

## Documentation Ancor2

31/05/2018

# Table des matières

<b>1</b>	<b>Informations générales</b>	<b>2</b>
	Dépôts git . . . . .	2
	Données des expériences . . . . .	2
	Documentation . . . . .	2
	Dépendances . . . . .	2
<b>2</b>	<b>Processus</b>	<b>3</b>
2.1	Appel des fonctionnalités . . . . .	3
<b>3</b>	<b>Architecture du projet</b>	<b>4</b>
3.1	com.democrat.ancortodemocrat . . . . .	5
	com.democrat.ancortodemocrat.element . . . . .	6
	com.democrat.ancortodemocrat.feature . . . . .	7
	com.democrat.ancortodemocrat.positioning . . . . .	8
	com.democrat.ancortodemocrat.treetagger . . . . .	9
3.2	com.democrat.classification . . . . .	10
3.3	com.democrat.expes . . . . .	11
<b>4</b>	<b>Entrées - Sorties</b>	<b>12</b>
4.1	Calcul des features . . . . .	12
	Etapes du calcul des features . . . . .	12
4.2	Génération des arff . . . . .	13
	Etapes génération Arff . . . . .	13
4.3	Génération du model . . . . .	14
	Etapes génération du model . . . . .	14
4.4	Classification . . . . .	15
	Etapes de la classification . . . . .	15
4.5	Chaînage . . . . .	16
	Etapes du Chaînage . . . . .	16
<b>5</b>	<b>Refactoring</b>	<b>18</b>

# Informations générales

## Dépôts git

**Alexis** <https://bitbucket.org/Slayug/ancortodemocrat.git>

**Augustin** <https://gitlab.com/augustinhoima/ancor2.git>

## Données des expériences

<https://drive.google.com/drive/u/0/folders/14mp2w690RVP6yh3H2dmxkYIeB-LMa2E2>

## Documentation

<https://docs.google.com/document/d/1vJP4o1z3eCfZSxo88vZTlS0OpT54B1Zu-Nfq4naSlWc/edit?usp=sharing>

## Dépendances

**git submodule** — reference-coreference-scorers

**maven javax.xml.bind** Parser XML

**log4j** Sortie text

**nz.ac.waikato.cms.weka** Classification

**org.annolab.tt4j** TreeTager : gestion des annotations *part-of-speech*

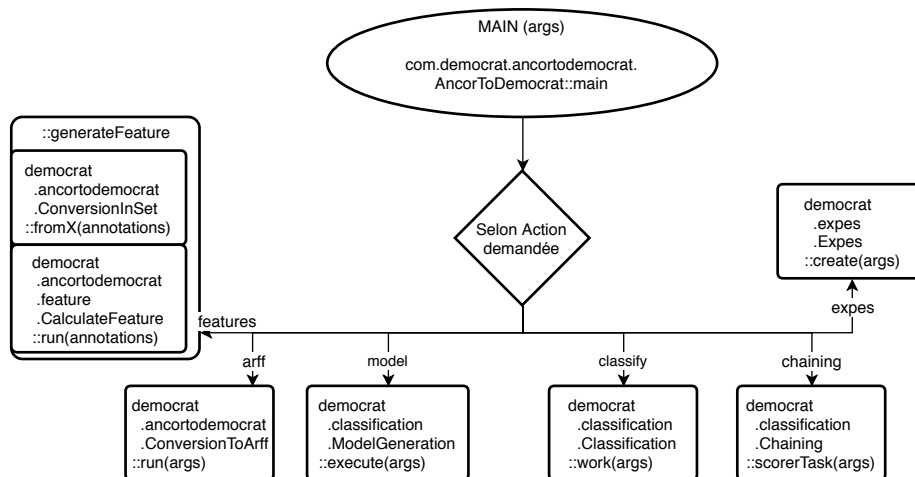
**commons-cli** Gestion des arguments en ligne de commande (Remplacement de l'ancienne gestion des arguments à faire)

**commons-io** Gestion des Fichiers

**org.odftoolkit** Gestion des fichiers OpenDocument (sortie des résultats pour les expériences, en cours de dvlp)

# Processus

## Appel des fonctionnalités



# Architecture du projet

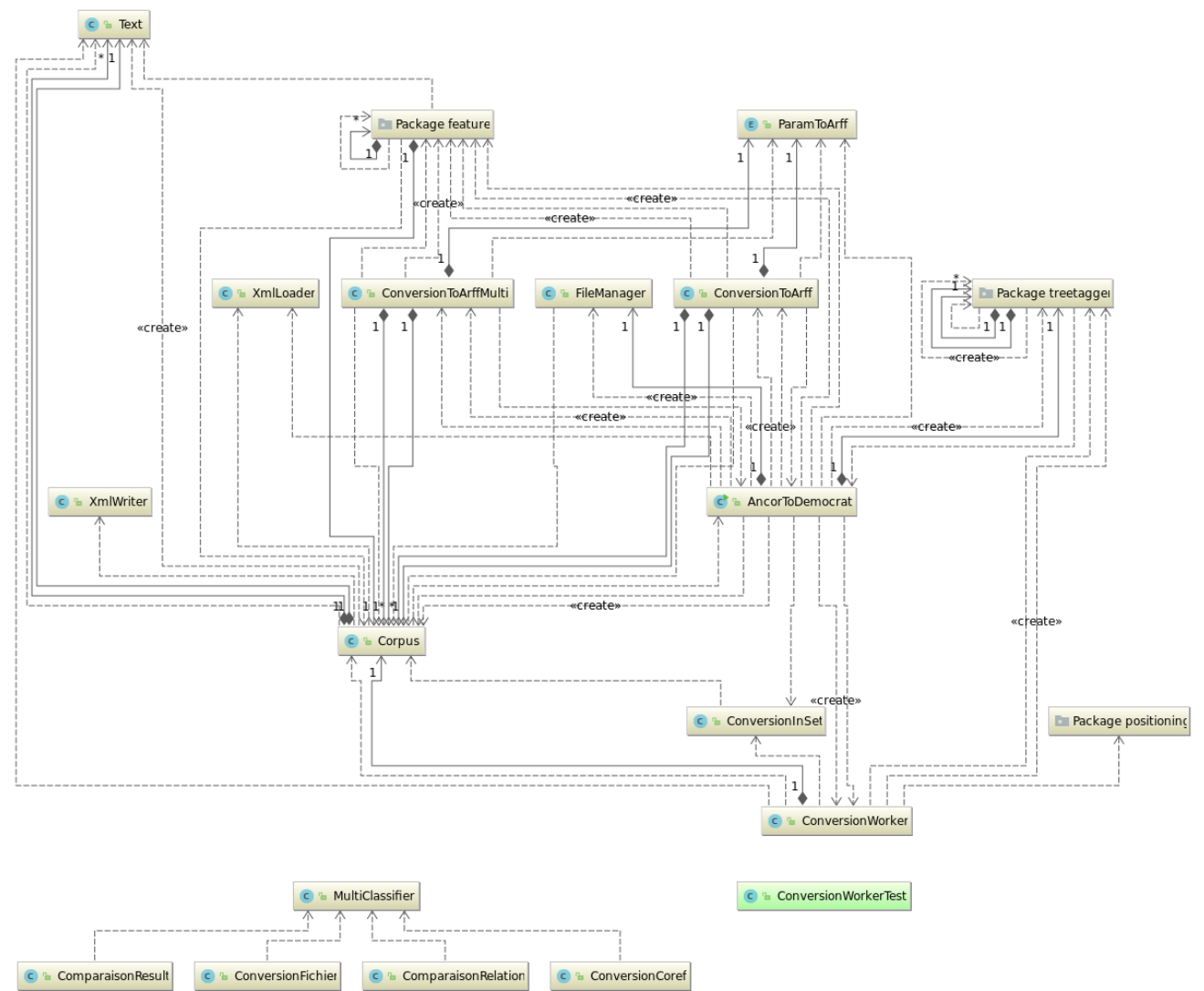
Packages du projet

```
com.democrat
├── com.democrat.ancortodemocrat
│   ├── com.democrat.ancortodemocrat.element
│   ├── com.democrat.ancortodemocrat.feature
│   ├── com.democrat.ancortodemocrat.positioning
│   └── com.democrat.ancortodemocrat.treetager
├── com.democrat.classification
├── com.democrat.expes
│   └── com.democrat.expes.rjc18
```

# com.democrat.ancortodemocrat

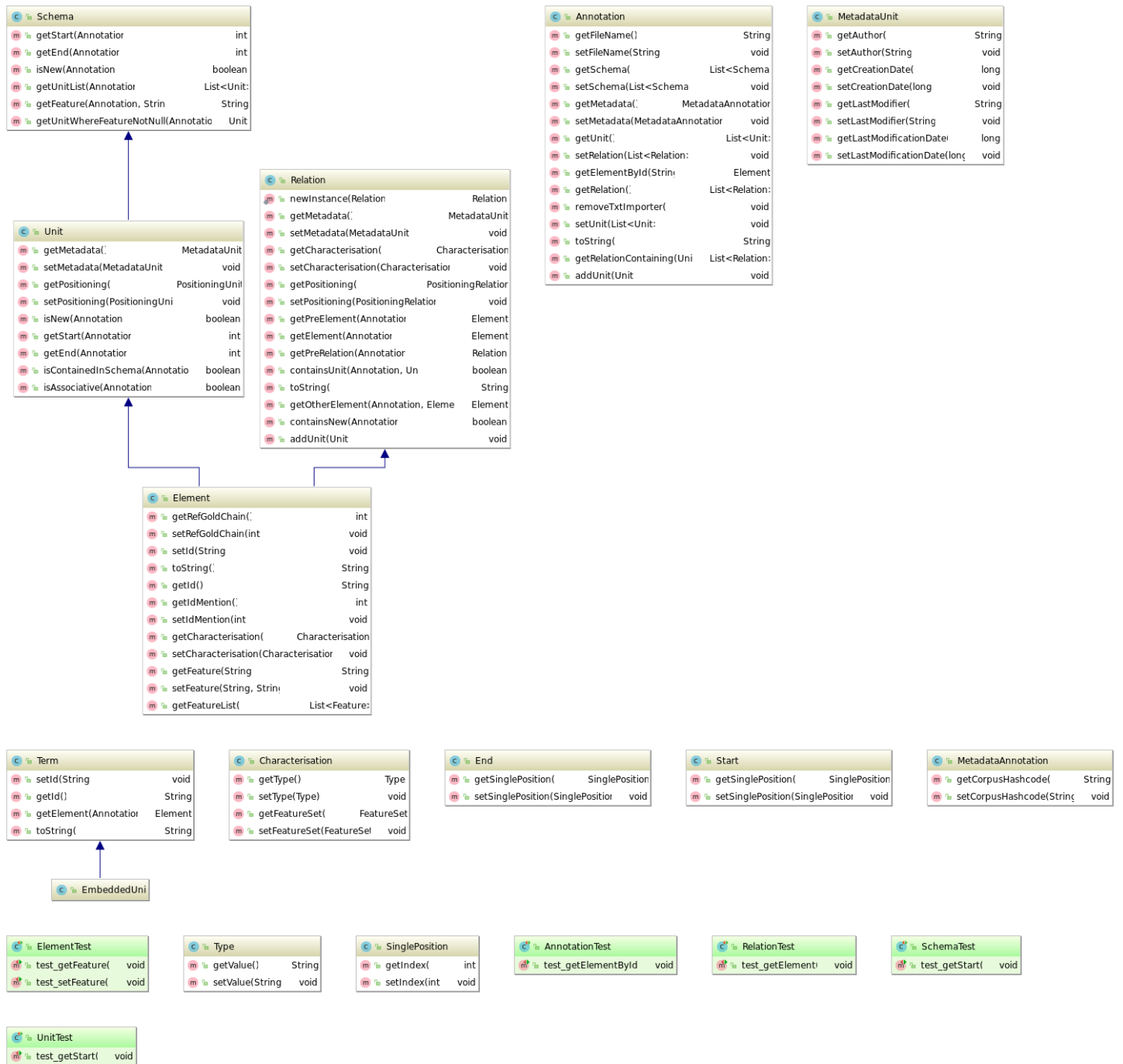
sous-packages

```
com.democrat.ancortodemocrat
├── com.democrat.ancortodemocrat.element
├── com.democrat.ancortodemocrat.feature
├── com.democrat.ancortodemocrat.positioning
├── com.democrat.ancortodemocrat.treetager
```



Powered by yhlies

com.democrat.ancortodemocrat.element



com.democrat.ancortodemocrat.feature

Feature		
m	getValue()	String
m	setValue(String	void
m	getName()	String
m	setName(String	void

CalculateFeature		
m	calculateFeatureOnRelation(Annotation, Relativ	void
m	run()	void

FeatureSet		
m	getFeature('	List<Feature:
m	setFeature(List<Feature:	void

Powered by yFiles



com.democrat.ancortodemocrat.positioning

PositioningUnit		
m	getStart()	Start
m	setStart(Start	void
m	getEnd()	End
m	setEnd(End	void
m	toString()	String
m	getEmbeddedUnit	List<EmbeddedUnit
m	setEmbeddedUnit(List<EmbeddedUn	void

PositioningRelation		
m	getTerm()	List<Term>
m	setTerm(List<Term>	void
m	toString()	String

PositioningSchema		
m	getEmbeddedUnit	List<EmbeddedUnit
m	setEmbeddedUnit(List<EmbeddedUn	void

Powered by y-files

com.democrat.ancortodemocrat.treetagger

C	ResultMention
m	newToken(ResultToken: void
m	getTokenList(): List<ResultToken:
m	containsNoun() boolean
m	getNounList() List<ResultToken:
m	toString() String

C	TokenConvertMentionHandler
m	token(String, String, Strir void
m	isDone() boolean
m	getResultMention() ResultMention
m	getMentionSplitted() String[

C	ResultToken
m	getToken() String
m	getPos() String
m	getLemma() String
m	isNoun() boolean

C	TokenConvertRelationHandle
m	run() void
m	isDone() boolean

C	TreeTagger
m	work(TokenConvertMentionHandle void

Powered by yFiles

## com.democrat.classification

C Chaining		
m	scorerTask(String[])	void
m	scorerTask(String, String, String, String[], boolean, boo	void
m	scorerTask(ScorerArgs)	void
m	createSet(String, String, TypeChai	ArrayList<ArrayList<String
m	writeCoNNL(String, String, ArrayList<ArrayList<String>>, HashMap<String, Integer>, TypeChai	void
m	removeChainFromList(List<Chain>,	void
m	containsChain(List<Chain>, i	boolean
m	getChainFromList(List<Chain>,	Chain
m	setRefIfNeed(List<Corpus:	void
m	eval(String, String, Strir	String
m	loadInstance(String	Instances

C Model		
m	learnModel(String, AbstractClassifie	Model
m	loadModel(String	Model
m	getPath()	String
m	setPath(String	void
m	classifyInstance(Instance:	void
m	classifyInstanceProba(Instance	Instances
m	export(String	void
m	crossValidate(Instances, in	Evaluation

C Chain		
m	getRef()	int
m	getMentionList(	List<Mention:
m	addMention(Mention	void
m	containsMention(Mention	boolean
m	containsMention(int	boolean
m	getMention(int	Mention
m	removeMention(int	void
m	size()	int

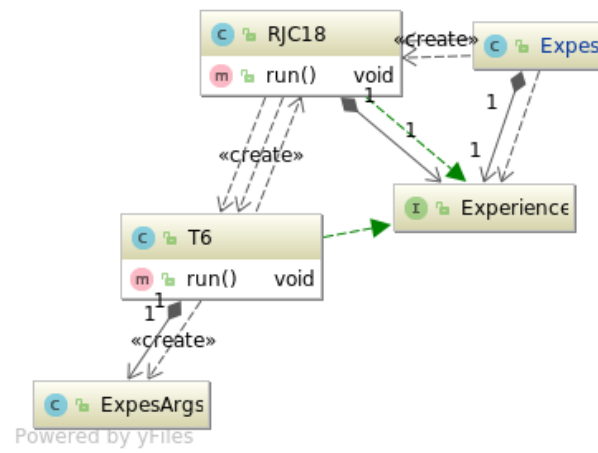
C Mention		
m	getId()	int
m	setId(int:	void
m	isCoref()	boolean
m	setCoref(boolean)	void

C Classification		
m	work(ClassifArgs)	void

C ModelGeneration		
-------------------	--	--

Powered by yFiles

com.democrat.expes



# Entrées - Sorties

## Calcul des features

**Entrée :** Corpus source : chemin/vers/Ancor

```
chemin/vers/corpus/Ancor
├── aa_fichiers
│   └── *.aa
└── ac_fichiers
    └── *.ac
```

**Sortie :** Corpus avec nouveaux features calculés : chemin/vers/features

```
chemin/vers/features
├── aa_fichiers
│   └── *.aa
└── ac_fichiers
    └── *.ac
```

## Etapas du calcul des features

```
com.democrat.AncorToDemocrat::generateFeature(corpus, output){
  Pour chaque annotation du corpus:
    ConversionInSet.toSetFrom{Chain/FirstMention}(annotation) // Génère le feature REF
  CalculateFeature::new(corpus,output){
    Pour chaque annotation du corpus:
      calculateNewFeature(annotation)
      calculateFeatureOnRelation(annotation)
    corpus.export(output)
  }
}
```

## Génération des arff

**Entrée :** Corpus contenant tous les features : chemin/vers/features

**Sortie :** nomfichier

- nomfichier(date).arff → Contient toutes les instances
- nomfichier(date).idff → Contient les identifiants de chaque instance et des mentions qui les composent

### Etapas génération Arff

```
com.democrat.ancortodemocrat.ConversionToArff::work(){
    sortInstance(){
        Pour chaque corpus:
            Pour chaque annotation du corpus:
                Pour chaque relation de l'annotation:
                    positiveRelationSelected.put(relation)
                    generateNegativeRelation(corpus,annotation, relation){
                        Pour chaque relation négative à générer:
                            newrelation = deux mentions au hasard qui ont
                                un champ REF différent
                            negativeRelationSelected.put(newrelation)
                    }
                }
    }
    selectInstance(){
        restriction nb_pos et nb_neg en fonction
        du nombre d'instances dans le corpus
        et en conservant le ratio
        mélange aléatoire de la liste des instances positives // à modifier
        mélange aléatoire de la liste des instances négatives
    }
    writeInstance()
}
```

## Génération du model

**Entrée :** Corpus d'apprentissage et de test

- train.arff
- test.arff

**Sortie :** Model

**Etapas génération du model**

```
com.democrat.classification.ModelGeneration::execute( classfierclass, args ){  
    AbstractClassifier.runClassifier(classfierclass.newInstance(),args);  
}
```

## Classification

**Entrée :**

- Model
- Fichier arff à classifier

**Sortie :** nomfichier.arff → Instances classifiées

**Etapas de la classification**

```
com.democrat.classification.Classification::work(args){  
    instances = ArffLoader::getDataSet();  
    model = Model.loadModel(args.model);  
    instances = model.classifyInstancesProba(instances);  
    ArffSaver::setInstances(instances);  
}
```



## Chaînage

**Entrée :** corpus gold et system

- Gold.arff
- System.arff

**Sortie :**

- *nomfichier\_conll\_to\_ancor.csv* → Table de correspondance des identifiants de mentions conll / Ancor
- *nomfichier\_GOLD.conll* → Chaines Gold au format conll
- *nomfichier\_SYSTEM.conll* → Chaines System au format conll

**Facultatif :**

- *nomfichier\_LOE\_GOLD.csv* → List of Edges Gold (ensemble des relations en première mention pour Gephi)
- *nomfichier\_LOE\_SYSTEM.csv* → List of Edges System (ensemble des relations en première mention pour Gephi)
- *nomfichier\_LOM.csv* → List of Mentions (ensemble des mentions (An-cor\_ID, Conll\_ID, Chain\_ID, Num\_Antecedents\_Before\_BestFirst))

### Etapas du Chaînage

```
com.democrat.classification.Chaining::scorerTask(args){
    goldChains = createSet(goldArff)
    // idem pour system

    pour chaque mention
        ecrire equivalence id_Ancor / id_Conll dans mention_str_to_int

    writeCoNNL(goldOut, mentions_str_to_int, goldChaine);
    // idem pour system
}

createSet(fichier){
    instances = loadInstances(fichier);

    boucle: Ajout de chaque mention dans un ensemble

    antécédents_possibles = HashMap<mention, HashMap<antécédent, proba> >
    Pour chaque instance coref:
        ajout de (instance.ancestor,proba) pour instance.element
        dans antécédents_possibles

    renvoyer constructChains(antécédents_possibles, mentions){

        chaines = ArrayList<HashMap<String,Integer>> // liste de chaines
        // chaque mention connaît son nombre d'antécédents possibles avant best-first

        pour chaque mention parmi les clés de antécédents_possibles:
            antécédent = antécédent max par proba dans antécédents_possibles.get(mention);
```

On enlève mention et antécédent du dictionnaire mentions pour ne laisser que les singletons

boucle sur chaine: on place mention et antécédent dans la chaîne qui contient l'un des deux

Si aucune chaîne ne contient mention ou antécédent, on crée une nouvelle chaîne qui les contient

Si mention et antécédent sont placés dans deux chaînes, on les fusionne.

boucle: on crée et ajoute une chaine par mention restante dans le dictionnaire mentions (ce sont les singletons)

renvoyer chaines;

}  
}

# Refactoring

**Refactoring de la classe principale** sous-traiter à des sous-classes principales pour chaque fonctionnalité

**Gestion des arguments** utiliser la librairie *commons-cli* pour avoir une gestion des arguments plus simple et maintenable, et locale à chaque fonctionnalité (sous-classes principales).

**Refactoring des duplicatats de code** Présents à beaucoup d'endroits

**ConversionToArff\*** Suppression des classes actuellement inutiles

**Multi-Threading** Certains traitements sont réalisés dans des threads qui n'exploitent pas forcément le multi-threading.

→Suppression de ces threads ou optimisation ?