Présentation MACAON

Alexis Nasr
Franck Dary

M2 - IAAA

17 octobre 2019

Table of contents

Multi Column File (mcf)

Format de fichier qui permet de représenter les données textuelles et les annotations qui y sont attachées.

- format "en colonne" :
  - chaque ligne correspond à une unité textuelle minimale (token)
  - chaque colonne correspond à un attribut du token
- les colonnes sont séparées les unes des autres par une tabulation
- le nombre de colonnes est illimité
- l'interprétation de chaque colonne est décrite dans un fichier mcd (Multi Column Description)
- les lignes commençant par "#" sont des commentaires (potentiellement des métadonnées)

mcf : exemple

```
La          det     le          1    det         0
diane       nc      diane       1    suj         0
chantait    v       chanter     0    root        0
dans        prep    dans       -1    mod         0
la          det     le          1    det         0
cour        nc      cour       -2    obj         0
des         prep    des        -1    dep         0
casernes    nc      caserne    -1    obj         0
.           poncts  .          -6    eos         1
Et          coo     et          0    root        0
le          det     le          1    det         0
vent        nc      vent        3    suj         0
du          prep    du         -1    dep         0
matin       nc      matin      -1    obj         0
soufflait   v       souffler   -5    dep_coord   0
sur         prep    sur        -1    mod         0
les         det     le          1    det         0
lanternes   nc      lanterne   -2    obj         0
.           poncts  .          -9    eos         1
```

Multi Column Description (mcd)

Un fichier mcd associe une étiquette à chaque colonne d'un fichier mcf, exemple mcd du format conllu :

```
0 ID
1 FORM
2 LEMMA
3 POS
4 EMPTY
5 MORPHO
6 GOV
7 LABEL
8 EMPTY
9 EMPTY
```

Etiquettes

- Une étiquette permet d'associer un nom explicite à une colonne
- Elles permettent d'accéder au contenu de la colonne à l'aide de *Word Features* décrites dans un fichier `fm`.
- Exemples d'étiquettes :
  - `ID` indice du token dans sa phrase
  - `FORM` la forme du token
  - `POS` sa partie de discours
  - `LEMMA` son lemme
  - `MORPHO` traits morphologiques
  - `GOV` l'indice de son gouverneur
  - `LABEL` sa fonction syntaxique
  - `EMPTY` colonne vide

Features

Fonctions permettant de représenter différents aspects d'une configuration

Il existe plusieurs types de features, pour les construire, on utilise des **fonctions d'adresse** et des **fonctions d'attribut**.
Toutes les Features sont codées dans le fichier
`transition_machine/src/FeatureBank.cpp`

Fonction d'adresse

Une fonction d'adresse indique à quel token on fait référence dans une configuration. Elle se compose de :

- Un objet cible b(buffer) ou s(stack)
- Un indice relatif dans l'objet (ex : 2 ou -3)
- un certain nombre (voir 0) de ces fonctions qui vont se combiner entres elles :

ldep	dépendent le plus à gauche
rdep	dépendent le plus à droite
l2dep	dépendent le 2ème plus à gauche
r2dep	dépendent le 2ème plus à droite
gov	gouverneur

Exemple : s.2.gov.l2dep qui désigne le deuxième dépendent le plus à gauche du gouverneur du token en position 2 dans la pile.

Fonction d'attribut

Une fonction d'attribut permet d'extraire une certaine information d'un token. Elle se compose d'une combinaison des éléments suivants :

Étiquette	une colonne du `mcd`
U	L'étiquette commence-t-elle par une majuscule ?
fasttext	découpe l'étiquette en paquets de lettres
nbr	nombre de dépendants droits
nbl	nombre de dépendants gauches

Exemple : FORM.U est-ce que la forme du token commence par une majuscule ?

Features composées

La majorités des features sont la concaténation (avec "#") d'une fonction d'adresse et d'une fonction d'attribut, exemples :

- s.1.ldep.ldep#LABEL La fonction syntaxique du dépendant le plus à gauche du dépendant le plus à gauche du token en position 1 de la pile.
- b.-3#LEMMA.fasttext La décomposition en paquets de lettres du lemme du token en position -3 dans le buffer.
- s.0.gov#nbr Le nombre de dépendants droits du gouverneur du token en sommet de pile.

Features spéciales

Il existe 2 autres types de features :

- tc.X la Xème transition la plus récente dans l'historique des transitions qui ont menées à cette configuration. (X commence à 0)
- X#DIST.Y Distance entre le token X et le token Y dans la phrase. Exemple : b.0#DIST.s.0

Feature Model (`fm`)

- Un fichier `fm` définit une collection de Features qui vont permettre de décrire une configuration
- Ces Features sont utilisées par le classifieur pour prédire le prochain mouvement
- Chaque ligne d'un fichier `fm` définit une Feature

fm : Exemple

```
b.1#POS
b.1#MORPHO
b.2#POS
b.2#MORPHO
b.-2#LABEL
b.-1#LABEL
b.0#LABEL
s.0#LABEL
s.1#LABEL
b.-1.gov#POS
b.-1.gov#MORPHO
s.0.gov#POS
s.0.gov#MORPHO
s.1.gov#POS
s.1.gov#MORPHO
```

dicts

Fichier qui définit des dictionnaires qui vont contenir les paires *feature value* $\rightarrow$ *embedding*, exemple :

```
#Name           Dimension Mode       PretrainedFilename #
#################################################################
Parser_actions 18        Embeddings _
Parser_bool    2         OneHot     _
Parser_int     16        Embeddings _
Parser_form    100       Embeddings _
```

- Name : nom du dictionnaire
- Dimension : taille des embeddings
- Mode : Embeddings ou OneHot
- PretrainedFilename : chemin vers un dictionnaire déjà complet (pretraining des embeddings)

cla

Type de fichier qui permet de décrire un classifieur (qui traite l'information à un certain niveau) exemple : parser, tagger, lemmatizer...

```
Name : Parser
Type : Prediction
Oracle : parser
Feature Model : parser.fm
Action Set : parser.as
Topology : (500,RELU,0.3)
Dynamic : yes
```

Ici Topology représente un MultiLayerPerceptron d'une couche cachée de taille 500 de fonction d'activation REctifiedLinearUnit et de 30% de dropout.

bd

Type de fichier qui permet de décrire le buffer de la machine.
Chaque ligne du fichier bd décrit une bande du buffer, exemple :

```
#Index Name   ref/hyp dict    Policy   Must print?#
##############################################
0      ID     ref     none    FromZero 1
1      FORM   ref     form    FromZero 1
3      POS    ref     pos     FromZero 1
5      MORPHO ref     morpho  FromZero 1
0      SGN    hyp     sgn     FromZero 0
0      EOS    hyp     eos     FromZero 0
```

1. Index : Indice de la colonne du format de sortie dans laquelle le contenu de cette bande sera écrit.
2. Name : Nom de cette bande.
3. ref/hyp : Le contenu de cette bande est donné (ref) ou prédit (hyp).
4. dict : Le nom du dictionnaire qui contient les valeurs de cette bande.
5. Policy : Le dictionnaire est figé (Final) ou peut évoluer (FromZero) ?
6. Must print ? : Faut-il afficher cette bande dans la sortie ?

tm

Type de fichier qui décrit la Transition Machine.
Une telle machine ressemble à un automate, elle comporte des états et des transitions étiquetées qui relient ces états entre eux.
A chaque état est associé un classifieur. Exemple :

```
Name : Parser Machine
Dicts : parser.dicts
%CLASSIFIERS
strategy strategy.cla
signature signature.cla
parser parser.cla
%STATES
strategy strategy
signature signature
parser parser
%TRANSITIONS
strategy signature MOVE signature
strategy parser MOVE parser
parser strategy *
signature strategy *
```

tm

1. Name : Le nom de la machine.
2. Dicts : Le fichier qui décrit les dictionnaires.
3. CLASSIFIERS : Sur chaque ligne le nom d'un classifieur et le fichier qui le décrit.
4. STATES : Sur chaque ligne le nom d'un état de la machine et le nom du classifieur correspondant.
5. TRANSITIONS : sur chaque ligne on a un état de départ, un état d'arrivée, et l'étiquette sur la transition.

Dans l'exemple donné plus haut, l'état parser s'occupe de construire l'arbre syntaxique, l'état signature appose une signature sur chaque mot (qui comporte des informations sur les parties de discours possibles pour ce mot) et l'état strategy s'occupe de décider à qui il donne la main à un instant t (à parser ou à signature).

Toujours dans le même exemple, l'état initial est strategy car il est définit le premier.

1. On peut passer de l'état strategy à l'état signature avec l'action MOVE signature.
2. On peut passer de l'état strategy à l'état parser avec l'action MOVE parser.
3. On passe de l'état parser à l'état strategy avec n'importe quelle action.
4. On passe de l'état signature à l'état strategy avec n'importe quelle action.

as

Type de fichier qui définit la liste des actions que peut effectuer un classifieur. Exemple :

```
REDUCE
LEFT ccomp
RIGHT ccomp
LEFT dep
RIGHT dep
LEFT cop
RIGHT cop
LEFT conj
RIGHT conj
LEFT iobj
RIGHT iobj
EOS
Default : SHIFT
```

L'action préfixée par Default : est celle utilisée par defaut quand aucune autre action n'est applicable. Les actions sont ordonnées par priorité décroissante, lorsque 2 actions sont possibles on préfère celle de plus grande priorité.

Dépots GIT

Le projet macaon est réparti sur deux dépôts GIT :

- `macaon` contient le code des différents modules
  `https://gitlab.lis-lab.fr/franck.dary/macaon`
- `macaon_data` contient les données primaires ainsi que les procédures de compilation des modèles à partir des données primaires
  `https://gitlab.lis-lab.fr/franck.dary/macaon_data`

macaon

macaon dépend des bibliothèques :

- dynet
- fasttext
- boost_program_options

Et des logiciels g++, CMake, python3

installer dynet

```
mkdir eigen3 && cd eigen3
git clone https://github.com/eigenteam/eigen-git-mirror.git .
```

https://dynet.readthedocs.io/en/latest/install.html#

Installer fasttext

```
mkdir fasttext && cd fasttext
git clone https://github.com/facebookresearch/fastText.git .
mkdir build && cd build
cmake .. && make -j && sudo make install
```

Installer boost_program_options

Télécharger boost `https://dl.bintray.com/boostorg/release/1.71.0/source/boost_1_71_0.tar.gz`

```
tar -xvf boost_1_71_0.tar.gz
cd boost_1_71
./bootstrap.sh --with-libraries=program_options
sudo ./b2 install
```

Installer macaon

```
cd macaon
mkdir build
cd build
cmake .. && make -j && sudo make install
```

Télécharger les corpus

TODO

Entraîner un outil

Une fois macaon_data présent sur l'ordinateur, on ajoute la ligne suivante au .bashrc

```
export MACAON_DIR=chemin/vers/macaon_data
```

Ensuite on se rend dans un dossier où les outils sont déjà configurés :

```
cd macaon_data/UD_any/
```

On indique où se trouvent nos données (corpus) : Dans le fichier config renseigner le dossier où se trouvent les corpus.

On extrait certaines informations des corpus (à ne faire qu'une fois) :

```
cd data
make
cd ..
```

Entraîner un outil

Toujours dans le dossier macaon_data/UD_any/ on lance :

```
./train.sh NOMLANGUE parser parser_maLangue
```

Qui va entraîner un parser nommé “parser_maLangue”, en utilisant les données issues des corpus de NOMLANGUE.

Par exemple pour entraînter un parser sur le français on peut faire :

```
./train.sh fr parser parser_fr
```

On peut ajouter des arguments à la commande, par exemple pour le mode debug :

```
./train.sh fr parser parser_fr --debug
```

Pour voir la liste de tous les arguments possibles :

```
macaon_train --help
```

Entraîner un outil

Une fois l'entraînement terminé, le modèle sera automatiquement évalué sur le corpus de test, le résultat sera affiché comme ça :

Metric	Precision	Recall	F1 Score	AligndAcc
Tokens	100.00	100.00	100.00	
Sentences	98.48	98.01	98.25	
Words	100.00	100.00	100.00	
UPOS	100.00	100.00	100.00	100.00
XPOS	100.00	100.00	100.00	100.00
UFeats	100.00	100.00	100.00	100.00
AllTags	100.00	100.00	100.00	100.00
Lemmas	100.00	100.00	100.00	100.00
UAS	91.40	91.40	91.40	91.40
LAS	88.53	88.53	88.53	88.53
CLAS	90.88	84.32	87.48	84.32
MLAS	90.70	84.16	87.31	84.16
BLEX	90.88	84.32	87.48	84.32

Pour le parsing, on s'intéresse aux lignes UAS et LAS.

Évaluer un outil

Une fois l'entraînement terminé, on peut aussi évaluer l'outil manuellement comme suit :

```
bin/maca_tm_parser_fr \
~/ud/fr/fr_test.conllu data/conllu.mcd > output.txt
```

Ainsi output.txt contiendra la prédiction du parser, on va ensuite ajouter des colonnes vides à cette sortie car le script d'évaluation a besoin de 10 colonnes :

```
../tools/conlluAddMissingColumns.py \
output.txt data/conllu.mcd > output.conllu
```

Maintenant on utilise un script d'évaluation pour comparer la prédiction et le gold :

```
../scripts/conll18_ud_eval.py \
~/ud/fr/fr_test.conllu output.conllu -v
```