Sorbonne Université

Projet ANDROIDE 2020/2021

Identification et indexation d'entités nommées

Etudiants :

- **DERRAS Khalil**
- **WILLAIME-ANGONIN Julien**
- **ZIDELMAL Smail**

Encadrant :

- **M. Gauvain Bourgne**

Table des Matières

1. Introduction :

La reconnaissance d'entités nommées, est la tâche de la catégorisation des entités dans le texte, son but est de détecter et d'étiqueter ces noms avec les concepts du monde réel qu'ils représentent, c'est une forme de traitement du langage naturel (NLP), un sous-domaine de l'intelligence artificielle. Le NLP s'intéresse au traitement informatique et à l'analyse du langage naturel.

Le but du projet sur lequel nous avons travaillé était d'améliorer un script python aidant les personnes souhaitant annoter un corpus de textes pour en tirer les noms propres désignant des entités dans les catégories choisies par ces personnes. Le programme va donc remplir les dictionnaires avec à la fois les noms et leurs synonymes (e.g. Joe Biden doit être sur la même ligne que Joseph R. Biden) et l'emplacement de leurs occurrences (nom du fichier, numéro de la ligne et du mot) afin d'avoir un retour possible vers le corpus d'origine pour pouvoir éventuellement mener des analyses sur les textes.

L'objectif de notre projet est de rendre ce processus le plus simple possible pour les utilisateurs (une seule commande dans le cas où l'entité est déjà présente dans les dictionnaires) en utilisant une comparaison des chaînes de caractères pour vérifier la présence de l'entité dans les dictionnaires et des techniques d'apprentissage pour suggérer une catégorie à l'utilisateur dans le cas où l'entité n'est pas dans le dictionnaire.

2. Description :

Nous avons inclus le cahier des charges directement à ce rapport et il correspond à cette partie (avec les solutions envisagées dans la partie Etat de l'art).

2.1 Contexte de projet :

Ce projet est dédié d'être utilisé dans le domaine des humanités numériques pour l'extraction des informations des larges articles, un outil prototype basique nous a été fourni au début de projet et le but c'était d'améliorer cet outil pour le rendre plus facile à utiliser et répondre mieux aux besoins des utilisateurs.

2.2 Fonctionnalités déjà présentes dans le prototype :

- L'identification des entités par des règles grammaticaux
- Demander à l'utilisateur de mettre le mot identifié dans un dictionnaire

2.3 Objectifs :

L'outil final doit contenir les fonctionnalités suivantes :

- Identification et d'indexation d'entités nommées par construction d'un dictionnaire d'alias
- Détection et étiquetage des ambiguïtés, dans un corpus important et structuré de textes
- Faciliter la construction des dictionnaires et l'annotation des corpus en facilitant le retour au texte
- La mise en forme de l'ancienne version de code

3. Etat de l’art :

Reconnaissance des entités nommées (Named Entity Recognition : NER) :

NER est une technique d'extraction d'informations pour identifier et classer les entités nommées dans le texte. En détail, il s'agit d'un processus dans lequel un algorithme prend une chaîne de texte (phrase ou paragraphe) comme entrée et identifie les noms pertinents (principalement des personnes, des lieux et des organisations…) qui sont mentionnés dans cette chaîne.

Les articles de recherche qu’on a consulté proposent un modèle d’apprentissage pour faire la prédiction des types des entités.

Approches d’apprentissage :

- **A- Tâche de classification multi-classes** où les entités nommées sont nos étiquettes afin que nous puissions appliquer différents algorithmes de classification. Le problème ici est que l'identification et l'étiquetage des entités nommées nécessitent une compréhension approfondie du **contexte** d'une phrase et de la séquence des étiquettes de mots qu'elle contient, ce que cette méthode ignore.

- **B- Conditional Random Field (CRF) model**. Il s'agit d'un modèle graphique probabiliste qui peut être utilisé pour modéliser des données séquentielles telles que des étiquettes de mots dans une phrase. Le modèle CRF peut capturer les caractéristiques des noms présents et passés dans un arrangement, mais il ne peut pas comprendre la configuration des étiquettes avant ; cette lacune, ainsi que l'ingénierie des fonctionnalités supplémentaires impliquées dans la formation d'un modèle CRF, rend son adoption par l'industrie moins attrayante.

- **C-** Le type de réseau des neurones qui se répète le plus et qui fonctionne le mieux pour s'attaquer au problème du NER étant donné que le texte est un format de données séquentiel c’est **Long short Term Memory (LSTM),** LSTM bidirectionnels car l'utilisation d'un LSTM standard pour faire des prédictions ne prendra en compte que les informations « passées » dans une séquence du texte. Pour le NER, étant donné que le contexte couvre les étiquettes passées et futures dans une séquence, nous devons prendre en compte les informations passées et futures. Un LSTM bidirectionnel est une combinaison de deux LSTM - l'un va de « droite à gauche » et l'autre de « gauche à droite ».

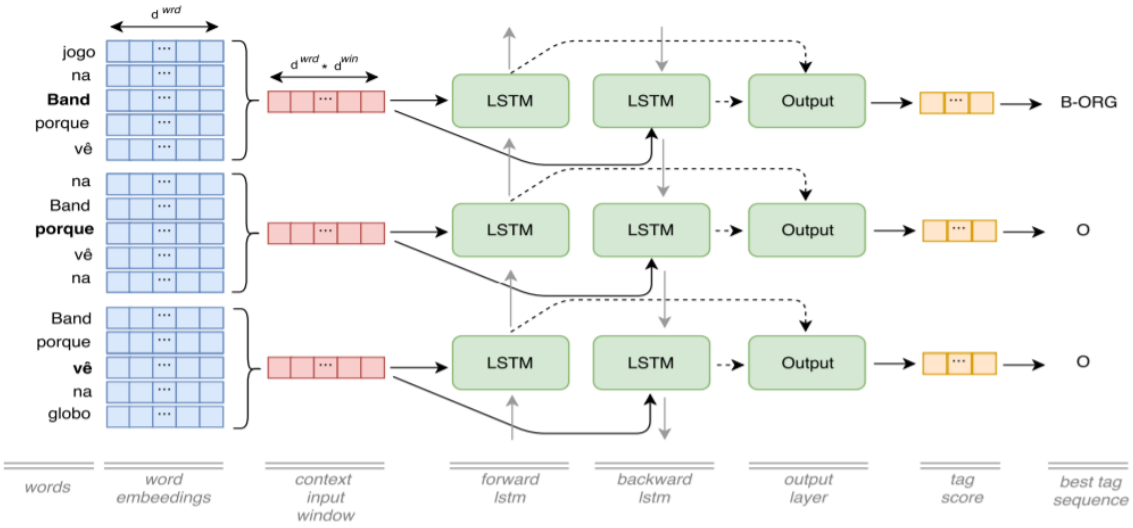


Figure 2 de l'article [11] de bibliographie : l'architecture d'une bi-lstm.

Embedding :

Embedding est un mappage d'une variable discrète - catégorielle - à un vecteur de nombres continus. Dans le contexte des réseaux de neurones, les Embeddings sont des représentations vectorielles continues apprises de faible dimension de variables discrètes. Les Embeddings de réseaux de neurones sont très utilisés car ils peuvent réduire la dimensionnalité des variables catégorielles et représenter de manière significative les catégories dans l'espace transformé.

En NLP, Words Embedding c'est le fait de soumettre les mots au processus d'Embedding, où ils sont convertis en vecteurs de nombres. Ces vecteurs capturent la fonction grammaticale (syntaxe) et le sens (sémantique) des mots, ce qui nous permet d'effectuer diverses opérations mathématiques sur eux.

4. Réalisation :

Nous avons divisé le travail en deux pour pouvoir travailler parallèlement sur les deux parties principales du projet : l'aspect interface avec l'utilisateur, implémentation des outils et l'aspect traitant de l'apprentissage pour la suggestion de classe pour les entités.

4.1 Description du programme final :

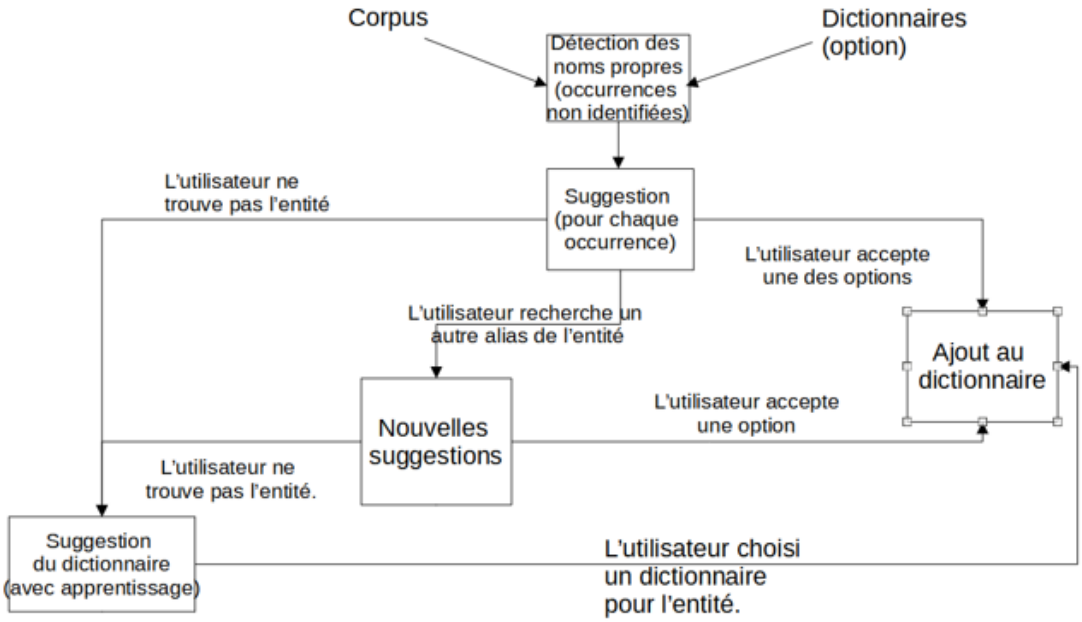


Diagramme représentant le processus du programme.

La détection se fait en fonction des majuscules au début des mots et si une occurrence est déjà présente dans un des dictionnaires alors elle est retirée de la liste des occurrences à traiter.

La suggestion d'entité se fait en fonction de la proximité entre la chaîne de caractères et les alias de toutes les entités avec un indice de Sørensen-Dice (plus l'indice est grand plus les chaînes sont proches), on affiche à l'utilisateur les 5 meilleurs choix et l'utilisateur à alors la possibilité de choisir entre l'une de ces 5 propositions, rechercher un autre alias de l'entité ou directement passer au choix d'un dictionnaire pour rajouter une nouvelle entité.

Dans le cas où l'utilisateur recherche un alias de l'entité, le procédé est le même, on compare les chaînes de caractères avec celle donnée par l'utilisateur et on renvoie les 5 plus proche pour que l'utilisateur choisisse l'une d'elle ou décide de créer une nouvelle entité dans un dictionnaire.

Dans le cas où l'utilisateur décide d'ajouter une nouvelle entité, le système va proposer soit un dictionnaire ne soit rien et l'utilisateur va choisir dans quel dictionnaire l'entité va être créée.

Dans tous les cas avant l'ajout de l'occurrence au dictionnaire, on demande à l'utilisateur si la chaîne de caractères traitée peut être ambiguë ou non. Si c'est le cas, l'occurrence courante est rajoutée au dictionnaire au bon endroit et on renvoie à l'utilisateur l'occurrence suivante de la même chaîne (si elle existe) dans le cas contraire toutes les occurrences de la chaîne sont ajoutées au même endroit dans le dictionnaires choisis (ou pour l'entité choisie si elle est déjà dans le dictionnaire).

L'ajout d'une occurrence existant déjà dans un des dictionnaires consiste en l'ajout de l'alias dans la liste des alias de l'entité et de l'ajout de l'occurrence (ou des occurrences) dans la liste des occurrences de l'entité.

Si l'entité n'existe pas alors on créer un nouvel identifiant (idmax+1) et on ajoute l'alias ainsi que la ou les occurrences de l'alias dans le texte.

Dans ce système chaque action est définitive et a des conséquences sur les suggestions suivantes. Cependant, la sauvegarde dans les fichiers dictionnaires ne se fait qu'à l'arrêt du programme.

4.2 Interfaces / Implémentation :

Modifications du script initial :

Initialement, le script permettait de parcourir un corpus spécifique et d'en tirer les chaînes de caractères supposées être des noms propres. Ensuite un deuxième script permettait à l'utilisateur d'attribuer un dictionnaire pour chacune des chaînes de caractères.

Le script initial était uniquement dédié à un seul corpus et prévu pour une seule liste de dictionnaires, le premier travail que nous avons effectué à été de généraliser ce script afin de permettre l'utilisation de ce programme avec des corpus et des dictionnaires différents. Finalement notre système permet de paramétrer les programmes à l'aide d'un fichier de configuration.

Stockage des données :

Un autre point sur lequel nous avons travaillé est le fait que le script initial traitait tous les homographes d'un seul coup, il n'y avait donc pas la possibilité de séparer les entité ayant la même chaîne de caractère pour les désigner. Le système de stockage des mots dans les dictionnaires a donc de être modifié pour pouvoir différencier les homographes, de plus nous devions aussi pouvoir retrouver les occurrences de mots dans le texte. Nous avons donc finalement décider de stocker les entités dans les dictionnaires de la façon suivante :

<identifiant de l'entité (un entier)> :<alias1>|<alias2>|... :(<fichier>,<ligne>,<numéro mot>)

L’identifiant de l’entité est unique parmi les dictionnaires ce qui permet d’éviter les ambiguïtés entre les identifiants. Le but étant de référencer les entités, les occurrences ne sont pas associées à un alias spécifique.

Informations données à l’utilisateur :

Dans le script initial seul la chaîne de caractère était affichée à l’utilisateur. Pour pouvoir différencier des entités qui aurait un alias en commun, il nous a fallu rajouter du contexte à la chaîne de caractère. C’est pourquoi on affiche, en plus de la chaîne, la ligne de texte qui contient la chaîne en question ainsi que le nom du fichier d’où est tirée la ligne.

Nous avons rajouté au système initial un étape supplémentaire qui donne à l’utilisateur les 5 entités ayant un alias le plus proche possible de la chaîne détectée. Nous fournissons à l’utilisateur ces entités dans l’ordre décroissant de proximité entre les deux chaînes.

Nous avions envisagé au départ l’utilisation d’un système d’entity linking mais la plupart de ces systèmes n’étaient pas adaptés à notre situation, c’est à un dire un contexte où nous n’avons peu ou pas d’information dans les dictionnaires au départ.

4.3 Reconnaissance des entités nommées (Named Entity Recognition : NER) :

Pour faciliter la construction des dictionnaires et l’annotation des corpus :

La solution adoptée :

Dans notre projet on a opté pour un modèle séquentiel en utilisant les LSTMs qui prédit le type de mot en se basant sur son contexte dans une séquence des mots.

L’approche que nous avons adoptée c’est d’utiliser une couche LSTM bidirectionnelle au niveau des mots et appliqué des Embeddings de mots. Bien que cette approche soit simple et donne souvent de bons résultats, il existe des améliorations potentielles. Si nous n’avons pas vu un mot, nous devons l’encoder comme inconnu et en déduire sa signification par les mots qui l’entourent. Souvent, le suffixe ou préfixe de mot contient beaucoup d'informations sur la signification du mot. L'utilisation de ces informations est très importante si vous traitez avec des textes contenant beaucoup de mots rares et que vous vous attendez a beaucoup de mots inconnus au moment de l'inférence.

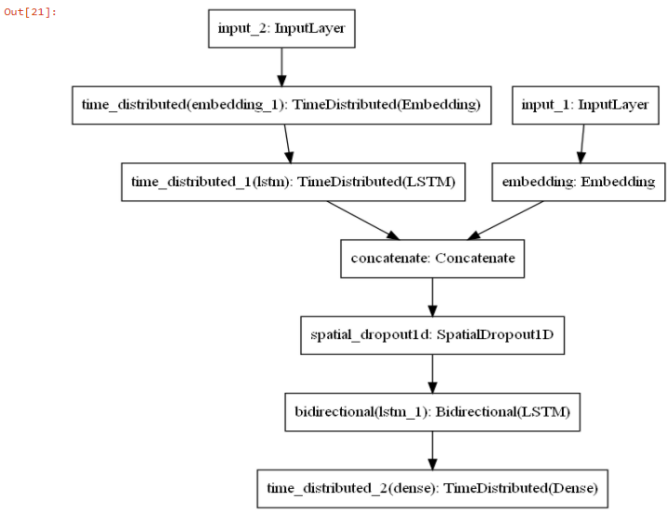
Pour encoder les informations au niveau des caractères, nous avons utilisé des Embeddings de caractères et un LSTM pour encoder chaque mot en un vecteur. Nous avons utilisé essentiellement tout ce qui produit un seul vecteur pour une séquence de caractères représentant un mot. Ensuite, nous avons transmis le vecteur à un autre LSTM avec l'Embedding du mot appris.

L'implémentation de solution :

- La première chose à faire c'est réaliser un Dataset annoté à partir des dictionnaires déjà remplis pour cela on a fait un script de conversion pour stocker les entités annotées avec leurs contextes dans un fichier csv comme des séquences des mots de la façon suivant :

	Line#	Word	Tag
0	Line0	De	NoTag
1	Line0	tous	NoTag
2	Line0	les	NoTag
3	Line0	poëtes	NoTag
4	Line0	de·l'	NoTag
5	Line0	Antiquité,	NoTag
6	Line0	Pindare	OEUVRES
7	Line1	est	NoTag
8	Line1	certainement	NoTag
9	Line1	aujourd'	NoTag
10	Line1	hui	NoTag
11	Line1	le	NoTag
12	Line1	plus	NoTag
13	Line1	éloigné	NoTag
14	Line1	de	NoTag
15	Line1	nous.	NoTag
16	Line1	Plus	NoTag
17	Line1	qu'	NoTag
18	Line1	aucun	NoTag

- Les couches LSTM n'acceptent que des séquences de même longueur. Par conséquent, chaque phrase représentée sous forme de nombres entiers doit être complétée (appliquer un padding) pour avoir la même longueur. Nous travaillerons avec la longueur maximale de la séquence la plus longue et remplirons les séquences les plus courtes pour y parvenir. Ensuite, on divise notre dataset en données de test et données d'apprentissage
- L'implémentation de l'architecture de modèle :
  Dans cette architecture, nous travaillons principalement avec trois couches (embedding, bi-lstm, lstm) :



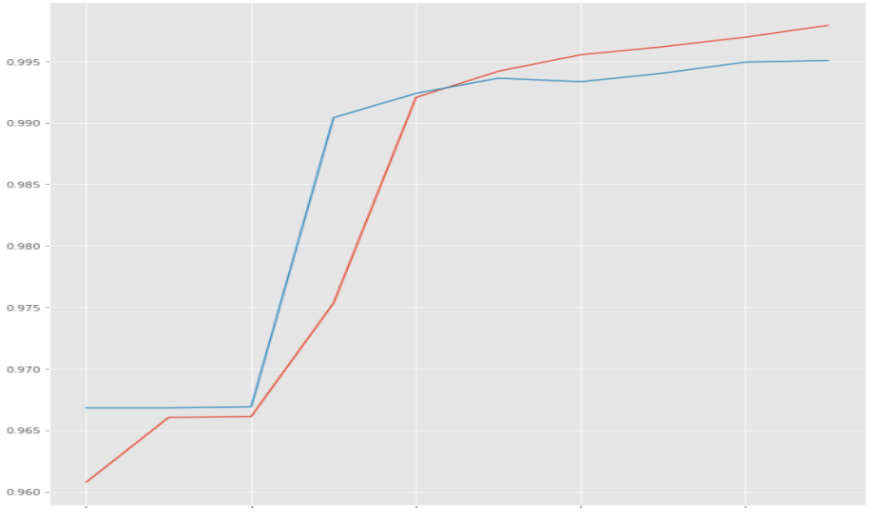
Comme indiqué dans la figure, il s'agit de combiner deux listes Lstm, une pour les caractères et l'autre (bi-lstm) pour les mots.

Comme il s'agit d'un lstm bidirectionnel pour les mots, nous aurons des sorties vers l'avant et vers l'arrière, qui combine ces sorties avant de passer à la couche finale TiemeDistibuted(Dense) qui produit le résultat .

- Nous allons apprendre le modèle avec model.fit en enregistrant et visualisant la perte à chaque époque.
- Enregistrer les poids et l'architecture de modèle pour le réutiliser dans notre programme principal.

Résultats /Evaluations :

En utilisant le « accuracy test » On a obtenu des très bons résultats sur les données de test.



Voici quelques prédictions sur les données de test (la case à gauche et l'autre c'est la valeur prédite par le modèle) :

Vous trouvez sur notre repository Github un **notebook** expliquant les détails de l'implémentation de modèle de chargement données jusqu'à le test et l'enregistrement.

5. Documentation de projet :

Description des fichiers de projet :

- Filter.py : Contient tous ce qui est annotation, suggestions et interface
- Identification.py : Contient l'identification des entités et la vérification si l'occurrence est présente dans un dictionnaire.
- Model.py : Contient L'implémentation de fonction de prédiction à partir de modèle déjà appris.
- Main.py : Contient Le programme principale.
- Modèle d'apprentissage. ipynb : Contient les détails de l'implémentation de modèle d'apprentissage .
- generate model input.py : le script qui génère le dataset en fichier csv pour l'utiliser dans l'apprentissage
- dataAnnotated.csv: le dataset
- Model.h5, Model.json : les poids et l'architecture de modèle après apprentissage.

Dependences:

- Keras
- Numpy
- Pandas
- Textdistance
- Pypandoc
- Configparser

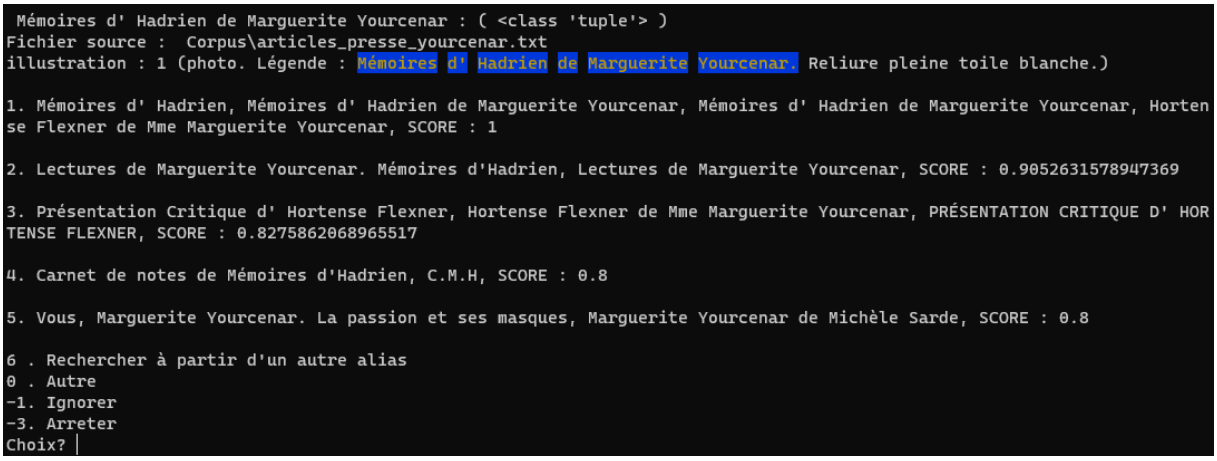
Carnet de Bord :

- https://drive.google.com/file/d/1cxC_oFifpr0KWp20BhksPoT12-xuoSux/view?usp=sharing

6. Manuel d'utilisation :

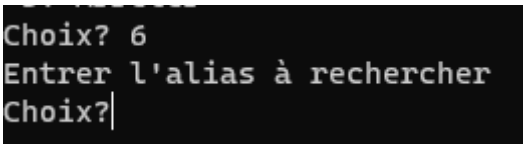
- Pour configurer ce programme, il faut utiliser le modèle du fichier config.txt, soit le modifier lui-même soit utiliser le modèle pour créer un nouveau fichier et l'utiliser en paramètre de la ligne de commande pour lancer le programme.

- Pour lancer le programme il faut utiliser la commande « python main.py » dans un terminal ouvert dans le répertoire du programme.

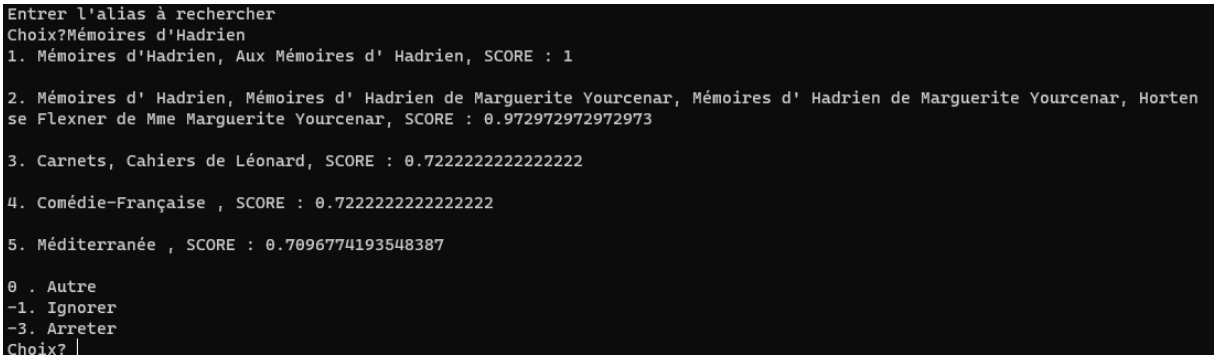


- On peut voir ici :
    - Le nom détecté par le programme (première ligne)
    - Le fichier d'où vient ce nom (deuxième ligne)
    - La ligne d'où provient ce nom qui est surligné (troisième ligne)
    - Les 5 meilleures correspondances par rapport au nom trouvé (5 premières options)
    - Le choix de chercher un autre nom qui correspond à la même chose (Choix 6)
    - Le choix 0 qui correspond à choisir un dictionnaire pour ajouter une nouvelle entité dans l'un des dictionnaires.
    - Le choix -1 qui permet d'ignorer le mot courant
    - Le choix -3 qui permet de quitter le programme et qui permet de sauvegarder dans les fichiers les affectations précédant /!\ LE CHOIX -3 OU LA FIN DE TOUS LES NOMS SONT LES SEULS MOYEN DE SAUVEGARDER L'AVANCEMENT. Toute autre méthode pour quitter le programme perd les données reçues jusque-là.

- Les suggestions sont faites à partir d'une comparaison entre les chaînes de caractères avec l'indice de Sørensen-Dice.



- En choisissant l'option de la recherche, il suffit de rentrer l'alias en question pour faire la recherche.



De nouveau un choix avec des suggestions (basées sur le même principe qu'à la première étape). On peut choisir ici, par exemple de prendre le choix « Autre » pour ajouter une entité aux dictionnaires.

Dans ce cas il y a 2 cas de figures :

Le programme trouve, grâce au système d'apprentissage, une suggestion de dictionnaire pour le nom :

Ici le premier choix correspond à la suggestion que fais le programme, les autres propositions sont aussi disponibles.

- Le programme ne trouve pas de suggestion :

Le choix se fait entre tous les dictionnaires sans aucun mis en avant.

- Dans tous les cas un choix se termine par cette décision :

S’il n’y a pas d’ambiguïté possible sur le nom proposé, toutes les occurrences de ce mot sont ajoutées dans le dictionnaire au même endroit, sinon seule l’occurrence dont le contexte a été affiché dans la première étape est ajoutée au dictionnaire.

/!\ TOUTE ACTION EST DÉFINITIVE, ON NE PEUT PAS CORRIGER LES ERREURS DANS L’EXÉCUTION DU PROGRAMME.

7. Conclusion :

Le cœur de notre projet était d'améliorer et de généraliser un script permettant d'annoter plus facilement des textes afin d'en retirer les entités nommées.

Nous avons donc, pour résumer, donné la possibilité à l'utilisateur de différencier des entités ayant des alias en communs, pour que l'utilisateur puisse faire la différence entre ces homographes nous lui présentons le contexte de l'entité ; nous avons aussi permis à l'utilisateur de rechercher un synonyme dans le cas où une entité pourrait avoir plusieurs nom très différents (par exemple dans le cas d'un auteur et de son pseudonyme) ; enfin nous avons aussi créer un système qui permet d'aider l'utilisateur à choisir le dictionnaire dans lequel ajouter une entité qui ne serait pas présente dans les dictionnaires de base à l'aide de l'apprentissage automatique.

Notre programme pourrait à son tour être amélioré notamment en effectuant plus de tests et d'ajustement pour rendre les suggestions plus précises voire utiliser des système d'entity linking pour trouver une même entité malgré des alias très différents. De même le système d'apprentissage pour la suggestion de dictionnaire pourrait être plus testé pour améliorer la fiabilité de la réponse et ainsi rendre le processus d'annotation encore plus simple pour les utilisateurs. Une dernière amélioration possible serait au niveau de l'ergonomie du programme qui est plutôt rudimentaire, un affichage graphique pourrait aussi faciliter la tâche des personnes pouvant utiliser ce programme.