

# Documentation ancor2

## [Présentation](#)

### [1.1\) Contexte](#)

### [1.2\) Téléchargement des sources](#)

### [1.3\) Installation](#)

[Linux & Windows, étape commune:](#)

[Windows](#)

[Configuration de TreeTagger](#)

[Linux](#)

[Configuration de TreeTagger](#)

[Variable d'environnement](#)

### [1.4\) Utilisation](#)

[Conversion des corpus en première mention vers en chaîne.](#)

[Pour générer les nouveaux traits \(features\):](#)

[Pour générer un fichier arff](#)

[Pour faire appel au scorer](#)

## [2\) Principe du traitement](#)

[Conversion des types de chaînes](#)

[Conversion des chaînes avec apprentissage](#)

## [3\) Utilisation du code:](#)

### [Packages](#)

[Package principal: ancortodemocrat](#)

[AncorToDemocrat](#)

[XmlLoader](#)

[XmlWriter](#)

[Corpus](#)

[Text](#)

[ConversionInSet](#)

[ConversionWorker](#)

[ConversionToArff](#)

[FileManager](#)

[Second package:](#)

[Annotation](#)

[Element](#)

[Unit extends Element](#)

[Schema extends Unit](#)

[Relation](#)

[MetadataAnnotation](#)

[MetaDataUnit](#)

[Characterisation](#)

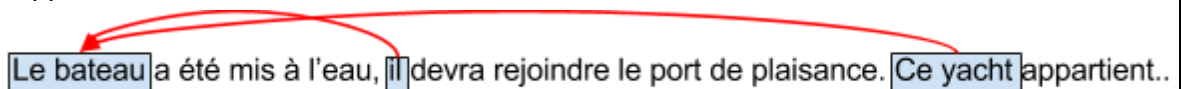
[Start & End](#)  
[SinglePosition](#)  
[PositionningUnit](#)  
[PositioningRelation](#)  
[Term](#)  
[CalculateFeature](#)  
[Package treetagger](#)  
[ResultMention](#)  
[ResultToken](#)  
[TokenConvertMentionHandler](#)  
[TokenConvertRelationHandler](#)  
[TreeTagger](#)  
[Package Speech](#)  
[Trans](#)  
[Episode](#)  
[Section](#)  
[Turn](#)  
[Package Classification](#)  
[Chain](#)  
[Mention](#)  
[Model](#)  
[Scorer](#)  
[Liste des traits](#)

# 1) Présentation

## 1.1) Contexte

Ce système a été conçu dans l'objectif de convertir des corpus annotés en anaphore dans le cadre du projet Ancor. Les corpus ciblés ont été annotés avec le logiciel Glozz. Ces corpus sont annotés en 1<sup>er</sup> mention.

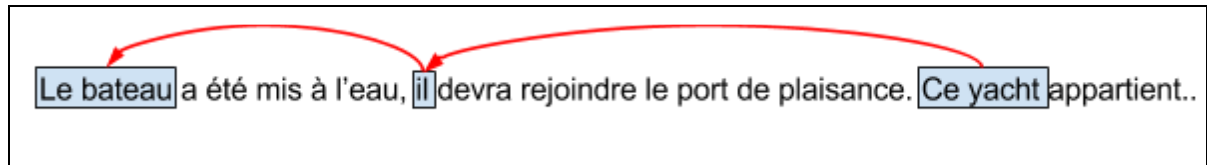
Rappel de 1<sup>er</sup> mention:



Le bateau a été mis à l'eau, il devra rejoindre le port de plaisance. Ce yacht appartient..

**Annotation en 1er mention**

Et le système a pour objectif de convertir ces annotations en chaîne:



### Annotation en chaîne

Sachant que les annotations ont été traitées avec Glozz, le logiciel a besoin des fichiers .aa et .ac. Un corpus doit donc avoir la structure suivante:

```
---- Corpus
----- aa_fichiers
----- fichier.aa
----- fichier2.aa
----- ac_fichiers
----- fichier.ac
----- fichier2.ac
```

## 1.2) Téléchargement des sources

Récupérer les sources depuis le dépôt:

```
git clone https://bitbucket.org/Slayug/ancortodemocrat.git
```

Vous pouvez alors travailler dessus en important le tout comme un projet Maven.

Penser à faire un **maven install** une fois le projet importé.

Vous devez aussi faire l'étape configuration de TreeTagger en fonction de votre système d'exploitation. (voir ci-dessous).

## 1.3) Installation

Pour le moment le programme compilé n'est pas récupérable directement.

Vous devez donc passer par le [téléchargement des sources](#) pour ensuite compiler le projet et pouvoir l'utiliser comme vous le souhaitez.

Sans avoir à importer le projet dans un IDE, vous pouvez en ligne de commande faire les étapes suivantes après avoir récupérer le dépôt.

### Linux & Windows, étape commune:

Placez vous dans le dossier ancortodemocrat que vous venez de récupérer.

Et lancer l'installation via maven. Rien ne vous empêche de le faire depuis votre IDE (Eclipse, NetBeans, etc..)

```
cd ancortodemocrat
mvn install
```

Aller ensuite dans le dossier générer à partir de la commande précédente.

```
cd target
```

Vous remarquez alors que ce dossier contient le .jar avec les dossiers nécessaires au bon fonctionnement du programme.

## Windows

### Configuration de TreeTagger

Remplacer le dossier target/TreeTagger par celui téléchargeable ici:

<http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger-windows-3.2.zip>

Si c'est juste pour utiliser les sources remplacer le dossier TreeTagger à la racine du dossier récupéré.

## Linux

### Configuration de TreeTagger

Avant de commencer à utiliser le programme vous devez configurer TreeTagger.

Si vous l'avez déjà d'installer vous pouvez passer à l'étape [variable d'environnement](#).

Téléchargement de TreeTagger pour Linux:

<http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger-linux-3.2.1.tar.gz>

Vous pouvez mettre l'ensemble des fichiers de l'archive dans un dossier tel que:

```
/home/username/TreeTagger/
```

L'étape suivante est aussi nécessaire si vous utilisez seulement les sources.

### Variable d'environnement

Ajouter la variable suivante à votre .bashrc:

Pensez à remplacer **username**, par le votre !

```
echo "export TREETAGGER_HOME=/home/username/TreeTagger/" >>  
/home/username/.bashrc
```

## 1.4) Utilisation

Pour indiquer quel corpus le logiciel doit convertir, il faut indiquer le ou les chemin(s) absolu(s) dans le fichier *path.txt*. Si ce fichier n'existe pas lancer juste une fois le logiciel, il sera créé dans le dossier contenant le .jar.

Exemple de path.txt avec un chemin absolu:

```
#Vous pouvez mettre des commentaires comme ceci  
#Ajouter le(s) chemin(s) absolu(s) que vous souhaitez des corpus que vous souhaitez.  
#chaque corpus doit comporter un dossier 'aa_fichiers' et 'ac_fichiers'  
C:\Users\votreNomDutilisateur\Document\corpusTest
```

Exemple 1. Contenu d'un path.txt

À partir du moment où vous avez indiqué le ou les corpus voulu(s) il suffit de (re)lancer le logiciel en double cliquant sur le .jar.

Une fois un corpus converti il se retrouve dans le dossier *generated/corpus/* avec les nouveaux .aa.

L'application peut aussi être invoquée en ligne de commande en fonction de la fonction que l'on souhaite utiliser.

### 1. Conversion des corpus en première mention vers en chaîne.

Ici il y a deux méthodes pour convertir des corpus en première mention vers en chaîne.

La première méthode consiste à indiquer les chemins des corpus dans le fichier path.txt vue précédemment, puis lancer le jar.

```
java -jar ancor2.jar
```

La seconde méthode se fait juste en ligne de commande, on lance le jar avec comme argument chain et ensuite le ou les chemins des corpus à convertir.

```
java -jar ancor2.jar chain
```

```
C:\Users\votreNomDutilisateur\Document\corpusTest
```

```
C:\Users\votreNomDutilisateur\Document\corpusTest2
```

Les corpus convertis seront dans generated/corpus/

### 2. Pour générer les nouveaux traits (features):

Cette fonctionnalité prend un corpus en entrée pour ensuite ajouter de nouveaux traits (cf. [Annexe](#))

Paramètres:

1. Type de corpus en entrée
  - a. "p" --> en première mention
  - b. "c" --> en chaîne
2. Chemin du corpus en entrée
3. -o chemin de sortie pour le corpus avec ses traits calculés (si non spécifié, generated/feature/nomDuCorpus)

Exemple:

1. 

```
java -jar ancor2.jar feature p
```

```
C:\Users\votreNomDutilisateur\Document\corpusTest
```
2. 

```
java -jar ancor2.jar feature c
```

```
C:\Users\votreNomDutilisateur\Document\corpusTest -o
```

```
C:\Users\votreNomDutilisateur\Document\corpusTest_feature
```

### 3. Pour générer un fichier arff

Ici la commande prend en entrée un dossier comprenant une liste de corpus ou un corpus avec **les nouveaux traits ajoutés** (ceci est important, car sinon vous allez vous retrouver avec des valeurs erronées pour les instances NOT\_COREF). Ensuite il faut préciser si l'on garde les relations associatives ou non.

Paramètres:

1. paramètre de sortie
  - a. "all" → toutes les relations (ratio  $\frac{1}{3}$  soit une à trois relations négatives pour chaque relation positive)

- b. "no\_assoc" → toutes les relations sans les associatives, ni les associatives pronominales (ratio 1/3 soit une à trois relations négatives pour chaque relation positive)
2. -i chemin du ou des corpus à extraire, peut aussi être un simple fichier .aa (pour garder l'intégrité d'un texte) (si non spécifié, prend tous les corpus dans generated/feature/) (le(s) corpus doit avoir ses features calculés (précédente commande))
3. -q quantité d'instances positives, et d'instances négatives
4. -o nom du fichier .arff qui sera exporté (si non spécifié, generated/arff/dateDuJour\_Heure.arff)
5. -s Indique en combien de partie doit être découpé le/les corpus en entrée

Exemples:

- `java -jar ancor2.jar arff all`
- `java -jar ancor2.jar arff no_assoc -i C:/Users/toast/Documents/stage/ancor/corpus_ESLO`
- `java -jar ancor2.jar arff -i C:/Users/toast/Documents/stage/ancor/corpus_ESLO -o jeSuisUnNomDeFichier`
- `java -jar ancor2 arff -i C:/Users/toast/Documents/stage/ancor/corpus_ESLO/aa_fichiers/023_C.aa -q 1200 500`
- `java -jar ancor2.jar arff all -i generated/feature/ -s 4`

#### 4. Pour faire appel au scorer

Cette commande permet d'avoir les différentes mesures (muc, B3, ceaf)

Elle prend en entrée

Paramètres:

1. paramètre de sortie
  - a. "all" → toutes les relations
  - b. "no\_assoc" → toutes les relations sans les associatives, ni les associatives pronominales
2. -i chemin du ou des corpus à extraire, peut aussi être un simple fichier .aa (pour garder l'intégrité d'un texte) (si non spécifié, prend tous les corpus dans generated/feature/) (le(s) corpus doit avoir ses features calculés (précédente commande))
3. -q quantité de nombre d'instances positives, et d'instances négatives
4. -o chemin de sortie des différentes fichiers
5. -s Indique en combien de partie doit être découpé le/les corpus en entrée
6. -r Liste des traits à supprimer de l'apprentissage
7. -a Type de traits à supprimer pour l'apprentissage (NO\_ORAL OU ONLY\_RELATIONAL)
8. -f Fichier contenant une liste de traits à ignorer, les traits doivent être séparés par des virgules

Les arguments 6, 7 et 8 peuvent être utilisé en même temps.

Exemples:

- `scorer no_assoc -i C:\Users\buggr\workspace\AncorToDemocrat\generated\feature\ -q 100 400 -o C:\Users\buggr\Documents\stage\callScorer -m C:\Users\buggr\Documents\stage\classification\model\en_chaine\2016_07_04_10000_20_80_no_assoc_C.model -a NO_ORAL`

- `scorer no_assoc -i`  
`C:\Users\buggr\workspace\AncorToDemocrat\generated\feature\ -q 100`  
`400 -o C:\Users\buggr\Documents\stage\callScorer -m`  
`C:\Users\buggr\Documents\stage\classification\model\en_chaine\2016_07`  
`_04_10000_20_80_no_assoc_C.model -r id_genre id_nombre`

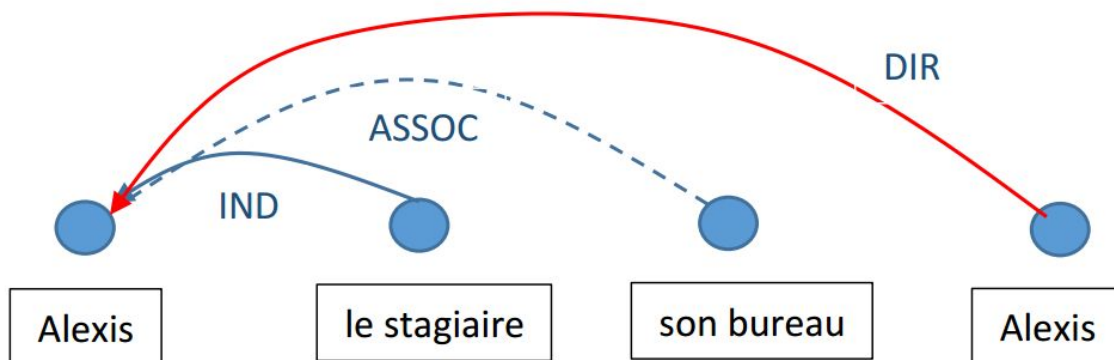
```
@RELATION coreference
@ATTRIBUTE m1_type {N, PR, UNK, NULL}
@ATTRIBUTE m2_type {N, PR, UNK, NULL}
@ATTRIBUTE m1_def {INDEF, EXPL, DEF_SPLE, DEF_DEM, UNK}
...
@ATTRIBUTE distance_word real
@ATTRIBUTE distance_char real
@ATTRIBUTE class {COREF, NOT_COREF}
@DATA

N PR INDEF INDEF M M PL PL YES NO NO NO NO NO 0.0 0.0 YES YES NULL YES YES YES 1 0 0 0 COREF
N PR DEF_SPLE INDEF M M PL PL YES NO AMOUNT NO NO NO 0.0 0.0 NO YES NULL YES YES YES 1 0 0 0 NOT_COREF
PR PR INDEF DEF_SPLE M M PL PL NO NO NO NO NO NO 0.0 0.0 NO YES NULL YES YES YES 5 0 0 0 COREF
PR PR DEF_SPLE INDEF M UNK PL UNK NO NO NO NO NO NO 0.0 0.0 NO YES NULL NO NO YES 1347 0 0 0 NOT_COREF
N PR DEF_SPLE DEF_SPLE M M PL PL YES NO NO NO NO NO 0.0 0.0 YES YES NULL YES YES YES 1 0 0 0 COREF
PR PR DEF_SPLE INDEF M M PL PL NO NO NO NO NO NO 0.0 0.0 NO YES NULL YES YES YES 1978 0 0 0 NOT_COREF
...
```

Exemple 2. Contenu d'un fichier .arff

## 2) Principe du traitement

### Conversion des types de chaînes



Si l'on reprend ce qui a été indiqué au début, nous devons aussi préciser que chaque relation (flèche) contient ses propres caractéristiques.

Ici le type a été mis en avant sur les flèches.

Pour savoir si le type doit être changé, il suffit de suivre le tableau que nous avons établi:

Par exemple pour le cas au dessus, le type actuel est celui en rouge, et celui avant est IND (Indirect), donc le dernier "Alexis" va pointer vers "le stagiaire" avec comme nouveau type: "INDIR".

TYPE	TYPE_AVANT	TYPE_CHAINE
DIR	DIR	DIR
DIR	INDIR	INDIR
DIR	PR	INDIR
INDIR	DIR	INDIR
INDIR	INDIR	TreeTagger si la chaîne n'est pas exactement la même
INDIR	PR	INDIR
PR	-	PR
PR	DIR	PR
PR	INDIR	PR
PR	PR	PR
ASSOC	-	Les associatives ne sont pas dans une chaîne coréférentielle et pointent toujours sur la première mention du référent qui permet leur interprétation. C'est ce que nous avons décidé lors du passage manuel au codage en chaîne d'Accueil-UBS par JY. Donc la relation doit rester inchangée. Avec Alexis, nous avons refait cette proposition sur le Wiki DEMOCRAT.
ASSOC_PR	-	Même remarque.

Comme le projet Ancor contient des corpus provenant de conversation entre plusieurs locuteurs, nous avons aussi la caractéristique: IDLOC qui a deux valeurs, YES ou NO. Celui-ci indique si les deux mentions d'une relation sont citées par le même locuteur ou non. On a aussi établi différentes règles:

IDLOC	IDLOC_AVANT	IDLOC_CHAINE
YES	YES	YES
YES	NO	NO
YES	UNK	UNK
NO	YES	NO
NO	NO	YES



NO	UNK	UNK
UNK	indifférent	UNK

Deux autres traits sont à recalculer pour chaque relation, l'accord en genre et l'accord en nombre. Pour les recalculer il faut donc prendre les deux mentions de la relation, et vérifier qu'ils aient le même genre et le même nombre.

Par exemple dans notre cas "Jim" (singulier, masculin) et "le stagiaire" (singulier, masculin)

Ici rien ne change mais par exemple pour "Ce bateau" (masculin, singulier) .. "Cette voile" (féminin, singulier), le trait d'accord en genre sera à NO.

## Conversion des chaînes avec apprentissage

Cette partie fait référence au changement des chaînes une fois l'apprentissage fait pour créer le fichier SYSTEM depuis les chaînes présentes dans le fichier GOLD.

Rappel des étapes pour la création des fichiers GOLD et SYSTEM au format CoNNL, Nous partons avec des relations coréférentes et nons coréférentes et nous voulons des chaînes pour le format CoNNL.

Exemple avec les mentions suivantes:

*Un chat et une souris vivaient ensemble dans un grenier en toute amitié. La souris, raisonnable et économe,[..]. proposa le chat. [..] Mais, le minou, très gourmand.*

On identifie les mentions comme suit:

- Un chat ( 1 )
- Une souris ( 2 )
- La souris ( 3 )
- le chat ( 4 )
- Le minou ( 5 )

Relation ( xMention, yMention )	GOLD	Chaîne produite pour GOLD	SYSTEM (test sur le model)	Chaîne(s) produite(s) pour SYSTEM
Relation A ( 1, 4 )	COREF	( 1, 4, 5 )	COREF	( 1, 4, 5 )
Relation B ( 4, 5 )	COREF		COREF	
Relation A ( 1, 4 )	COREF	( 1, 4, 5 )	COREF	( 1, 4 ), ( 5 )
Relation B ( 4, 5 )	COREF		NOT-COREF	
Relation C ( 1, 2 )	NOT-COREF	( 1 ), ( 2 ), ( 3 ), ( 4 )	COREF	( 1, 2 ), ( 3 ), ( 4 )
Relation D ( 4, 5 )	NOT-COREF		NOT-COREF	

--	--	--	--	--

Les chaînes sont ensuite écrites au format CoNNL 2011/2012

```
#begin document 01_09_16_12H02.txt
13684 (28)
13612 (28)
986 (28)
985 (28)
1818 (28)
1817 (28)
29567 (21)
29630 (21)
1040 (21)
1039 (21)
4918 (3)
#end document
```

Le premier nombre d'une ligne étant le numéro d'une mention et le nombre entre parenthèse est le numéro de la chaîne.

## 3) Utilisation du code:

### Packages

#### Package principal: ancortodemocrat

C'est dans ce package où sont contenues les classes pour traiter les fichiers et les différents algorithmes. Tout commence par [AncorToDemocrat](#).

#### AncorToDemocrat

Au départ le *main* configure le logger depuis le fichier de configuration.

Ensuite il configure TreeTagger et FileManager, en gardant les instances en static.

Après en fonction des arguments passés à l'application, le main choisi son chemin comme indiqué plus haut. Si aucun argument est passé alors on démarre ensuite le chargement des corpus par la lecture du fichier path.txt, puis le chargement des annotations (.aa) et des textes associés (.ac).

Et en dernier temps se lance la conversion des corpus.

#### XmlLoader

Cette classe contient une seule méthode static.

- **loadAnnotationFromFile**: Elle permet de charger un .aa depuis le chemin du fichier. Elle retourne ensuite une classe **Annotation** qui contient tous les éléments du .aa.

## XmlWriter

Même principe que précédemment mais dans l'autre sens.

- **writeXml**: Prend deux arguments, l'objet à parser dans le xml, et le nom du fichier qui sera enregistré.

## Corpus

Cette classe représente un corpus entier, et contient donc toutes les annotations de celui-ci.

- **Attributs**:
  - name: nom du corpus
  - path: chemin absolu du corpus sans le nom
- **Méthodes**:
  - getAnnotation(): Retourne la liste des annotations de ce corpus
  - loadAnnotation(): Charge les annotations depuis les fichiers .aa
  - getText( *nom* ): Retourne le texte contenu dans le fichier *nom*
  - loadText(): Charge les textes depuis les fichiers .ac
  - getPath(): Retourne le chemin absolu de ce corpus
  - setPath(): Permet de modifier le chemin absolu de ce corpus
  - setName(): Permet de modifier le nom de ce corpus
  - getName(): Retourne le nom du corpus
  - setDone( boolean ): Permet de définir le corpus comme terminé ou non, ce paramètre représente si le corpus est converti ou non
  - isDone(): Retourne donc si le corpus est converti ou non
  - export(): Enregistre les nouveaux fichiers .ac dans le dossier du corpus dans generated/

## Text

Classe contenant le contenu d'un fichier .ac mais aussi le nom du fichier.

- **Attributs**:
  - content: contenu du fichier
  - fileName: nom du fichier .ac
- **Méthodes**:
  - getContentFromUnit( Annotation, Unit ): Permet de retourner la mention de l'unité indiqué, l'annotation doit correspondre.
  - getContent(): Retourne tout le contenu du fichier .ac
  - toTrans(): Retourne l'objet de type Trans qui contient les noeuds/objets du fichier .ac, et donc qui contient les Turn
  - indexOf( Turn ): Retourne la position du premier caractère de ce tour dans le contenu du fichier.
  - contentWithoutTag( ): Retourne le contenu du fichier sans les balises XML
  - isCorresponding( Annotation, Turn, Unit ): Test et retourne la réponse si l'unité (Unit) est contenu dans le tour (Turn) précisé.
  - getTurnCorresponding( Annotation, Unit ): Retourne Le tour (Turn) qui contient l'unité passée en argument (Unit).

## ConversionInSet

Contient deux méthodes accessibles publiquement, le reste est pour la logique de ces deux précédentes.

L'une pour ajouter l'élément REF sur les chaînes depuis un texte annoté en première mention et un autre depuis un texte annoté en chaîne.

- toSetFromFirstMention( Annotation )
- toSetFromChain( Annotation )

## ConversionWorker

À besoin d'un corpus pour travailler.

Une fois instanciée, nécessite juste d'appliquer la méthode start() et un nouveau processus est crée pour convertir le corpus entièrement. Le corpus est ensuite exporté.

La conversion comprend

- le passage de première mention vers en chaîne
  - en prenant en compte l'accord des relations (genre et nombre)
  - le type des relations (directe, indirecte, anaphore, associative)
- Conversion en SET

Une fois que la conversion du corpus s'est déroulée entièrement le corpus est considéré comme réalisé grâce à un attribut done accessible via isDone().

## ConversionToArff

Cette classe s'occupe d'écrire les relations coréférentes d'un ou plusieurs corpus en ajoutant des relations non coréférentes. Tout ceci dans un fichier .arff avec les attributs au début pour respecter le format de Weka (voir Exemple 2.).

- Attributs:
  - List Corpus: Liste des corpus qui seront utilisés
  - int positif: nombre de positif à générer
  - int negatif: nombre de négatif à générer
  - List positiveRelation: Liste des relations positives
  - List negativeRelation: Liste des relations négatives
  - Param param: Si il faut extraire toutes les relations, ou exclure les associatives
  - String outpath: Chemin de sortie du fichier arff
  - int split: Quantité de sous arff à générer depuis le ou les corpus
  - String ARFF\_ATTRIBUTE: contient la liste des modèles des attributs sous la forme de Weka.
- Méthodes:
  - Pour une relation donnée, écrit dans un string la liste des valeurs de ses traits, et renvoie cette ligne.
  - work(): Démarre les étapes pour écrire le fichier .arff en fonction des paramètres
  - generateNegativeRelation( Corpus, Annotation, Relation, PrintWriter): génère entre une à trois relation négative en fonction de la relation passée, puis écrit celle(s)-ci sur le fichier .arff.

## FileManager

Cette classe contient des outils pour gérer les fichiers, une instance static est disponible de AncorToDemocrat.fileManager.

Elle est instanciée au début pour vérifier si le fichier path.txt existe bien, sinon il est généré.

Même chose pour le dossier generated/

La classe contient aussi différents éléments:

- getFileFromFolder( cheminDuDossier )
  - Retourne la liste des noms des fichiers dans un dossier
- loadPathFile()
  - Retourne la liste des corpus contenu dans le fichier *path.txt*
- mkdir( cheminDuDossier )
  - créer un dossier si il n'existe pas
- loadAaFile( Corpus )
  - Retourne une liste des noms des fichiers .aa d'un corpus
- loadAcFile( Corpus )
  - Retourne une liste des noms des fichiers .ac d'un corpus

## Second package:

element ET feature ET positionning

Dans ces trois paquets sont contenus les objets nécessaires pour charger les .aa. Chaque classe représente un noeud, un type contenu dans les .aa.

Si on prend la DTD de Glozz on retrouve les classes présentes dans ces packets.

J'ai volontairement omis d'ajouter certaines classes, car elles représentent seulement des types présents dans les .aa et n'ont pas de fonction de traitement. Seulement des getters et des setters, donc compréhensible sans explications.

```

1  <?xml version="1.0" ?>
2  <annotations>
3    <metadata corpusHashCode="787-51594840"/>
4    <unit id="agoudjo_1331895064938">
5      <metadata>
6        <author>agoudjo</author>
7        <creation-date>1331895064938</creation-date>
8        <lastModifier>jmuzerelle</lastModifier>
9        <lastModificationDate>1332410380589</lastModificationDate>
10     </metadata>
11     <characterisation>
12       <type>N</type>
13       <featureSet>
14         <feature name="NEW">YES</feature>
15         <feature name="GEN_REF">SPEC</feature>
16         <feature name="EN">PERS</feature>
17         <feature name="DEF">DEF_SPLE</feature>
18         <feature name="GP">NO</feature>
19         <feature name="GENRE">UNK</feature>
20         <feature name="NB">SG</feature>
21       </featureSet>
22     </characterisation>
23     <positioning>
24       <start>
25         <singlePosition index="635"/>
26       </start>
27       <end>
28         <singlePosition index="647"/>
29       </end>
30     </positioning>
31   </unit>
32   <relation id="jmuzerelle_1378375156971">
33     <metadata>
34       <author>jmuzerelle</author>
35       <creation-date>1378375156971</creation-date>
36       <lastModifier>n/a</lastModifier>
37       <lastModificationDate>0</lastModificationDate>
38     </metadata>
39     <characterisation>
40       <type>ANAPHORE</type>
41       <featureSet>
42         <feature name="ID_LOC">YES</feature>
43         <feature name="NOMBRE">YES</feature>
44         <feature name="GENRE">YES</feature>
45       </featureSet>
46     </characterisation>
47     <positioning>
48       <term id="jmuzerelle_1378374445856"/>
49       <term id="jmuzerelle_1378374358776"/>
50     </positioning>
51   </relation>
52   <schema id="jmuzerelle_1378374374106">
53     <metadata>
54       <author>jmuzerelle</author>
55       <creation-date>1378374374106</creation-date>
56       <lastModifier>n/a</lastModifier>
57       <lastModificationDate>0</lastModificationDate>
58     </metadata>
59     <characterisation>
60       <type>default</type>
61       <featureSet>
62         <feature name="default">default</feature>
63       </featureSet>
64     </characterisation>
65     <positioning>
66       <embedded-unit id="jmuzerelle_1378374372556"/>
67       <embedded-unit id="jmuzerelle_1378374365936"/>
68     </positioning>
69   </schema>
70 </annotations>

```

Exemple de fichier .aa

## Annotation

- Attributs:
  - MetadataAnnotation: class contenant le hashCode du corpus
  - List unit: Liste des unités du fichier.
  - List relation: Liste des relations du fichier.
  - List schema: Liste des schéma du fichier.
  - String fileName: Nom du fichier.
- Méthodes:
  - getFileName( )
  - setFileName( String )
  - getSchema( )
  - setSchema( List )
  - getMetadata( )
  - setMetadata( MetadataAnnotation )
  - getUnit( )
  - setUnit( List )
  - getRelation( )
  - setRelation( List )
  - getElementById( String id ): Retourne l'élément correspond à cet id.
  - removeTxtImpoter( ): Supprimer les unités qui contiennent comme id "TXT\_IMPORTER"
  - getRelationContaining( Unit ): Retourne la ou les relation(s) qui ont l'unité passée en argument en commun.

## Element

L'élément est la classe parent de Unit et Schema car ils ont des attributs en commun

- Attributs:
  - Characterisation
  - String id
- Méthodes:
  - getFeature( featureName): Retourne la feature en fonction du nom passé en argument, si la feature n'existe pas, retourne un String("NULL")
  - setFeature( featureName, value ): Ajoute une nouvelle feature à l'élément, avec sa valeur, si la feature existe déjà, alors il y'a juste la valeur qui est changée.

## Unit **extends** Element

- Attributs
- Méthodes:
  - isNew( Annotation ): Test si l'unité contient la feature NEW à YES
  - getStart( Annotation ): Retourne la première position du caractère
  - getEnd( Annotation ): Retourne la position du dernier caractère de l'unité.
  - isContainedInSchema( Annotation ): Test si l'unité est contenu dans un schéma ou non.

- isAssociative( Annotation ): Test si une relation est liée à cette unité et si cette relation est associative.

## Schema **extends** Unit

Certaines des fonctions présentes dans cette classe, sont présentes dans Unit, donc l'explication est la même, il y a juste la logique qui est différente.

- Attributs:
- Méthodes:
  - getUnitList( Annotation ): Retourne la liste des unités associées à ce schéma.
  - getUnitWhereFeatureNotNull( Annotation ): Retourne l'unité qui a ses features non null. Peut retourner null, si toutes les unités de ce schéma ont des features à null.

## Relation

Méthodes:

- newInstance( Relation): Retourne une copie de la Relation passée en argument
- getMetadata(): Retourne les métas (données) de la relation
- setMetadata(Metadata ): Change les Metadatas de la relation
- getCharacterisation(): Retourne la Characterisation de la relation
- setCharacterisation( Characterisation ): Change la Characterisation de la Relation
- getElement( Annotation ): Retourne l'élément qui se trouve à droite dans une relation (e.g: "Le petit bateau .. Il .." ici l'élément sera "Il"). Si la relation ne pointe pas sur un Schema ou une unité, alors elle renvoie null.
- getPrelement( Annotation ): Retourne le premier élément de la relation, depuis l'exemple précédent, le résultat sera "Le petit bateau". Si la relation ne pointe pas sur un Schema ou une unité, alors elle renvoie null.
- getPreRelation( Annotation ): Retourne la relation se trouvant juste avant celle-ci: (e.g: "Le chat .. Il .. Cette boule de poils", si on est sur la relation entre "Le chat" et "Cette boule de poils" alors la fonction retourne la relation entre "Le chat" et "Il". Fonctionne seulement en première mention, peut être adapté pour fonctionner avec des relations en chaîne.
- containsUnit( Annotation, Unit ): Test si l'unité passé en argument est contenue dans la relation, donc soit l'élément ou le pré-élément.
- getFeature( String ): Retourne la feature en fonction de son nom passé en argument, si la feature n'existe pas, alors renvoie un String("NULL")
- getOtherElement( Annotation, Element ): Retourne l'autre élément de la relation, (e.g: "Ce chaton ... Ce minou", si on passe en argument l'élément correspondant à "Ce minou" alors la fonction retourne l'élément correspondant à "Ce chaton").
- containsNew( Annotation ): Test si la relation contient un élément qui a la feature NEW à YES.

## MetadataAnnotation

- Attributs:
  - String corpusHashCode: valeur du hash code du corpus.
- Méthodes:
  - getCorpusHashCode( )
  - setCorpusHashCode( String )



## MetaDataUnit

- Attributs:
  - String author: Auteur de l'unité.
  - long createDate: Date de la creation de l'unité.
  - String lastModifier: id du dernier Auteur qui a modifié l'unité.
  - long modificationDate: Date de la dernière modification de la date.
- Méthodes:
  - getters et setters de chaque attribut.

## Characterisation

- Attributs:
  - Type: Classe contenu le type de la caracterisation.
  - FeatureSet: Classe contenant la liste des features/traits.
- Méthodes:
  - getType( )
  - setType( Type )
  - getFeatureSet( )
  - setFeatureSet( FeatureSet )

## Start & End

Ces classes contiennent respectivement, la position de début et de fin d'un élément.

- Attributs:
  - SinglePosition: classe contenant la valeur de la position.
- Méthodes:
  - getSinglePosition( )
  - setSinglePosition( SinglePosition )

## SinglePosition

- Attributs:
  - int index: Valeur de la position
- Méthodes:
  - getIndex( )
  - setIndex( int )

## PositionningUnit

- Attributs:
  - Start: Classe contenant la position du début de l'unité
  - End: Classe contenant la position de fin de l'unité.
- Méthodes:
  - getters et setters

## PositioningRelation

- Attributs:
  - List term: Liste des unités liées à cette relation

- Méthodes:
  - getters et setters

## Term

Cette classe contient une unité liée à une relation.

- Attributs:
  - String id
- Méthodes:
  - getId( )
  - setId( String )
  - getElement( Annotation ): Retourne l'élément attaché à cette classe.

## CalculateFeature

Classe permettant de calculer les nouveaux traits des relations, unités.

- Attributs:
  - Corpus: Corpus sur lequel les nouveaux traits seront calculés.
- Méthodes:
  - work( ): Méthode qui calcule les nouveaux traits de toutes les relations
  - calculateFeatureOnRelation( Annotation, Relation ): Calcule les features/traits d'une relation
  - calculateFeatureOnRelation( Annotation ): Calcule les nouveaux features/traits des relations d'une Annotation.
  - calculateNewFeature: Calcule les nouveaux traits tels que speaker, m1\_next, m2\_next et m1\_previous, m2\_previous.
  - calculatePreviousNextToken(Annotation, Relation ): Calcule les traits m1\_next, m2\_next et m1\_previous, m2\_previous.
  - calculateSpeaker( Annotation ): Calcule sur les unités quel est son locuteur.
  - calculateToken( Annotation ): Ajoute à chaque unité le mot suivant et précédent à la mention
  - getNextToken( Annotation, Text, Unit ): Retourne le mot suivant à la mention de l'unité
  - getPreToken( Annotation, Text, Unit ): Retourne le mot précédent à la mention de l'unité.

## Package treetagger

Ce package contient les outils pour travailler avec TreeTragger et aussi donc pour convertir le cas de (NO, IND, IND) → ?

Pour lancer la conversion entre deux séquences où le doute est permis et où TreeTagger doit intervenir, on doit instancier TokenConvertRelationHandler( Relation, premièreSequence, secondeSequence ).

La classe s'occupe elle-même de convertir la relation en fonction des deux séquences.

Un cas d'exemple d'utilisation de cette classe est présent dans ConversionWorker, ligne 217.

Une fois le handler terminé, on peut vérifier avec isDone si le travail est réalisé.

Dans le cas présent dans ConversionWorker, j'ai décidé d'attendre la fin de la tâche avant de passer à la relation suivante un accès concurrentiel à cette ressource de relation.

## ResultMention

Cette classe contient le résultat d'une séquence analysé par TreeTagger, la séquence est découpée avec une liste de ResultToken.

Méthodes:

- newToken( ResultToken ): Ajoute à la liste un nouveau token
- containsNoun(): Test si parmi la liste des token, au moins un est un nom ou non
- getNounList(): Retourne la liste des noms contenu dans la liste, peut retourner une liste vide

## ResultToken

Contient le résultat pour un mot (token) de TreeTagger. C'est à dire le type (Nom, déterminant, verbe, proposition, ..) et son lemma (ex: chevaux → cheval, seras → être), et le mot (token) lui même.

Méthodes:

- getToken(): Récupère le token (mot)
- getPos() Retourne le type du token
- getLemma(): Retourne le lemma du token
- isNoun(): Retourne vrai si le token est un nom, sinon faux.

## TokenConvertMentionHandler

Handler faisant appel à TreeTagger pour convertir une séquence.

Contient un attribut done pour savoir si TreeTagger a répondu le résultat (qui est donc contenu dans ResultMention) ou non.

- Attributs:
  - ResultMention: Représente la mention qui sera convertie à la fin de l'appel de TreeTagger
  - boolean done: état dans lequel est la classe, si elle a eu la réponse de TreeTagger au complet ou non.
  - String[] mentionSplitted: Contient les éléments de la mention à passer au TreeTagger pour connaître leur type et leur lemma.
- Méthodes:
  - token( String token, String pos, String lemma): appelé par TreeTagger pour chaque élément de la mention.
  - isDone() Retourne l'état de la classe.
  - splitMention( String sentence ): Découpe la phrase en tableau pour identifier chaque mot, ponctuation.

## TokenConvertRelationHandler

Cette classe utilise un processus pour faire sa tâche. Elle s'occupe de convertir le type d'une relation. Elle fait appel à deux TokenConvertMentionHandler pour comparer les deux chaînes et savoir si les deux séquences sont directes entre elles ou non.

- Attributs:
  - Relation: La relation sur laquelle la classe travail pour obtenir deux ResultsMention
  - firstMention: Première mention de la relation

- secondMention: Seconde mention de la relation
- boolean done: État de la classe pour savoir si elle a fini ou non
- Méthodes:
  - run(): Méthode pour travailler sur les données (en Thread)

## TreeTagger

Configure TreeTagger pour être utilisée plus tard, grâce à la méthode work().

- Attributs:
  - String pathTreeTagger: Nom du dossier de TreeTagger
- Méthodes:
  - work( TokenConvertMentionHandler ): Fait appel à TreeTagger pour la mention passée.

## Package Speech

Ce packet comprend les objets pour parser les fichiers .ac

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE Trans SYSTEM "trans-13.dtd">
3  <Trans scribe="Foulon J" audio_filename="052_00000035" version="1" version_date="041124">
4    <Speakers>
5      <Speaker id="spk1" name="hotesse" check="no" dialect="native" accent="" scope="local"/>
6      <Speaker id="spk2" name="client" check="no" dialect="native" accent="" scope="local"/>
7    </Speakers>
8    <Episode>
9      <Section type="report" startTime="0" endTime="13.968">
10     <Turn startTime="0" endTime="1.868" speaker="spk1">
11       <Sync time="0"/>U B S Bonjour
12     </Turn>
13     <Turn speaker="spk2" startTime="1.868" endTime="9.786">
14       <Sync time="1.868"/>bonjour madame excusez-moi de vous déranger j'ai appelé tout à
15       l'heure pour savoir si on pouvait [pf] venir récupérer les diplômes de licence
16     </Turn>
17     <Turn speaker="spk1 spk2" startTime="9.786" endTime="12.1">
18       <Sync time="9.786"/>
19       <Who nb="1"/># oui alors attendez je vais vous
20       <Who nb="2"/>et
21     </Turn>
22     <Turn speaker="spk1" startTime="12.1" endTime="12.546">
23       <Sync time="12.1"/>passer la
24     </Turn>
25     <Turn speaker="spk1 spk2" startTime="12.546" endTime="13.968">
26       <Sync time="12.546"/>
27       <Who nb="1"/>personne
28       <Who nb="2"/>ouais [rire]
29     </Turn>
30   </Section>
31 </Episode>
32 </Trans>

```

### Exemple de fichier .ac

## Trans

- Attributs:
  - Episode
- Méthodes:
  - getEpisode()
  - setEpisode( Episode )

## Episode

- Attributs
  - Section
- Méthodes:
  - getSection( )
  - setSection( Section )

## Section

- Attributs:
  - List turnList: Liste des tours de parole.
- Méthodes:
  - getTurn( )
  - setTurn( Turn )

## Turn

- Attributs:
  - String speaker: id du locuteur.
  - float startTime: date de début du tour de parole (exprimés en secondes depuis le début de la conversation.
  - float endTime: date de fin du tour de parole.
  - List text: Liste du texte contenu dans le tour de parole, souvent le texte est séparé avec des <Who>.
- Méthodes:
  - getSpeaker( )
  - setSpeaker( String )
  - getStartTime( )
  - setStartTime( float )
  - getEndTime( )
  - setEndTime( float )
  - getText( ): Retourne le contenu du texte du tour de parole en liste, directement parser depuis le XML.
  - setText( List ): change le contenu du tour de parole
  - getContent( ): Retourne le contenu du tour de parole sous un seul String.

## Package Classification

### Chain

Classe permettant de représenter une chaîne

- Attributs
  - ref: champ référent de la chaîne
  - mentionList: Liste des mentions de la chaîne
- Méthodes
  - getRef()

- `getMentionList()`
- `addMention( Mention mention )` Ajoute une mention si elle n'est pas déjà présente
- `containsMention( Mention mention )` Test si la mention est présente ou non dans la chaîne en comparant l'id des mentions
- `containsMention( idMention )`
- `getMention ( idMention )`
- `removeMention( idMention )`
- `size( )` Retourne le nombre de mention présente dans la chaîne

## Mention

- Attributs

- `id`
- `boolean coref`: Si la mention est contenu dans une relation COREF
- `boolean corefSet`: Si l'attribut précédent a été mis ou non

Les deux derniers attributs permettent de savoir après apprentissage si une mention doit être supprimée ou non d'une chaîne

Par exemple prenons une chaîne qui contenait trois mentions avant apprentissage:

1(T, F) - 2(T, F) - 4(T, F)

où T: True, F: False et respectivement le premier est coref et le second est corefSet.

Et supposons qu'après apprentissage les mentions 2 et 4 ne soient plus coréférentes, mais 1 et 2 le soient toujours.

on aura donc:

1(T, T) - 2(T OR F, T)

et

2(T OR F, T) - 4(F, T)

Sachant que l'apprentissage est fait avec les relations.

On applique donc la logique du OR et on maintenant les chaînes suivantes:

( 1, 2 ), ( 4 )

- Méthodes

- `getId()`
- `setId ( id )`
- `isCoref( )`
- `setCoref( coref )` Si coref non défini on met la valeur et on change corefSet à true, sinon on fait un OU logique entre le coref actuel et la nouvelle valeur.

## Model

- Attributs

- `classifier`: classifier issu de Weka
- `path`: chemin du model

- Méthodes

- `loadModel( )`
  - `arffFile` données d'apprentissage
  - `classifier` type d'apprentissage à faire
- `loadModel( )`
  - `modelFile` chemin du fichier à charger
- `getPath( )`

- `setPath( )`
- `classifyInstance( )` Classifie une liste d'instance (trouve la classe pour chaque instance)
  - `unlabeled`: instance à classifier
- `export( )`
  - `fileName` chemin de sortie
- `crossValidate( )` Apprentissage par validation croisée
  - Instances données d'apprentissage
  - `nbFolds`

## Scorer

Ce fichier contient toutes les méthodes pour faire appel au scorer, il sert pour la command "scorer"

- Méthodes
  - `scorerTask( )`
    - `Command String` de la commande exécutée pour cette fonction
    - `corpusList` liste des corpus où sont extrait les relations
    - `modelPath` chemin du fichier model à appliquer pour faire le test
    - `positif` nombre d'instance positive
    - `negatif` nombre d'instance négative
    - `param` Quel type de relation l'on doit prendre (ALL, NO\_ASSOC)
    - `outputPath` Chemin de sortie des résultats
    - `split` Si on doit splter le/les corpus indiqué(s), sinon 0 et aucun split est appliqué
    - `listRemoveAttribute` Liste des attributs à ignorer pour l'apprentissage
  - `createGoldSet( )` Créer un ensemble de chaîne pour chaque fichier passé qui correspond au jeu de donnée du fichier GOLD.
    - `perFile` Tableau des listes des chaînes qui sont à créer par fichier
    - `positiveRelationSelected` Liste des relations positives pour tous les fichiers
    - `negativeRelationSelected` Liste des relations négatives pour tous les fichiers
    - `split` En combien de partie le/les fichier(s) sont découpés
    - `fileArff` Liste des fichiers
    - `lastChainSingleton` ID des nouvelles chaînes pour les relations négatives
  - `createSystemSet( )` Modifie l'ensemble des chaînes passées en fonction de l'apprentissage des données sur le model
    - `perFile` Tableau des listes des chaînes qui sont à modifier en fonction des résultats de l'apprentissage avec le model
    - `positiveRelationSelected` Liste des relations positives pour tous les fichiers
    - `negativeRelationSelected` Liste des relations négatives pour tous les fichiers
    - `split` En combien de partie les fichiers ont été découpé
    - `fileArff` Les des fichiers
    - `lastChainSingleton` ID des nouvelles chaînes pour les relations négatives
    - `model` Model qui servira d'apprentissage pour classer les relations

- removeAttribute Liste des traits qui ne sont pas à prendre en compte pour l'apprentissage
- writeCoNNL( ) Écrit le/les fichiers CoNNI depuis la liste donnée avec les chaînes correspondantes.
  - fileArff Liste des fichiers qui sont à créer
  - chainListPerFile[] Liste des chaînes par fichier
  - outputPath Chemin de sortie
  - fileName extension du nom des fichiers (GOLD ou SYSTEM par exemple)
- removeChainFromList( ) Supprime une chain dans le liste donnée
  - chainList: liste des chaînes
  - ref: champ référent de la chaîne à enlever de la liste
- containsChain( ) Vérifie si une chaîne est contenu dans une liste ou non, avec son champ référent
  - chainList: liste des chaînes
  - ref: champ référent de la chaîne à chercher
- getChainFromList( ) Retourne la chaîne trouvée dans la liste en cherchant par son champ référent
  - chainList: liste des chaînes
  - ref: champ référent de la chaîne à chercher
- setRefIfNeed( ) Ajoute le champ référent si il n'est pas présent sur le/les corpus
  - corpusList: Liste des corpus
- eval() Evalue le jeu de donnée
  - metric: muc OU B3 ou ceaf
  - trueFile: correspond au fichier GOLD
  - systemFile: correspond au fichier SYSTEM, réponse de l'apprentissage depuis le fichier GOLD
- loadInstance( ) Charge les instances depuis un fichier arff
  - arffFile: Chemin du fichier arff

## Liste des traits

Les traits en gris sont ceux déjà présent dans le corpus ANCOR, les autres sont donc les traits calculés automatiquement.

	Traits	Définition	Valeurs possibles
1	m1_type	Catégorie syntaxique de m1	{N, PR, UNK, NULL}
2	m2_type	Catégorie syntaxique de m2	{N, PR, UNK, NULL}
3	m1_def	Détermination de définition de m1	{INDEF, EXPL, DEF_SPLE, DEF_DEM, UNK}
4	m2_def	Détermination de définition de m2	{INDEF, EXPL, DEF_SPLE, DEF_DEM, UNK}
5	m1_genre	genre de m1	{M, F, UNK, NULL}
6	m2_genre	genre de m2	{M, F, UNK, NULL}



7	m1_nombre	nombre de m1	{SG, PL, UNK, NULL}
8	m2_nombre	nombre de m2	{SG, PL, UNK, NULL}
9	m1_new	Si m1 introduit une nouvelle entité, ou une associative.	{YES, NO, UNK, NULL}
10	m2_new	Si m2 introduit une nouvelle entité, ou une associative.	{YES, NO, UNK, NULL}
11	m1_en	Type d'entité de m1	{PERS, FONC, LOC, ORG, PROD, TIME, AMOUNT, EVENT, NO, UNK, NULL}
12	m2_en	Type d'entité de m2	{PERS, FONC, LOC, ORG, PROD, TIME, AMOUNT, EVENT, NO, UNK, NULL}
13	ID_form	Si les deux mentions sont les mêmes tokens.	{YES, NO, NA}
14	ID_subform	Si un token de la mention la plus petite est contenu dans la plus grande.	{YES, NO, NA}
15	INCL_RATE	Taux d'inclusion de la plus petite mention dans la plus grande.	REAL
16	COM_RATE	Nombre de token en commun entre les deux mentions / nombre de token de la plus grande mention.	REAL
17	ID_DEF	Si les deux mentions ont la même DEF.	{YES, NO, NA}
18	ID_TYPE	Si les deux mentions ont le même type.	{YES, NO, NA}
19	ID_EN	Si les deux mentions ont la même fonction (EN).	{YES, NO, NA}
20	ID_GENRE	Si les deux mentions ont le même genre	{YES, NO, NA}
21	ID_NOMBRE	Si les deux mentions contiennent la même forme de quantité.	{YES, NO, NA}
22	DISTANCE_MENTION	Distance en nombre de mention entre ces deux mentions. (Pour la même relation).fquantité	REAL
23	DISTANCE_TURN	Distance en nombre de tour de parole entre ces mentions.	REAL
24	DISTANCE_WORD	Nombre de mot entre les deux mentions.	REAL
25	DISTANCE_CHAR	Nombre de caractère entre les deux mentions.	REAL
26	EMBEDDED	Si m2 est contenu dans m1	{YES, NO, NA}
27	ID_PREVIOUS	Si les deux tokens précédents les mentions sont semblables.	{YES, NO, NA}
28	ID_NEXT	Si les deux tokens suivants les mentions sont semblables.	{YES, NO, NA}

29	ID_SPK	Si les locuteurs sont les mêmes pour les deux mentions.	{YES, NO, NA}
30	ID_NEW	SI les deux mentions contiennent le trait NEW à vrai.	{YES, NO, NA}

**Liste des nouveaux traits calculés automatiquement**