Compte Rendu "Qui a tué qui? Où ? Quand"

Projet de Projet Ingénierie Linguistique, Groupe F

BIVER Alexis - BRICHE Arnaud - COLIN Emile - CULARD Mathieu - RICHIER Valère
L3 MIASHS parcours Sciences Cognitives - Ingénierie Linguistique

Table des matières

BIVER Alexis - BRICHE Arnaud - COLIN Emile - CULARD Mathieu - RICHIER Valère
L3 MIASHS parcours Sciences Cognitives - Ingénierie Linguistique

Introduction

Ce projet vise à créer un programme permettant de faire une recherche sur un corpus de texte. Plus précisément, nous travaillons en nous basant sur une question : “Qui a tué qui, quand et où ?”. En d’autres termes, le but ici est de rechercher des tueurs dans un corpus, le nom de leur victime, la date du crime ainsi que le lieu. Pour répondre à cette demande, le corpus a été tiré d’une série de pages wikipédia sur des tueurs. La recherche des noms des tueurs est ramenée à une unique lettre, F, afin de ne pas avoir une recherche prenant trop de temps algorithmiquement parlant. Le projet est codé en python, à l’aide de librairies comme NLTK, WordNet, XML, ou ElementTree. Pour vous expliquer le fonctionnement de cette recherche, nous aborderons tout d’abord la gestion du projet et la composition de l’équipe, pour ensuite discuter de la composition du corpus. Par la suite, nous aborderons le point de la tokenisation et l’Identification des Entités Nommées. Pour terminer, les algorithmes de la recherche de synonymes du verbe “tuer” seront étudiés en détails, pour finir sur les méthodes utilisées pour l’identification finale.

Composition de l'équipe

Équipe F :

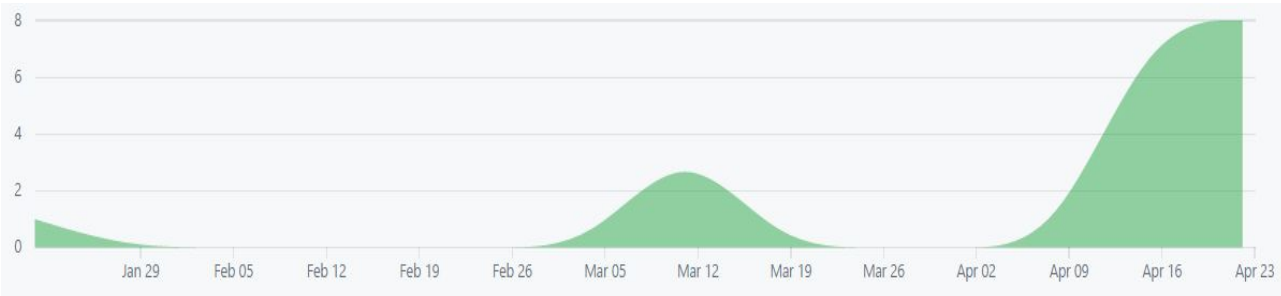
- BIVER Alexis
- BRICHE Arnaud
- COLIN Emile
- CULARD Mathieu
- RICHIER Valère

Avancée du projet

Gestion du projet

Mise en place d'un espace de travail collaboratif :

- Trello
- GitHub
    - Graphique des commits au cours du temps :



- Définition des tâches et répartition
- Communication via groupe messenger

Liste des tâches

Composition du corpus
Tokenization
Identification des entités nommées
Recherche de verbes synonymes de “kill”
Recherche des phrases à partir des verbes
Identification

Constitution du corpus

Mots clés choisis :

- Fictional murderers
- People executed for murder
- Male serial killers

Après avoir parcouru les pages spéciales de wikipédia regroupant différents types de meurtriers par catégories à partir de la page https://en.wikipedia.org/wiki/Category:Murderers, nous avons choisis nos mots clés de catégories parmis ceux qui contenaient une grand nombre de pages de meurtriers dont le nom commence par F (lettre qui ne nous a pas semblée être parmis les plus faciles car peu courantes dans les noms de meurtriers).
D'après notre recensement manuel et approximatif sur Wikipédia dans le classement par nom, la catégorie Fictional murderers contient 24 tueurs en F, la catégorie People executed for murder en contient 21 et la catégorie Male serial killers en contient 17.
Il nous paraissait tout d'abord important d'avoir plus de pages et de tueurs possibles afin d'être sûrs de trouver suffisamment de données sur chacun pour répondre à la consigne, pour les serial killers il pouvait être difficile d'identifier les victimes et pour les tueurs de fictions il pouvait être difficile de trouver un lieu et un moment.
Ainsi avec ces trois catégories nous avons (naïvement) exporté via wikipedia un XML de 25,5 Mo que nous avons essayé en vain de traiter avec le site du loria fourni dans la description du projet : http://talc2.loria.fr/import-text/ sans succès, puis avec un deuxième fichier légèrement plus léger duquel nous avons retiré les pages ne menant qu'à d'autres catégories.
Devant ces échecs nous avons alors fait le choix d'extraire uniquement les pages qui nous intéressent en les sélectionnant via les pages de catégories cités plus haut uniquement les pages de tueurs dont le nom commence par F dont le txt contenant le nom de ces pages entrés dans wikipedia est en annexe. Pensant ne plus répondre à la consigne, nous vint alors l’idée d’ajouter une centaine de pages “parasites”. L’export fût encore une fois impossible, notre corpus de base est donc uniquement composé des pages vues précédemment.

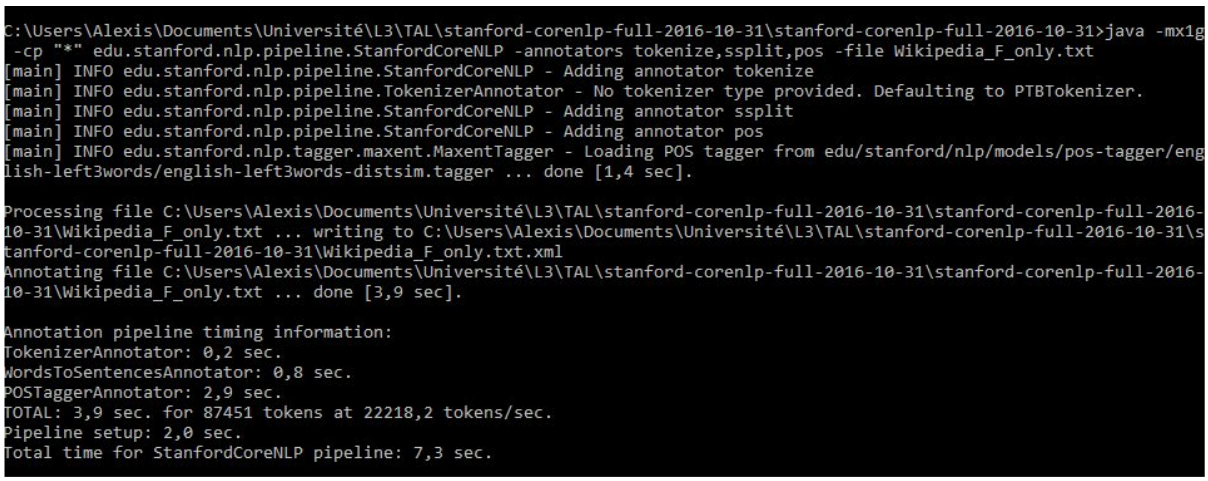
Nous voilà ainsi avec un fichier Wikipedia_F_only.txt de 430ko crée après traitement du site du loria du fichier Wikipedia-F only.xml de 726ko extrait de Wikipedia.

Tokenization

Traitement sur le corpus

Pour tokenizer le corpus extrait, nous avons utilisé CoreNLP comme vu lors du premier TD A partir de notre txt et de stanford-corenlp-full-2016-10-31, nous avons utilisé la commande suivante :

java -mx1g -cp ”*” edu.stanford.nlp.pipeline.StanfordCoreNLP -annotators tokenize,ssplit,pos -file Wikipedia_F_only.txt



Après 7,3 secondes d’attente plutôt stressantes, nous avons maintenant un fichier Wikipedia_F_only_Tokenizé par CoreNLP.xml de près de 20Mo sur lequel le traitement a bien été opéré

Scripts Python

Nous allons désormais vous présenter les différentes méthodes python permettant d’ouvrir un fichier XML. celles-ci sont assez simples et rapide, car utilisant la librairie XML. Ainsi, nous ouvrons un fichier à l’aide de la fonction open(), prenant en paramètre le fichier en question, le mode d’ouverture (‘r’ pour lecture, ou read), et l’encodage en UTF-8. Par la suite, nous faisons un parsing sur ce fichier à l’aide la libraire ElementTree en stockant le résultat dans “tree”, pour enfin récupérer la racine de cet arbre avec la méthode getroot().

Afin de lire le fichier et récupérer les verbes dans un cas, et les entités nommées dans un autre, nous recherchons successivement avec des boucles for dans les différents tags des tokens du fichier, et lorsque que le POS (Part Of Speech) est égale à l’un ou l’autre dépendant de la situation, nous rajoutons ce verbe dans une liste.

Identification des Entités Nommées

Traitement sur le corpus

Pour identifier les entités nommées dans notre corpus, nous avons besoin d'un nouvel outil, lui aussi fournis par Stranford :
Stanford Named Entity Recognizer (NER)

A partir de notre txt et stanford-ner-2016-10-31, nous avons utilisé la commande suivante :

java -mx600m -cp "*;lib/*" edu.stanford.nlp.ie.crf.CRFClassifier -loadClassifier classifiers/english.muc.7class.distsim.crf.ser.gz -outputFormat tabbedEntities -textFile Wikipedia_F_only.txt> Wikipedia_F_only_NamedEntities.tsv

Ainsi nous avions dans le même dossier un fichier Wikipedia_F_only_NamedEntities au format tsv qui contient tout le texte sous forme de ligne dont les composants sont séparés de tabulations avec :
Pour une ligne une entité nommée, sous la forme "Entité Nommée \t Type d'Entité Nommée \t texte brut jusqu'à la prochaine Entité Nommée" qu'il nous sera plus facile de traiter

Scripts Python

Après avoir essayé de traiter le fichier tsv avec le module csv de python en vain, nous avons choisi de le traiter comme un fichier txt et de se servir des marqueurs de ligne et de tabulation pour constituer les tableaux que nous utiliserons pour l'identification.

On découpe tout d'abord le fichier dans un tableau 2D de la forme :
EN[en numéro 1]= ("Jean Paul", "PERSON", "est parti manger des crêpes en")
EN[en numéro 2]= ("Bretagne", "LOCATION", "avec son ami")
EN[en numéro 3]= ("Pierre", "PERSON", "qu'il connaît depuis ")
EN[en numéro 4]= ("1996", "DATE", ".")

Ce tableau est ensuite utilisé pour trier ces entitées nommées par catégories dans les 4 tableaux suivants :
persons=[]#tableau de personnes
Fpersons=[]#tableau de personnes dont le nom commence par F
locations=[]#tableau de locations
dates=[]#tableau de dates

Expression régulière

Pour créer Fpersons nous avons besoin d'une reconnaissance par Expression régulière. Soit l'expression régulière "^(\w*\s)*F(\w*\s?)+", représentant un début de mot possible comprenant alors un nombre possiblement infini de lettres majuscules ou minuscules,

forcément suivi d'un espace, puis un F majuscule, forcément suivi d'au moins une suite de lettre minuscules ou majuscules et peut être un espace si le nom n'est pas terminé.
Si une EN est une personne et qu'elle correspond à cette regex, alors on l'ajoute au tableau Fpersons.

Après avoir créé chaque tableau, nous supprimons tous les doublons avec des fonctions spécifiques.
On a finalement :

```
#récupération des EN par catégories
persons=[]#tableau de personnes
Fpersons=[]#tableau de personnes dont le nom commence par F
pattern = re.compile("^(\w*\s)*F(\w*\s?)+")#regex F nom commençant par F
locations=[]#tableau de locations
dates=[]#tableau de dates
for x in EN:
        if x[1]=="PERSON":
                persons.append(x[0])
                if pattern.match(x[0]):#regex F nom commençant par F
                        Fpersons.append(x[0])
        if x[1]=="DATE":
                dates.append(x[0])
        if x[1]=="LOCATION":
                locations.append(x[0])
#surrpession des doublons
Fpersons=checkDoublon(Fpersons)
persons=checkDoublon(persons)
dates=checkDoublon(dates)
locations=checkDoublon(locations)
```

Les tableaux sont ensuite exportés aux format txt pour pouvoir être traités dans le script d'identification

Recherche de synonymes pour identification

La recherche des synonymes s'est basé sur le verbe 'killed', choix arbitraire représentant pour nous un bon point de départ. Le script python associé à cette recherche est nommé "RecuperationVerbesSynKill.py", et se décompose en plusieurs étapes et méthodes.

Tout d'abord, le corpus tokenisé en format XML est ouvert, puis nous mettons en place un parsing de ce dit document grâce à l'import de la libraire ElementTree (import xml.etree.ElementTree as etree). Ce fichier XML est ensuite traité par la fonction getVerbs, qui recherche les différents types de verbes présent dans le texte grâce à plusieurs tags (VB, VBD, VBG, VBN et VBZ). Ces verbes sont stockés, après avoir été lemmatisés, dans une liste, listOfVerbsInText. Le choix a été fait à partir d'ici de stocker ces verbes dans un fichier .txt, "Text des verbes", grâce à la méthode putListInTxt. Cette opération permet de gagner du temps de traitement lors de l'utilisation de cette liste, le programme n'est plus obligé de parcourir le corpus en entier.

La prochaine étape consiste à récupérer la liste des Synsets de ‘killed’ et d’y ajouter les synonymes de ces synsets. Cela est rendu possible à l’aide de la librairie WordNet de NLTK et deux méthodes, initKilledSynsets() et synKilled(). La première permet de trouver les synsets de ‘killed’, au nombre de 14. Un appel de fonction pour retirer les doublons est ensuite appelée (checkDoubleSynsets), faisant chuter ce nombre de synsets à 3. Cette liste est stockées dans listOfKilledSynsets, qui est ensuite modifiée dans la méthode synKilled(). Cette dernière permet de récupérer une liste contenant tous les synonymes des 3 synsets précédemment ajoutés. Cette méthode fait le tour de tous les synsets existant dans la librairie WordNet, et sous un seuil de 2.26 d’indice de ressemblance. La liste rendu contiendra 44 synonymes du mot ‘killed’, dont 4 doublons retirés par la suite, obtenant un nombre final de 40 synonymes.

Cette liste est enfin lemmatisée et, tout comme la première, ajoutée dans un fichier .txt pour gagner du temps de traitement par la suite, Text des synonymes de killed.

Pour terminer, la dernière étape consiste à comparer les verbes présents dans le texte aux synonymes du mot ‘killed’. Pour cela, on fait appel à la méthode findKillVerbsInText(), qui stocke les résultats dans une liste composée elle-même d’une liste en 2 dimensions. Cette dernière contient le verbe en question ainsi que l’index de celui-ci, autrement dit : “Le verbe kill est le 4501 sur 15042 verbes dans le texte”. Cet index nous permettra de récupérer l’emplacement dans le texte de ce verbe, et, par extension, la phrase qui le contient.

Identification

N’ayant pas eu le temps de mettre en place cette partie, voici les différentes étapes que nous aurions aimé développer pour parvenir à ce but :

1. Récupérer l’emplacement d’un synonyme de ‘killed’ dans une phrase à l’aide de son index dans la liste des verbes
2. Regarder si on trouve un nom en F faisant partie de la liste d'alexis dans la phrase qui contient le verbe précédent
3. Si oui, regarder si une date est présent dans la phrase, ou dans celle qui suit
4. Regarder si un deuxième nom, celui de la victime, est présent dans la phrase ou dans ce qui suit
5. Vérifier l’ordre d’apparition de ces noms, pour éviter que les victimes en F ne soit rendu comme tueurs

Conclusion

Malgré un corpus de base en dessous de nos espérances, nos traitements préliminaires sur celui ci nous paraissent plutôt bons, ainsi que les différentes mécaniques mises en place pour une identification future. Cependant, notre mauvaise gestion du temps parmi les différents projets qui composent ce cursus ne nous a pas permis de terminer ce que nous avions entrepris, et de la manière dont nous aurions souhaité le faire : la partie

identification a dû être abandonnée au profit de la rédaction de ce rapport et de la préparation de l'oral. Nous avons malgré tout pu délimiter les étapes nécessaires à la suite du traitement.