AuthorDetector

Un framework pour le développement rapide de méthodes d'identification d'auteur

Par Stephen Larroque

Sous la direction de M. Jean-Gabriel Ganascia

UPMC SORBONNE UNIVERSITÉS

Plan

- Problématique
- Framework
- Modélisation du workflow
- Usage et déploiement
- Futurs plans

Problématique

de l’identification d’auteur

Problématique

- Identifier l’auteur d’un texte non signé ou sous pseudonyme
- Principale hypothèse: à partir de caractéristiques du texte, on peut distinguer les auteurs

Problématique - Formulation

- Similaire à PAN13
- Étant donné:
  - corpus de textes etiquetes (entre 1 et 10 textes/auteur + class imbalance)
  - texte sans etiquette
- Déterminer l'étiquette du texte, avec un score de confiance

Problématique - Contrainte

- Rudman (1998): 1000 approches différentes proposées pour cette tache.
- Comment concilier cette variété d'approches et reproducibilité des résultats dans un workflow généralisé?

Première approche

- Framework modulaire:
  - pour réutiliser au maximum
  - tester de nombreuses combinaisons d'algos
  - ne se soucier que de son module
- Modélisation/workflow: ensemble de catégories de modules généralisant les méthodes d'identification d'auteur.
- Nomenclature de Stamatatos (2009)

Framework

AuthorDetector

- Framework de développement rapide
- Modulaire (tout est module)
- Flexible (chaque module est indépendant)
- User-friendly (GUI, notebook)
- Reproductible (notebook, config, sauvegarde automatique des paramètres)
- Codé en Python et Pandas/Numpy

Paradigme

- Tout est module
- Module = fichier (comme sous Linux)
  `eg: authordetector/preprocessor/module.py`
- Catégorie = dossier
  `eg: authordetector/preprocessor/`
- Modules indépendants et ordre interchangeable (voir Workflow)

Paradigme - 2

- Module enfant appelle et définit les variables, pas le parent

```
function mafunc(X=None, Y=None, *args, **kwargs):
…
return {‘X’: Y, ‘Y’:X, ‘Z’:0}
```

- Classes de base sert d’interface (spécification de ce qui est requis)

- Accès à la config ubiquitaire:

```
self.config.get(‘mavariable’)
```

Modélisation du workflow

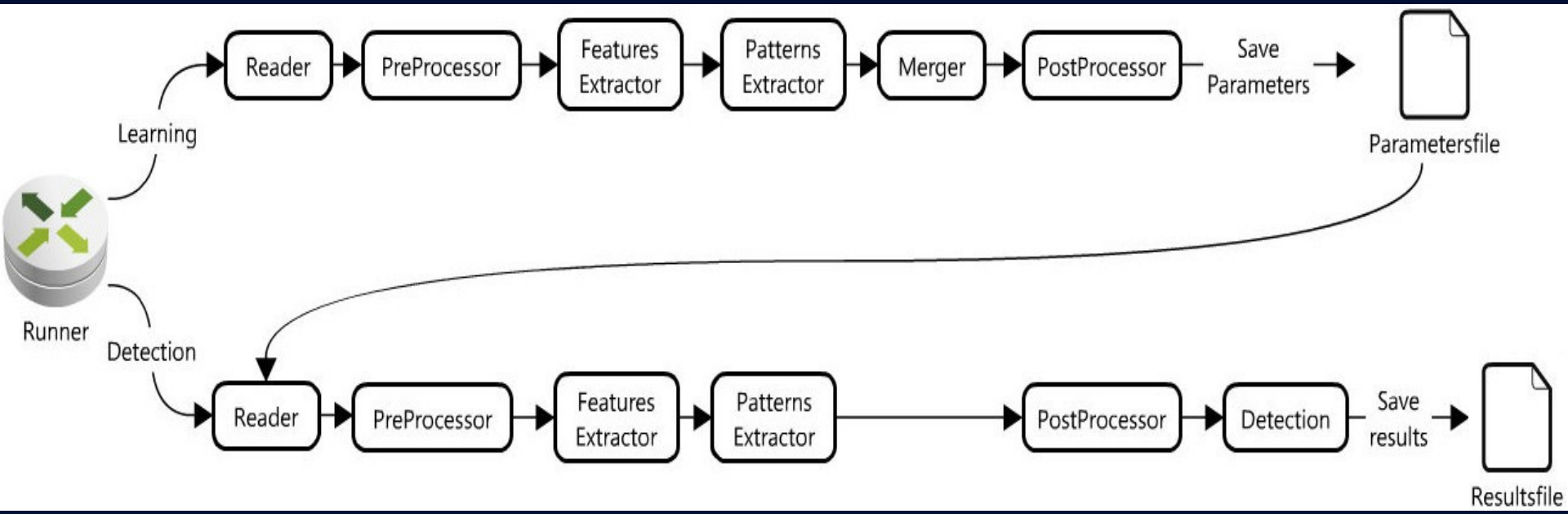
Modélisation

- Workflow = enchainement d'appels de modules
- Généralisation des méthodes d'identification d'auteur dans une procédure (workflow) standard mais flexible
- Totalement modifiable dans le fichier de configuration

Mécanisme global

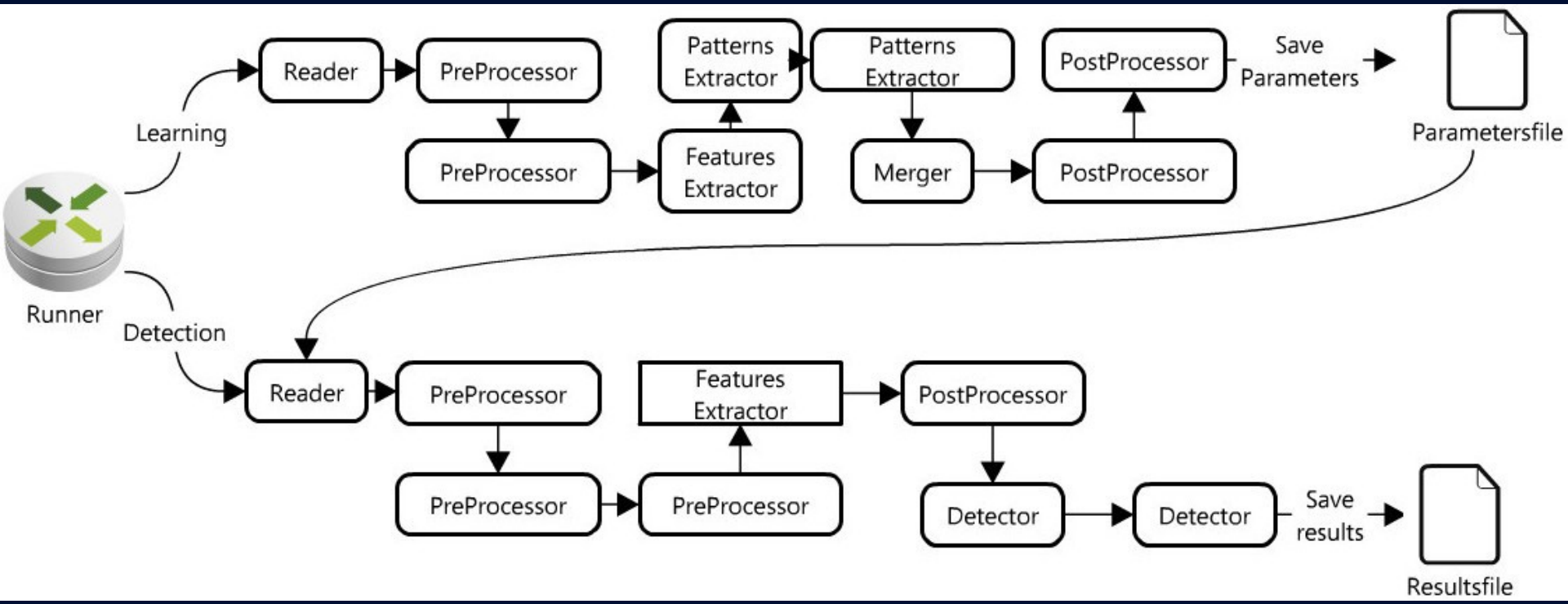
- 2 phases/modes:
  - Apprentissage: apprend les paramètres (motifs récurrents) à partir d'un corpus de textes étiquetés.
  - Détection/Identification: à partir des paramètres appris, identifie l'auteur d'un texte non étiqueté.
- Workflow similaire (sauf exception config)

Workflow



- Peut réutiliser (quasi) même routine sur données d'apprentissage et détection

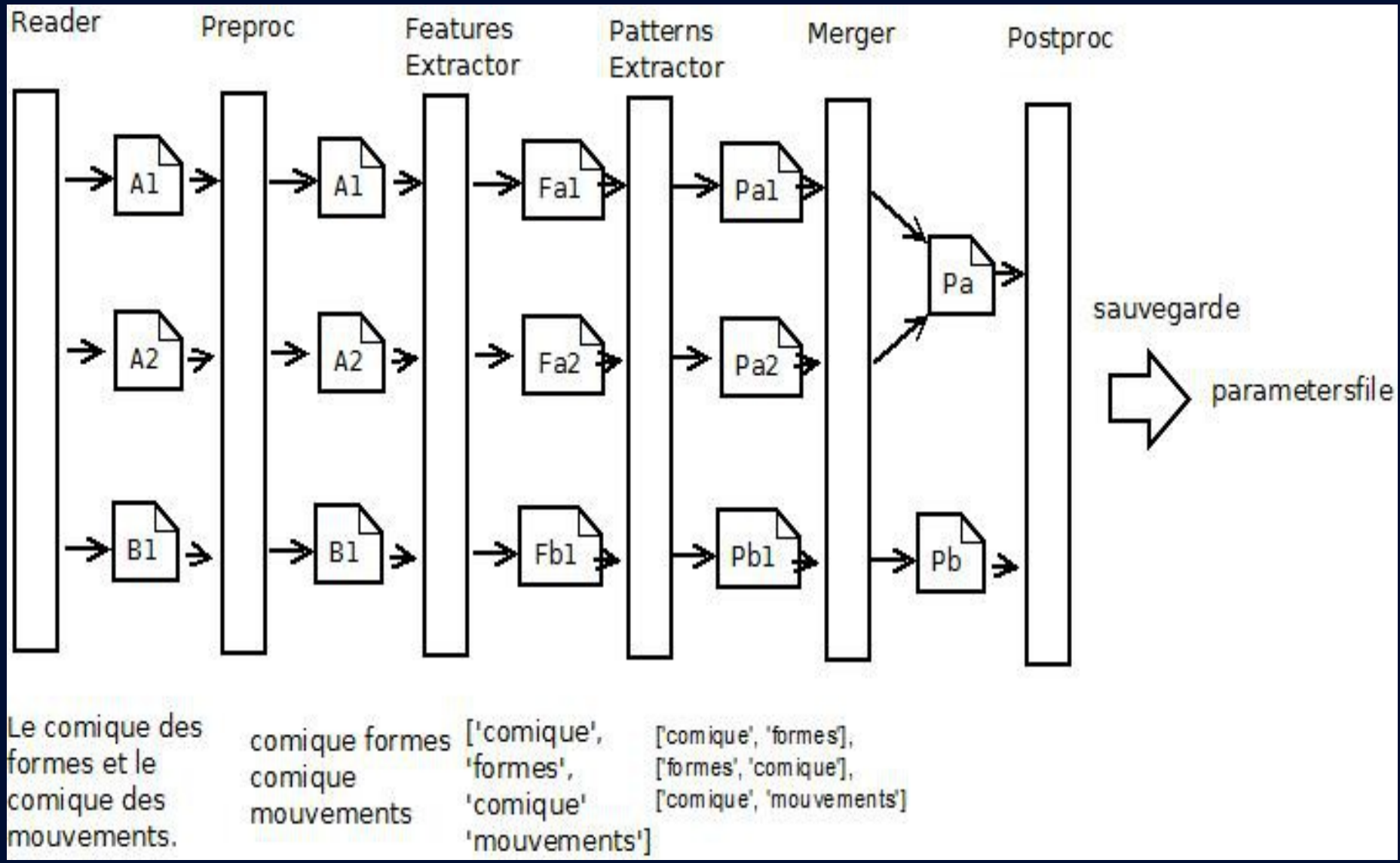
Workflow 2


Learning
Reader
PreProcessor
PreProcessor
Patterns Extractor
Features Extractor
Patterns Extractor
Merger
PostProcessor
PostProcessor
Save Parameters
Parametersfile
Runner
Detection
Reader
PreProcessor
PreProcessor
Features Extractor
PreProcessor
PostProcessor
Detector
Detector
Save results
Resultsfile

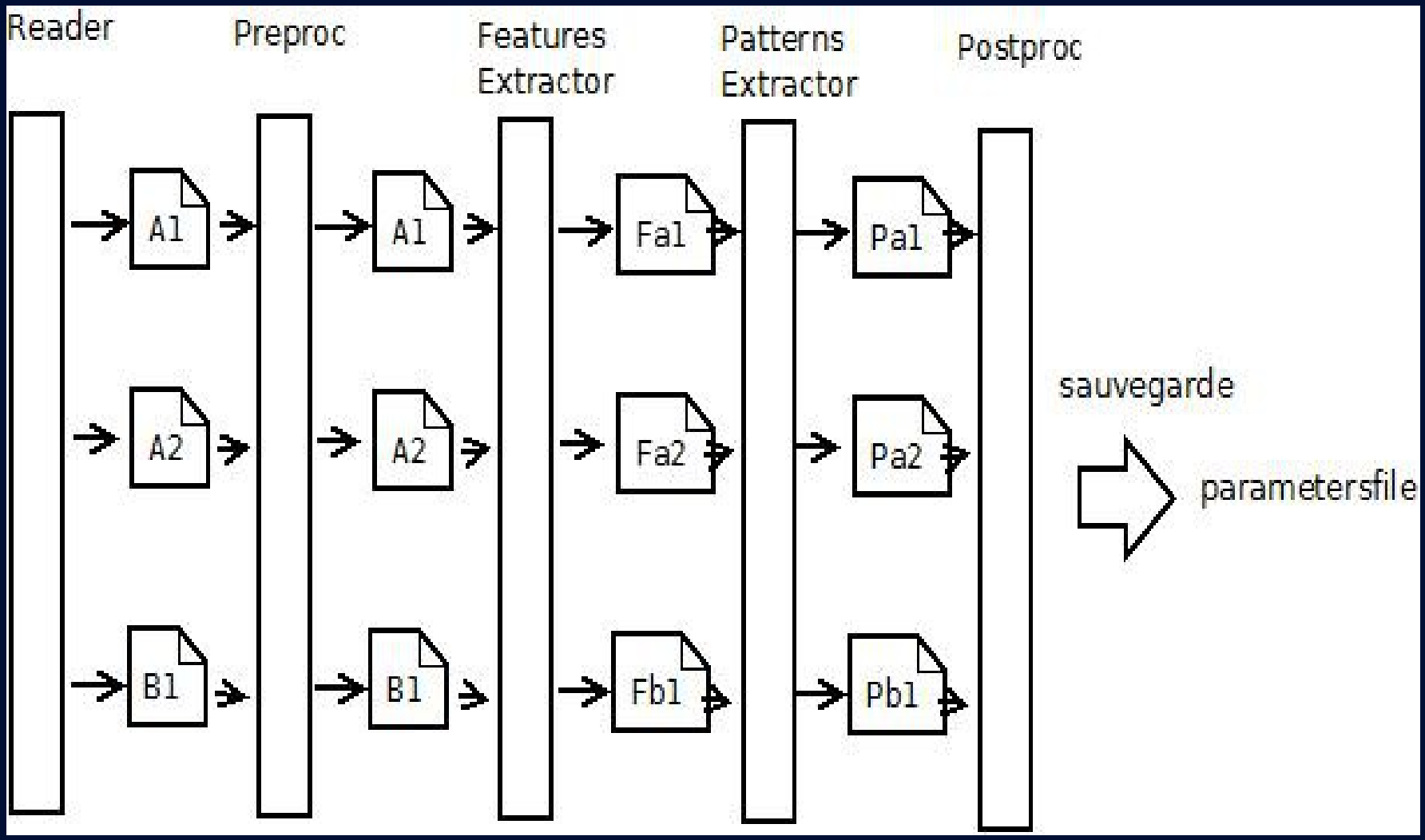
Modélisation - 2

- 3 grandes catégories de workflow selon Stamatatos (2009):
  - Profile-based: 1 style/auteur (textes concaténés)
  - Instance-based: 1 style/texte
  - Hybride (par défaut): commence par instance-based puis finit en profile-based

Exemple - Approche hybride

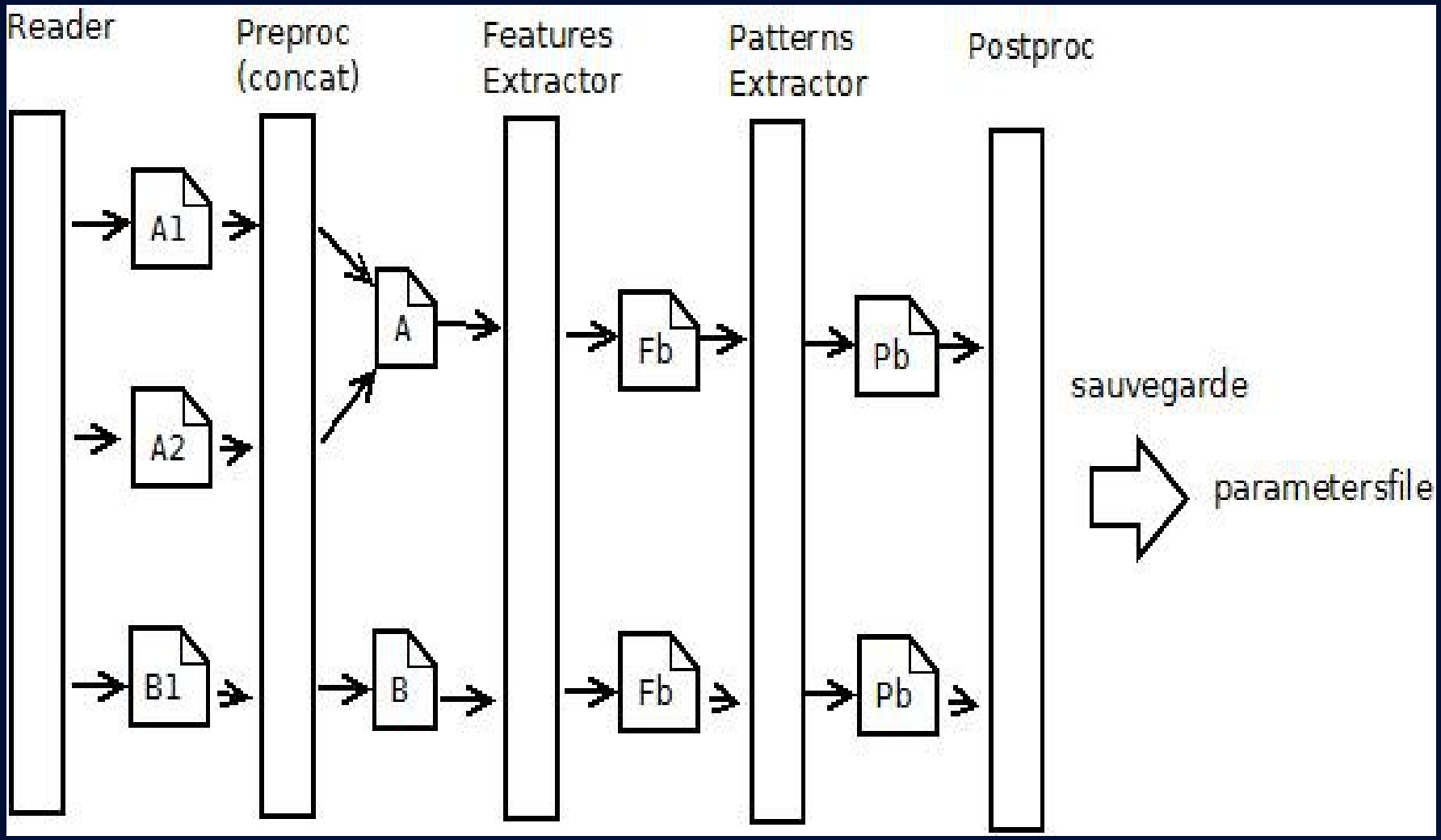


Approche instance-based



Supprime l'étape Merger

Approche profile-based



Rajoute concaténation dans Preprocessor

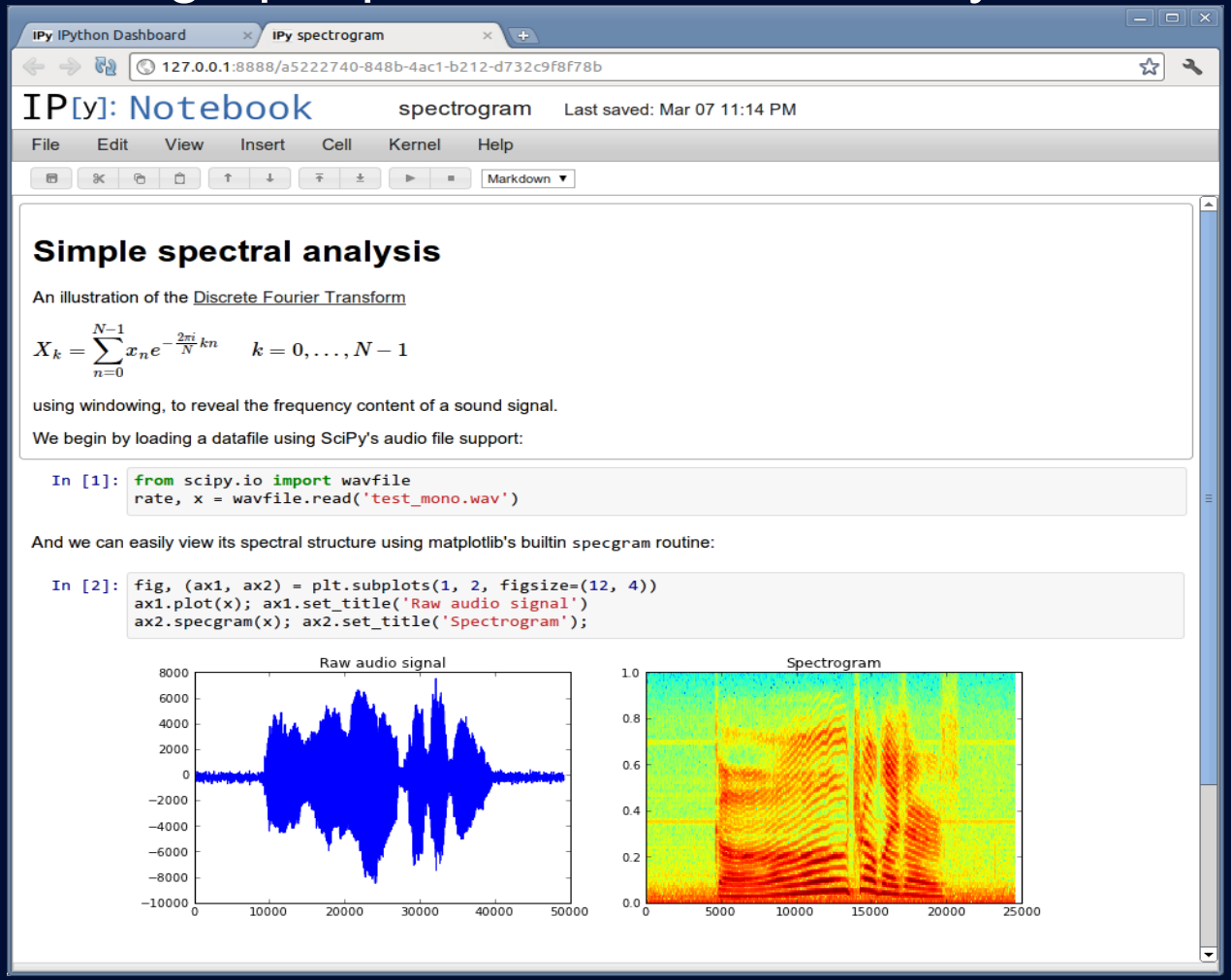
Usage et déploiement

Usage

- Ligne de commande
- Module de script (Python, Java, etc.)
- GUI: IPython Notebook

GUI - Notebook

- Interface graphique interactive avec IPython Notebook



Expérimentations avec reproducibilité et historique
Partage en ligne avec nbviewer

Module de script

- Totalement scriptable comme simple module:

```
import authordetector.main
runner = authordetector.main.main(['--script'])
```

Ligne de commande

- Mode apprentissage:

```
python authordetector.py --learn -c config.json
--textconfig textconfig.json -p parameters.txt
```

- Mode détection:

```
python authordetector.py -c config.json -p
parameters.txt --textconfig_detection
textconfig_detection.json
```

- Aide: `python authordetector.py --help`

Déploiement

- Pour reproducibilité des recherches:

  `AD application + config.json + textes apprentissage`

- Comme pre-logiciel utilisateur:

  `AD application + config.json + parameters.txt`

Config

- Paramètres généraux:
  - workflow: liste des catégories de modules à appeler, format: [{“cat1”: “method1”}, {“cat2”: “method2”}]
    Note: “method” = nom méthode principale à appeler pour chaque module de cette catégorie, qu’ils doivent tous partager selon interface dans module de base (eg: basepreprocessor.py)
  - workflow_learn: idem que workflow mais pour l’apprentissage
  - Classes: nom des classes (dans modules) à charger, format: {“cat1”: “module1”, “cat2”: [“module2”, “module3”], “cat3”: “all”}
    Note: filename d’un module = lowercase(nom classe qu’il contient)
    Note2: on peut charger:
    - soit un module
    - soit une liste de module (appelés dans l’ordre config ici)
    - soit tous les modules dans cette catégorie avec “all”.
  - Arguments commandline (voir --help) qu’on peut aussi mettre dans config (sauf --config)

- D’autres paramètres existent pour chaque module

Futurs plans

Futurs plans

- Interopérabilité avec Java via Jython
- Meilleur score similarité (cosinus? SPI?)
- PAN analyse (émotion)? Arbres stratifiés ordonnés pour groupe grammatical au lieu de catégorie grammatical par mot?
- Extension modèle bayésien par Peng, et al. (2004)
- Translater modèles topic-based, en particulier Rosen-Zvi et al (2010)
- S’inspirer des algos détection artiste musique (multi-critères)
- Module d’évaluation des performances?

Merci!

THE END