Rapport de projet d’Ingénierie Linguistique

Aurélien HERBIN, Morgane PAILLER, Nicolas PETITJEAN,

Laure POINT, Camille Rémi

Contents

[II. Introduction 3](#_Toc36389693)

[III. Extraction XML 3](#_Toc36389694)

[1. Présentation du langage XML 3](#_Toc36389695)

[2. Extraction des données du fichier XML 4](#_Toc36389696)

[IV. Traitement des données 4](#_Toc36389697)

[1. Recherche de règles générales 4](#_Toc36389698)

[2. Implémentations des règles en python 5](#_Toc36389699)

# Introduction

Le but du projet que nous avons étaient amené à développer cette année dans le cadre du cours d’Ingénierie Linguistique était de récupérer des pages XML importer de Wikipédia et pour une lettre choisie extraire du document tous les tueurs dont le nom commencé par cette lettre ainsi que d’extraire le nom et le nombre de ses victimes. Cet exercice est assez fréquent dans le Traitement Automatique des Langues car nous récupérons des données brutes (ici les fichiers XML) nous les rendons utilisables et ensuite nous travaillons dessus avec des bibliothèques comme *nltk* pour en avoir un traitement particulier.

Python est beaucoup utilisé dans ce genre de cas car c’est un langage de programmation orientée objet qui est facile d’accès et qui a un grand nombre de bibliothèques pour pouvoir satisfaire ces traitements.

# Extraction XML

## Présentation du langage XML

Le langage est un langage de balisage, le but était d’envoyer des pages sur le web et qu’elles puissent être traitées comme du code HTML, ce qui explique la similarité entre les deux langages. Le langage XML répond à une norme *unicode* c’est-à-dire qu’elle doit accepter les codages des caractères *UTF-8* et *UTF-16*. Pour simplifier un peu le propos on peut envisager un fichier XML comme un arbre avec une racine qui a généralement pour nom *mediawiki.*  Ensuite, on a plusieurs feuilles comme la feuille *siteinfo* et la feuille *page*. Il peut y avoir plusieurs feuilles *page* car cela va dépendre de si on exporte qu’une seule page ou plusieurs pages.

## Extraction des données du fichier XML

![A screenshot of a cell phone

Description automatically generated]()

‑ fonction qui permet de récupérer le texte des balises XML

La fonction *recuperationText()* est une fonction qui prends une chaine de caractère en paramètre et renvoie une chaine de caractère en sortie. Décortiquons-la un peu plus la variable qu’elle prends en entrée est le chemin vers le dossier XML. Une amélioration possible de cette fonction serait via une interface graphique de demander à l’utilisateur ou serait le dossier XML car l’inconvénient d’enregistrer dans le code le chemin c’est qu’on ne peut pas le changer autrement qu’en le cherchant dans le code.

Pour pouvoir extraire des données XML via python il faut utiliser la librairie *Etree* qui permet d’aller chercher le dossier XML et ensuite de l’exploiter via des fonctions comme *getRoot()*  qui renvoie un objet de type Root. Comme on le voit sur l’image on remplis une liste avec toute les balises *text* dedans ensuite on transforme via la méthode *join()* la liste en string et on retourne un assez grand texte au format *string* avec dedans toutes nos pages exportées.

# Traitement des données

## Recherche de règles générales

L’extraction des noms et prénoms des tueurs est généralement assez simple parce que dans le premier dossier que l’on utilise et qui se nomme : *listSerialKiller.xml*. Ils se présentent sous cette forme :

« *Martha Rendell : killed three stepchildren with chloridric acid in 1907-08; last woman to be hanged in Western Australia* »

Ce qui nous permet de faire un algorithme qui dirait que derrière chaque puce la première suite de Noms Propres représente le tueur. Il faut ensuite réfléchir à comment trouver ses crimes et il existe autant de moyens d’aborder ce sujet que de verbes pour designer un meurtre. En lisant 2-3 documents on a trouvés plusieurs verbes récurrents comme : *Poisoning, Assassinated, Demembred, Killed…* Une fois l’extraction de ces verbes est faite il faut voir si on peut déduire des règles.

On a donc notre mot cible qui est le verbe on va donc s’intéresser aux mots autours. Cela nous a permis de déduire des règles comme :

- Apres un *who* on aura toujours le verbe *murdered* et un nombre de personne comme par exemple : *who murdered five people …*

- Aprèsun « of », on sait que cela concerne les victimes :

• Cela peut être un **chiffre**, en nombre ou en lettre comme par exemple : *poisoner of three individuals*

• Cela peut être un **mot-clé** : en Xml, ça ressemble à [[mot-clé]] : si le mot-clé peut être un nom commun (comme le mot prince), on peut tomber sur un nom propre (qui nous intéresse, car ça va être celui de la victime)

- Au lieu de « of », on peut tomber sur un mot de « viser » comme par exemple *perpetrated* ***against***

## Implémentations des règles en python

Aurélien :

Le but du programme est de reconnaître avant tout, les personnes qui ont été tué par le tueur : isoler seulement les noms propres ne suffit pas car certains peuvent ne pas être des victimes ou être des noms de bâtiments. C’est pourquoi, il faut commencer à partir des mots (noms communs et verbes) en relation avec le meurtre. On les mettra dans une liste (dans le programme, il est nommé « kill »).

On constate alors qu’il peut y en avoir beaucoup : avec un seul mot, on peut partir sur d’autres par « dérivations » (en ajoutant un suffixe par exemple) ; ce mot est nommé un lexème. Par exemple, avec le mot « kill », on peut obtenir « killing », « killing », « killer », « killed », etc. À noter que si le lexème peut être un verbe, on peut le dériver pour en faire un nom commun.

Afin de réduire efficacement la liste des mots en relation avec le meurtre, on y met seulement des mots basiques : lorsqu’on analyse le texte, on « réduit » si possible chaque mot lu en leur lexème (en supprimant les suffixes par exemple).

On remarque aussi que ce soit le lexème ou ses dérivations, ces derniers suivent les règles citées précédemment. De ce fait, se contenter des lexèmes simplifient le programme tout en gardent une certaine qualité de lecture.

De l’autre côté, il faut évidemment reconnaître les noms propres, mais aussi les regrouper : en rapport avec une des règles d’implémentation, après un mot en relation avec le meurtre, le mot « them » se rapport à plus d’une personne.

Or, si avec « him » ou « her », il suffit de se rapporter au dernier nom propre enregistré, pour « them », il faut savoir jusqu’à combien de noms propres enregistrés, il faut prendre. De ce fait, il faut regrouper des noms propres proches, entre eux. Après avoir lu un nom propre, le programme compte la distance entre ce mot, le dernier nom propre enregistré : s’ils sont assez proches, le mot est mis avec le dernier nom propre enregistré (dans une liste) ; sinon, le programme met ce mot dans un nouveau groupe (en créant une nouvelle liste). À noter qu’en absence d’une virgule ou de « and », le programme rallonge artificiellement la distance afin d’éviter tout confusion : un point provoque la séparation de deux groupes de noms propres entre eux.

Par ailleurs, tout ce qui se trouve entre les noms propres est ignoré : on sait que sur un mot en relation avec le meurtre, l’énumération de noms propres est logiquement terminée et le programme force donc une nouvelle séparation avec les prochains qu’il va rencontrer.

En conséquent, la liste qui va contenir tous les mots du texte est une liste de liste : les noms propres proches entre eux sont regroupés dans une liste ; afin de rendre la lecture de cette liste plus fluide, tous les mots (incluent les noms propres isolés), seront mis dans une liste (qui contient qu’un seul élément). De ce fait, une fois ce regroupement terminé, il suffit de lire cette liste, d’identifier que le mot en relation avec le meurtre, puis déduire avec ce qui suit, quel(s) noms propres y est/sont associés (et donc les mettre dans une liste qui contient logiquement les noms des victimes du tueur). À noter que le nom du tueur sera ignoré pour des raisons évidentes…