Manipulation des données textuelles utilisation des outils JAVA pour le projet MAIAD

Vincent Guigue
Laboratoire d'Informatique de Paris 6 (LIP6)

Traitements pour la classification de textes

Les données textuelles sont difficiles à gérer :

1. Les corpus sont volumineux, le vocabulaire est grand :
   - il faut des algorithmes rapides,
   - les données ne sont pas toujours stockables en mémoire.
2. La structure des phrases est difficile à gérer.
3. Les mots peuvent prendre plusieurs formes (pluriels...)
4. Les algorithmes de machine learning ont du mal sur des données de grande dimension

Traitements pour la classification de textes

Les données textuelles sont difficiles à gérer :

1. Les corpus sont volumineux, le vocabulaire est grand :
   - il faut des algorithmes rapides,
   - les données ne sont pas toujours stockables en mémoire.

   Le boosting est recommandé dans ce cas !
2. La structure des phrases est difficile à gérer.
   On supprime la structure...
3. Les mots peuvent prendre plusieurs formes (pluriels...)
   Plusieurs approches possibles... (cf plus loin)
4. Les algorithmes de machine learning ont du mal sur des données de grande dimension
   On cherche des heuristiques pour supprimer les mots inutiles.

Sacs de mots

Ne sachant pas prendre efficacement en compte la structure des phrases... On l'élimine totalement !
Un document devient alors un comptage des différents mots qui le composent :

Représentation *bag of words*

Soit V le vocabulaire et d un document : $d \in \mathbb{N}^{|V|}$

NB : d est (presque toujours) un vecteur *sparse*, c'est à dire composé essentiellement de 0.

Elimination du bruit : mots similaires

Etant donnée la représentation en sac de mots, il est pénalisant de compter les occurrences de président et présidents dans deux cases séparées... Nous allons donc traiter les mots du textes pour les ramener à leurs radicaux :

- mangeait, mangera, mangeoire,... $\longrightarrow$ manger

Lemmatisation : approche basée sur un dictionnaire

efficace / il faut un dictionnaire...
Exemple d'outil : treetagger (gratuit, récupérable sur internet)

Stemmatisation : approche statistique de suppression des suffixes

rapide, facile à utiliser / parfois approximatif
cf outils fournis dans le projet

Traitements discriminants : réduction de la dimensionnalité

Les algorithmes de machine learning sont mis en difficulté sur les problèmes de grandes dimensions... Nous cherchons donc à réduire la dimension des données :

Heuristiques :

- Elimination des mots peu fréquents
- Elimination des mots courts (articles...)
- Elimination de tous les éléments inutiles a priori (chiffres...)

Traitements discriminants

Le codage *tf-idf* permet de faire apparaître les mots *saillants*, caractéristiques d'un document.
Soit le document d_j tiré de l'ensemble D, $n_{i,j}$ désigne le nombre d'occurrences du mot t_i dans d_j :

$$\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}, \quad \mathrm{idf_i} = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

$\mathrm{tf_{i,j}}$: fréquence de t_i dans le document j.
$\mathrm{idf_i}$: pourcentage des documents où t_i apparaît (log de l'inverse).

On remplace le codage $n_{i,j}$ par le codage $\mathrm{tf-idf}(i,j) = \mathrm{tf_{i,j}} \times \mathrm{idf_i}$
`http://fr.wikipedia.org/wiki/TF-IDF`

Traitements discriminants : sélection de variables

Il est possible d'utiliser des critères discriminants (donnant un score à chaque mot) pour choisir un sous-ensemble du dictionnaire sur lequel travailler.
Parmi les classiques :

- Saillance : $S_{tf-idf}(i) = \frac{\sum_j \mathrm{tf-idf}(i,j)}{|\{\mathrm{tf-idf}(i,j) \neq 0\}|}$
- Odds ratio : $S_{odds}(i) = \frac{p_i/(1-p_i)}{q_i/(1-q_i)} = \frac{p_i(1-q_i)}{q_i(1-p_i)}$. (souvent utilisé en log). Où p_i est la probabilité d'observer le mot t_i dans la classe 1 et q_i est la probabilité d'observer t_i dans la classe 2.

Présentation des données

Données d'apprentissage :

```
<100:1:C> Quand je dis chers amis, ...
<100:2:C> D'abord merci de cet ...
...
<100:14:M> Et ce sentiment  ...
```

Le format est le suivant : <ID-Discours :ID-phrase :Etiquette>, C $\rightarrow$ Chirac, M $\rightarrow$ Mitterrand

Données de test, sans les étiquettes :

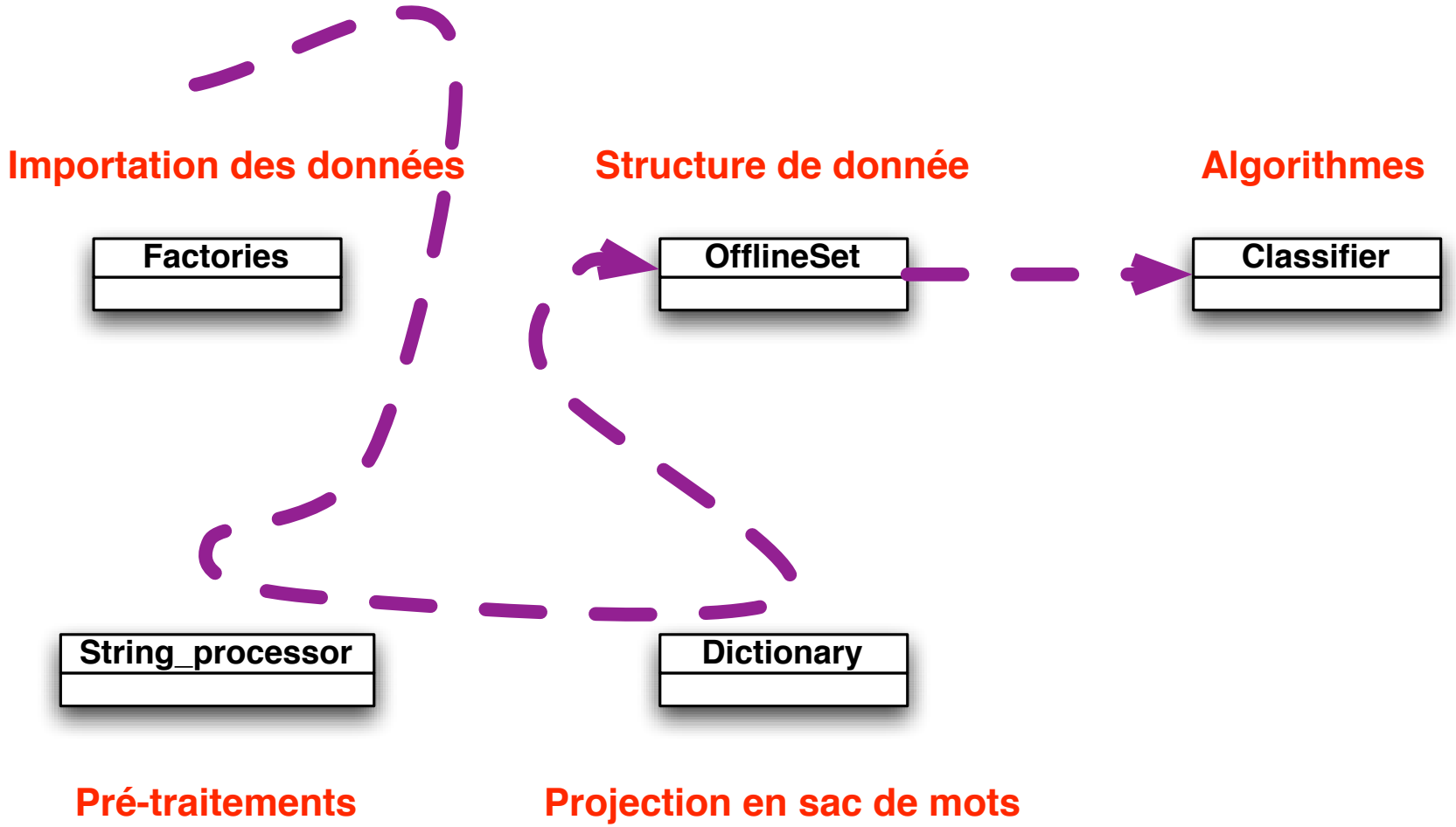
```
<100:1> Quand je dis chers amis, ...
<100:2> D'abord merci de cet ...
...
```

Outils JAVA mis à disposition

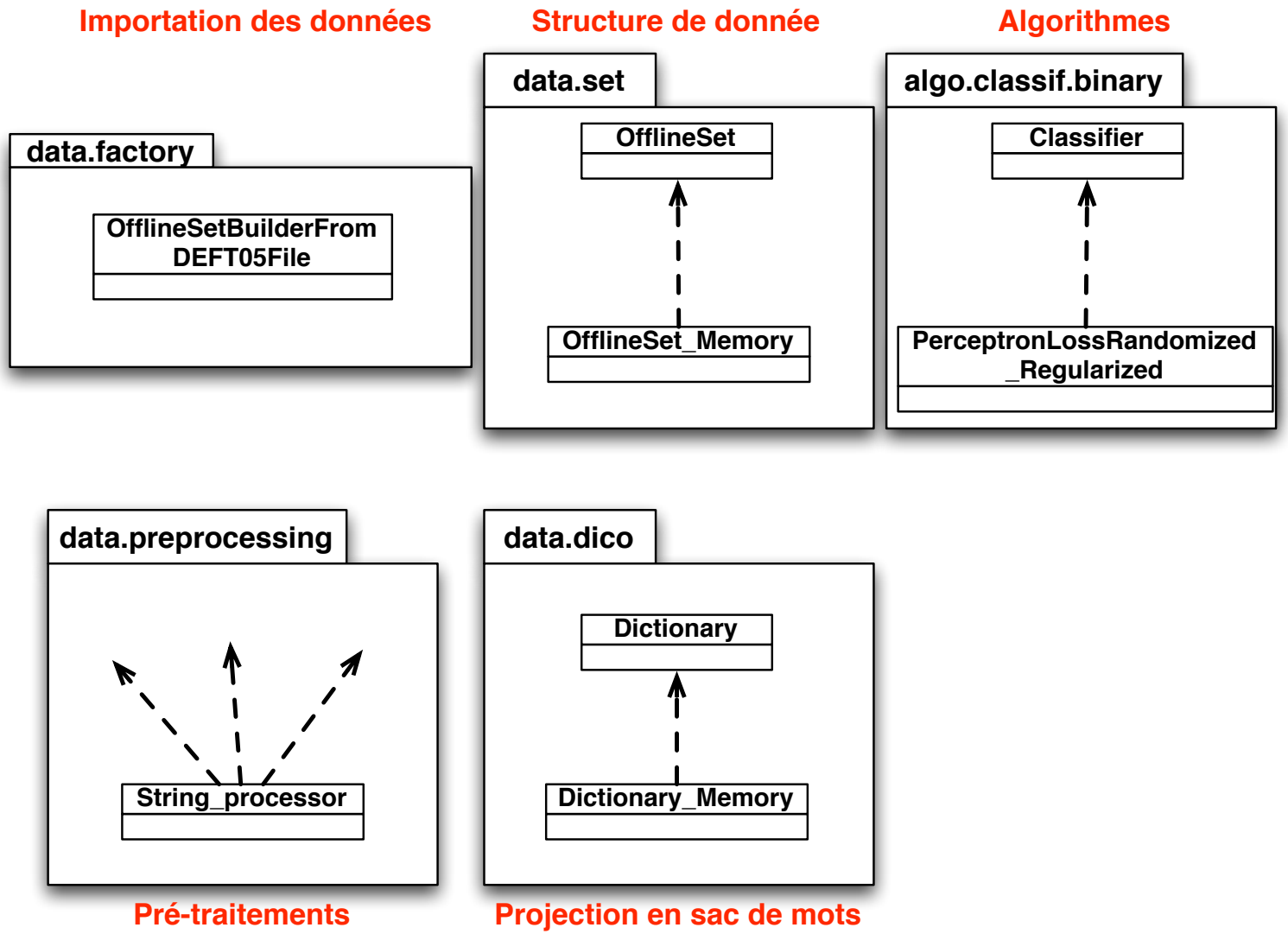
- Contraire aux bonnes pratiques...
- Pas de Javadoc ! ! !
- Mais un tutoriel
- En un prof. pour répondre aux questions

Bref, il faut regarder dans `TutorielPerceptron.java`

Organisation générale du code



Organisation générale du code (2)



TutorielPerceptron.java

Usage de base : chargement d'un corpus DEFT :

```
boolean learningSet = true;
Pair< OfflineSet<String>, OfflineSet<String> > c =
  OfflineSetBuilderFromDEFT05File.build(filename, learningSet);
OfflineSet<String> corpus = c.getX();
OfflineSet<String> labels = c.getY();

//=========================
// exemple d'utilisation:
// 1) definition d'iterateur sur les documents et les labels
OfflineSetIterator<String> iter = corpus.inputsetiterator();
OfflineSetIterator<String> iterLabels = labels.inputsetiterator();

// Afficher le premier document:
iter.next(); iterLabels.next();// passer au premier

System.out.println("Doc : \n " + iter.getCurrentObject()+"\n---\n")
System.out.println("Label : \n " + iterLabels.getCurrentObject()+"\
```

Pre-processing dans TutorielPerceptron.java

Construction :

```
StringProcessor_FromStringProcessors sp2 =
    new StringProcessor_FromStringProcessors();
sp2.add(new StringProcessor_RemoveLineAndSpace());
sp2.add(new StringProcessor_LowerCase());
sp2.add(new StringProcessor_RemovePunctuation());

sp2.add(new StringProcessor_TreeTagger("treetagger",
            "treetagger/models/french.par:iso8859-1", null));
sp = sp2;
```

Définition de l'interface :

```
public interface StringProcessor {
        public String map(String str);
}
```

Dictionnaire dans TutorielPerceptron.java

```
System.out.println("Construction du dictionnaire à partir du corpus");
dico = DictionaryBuilderFromOfflineSet.build(corpus, sp);

// si le mot n'apparait qu'une fois -> eliminé
int limiteApparition = 2;
DictionarySaverWithoutRareWords.save(dicoFile,
    dico, limiteApparition);
```

Définition de l'interface :

```
public interface Dictionary<X> extends OfflineSet<X>
{
  public int indexOf(X x);
  public int size();
  public OfflineSetIterator<X> offlinesetiterator();
}
```

Format de sauvegarde octave

Dans le code JAVA, nous sauvegarderons les données numériques au format octave ainsi que le dictionnaire. Les classes de sauvegardes sont fournies.

```
load("../data/dicoInOctave.dat");

dico
{
   [1,1] = quand
   [2,1] = cher
   [3,1] = amis
   [4,1] = agit
   [5,1] = formul
   [6,1] = diplomat
   [7,1] = mais
   [8,1] = express
   [9,1] = ressen
```

Bibliographie intéressante

Références, code...

`http://www.cs.princeton.edu/~schapire/boost.html`

`http://cseweb.ucsd.edu/~yfreund/`

Mots/idées clés :

- Maximisation AUC
- Ranking
- Sélection de variables
- Précision, Rappel, score F1