Implémentation d’un analyseur en transitions

Implémentation

Formats de fichiers

Les principaux outils

Les principales classes

Installation et utilisation

Formats de fichiers

- Multi Column File (`mcf`)
- Multi Column Description (`mcd`)
- Feature Model (`fm`)
- Vocabulaires (`dico`)
- Class Features File (`cff`)

Multi Column File (`mcf`)

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

- format “en colonnes” :
  - chaque ligne correspond à une unité textuelle minimale (token)
  - chaque colonne correspond à un attribut du token (par exemple son genre ou son nombre)
- 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 ignorées

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`

- Chaque ligne du fichier `mcd` décrit une colonne du fichier `mcf`.
- Chaque ligne du fichier `mcd` est formée de quatre colonnes :
  1. un entier indiquant la colonne décrite (à partir de `0`)
  2. l'étiquette correspondant à la colonne (voir plus loin)
  3. le type de la valeur s'y trouvant
     - `SYM` indique que les valeurs sont des symboles
     - `INT` indique que les valeurs sont des entiers
  4. le mot clef `KEEP` ou `IGNORE`
     - `KEEP` indique que la colonne est prise en compte
     - `IGNORE` qu'elle ne l'est pas

Etiquettes

- Une étiquette permet de donner un nom explicite à une colonne
- Elles permettent d'accéder au contenu de la colonne à l'aide de *Word Features*
- On peut créer autant d'étiquettes que l'on veut
- Conventions
  - `FORM` la forme du token
  - `POS` sa partie de discours
  - `LEMMA` son lemme
  - `MORPHO` traits morphologique (genre, nombre, temps, ...)
  - `GOV` la position relative de son gouverneur ($-n$ indique que le gouverneur se trouve n tokens à gauche, n indique qu'il se trouve n tokens à droite)
  - `LABEL` sa fonction syntaxique
  - `EOS` sa position dans la phrase (1 dernier mot de la phrase, 0 autre)

mcd exemple

- un extrait du fichier mcf

```
La          det     le          1     det          0
diane       nc      diane       1     suj          0
chantait    v       chanter     0     root         0
```

- le fichier mcd correspondant

```
0    FORM          SYM KEEP
1    POS           SYM KEEP
2    LEMMA         SYM IGNORE
3    GOV           INT KEEP
4    LABEL         SYM KEEP
5    EOS           SYM KEEP
```

- Convention, on nomme les fichiers mcd selon les colonnes qu'ils définissent, par exemple FPLGSE.mcd

Features

- Fonctions permettant d'accéder au contenu d'une configuration
- Deux types de Features :
  1. Word Features : attributs des tokens, généralement présentes dans le fichier `mcd` fourni en entrée à l'analyseur.
     Exemples :
     - les parties de discours (`POS`)
     - la fin de phrase (`EOS`)
     - ...
  2. Configuration Features : features prédites pendant l'analyse ou pendant l'exécution de l'oracle.
     Exemples :
     - la distance entre le mot courant (`B0`) et le mot en sommet de pile (`S0`)
     - Le nombre d'éléments dans la pile
     - Le nombre de dépendants droits ou gauche du mot en sommet de pile
     - ...

Word features

Quadruplets (W, Conteneur, Position, Etiquette) :

1. Conteneur :
   - `B` pour Buffer
   - `S` pour Stack
2. Position
   - pour `B` :
     - `0` token courant
     - `1` token directement à droite de `B 0`
     - `2` token directement à droite de `B 1`
     - `-1` token directement à gauche de `B 0`
     - `-2` token directement à gauche de `B -1`
     - ...
   - pour `S`
     - `0` token au sommet de la pile
     - `1` token au dessous de `S 0` dans la pile
     - ...
3. Etiquette, telle que définie dans un `mcd`

Configuration Feature

- Quatre type de Configuration Feature
  1. Distance entre un mot du buffer et un mot de la pile (en général mot courant dans le buffer et mot au sommet de pile)

     ```
     C  DIST    B  0  S 0
     ```

  2. Nombre de dépendants droits ou gauche d'un mot

     ```
     C  NLDEP   S  0
     C  NRDEP   S  0
     ```

  3. du dépendant le plus à gauche ou le plus à droite d'un mot

     ```
     C  LLDEP   S  0
     C  LRDEP   S  0
     ```

  4. taille de la pile

     ```
     C  SH
     ```

Feature Model (`fm`)

- Un fichier `fm` définit un ensemble 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

Exemple de fichier `fm`

```
W  B      -2  POS
W  B      -1  POS
W  B      0   POS
W  B      1   POS
W  B      2   POS
W  S      0   POS
W  S      1   POS
C  DIST   B   0     S  0
C  NLDEP  S   0
C  NRDEP  S   0
C  SH
C  LLDEP  S   0
C  LRDEP  S   0
```

`dico`

- Dictionnaire permettant de stocker les différentes valeurs possibles d'une Feature, sous la forme de couples (clé, valeur).
- La clé correspond à une chaîne de caractère et la valeur à un entier (entiers successifs à partir de 0)
- Un fichier `dico` est composé :
  - d'une ligne de la forme `##FEAT` indiquant le début des valeurs possibles de `FEAT`
  - de lignes comportant un symbole (les valeurs possible de `FEAT`)
- Un fichier `dico` peut comporter plusieurs dictionnaires différents.

Exemple de fichier `dico`

```
##POS
NOUN
VERB
ADJ
##LABEL
nsubj
root
obj
##EOS
0
1
```

Class Features File (`cff`)

- Format utilisé pour entraîner les classifieurs de type réseaux de neurones
- Première ligne : taille de la couche d'entrée
- Deuxième ligne : taille de la couche de sortie
- Puis alternance de :
  - valeur couche d'entrée
  - valeur couche de sortie
- Représentation one-hot :
  - pour représenter la valeur 2 parmi 4 valeurs possible : `0 0 1 0`
- Exemple

```
8
2
0 1 0 0 0 0 1 0
1 0
1 0 0 0 0 1 0 0
0 1
```

Les principaux outils

- `mcf2cff`
- `tbp_train`
- `tbp_decode`
- `eval_mcf`

mcf2cff

description :

- Transforme des arbres syntaxiques en séquences de configurations et de mouvements à l'aide d'un oracle.

entrées :

- un fichier `mcf` annoté en syntaxe (colonnes `GOV` et `LABEL`)
- un fichier `fm` décrivant les features utilisées pour la prédiction
- un fichier `mcd` décrivant la structure du fichier `mcf`
- un fichier `dico` contenant les vocabulaires correspondant aux features utilisées et le vocabulaire des features du classifieur
- le nombre minimal de mots à traiter

sorties :

- un fichier `cff` contenant les données pour l'apprentissage du classifieur

tbp_train

description :

- Apprend un classifieur de type MLP au format keras à partir d'un fichier `cff`
- Les hyperparamètres du MLP sont représentés en dur dans le programme

entrées :

- un fichier `cff`

sorties :

- un modèle au format keras/tensorFlow

tbp_decode

description :

- Réalise l'analyse syntaxique de phrases d'un fichier `mcf`

entrées :

- un fichier `mcf` contenant le texte à analyser
- un modèle keras
- un fichier `dico` contenant les vocabulaires correspondant aux features utilisées et le vocabulaire des features du classifieur
- un fichier `fm` contenant le modèle de features
- un fichier `mcd` décrivant la structure du fichier `mcf`
- le nombre minimal de mots à traiter

sorties :

- un fichier `mcf` avec deux colonnes supplémentaires pour `GOV` et `LABEL`

eval_mcf

description :

- Compare deux fichiers `mcf` et calcule le Labeled Accuracy Score et le Unlabeled Accuracy Score

entrées :

- un fichier `mcf` de référence
- un fichier `mcf` hypothèse (dont les colonnes `GOV` et `LABEL` ont été prédites)
- un fichier `mcd` pour la référence
- un fichier `mcd` pour l'hypothèse

sorties :

- LAS et UAS du fichier hypothèse

Les principales classes

- `Word.py`
- `WordBuffer.py`
- `Stack.py`
- `Config.py`
- `FeatModel.py`
- `Dico.py`
- `Dicos.py`
- `Mcd.py`
- `Moves.py`

Word

```
def __init__(self):
  self.featDic = {}           # dictionnaire dans lequel sont stoc
  self.leftDaughters = []     # liste des indices des dépendants g
  self.rightDaughters = []    # liste des indices des dépendants d

def getFeat(self, featName)
def setFeat(self, featName, featValue)
def addLeftDaughter(self, index)
def addRightDaughter(self, index)
```

WordBuffer

```
def __init__(self, mcfFileName=None, mcd=None):
  self.currentIndex = 0
  self.array = []
  self.mcd = mcd
  self.mcfFile = None

def empty(self)
def init(self, mcd)
def addWord(self, w)
def getLength(self)
def getCurrentIndex(self)
def getWord(self, index)
def getCurrentWord(self)
def readNextWord(self)
def readNextSentence(self)
def endReached(self)
```

Stack

Les éléments de la pile sont des indices de mots et pas des références à des `Word`

```
def __init__(self):
  self.array = []

def isEmpty(self)
def empty(self)
def push(self, elt)
def pop(self)
def top(self)
def getLength(self)
```

Config

```
def __init__(self, filename, mcd, dicos):
  self.wb = WordBuffer(filename, mcd)
  self.st = Stack()

def isFinal(self)
def getStack(self)
def getBuffer(self)
def fwd(self)
def shift(self)
def red(self)
def right(self, label)
def left(self, label)
def applyMvt(self, mvt)
def getWordFeat(self, featTuple)
def extractFeatVec(self, FeatModel)
```

FeatModel

En gros un tableau de features
Permet aussi de calculer la taille du vecteur d'entrée du classifieur

```
def __init__(self, featModFilename, dicos):
  self.featArray = self.readFeatModelFile(featModFilename)
  self.inputVectorSize = self.computeInputSize(dicos)

def computeInputSize(self, dicos)
def getInputSize(self)
def getFeatArray(self)
def getNbFeat(self)
def getFeatContainer(self, featIndex)
def getFeatPosition(self, featIndex)
def getFeatLabel(self, featIndex)
def buildInputVector(self, featVec, dicos)
```

Dico

Un `Dico` permet d'associer des symboles et des entiers.

```
def __init__(self, name):
  self.name = name
  self.hash = {}
  self.array = []

def add(self, symbol)
def getCode(self, symbol)
def getSymbol(self, code)
def getSize(self)
def printToFile(self, dicoFile)
```

Dicos

Ensemble de dico, chacun étant accessible à partir de son nom

```
def __init__(self, mcd=False, fileName=False):
  self.content = {}
  if mcd :
    self.initializeWithMcd(mcd)
  if fileName :
    self.initializeWithDicoFile(fileName)

def getDico(self, dicoName)
def addDico(self, dicoName)
def getCode(self, dicoName, symbol)
def getSymbol(self, dicoName, code)
def add(self, dicoName, symbol)
def populateFromMcfFile(self, mcfFilename, mcd)
def printToFile(self, filename)
```

Mcd

Chaque colonne d'un `Mcd` est un triplet `(name, type, status)`.

```
def __init__(self, mcdFilename):
  self.array = self.readMcdFile(mcdFilename)

def readMcdFile(self, mcdFilename)
def getNbCol(self)
def getColName(self, colIndex)
def getColType(self, colIndex)
def getColStatus(self, colIndex)
def locateCol(self, name)
```

Moves

Associe un code à une action
Permet aussi de construire le vecteur de sortie pour le classifieur

```
def __init__(self, dicos):
  self.dicoLabels = dicos.getDico('LABEL')
  if not self.dicoLabels :
    print("cannot find LABEL in dicos")
    exit(1)
  self.nb = 2 * self.dicoLabels.getSize() + 3

def getNb(self)
def mvtCode(self, mvt)
def mvtDecode(self, mvt_Code)
def buildOutputVector(self, mvt)
```

Dépot GIT

- Le code de l'analyseur, ainsi que les données et les scripts permettant de lancer des expériences se trouvent sur le dépôt GIT suivant :
  `https://gitlab.lis-lab.fr/alexis.nasr/tbp.git`
- Sa structure est la suivante :
  - `src` contient tout le code
  - `data` contient les données (pas sur le git pour l'instant, mais sur la page du cours)
  - `expe`
    - un fichier `fm` (`basic.fm`)
    - un fichier `Makefile` qui permet de lancer une expérience
    - un fichier `mcd` (`PGLE.mcd`)

Le Makefile (de la mort)

```
train_conll=../data/train_$(lang).conllu
train_proj_conll=./out/train_$(lang)_proj.conllu
train_mcf=./out/train_$(lang)_pgle.mcf
train_cff=./out/train_$(lang).cff
train_word_limit=40000

dev_conll=../data/dev_$(lang).conllu
dev_proj_conll=./out/dev_$(lang)_proj.conllu
dev_mcf=./out/dev_$(lang)_pgle.mcf
dev_cff=./out/dev_$(lang).cff
dev_word_limit=5000

test_conll=../data/test_$(lang).conllu
test_mcf=./out/test_$(lang)_pgle.mcf
test_mcf_hyp=./out/test_$(lang)_hyp.mcf
test_word_limit=700

feat_model=basic.fm

dicos=./out/$(lang)_train.dic
model=./out/$(lang).keras
results = ./out/$(lang).res

mcd_pgle=PGLE.mcd
```

Le Makefile (de la mort)

```
eval: $(test_mcf_hyp)
python3 ../src/eval_mcf.py $(test_mcf) $(test_mcf_hyp) $(mcd_pgle) $(mcd_pgle) $(lang) > $(results)

$(test_mcf_hyp): $(test_mcf) $(model)
python3 ../src/tbp_decode.py $(test_mcf) $(model) $(dicos) $(feat_model) $(mcd_pgle) $(test_word_limit) > $(test_mcf_hyp)

$(model): $(train_cff) $(dev_cff)
python3 ../src/tbp_train.py $(train_cff) $(dev_cff) $(model)

$(train_cff): $(train_mcf) $(dicos)
python3 ../src/mcf2cff.py $(train_mcf) $(feat_model) $(mcd_pgle) $(dicos) $(train_cff) $(train_word_limit)

$(dev_cff): $(dev_mcf) $(dicos)
python3 ../src/mcf2cff.py $(dev_mcf) $(feat_model) $(mcd_pgle) $(dicos) $(dev_cff) $(dev_word_limit)

$(train_mcf): $(train_proj_conll)
python3 ../src/conll2mcf.py $(train_proj_conll) $(mcd_pgle) > $(train_mcf)

$(dev_mcf): $(dev_proj_conll)
python3 ../src/conll2mcf.py $(dev_proj_conll) $(mcd_pgle) > $(dev_mcf)

$(test_mcf):
python3 ../src/conll2mcf.py $(test_conll) $(mcd_pgle) > $(test_mcf)

$(train_proj_conll):
python3 ../src/remove_non_projective_sentences_from_conll.py $(train_conll) > $(train_proj_conll)

$(dev_proj_conll):
python3 ../src/remove_non_projective_sentences_from_conll.py $(dev_conll) > $(dev_proj_conll)

$(dicos): $(train_mcf)
python3 ../src/create_dicos.py $(train_mcf) $(mcd_pgle) $(dicos)
```

Utilisation

pré-requis :

- une machine avec python et keras
- accès à collab

Installation et exécution :

```
git clone https://gitlab.lis-lab.fr/alexis.nasr/tbp.git
cd tbp
wget http://pageperso.lif.univ-mrs.fr/~alexis.nasr/Ens/TLNL/data.tgz
tar xvfz data.tgz
cd expe
make lang=fr
```