IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

UE TLFT

TP Irma
(mercredi 11 mars 2020)

Yannis Haralambous (IMT Atlantique)

1 Le corpus

Il s'agit de 60 articles du journal *Le Monde* sur l'ouragan Irma qui a frappé l'Atlantique Nord en septembre 2017. Le fichier `irma.txt` (sur Moodle) contient les articles, nettoyés, codés en UTF-8, séparés par des

```
==================================================
```

Nous avons calculé les POS tags et les dépendances de ce corpus en nous servant des outils *Talismane* [3, 4] et *grew* [2], le résultat est sur Moodle (fichier `irma.suf.conll`). Voir l'annexe de l'énoncé pour des instructions comment utiliser vous-même ces outils.

2 Calcul de la termitude

On va appliquer la formule de C-valeur de Frantzi-Ananiadou-Tsujii (article [1] de 1999 que vous trouverez sur Moodle) qui se calcule comme suit :

Soit $\mathcal{M}$ un ensemble de motifs de POS tags, et soit C l'ensemble des groupes de mots reconnus par ces motifs («C» comme «candidats»). Entre ces groupes de mots $(m_1, \ldots, m_n)$ il y a des inclusions : $(m_1, \ldots, m_n) \subset (m_1, \ldots, m_{n+1})$, comme par exemple «bloc opératoire» $\subset$ «bloc opératoire opérationnel».

Soit a un candidat de n mots ($|a| = n$) et T_a l'ensemble des candidats qui contiennent a et notons $\#X$ le cardinal d'un ensemble X et $f(x)$ la fréquence d'un terme dans le corpus. Alors on définit la C-valeur de a par

$$C(a) = \begin{cases} \log_2 |a| \cdot f(a) & \text{si } T_a = \emptyset, \\ \log_2 |a| \cdot \left(f(a) - \frac{1}{\#Ta} \sum_{b \in T_a} f(b) \right) & \text{sinon.} \end{cases}$$

Pour calculer efficacement la C-valeur de tous les candidats, on commence par les groupes les plus longs, ce qui nous donne les ensembles T_* pour le calcul des sous-groupes de ceux-ci.

Quand on a calculé la C-valeur de tous les groupes de mots respectant les motifs $\mathcal{M}$, on affiche les candidats par ordre décroissant le C-valeur, et on pose manuellement un seuil au-delà duquel les candidats sont appelés *termes* et en-deçà duquel les candidats sont ignorés.

2.1 Exercice

Trouver les termes du corpus Irma en implémentant la formule de calcul de la C-valeur et en l'appliquant au document `irma.surf.conll`. En utilisera

$$\mathcal{M} = \begin{cases} \text{NC ADJ+} \\ \text{NC P DET ? NC (ou P peut être «de».} \end{cases}$$

Exemples : «innovation conceptuelle», «centrale de pharmacie» (pour N P N), «droits de l'homme» (pour N P DET N).

Attention :

1. certains nombres sont taggés en tant que NC ou en tant que ADJ, les éviter
2. ne garder que la préposition *de* et le déterminant défini dans les motifs
3. éviter les valeurs _ en tant que NC.

SOLUTION

```
import re, io, numpy

conll_re=re.compile(r'^([0-9]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]

motif_a_re=re.compile(r'NC([0-9]+) ADJ([0-9]+) ADJ([0-9]+)')
motif_b_re=re.compile(r'NC([0-9]+) ADJ([0-9]+)')
motif_c_re=re.compile(r'NC([0-9]+) Pde([0-9]+) DETle([0-9]+) NC([0-9]+)')
motif_d_re=re.compile(r'NC([0-9]+) Pde([0-9]+) NC([0-9]+)')

motif_num_re=re.compile(r'[0-9]+$')

SENTS=[]
SENT=[]

f=io.open("irma.surf.conll",encoding="utf-8")
for ligne in f:
    m=conll_re.match(ligne)
    if m:
        if m.group(1)=="1":
            if SENT:
                SENTS.append(SENT)
            SENT=[]
        if (m.group(5)=="NC" and motif_num_re.match(m.group(3))):
            five="NCnum"
        elif (m.group(5)=="ADJ" and motif_num_re.match(m.group(3))):
            five="ADJnum"
        elif (m.group(3)=="de" and m.group(5)=="P"):
            five="Pde"
        elif (m.group(3)=="le" and m.group(5)=="DET"):
            five="DETle"
        else:
            five=m.group(5)
        SENT.append((m.group(3),five+m.group(1)))
f.close()

CANDIDATS=[]

for sentence in SENTS:
    tags=" ".join([x[1] for x in sentence])
    #print(tags)
    mm=motif_a_re.findall(tags)
    for m in mm:
```

```
      CANDIDATS.append((sentence[int(m[0])-1][0],sentence[int(m[1])-1][0],sentence[int(m[2])-1][0]))
    mm=motif_b_re.findall(tags)
    for m in mm:
        CANDIDATS.append((sentence[int(m[0])-1][0],sentence[int(m[1])-1][0]))
    mm=motif_c_re.findall(tags)
    for m in mm:
      CANDIDATS.append((sentence[int(m[0])-1][0],sentence[int(m[1])-1][0],sentence[int(m[2])-1][0],sentence[i
    mm=motif_a_re.findall(tags)
    for m in mm:
      CANDIDATS.append((sentence[int(m[0])-1][0],sentence[int(m[1])-1][0],sentence[int(m[2])-1][0]))

CANDIDATS=sorted(CANDIDATS,key=lambda x: len(x),reverse=True)

FREQ={}

for c in CANDIDATS:
    if not "_" in c:
        if c in FREQ.keys():
            FREQ[c] += 1
        else:
            FREQ[c] = 1

T_C={}

for c in FREQ.keys():
    if len(c)>=3:
        if c[:-1] in T_C.keys():
            T_C[c[:-1]].append(c)
        else:
            T_C[c[:-1]] = []
            T_C[c[:-1]].append(c)

C={}

for c in FREQ.keys():
    if c in T_C.keys():
        C[c] = numpy.log2(len(c))*(FREQ[c] - (1/len(T_C[c])) * sum([FREQ[b] for b in T_C[c]]))
    else:
        C[c] = numpy.log2(len(c))*FREQ[c]

Ckeys=C.keys()
Ckeys=sorted(Ckeys,key=lambda x: C[x],reverse=True)

for c in Ckeys:
    print(" ".join(c),C[c])
```

Pourquoi «bureau météorologique» a une C-valeur de -1 et «consommation énergétique» une C-valeur de -3 ?

2.2 La suite des évènements

On ne le fera pas en TP, mais normalement, une fois détectés les termes complexes, on les remplace dans le fichier CoNLL en mettant tout dans la tête du terme :

```
4 le le D DET g=m|n=s 5 det _ _
5 bureau bureau N NC g=m|n=s|s=c 8 suj _ _
6 météorologique météorologique A ADJ n=s 5 mod _ _
7 américain américain A ADJ g=m|n=s 5 mod _ _
8 baptise baptiser V V m=ind|n=s|t=pst _ _ _ _
```

doit devenir

```
4 le le D DET g=m|n=s 5 det _ _
5 bureau_météorologique_américain bureau_météorologique_américain N NC g=m|n=s|s=c 6 suj _
_
6 baptise baptiser V V m=ind|n=s|t=pst _ _ _ _
```

Attention aux dépendances : leurs numéros doivent changer aussi.

SOLUTION Parce que «bureau météorologique américain» apparaît 2 fois et «bureau météorologique» seulement une fois. De même «consommation énergétique française» apparaît 6 fois et «consommation énergétique» seulement 3 fois.

3 Entités nommées

Comment *Talismane+grew* gèrent-ils les entités nommées ? Comment récupérer celles qui manquent ?

Quand les entités nommées occupent plusieurs tokens dans le CoNLL, on remplace :

```
16 de de P P _ 13 dep _ _
17 l'le D DET n=s 18 det _ _
18 hôpital hôpital N NC g=m|n=s|s=c 16 obj.p _ _
19 de de P P _ 18 dep _ _
20 Saint _ N NPP s=p 19 obj.p _ _
21 --PONCT PONCT _ 22 ponct _ _
22 Martin martin N NC g=m|n=s|s=c _ _ _ _
```

devient

```
16 de de P P _ 13 dep _ _
17 l'le D DET n=s 18 det _ _
18 hôpital hôpital N NC g=m|n=s|s=c 16 obj.p _ _
19 de de P P _ 18 dep _ _
20 Saint-Martin _ N NPP s=p 19 obj.p _ _
```

Il faut le faire au cas par cas (ici, dans l'exemple, on a un NPP suivi d'un trait d'union et d'un NC : on peut imaginer que si un nom est séparé d'un nom propre qui le précède par un trait d'union, alors il fait partie de ce nom, et donc on peut les réunir).

Réunir également les noms propres en plusieurs mots sans ponctuation :

```
16 New _ N NPP s=p 15 obj.p _ _
17 York York N NPP g=m|n=s|s=p 16 mod _ _
```

devient

```
16 New_York _ N NPP s=p 15 obj.p _ _
```

Écrire le code qui (a) réunit les cas (NPP, "-", NC) et (b) réunit les cas (NPP, NPP). Faites attention aux dépendances, *avant et après* l'entité nommée.

Tuyau : utiliser des objets pour stocker les tokens du fichier CoNLL avec un attribut de plus : le nouveau numéro.

```
class token ():
    num=0
    w=""
    lem=""
    tagsimple=""
    tag=""
```

```
    feats=""
    dep=0
    depnat=""
    newnum=0
    def __init__(self,num,w,lem,tagsimple,tag,feats,dep,depnat,oldnum):
        self.num=num
        self.w=w
        self.lem=lem
        self.tagsimple=tagsimple
        self.tag=tag
        self.feats=feats
        self.dep=dep
        self.depnat=depnat
        self.newnum=newnum
```

Normalement il y a 128 cas (a) et 465 cas (b).

SOLUTION

```
import re, io, numpy

conll_re=re.compile(r'^([0-9]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]

debarrasser_barre_re=re.compile(r'([^|]+)|.+')

SENTS=[]
SENT=[]

class token:
    num=0
    w=""
    lem=""
    tagsimple=""
    tag=""
    feats=""
    dep=0
    depnat=""
    newnum=0
    def __init__(self,num,w,lem,tagsimple,tag,feats,dep,depnat,newnum):
        self.num=num
        self.w=w
        self.lem=lem
        self.tagsimple=tagsimple
        self.tag=tag
        self.feats=feats
        self.dep=debarrasser_barre_re.sub(r'\1',dep)
        self.depnat=debarrasser_barre_re.sub(r'\1',depnat)
        self.newnum=newnum

f=io.open("irma.surf.conll",mode="r",encoding="utf-8")
for ligne in f:
    m=conll_re.match(ligne)
    if m:
        if m.group(1)=="1":
            if SENT:
                SENTS.append(SENT)
            SENT=[]
      o=token(m.group(1),m.group(2),m.group(3),m.group(4),m.group(5),m.group(6),m.group(7),m.group(8),0)
        SENT.append(o)
f.close()
```

```
SENTS.append(SENT)

for sentence in SENTS:
    for i in range(len(sentence)):
      if (i+2<len(sentence) and sentence[i].newnum>=0 and sentence[i].tag=="NPP" and sentence[i+1].w=="-" and s
            sentence[i].w=sentence[i].w+"-"+sentence[i+2].w
            sentence[i+1].newnum=-int(sentence[i].num)
            sentence[i+2].newnum=-int(sentence[i].num)
            chang_a+=1
      elif (i+1<len(sentence) and sentence[i].newnum>=0 and sentence[i].tag=="NPP" and sentence[i+1].tag=="NPP
            sentence[i].w=sentence[i].w+"_"+sentence[i+1].w
            sentence[i+1].newnum=-int(sentence[i].num)
            chang_b+=1

for sentence in SENTS:
    compteur=1
    for i in range(len(sentence)):
        if (sentence[i].newnum >= 0):
            sentence[i].newnum=compteur
            compteur += 1

for sentence in SENTS:
    for i in range(len(sentence)):
        if (sentence[i].dep != "_" and sentence[i].dep != "0"):
            sentence[i].dep=sentence[int(sentence[i].dep)-1].newnum

for sentence in SENTS:
    for c in sentence:
        c.num=c.newnum

f=io.open("irma.surf.conll2",mode="w",encoding="utf-8")
for sentence in SENTS:
    for c in sentence:
        if int(c.num) > 0:
          print("\t".join([str(x) for x in (c.num,c.w,c.lem,c.tagsimple,c.tag,c.feats,c.dep,c.depnat,"_","_")]
    print("",file=f)
f.close()
```

4 Résolution d'anaphore

Il s'agit de trouver le référent d'une expression référentielle. Pour simplifier on va s'intéresser juste aux pronoms, par exemple dans «Mon père est venu. Il est arrivé hier», on cherche à détecter que «Il» se réfère à «père» (on écrira «Mon père$_1$ est venu. Il$_1$ est arrivé hier».

D'après le Bescherelle il y a des pronoms

— personnels (je, tu, il, nous, le, lui, etc.),
— démonstratifs (ce, ceci, cela, etc.),
— possessifs (le mien, le tien, le sien, etc.),
— interrogatifs (qui, que, lequel, etc.),
— relatifs (qui, que quoi, dont, où, etc.),
— etc.

Comme dans ce TP le corpus est journalistique, le texte est le plus souvent écrit à la troisième personne. On va s'intéresser aux pronoms personnels «il», il sera facile d'étendre les résultats aux autres pronoms.

Pour trouver le référent d'un «il», problèmes se présentent :

1) il peut s'agir d'un *verbe impersonnel*.

Il existe des verbes qui sont tout le temps impersonnels : il faut, il gèle, il grêle, il grésille, il importe, il pleut, il tonne, il neige ; et des verbes qui sont souvent impersonnels : il arrive, il convient, il résulte, il suffit, il paraît, il tarde, il semble, etc.

2) le référent peut être situé avant ou après le pronom, dans le deuxième cas on parle de *cataphore* : «il$_1$ est malin, mon ami$_1$». Il peut se trouver dans la même phrase ou dans une phrase qui précède ou qui suit. On va considérer que cela ne peut être que la phrase précédent ou la phrase suivante, dans le même document.

Pour trouver des candidats, on appliquera la méthode suivante :

1. on cherchera les NC ou NPP qui se trouvent dans la phrase précédente, la même phrase ou la phrase suivante, en restant dans le même document ;
2. on ne gardera que ceux qui sont masculins et au singulier ;
3. on calculera la distance entre référence et référent, pour ceux qui se trouvent après la référence on multipliera par un facteur ρ (par exemple 1.5, mais qui peut varier) ;
4. on gardera celui de plus faible poids.

Cet algorithme est bien sûr largement perfectible (par exemple en mesurant des co-occurrences verbe-référent, etc.). Ne soyez pas frustrés si le premier cas de pronom «il» du corpus donne déjà un mauvais résultat :

«...l'ouragan$_1$ Harvey a frappé le Texas$_2$ le 25 août$_3$. Reclassé en tempête$_4$ tropicale, il$_3$ s'est accompagné»
(«il» va pointer vers «août» puisque c'est le NC le plus proche). Le deuxième «il», par contre, doit donner un résultat correct :

«Un ouragan$_1$ de catégorie$_2$ 3 ou 4 inquiète désormais les Caraïbes$_3$. Il$_1$ pourrait passer...»
(ici «il» pointe vers «ouragan» puisque «catégorie» et «Caraïbes» sont féminins).

4.1 Exercice

Écrire le code qui va détecter les référents et ajouter cette information dans le champ 9 du format CoNLL. On indiquera le numéro du token ciblé, précédé de `(-1)` si c'est dans la phrase précédente ou de `(+1)` si c'est dans la phrase suivante.

Vous allez faire 244 cas, essayer de les évaluer manuellement.

SOLUTION

```
import re, io, numpy

conll_re=re.compile(r'^([0-9]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]

rho=3.

debarrasser_barre_re=re.compile(r'([^|]+)|.+')

DOCS=[]
SENTS=[]
SENT=[]

class token:
    num=0
    w=""
    lem=""
    tagsimple=""
    tag=""
    feats=""
    dep=0
    depnat=""
    ref="_"
    doc=0
    def __init__(self,num,w,lem,tagsimple,tag,feats,dep,depnat,ref,doc):
        self.num=num
        self.w=w
```

```
        self.lem=lem
        self.tagsimple=tagsimple
        self.tag=tag
        self.feats=feats
        self.dep=debarrasser_barre_re.sub(r'\1',dep)
        self.depnat=debarrasser_barre_re.sub(r'\1',depnat)
        self.ref=ref
        self.doc=doc

f=io.open("irma.surf.conll",mode="r",encoding="utf-8")
for ligne in f:
    m=conll_re.match(ligne)
    if m:
        if m.group(1)=="1":
            if SENT:
                SENTS.append(SENT)
                if m.group(2)=="================================================":
                    DOCS.append(SENTS)
                    SENTS=[]
            SENT=[]
      o=token(m.group(1),m.group(2),m.group(3),m.group(4),m.group(5),m.group(6),m.group(7),m.group(8),"_",0)
        SENT.append(o)
f.close()
SENTS.append(SENT)
DOCS.append(SENTS)

compteur=0

for doc in DOCS:
    for I in range(len(doc)):
        for i in range(len(doc[I])):
            if doc[I][i].w=="Il" or doc[I][i].w=="il":
                CANDIDATS=[]
                CANDIDATS2=[]
                #on regarde dans la meme phrase, avant i
                if i>0:
                    for j in range(i-1,0,-1):
                   if ((doc[I][j].tag=="NC" or doc[I][j].tag=="NPP") and re.match(r'g=m\|n=s',doc[I][j].feats)
                            #on a trouve un candidat
                          CANDIDATS.append((doc[I][j],0,-1,i-j))# objet, meme doc, arriere, distance
                #on regarde dans la meme phrase, apres i
                if i<len(doc[I]):
                    for j in range(i+1,len(doc[I]),1):
                   if ((doc[I][j].tag=="NC" or doc[I][j].tag=="NPP") and re.match(r'g=m\|n=s',doc[I][j].feats)
                            #on a trouve un candidat
                            CANDIDATS.append((doc[I][j],0,1,j-i))#objet, meme doc, devant, distance
                #on regarde dans la phrase precedente
                if I>0:
                    for j in range(len(doc[I-1])-1,0,-1):
                   if ((doc[I-1][j].tag=="NC" or doc[I-1][j].tag=="NPP") and re.match(r'g=m\|n=s',doc[I-1][j].
                            #on a trouve un candidat
                      CANDIDATS.append((doc[I-1][j],-1,-1,i+(len(doc[I-1])-j)))# objet, doc precedent, arriere
                #on regarde dans la phrase suivante
                if I<len(doc)-1:
                    for j in range(0,len(doc[I+1]),1):
                   if ((doc[I+1][j].tag=="NC" or doc[I+1][j].tag=="NPP") and re.match(r'g=m\|n=s',doc[I+1][j].
                            #on a trouve un candidat
                      CANDIDATS.append((doc[I+1][j],1,1,(len(doc[I])-i+j)))# objet, doc suivant, devant, dista
                #on compare les candidats
                for (c,ii,jj,kk) in CANDIDATS:
                    if jj==-1:
```

```
                    CANDIDATS2.append((c,ii,jj,kk))
                else:
                    CANDIDATS2.append((c,ii,jj,rho*kk))
            CANDIDATS2=sorted(CANDIDATS2,key=lambda x: float(x[3]),reverse=False)
            if CANDIDATS2:
                compteur += 1
                if CANDIDATS2[0][1]==-1:
                    doc[I][i].ref = "(-1)"+str(CANDIDATS2[0][0].num)
                elif CANDIDATS2[0][1]==1:
                    doc[I][i].ref = "(+1)"+str(CANDIDATS2[0][0].num)
                else:
                    doc[I][i].ref = str(CANDIDATS2[0][0].num)

f=io.open("irma.surf.conll2",mode="w",encoding="utf-8")
for doc in DOCS:
    for sentence in doc:
        for c in sentence:
            if int(c.num) > 0:
             print("\t".join([str(x) for x in (c.num,c.w,c.lem,c.tagsimple,c.tag,c.feats,c.dep,c.depnat,c.ref,
        print("",file=f)
f.close()

print(compteur)
```

A Utilisation de *Talismane* et de *grew*

Récupérer *Talismane* sur http://redac.univ-tlse2.fr/applications/talismane.html et *grew* sur http://www.grew.fr/.

Pour le fichier irma.txt il suffit de lancer :

```
java -Xmx1G -Dconfig.file=talismane-fr-5.2.0.conf -jar talismane-core-5.2.0.jar --analyse
    --endModule=posTagger --sessionId=fr --encoding=UTF8 --inFile=../irma.txt
    --outFile=../irma.tal
sed -f tal2grw.sed irma.tal > irma.pos.conll
grew transform -grs POStoSSQ/grs/surf_synt_main.grs -i irma.pos.conll -o irma.surf.conll
```

Références

[1] Katerina T. Frantzi, Sophia Ananiadou, and Junichi Tsujii. The C-value/NC-value Method of Automatic Recognition for Multi-word Terms. In *Proceedings of ECDL'98*, volume 1513 of *Springer LNCS*, pages 585–604, 1999.

[2] Bruno Guillaume and Guy Perrier. Dependency parsing with graph rewriting. In *IWPT 2015, 14th International Conference on Parsing Technologies*, pages 30–39, 2015.

[3] Assaf Urieli. *Robust French syntax analysis : reconciling statistical methods and linguistic knowledge in the Talismane toolkit.* PhD thesis, Université de Toulouse II le Mirail, 2013.

[4] Assaf Urieli and Ludovic Tanguy. L'apport du faisceau dans l'analyse syntaxique en dépendances par transitions : études de cas avec l'analyseur Talismane. In *Actes de la 20e conférence sur le Traitement Automatique des Langues Naturelles (TALN'2013)*, pages 188–201, 2013.