

More gems in scikit-learn

Outlier detection and isolation forests (0.18)



Time to wrap up

Time to wrap up

Code, code, code

Scipy-lectures: learning numerical Python

Many problems are better solved by documentation than new code

Scipy-lectures: learning numerical Python

- Comprehensive document: numpy, scipy, ...
  1. Getting started with Python for science
  2. Advanced topics
  3. Packages and applications

Scipy Lecture Notes

One document to learn numerics, science, and data with Python

Tutorials on the scientific Python ecosystem: a quick introduction to central tools and techniques. The different chapters each correspond to a 1 to 2 hours course with increasing level of expertise, from beginner to expert.

Download

- PDF, 2 pages per side
- PDF, 1 page per side
- HTML and example files
- Source code (github)

About the scipy lecture notes

Authors What's new License Contributing

http://scipy-lectures.org

1. Getting started with Python for science

▸ 1.1. Scientific computing with tools and workflow

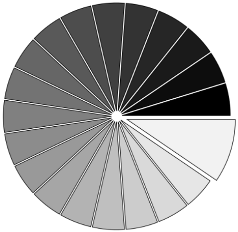
Scipy-lectures: learning numerical Python

- Code examples

sphinx-gallery

Pie chart

A simple pie chart example with matplotlib.



Python source code: `plot_pie_ex.py`

```
import numpy as np
import matplotlib.pyplot as plt

n = 20
Z = np.ones(n)
Z[-1] *= 2

plt.axes([0.025, 0.025, 0.95, 0.95])

plt.pie(Z, explode=Z*.05, colors = ['%f' % (i/float(n)) for i in range(n)])
plt.axis('equal')
plt.xticks(())
plt.yticks()
```

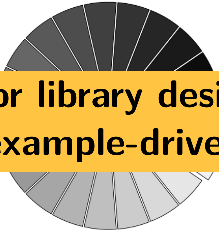
Scipy-lectures: learning numerical Python

- Code examples

sphinx-gallery

Pie chart

A simple pie chart example with matplotlib.



Useful for library design too: example-driven design

Python source code: `plot_pie_ex.py`

```
import numpy as np
import matplotlib.pyplot as plt

n = 20
Z = np.ones(n)
Z[-1] *= 2

plt.axes([0.025, 0.025, 0.95, 0.95])

plt.pie(Z, explode=Z*.05, colors = ['%f' % (i/float(n)) for i in range(n)])
plt.axis('equal')
plt.xticks(())
plt.yticks()
```

Writing code for science and data

1. Go fast: Experimentation & progressive consolidation

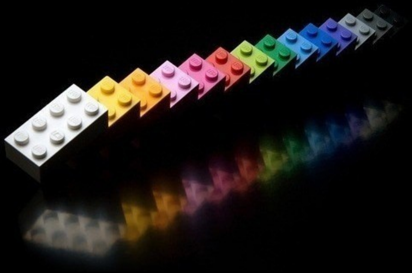
Agility is key for experimentation
Don't adopt engineering practices too early
Do adopt them in time

@GaelVaroquaux

Writing code for science and data

1. Go fast: Experimentation & progressive consolidation
2. Go far: Quality software is the cement of science

Components made for reuse
Quality & testing



@GaelVaroquaux

Writing code for science and data

1. Go fast: Experimentation & progressive consolidation

2. Go far: Quality software is the cement of science

3. Facilitate: Make it easy to use

API, docs, & examples

scikit-learn

Machine learning without learning the machinery

@GaelVaroquaux