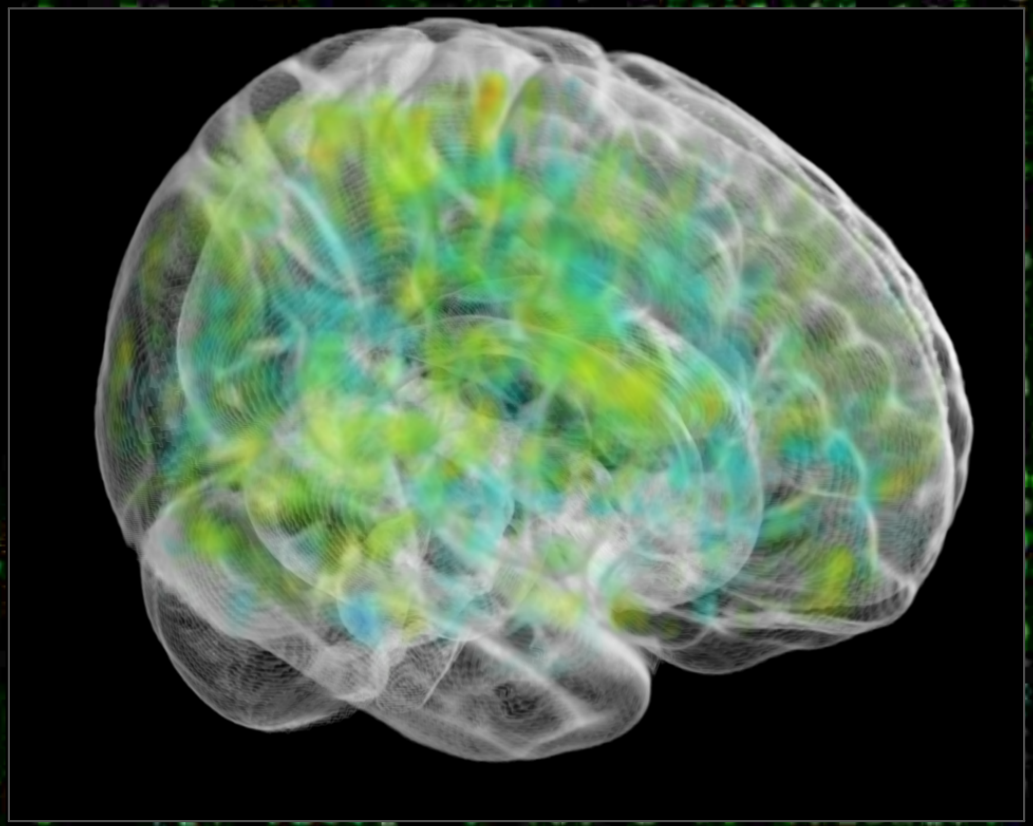
Aiguille et botte de foin :

`scikit-learn` et `joblib` pour explorer des données volumeuses

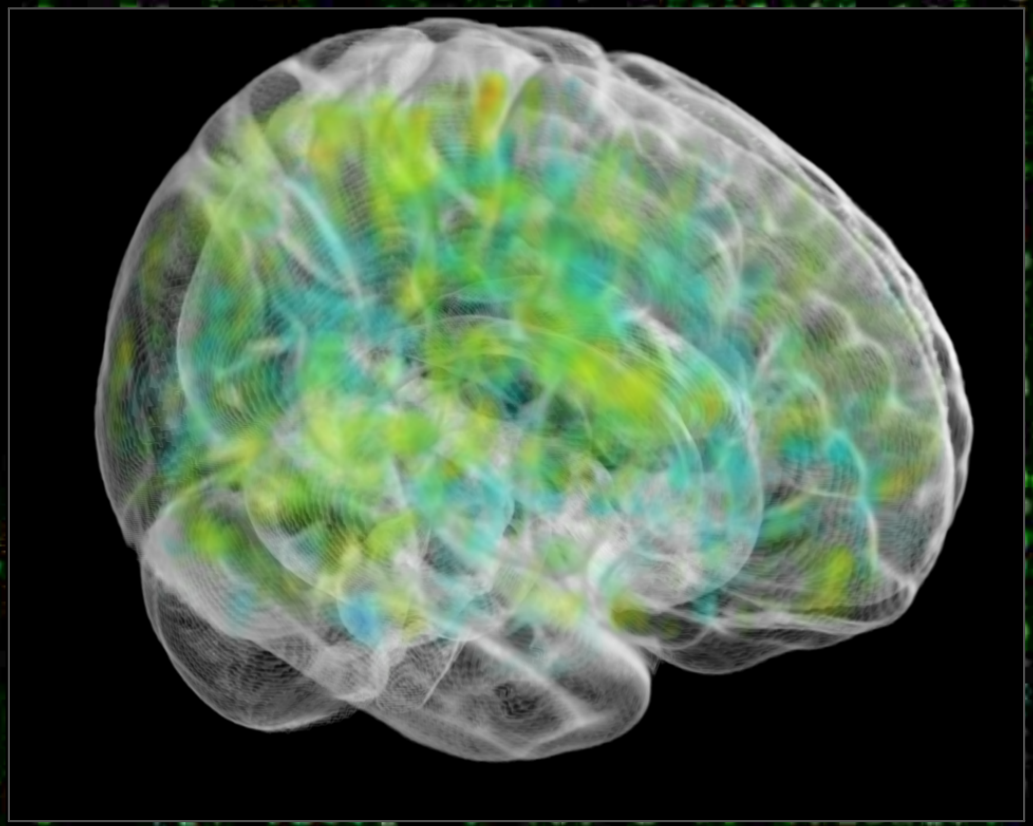
G. Varoquaux

Neurospin + INRIA

Explosion de la quantité de données



Explosion de la quantité de données



Traitement statistique de donnée (apprentissage)

Résumer l'information

Prédire de nouvelles informations

- Statistiquement optimal
- Algorithmiquement rapide

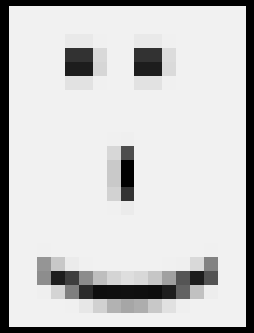
1. **Prédire** – `scikit-learn`

2. **Résumer** – `scikit-learn`

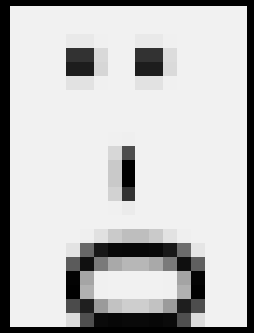
3. **Être fainéant (et efficace)** – `joblib`

1 Prédire — `scikit-learn`

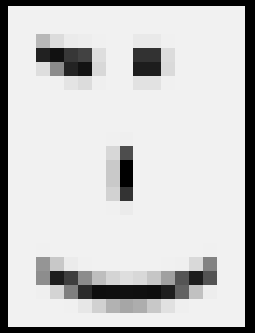
1 Reconnaissance de visage



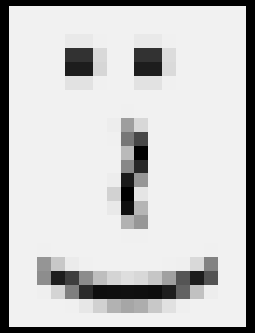
André



Bernard

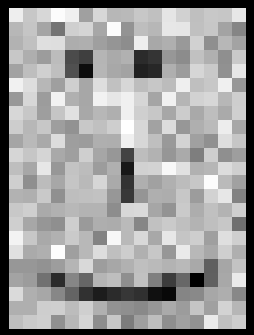


Charles

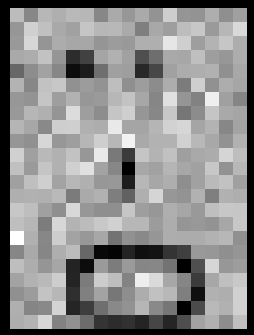


Didier

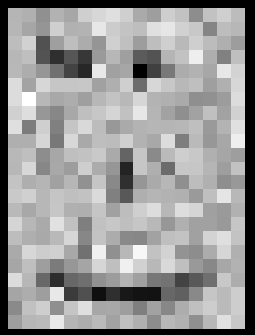
1 Reconnaissance de visage



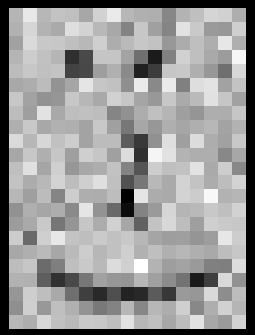
André



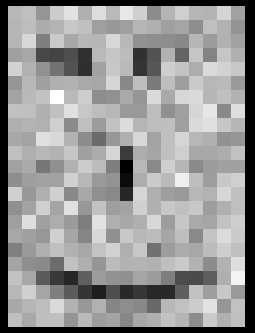
Bernard



Charles



Didier



?

1 Méthode naïve

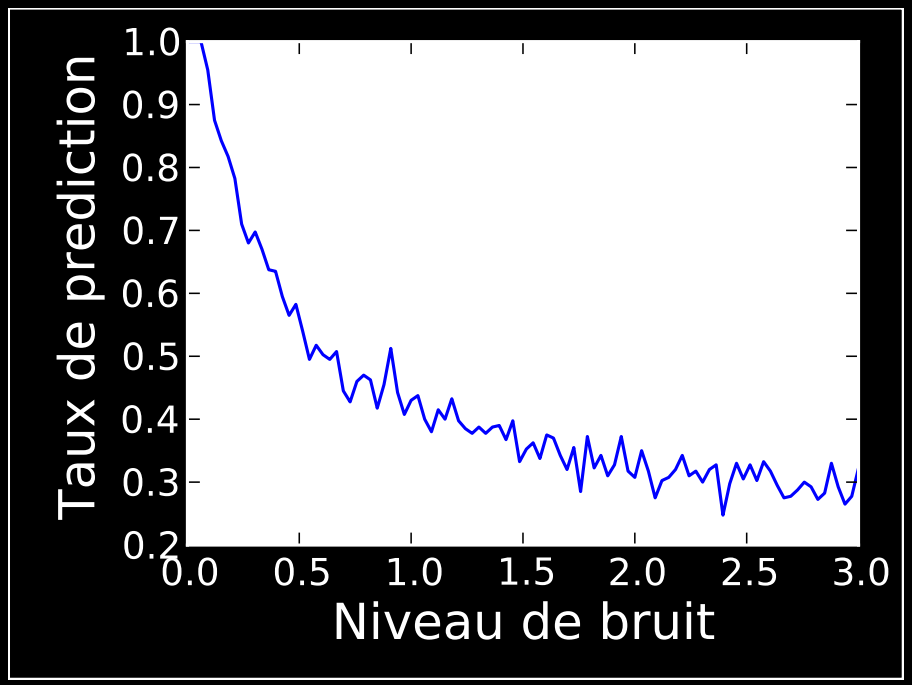
1. Estimer les visages *typiques* à partir d'images connues (bruitées).

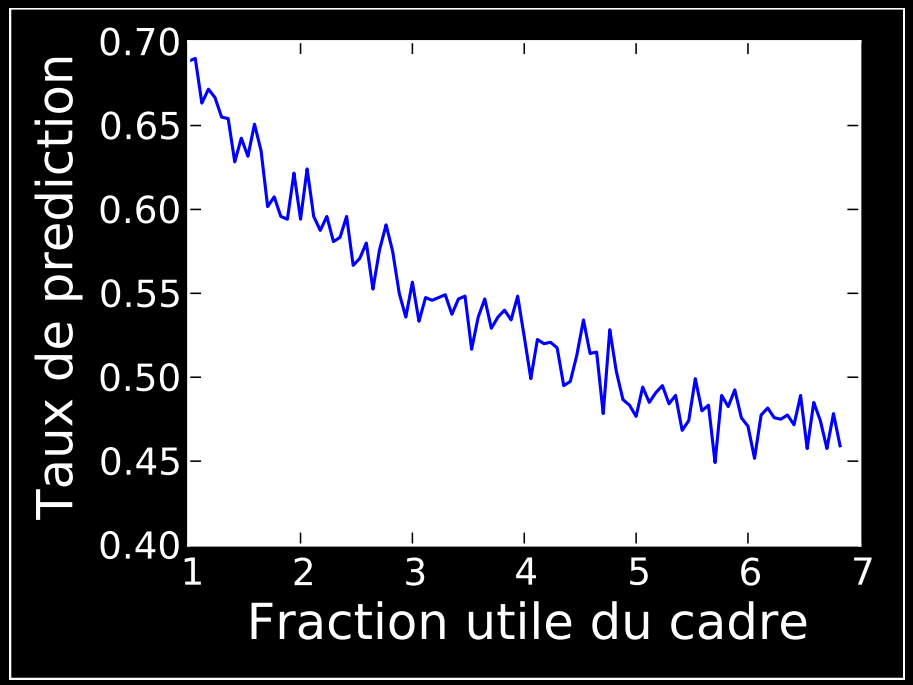
2. A partir d'une photo (bruitée aussi), trouver le visages *typique* qui lui ressemble le plus.

1 Une botte de foin

La fraction d'information intéressante est faible :

- Il y a du bruit
- L'image est grande





1 Les vrais données sont pourries

http://cswww.essex.ac.uk/mv/allfaces

100 individus, 10 photos par individu,

à peu près alignées

1 Validation croisée

1. Choisir une fraction des photos pour les visages connus (*jeu d'apprentissage*).
2. Apprendre les visages *typiques* et la fonction de prédiction dessus.
3. Tester la prédiction en essayant de nommer les photos non utilisées pour l'apprentissage (*jeu de test*) et en mesurant le taux d'erreur.

1 Du code : scikit-learn

- Objets avec fit/predict prenant des tableaux numpy

```
e = Estimator()
e.fit(known_faces, known_names)
guessed_names = e.predict(unkown_faces)
```

- Générateurs de validation croisée renvoyant des masques

```
from scikits.learn import cross_val
cv = cross_val.StratifiedKFold(names)
for train, test in cv:
    e.fit(faces[train], names[train])
    e.predict(faces[test])
```

Ingrédients de base d'un framework d'apprentissage

1 Au final : .5% d'erreur de prédiction

- Utiliser les k plus proches voisin (kNN) :

```
import numpy as np
from scikits.learn import cross_val
from scikits.learn.neighbors import Neighbors

cv = cross_val.StratifiedKFold(labels, k=5)
e = Neighbors(k=5)
errors = cross_val.cross_val_score(e, data,
                                   labels, cv=cv)
print 'score', np.sum(errors)/float(len(data))
```

- Choisir k pour optimiser le score.

  Attention 2 boucles imbriquées de validation croisée

  1 choisir k

  2 évaluer l'erreur sur des données inconnues

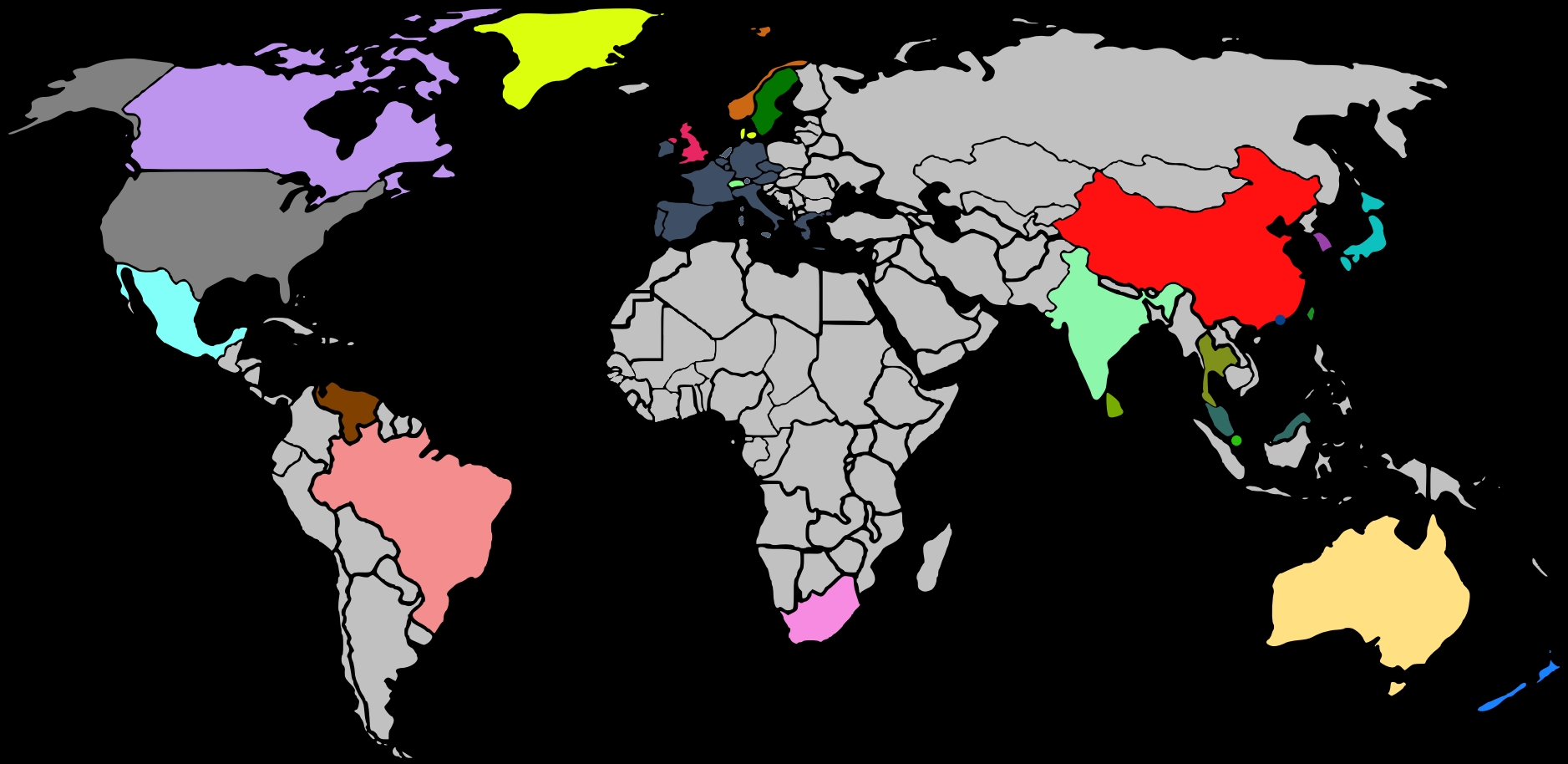
2 Résumer – `scikit-learn`

2 Réduire la taille des données

- Botte de foin : la taille des données nuit à la performance
- Les images contiennent beaucoup d'information redondante

Comment la réduire de façon optimale ?

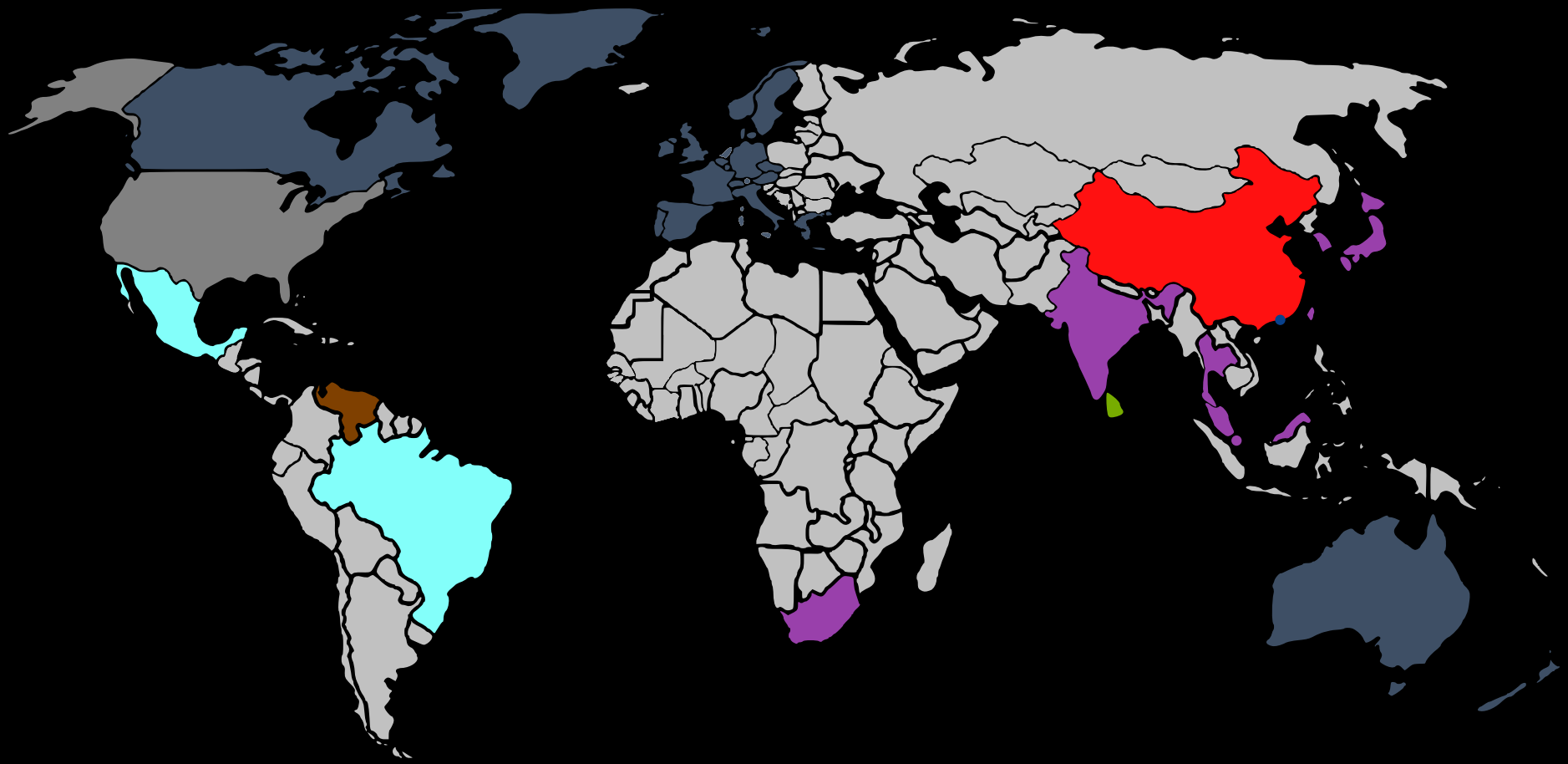
2 Former des groupes



Pays associés à une devise majeure

La corrélation des taux de changes fournit une mesure d'affinité

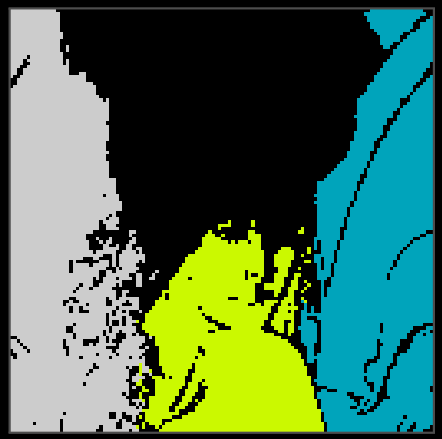
2 Former des groupes



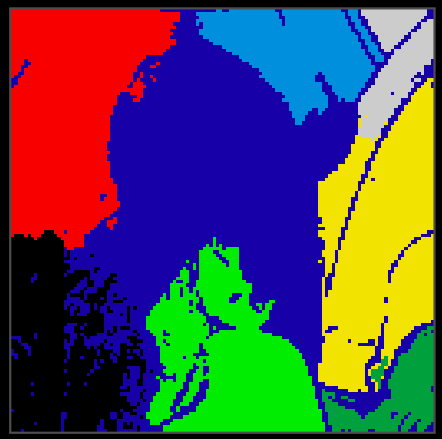
Qui se ressemble s'assemble

2 Former des groupes sur une image

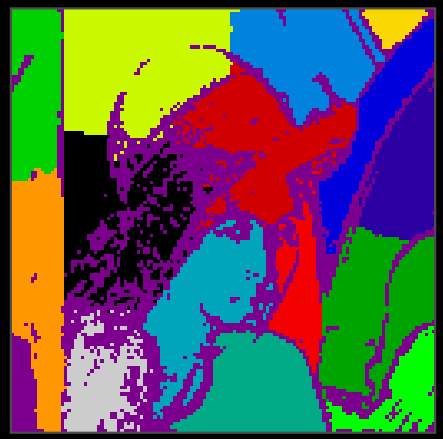
Lena



4



8

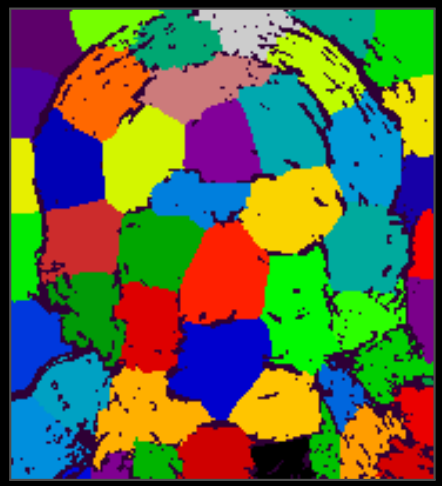


16

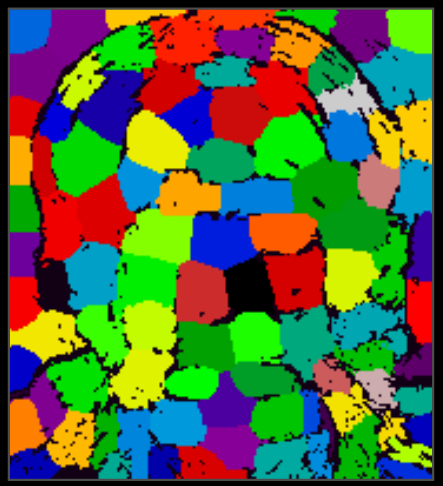
32

2 Former des groupes sur les 1000 visages

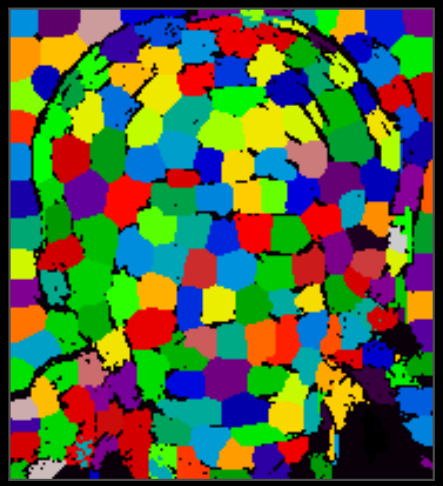
k=30



k=50



k=100



k=200

- Choisir k pour maximiser la prédiction (encore une boucle de validation croisée)

Apprentissage statistique en Python

Efficace

Bons algorithmes, `numpy` + `scipy`,
C + Cython + Python

Pythonesque

License BSD, objets simples,
pas de dépendences autres que `numpy` + `scipy`

Facile à utiliser

API uniforme, documentation,
paramêtres par défault

3 Être fainéant (et efficace) – `joblib`

3 Évaluation fainéante

On calcule toujours la même chose

Boucles de imbriquées avec transformations successives

Calculs en variant les paramêtres

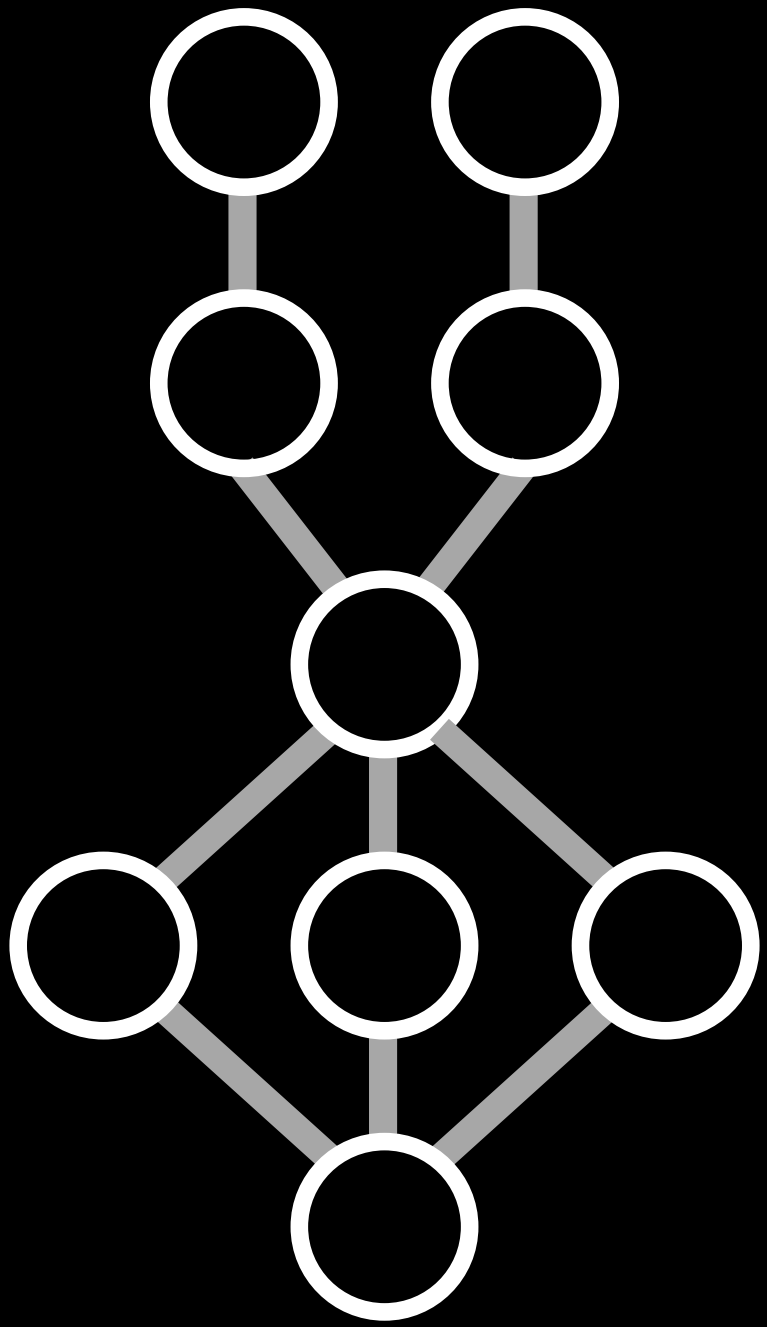
Difficultés

Connaître les dépendances entre les étapes

Suivre les paramêtres

Évaluation fainéante : ne pas recalculer ce qu'on a déjà calculé

3 pipeline/data flow programming



Philosophie

- Simple

  ne changez pas vos programmes
  n'apprenez pas de nouveau paradigmes

- Minimal

  pas de dépendances, embarquable
  peu de fonctionnalités

- Performant

  grosses données
  calcul parallèle

- Robuste

  tolérant aux erreurs
  debuggable

Évaluation fainéante

```
>>> from joblib import Memory
>>> mem = Memory(cachedir='/tmp/joblib')
>>> import numpy as np
>>> a = np.vander(np.arange(3))
>>> square = mem.cache(np.square)
>>> b = square(a)
________________________________________
[Memory] Calling square...
square(array([[0, 0, 1],
        [1, 1, 1],
        [4, 2, 1]]))
_____________________________square - 0.0s
>>> c = square(a)
>>> # Pas de re-evaluation
```

Évaluation fainéante

- Hash MD5 des paramêtres d'entrées (efficace)
  $\Rightarrow$ pas de modèle de données et d'exécution
- Stockage sur disk dans des répertoires séparés (efficace – `memmap`)
- Table globale `sqlite` (verrous `:(`)
- Nettoyage de cache à la volée (beta)

Calcul parallèle

```
>>> from joblib import Parallel, delayed
>>> from math import sqrt
>>> Parallel(n_jobs=1)(delayed(sqrt)(i**2)
...                    for i in range(7))
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0]
```

- La syntaxe est importante
- Le débugage aussi
- $\Rightarrow$ Fork agressif

Le futur

- Meilleur branchement parallèle/memoire
- Meilleurs bases de données/datastore
- 2 niveaux de cache (mémoire/disk)
- Suivit d'exécution

3 Traitement de données performant...

Scikit Learn :

- Algorithmes état de l'art
- Projet jeune

joblib

- Mieux calculer
- Pas que scientifique

`http://scikits-learn.sf.net`
`http://packages.python.org/joblib`