Projet TATIA : Rapport :

Conception et implémentation d’un système de questions réponses en langue naturelle sur des données structurées avec système de reconnaissance vocale intégré

Les systèmes question/réponse sont des systèmes qui répondent à une question posée en langage naturel, par l’extraction d’une réponse précise à partir d’un corpus de documents. Le projet que nous avons réalisé ici s’appuie sur base *DBpedia* qui propose une version structurée et normalisée au format du web sémantique des contenus de *Wikipedia*. Le système que nous avons conçu requiert l’utilisation d’un **interpréteur Python** afin de permettre son exécution ainsi que d’une **connexion internet active** afin d’extraire les réponses de la base *DBpedia*. Le programme a en particulier été testé avec Python en **version 3.9 et 3.9.1**, le programme devrait cependant fonctionner sous des versions plus anciennes ou plus récentes de **Python 3**. De plus un certain nombre de packages Python supplémentaires doivent être installés sur la machine afin de permettre l’exécution du programme. Le nom des packages et les différentes commandes d’installation sont détaillés dans la section ci-après.

**Prérequis au lancement du programme**

En plus **d’un interpréteur Python et d’une connexion internet active,** des packages Python supplémentaires sont nécessaires au fonctionnement du code de ce projet. La liste ci-après contient le nom des packages à installer ainsi que la commande permettant son installation avec le gestionnaire de paquet ***PIP*** accompagné d’une brève description (nous reviendrons en détail sur le rôle concret des éléments apportés par ces packages dans notre projet plus tard dans ce rapport).

Les packages peuvent être installés séparément ou d’un seul coup avec les 2 lignes suivantes :

pip install spacy bs4 lxml deep-translator SpeechRecognition gtts pyglet PyAudio

+ python -m spacy download fr\_core\_news\_lg

**Packages indispensables pour le fonctionnement du projet :**

* [**Spacy** :](https://spacy.io/usage) pip install spacy

*SpaCy est une bibliothèque logicielle Python libre de traitement automatique des langues. Cette bibliothèque inclut par exemple des outils tels qu’un tokenizer ou un PoS tagger, mais a aussi pour grand avantage d’inclure un NER prenant en charge la langue française.*

* [***Modèles statistiques pré-entraînés en français pour Spacy :***](https://spacy.io/models/fr)

python -m spacy download fr\_core\_news\_lg ***(après installation de spacy!)***

*Installation d’un modèle français pré-entraînés pour Spacy permettant la prédiction des entités nommée et la détermination des dépendances syntaxiques. Il s’agit ici d’un réseau neuronal convolutif formé sur UD French Sequoia et WikiNER en français.*

***Note : un bug interne à numpy version 1.19.4 peut faire planter, dans ce cas revenir à la version 1.19.3 :*** pip uninstall numpy ***et*** pip install numpy==1.19.3

* [***Beautiful Soup 4:***](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)pip install bs4

*Bibliothèque Python permettant d'extraire des données de fichiers HTML et XML.*

* [***Lxml:***](https://lxml.de/installation.html)pip install lxml

*Bibliothèque permettant le traitement de XML et HTML dans le langage Python.*

* [***Deep Translator:***](https://pypi.org/project/deep-translator/)pip install deep-translator

*Bibliothèque permettant la traduction entre différentes langues de manière simple en utilisant plusieurs traducteurs.*

**Packages nécessaires aux fonctionnalités vocales :**

* [***Speech Recognition:***](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)pip install SpeechRecognition

*Permet la transformation de phrases orales en texte écrit*

* [***gTTS:***](https://lxml.de/installation.html)pip install gtts

*Permet la transformation de texte écrit en fichier mp3*

* [***Pyglet:***](https://pypi.org/project/deep-translator/)pip install pyglet

*Permet la lecture de fichier mp3*

* [***PyAudio:***](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)pip install PyAudio

Outils assurant divers traitements audios nécessaires au fonctionnement de certaines fonctionnalités de reconnaissance vocale du projet

**Note concernant l’installation de PyAudio :**

Dans certains cas l’installation de PyAudio avec la commande *« pip »* peut s’avérer capricieuse, en particulier avec des versions récentes de Python et/ou de la commande *pip*. On notera ici deux solutions qui devraient permettre la résolution de ce problème selon le système d’exploitation utilisé.

Avec **Windows** les deux commandes suivantes devraient permettre la bonne installation de PyAudio : pip install pipwin suivi de pipwin install pyaudio

Sous Linux (**Ubuntu)** les deux commandes suivantes devraient permettre la bonne installation de PyAudio : sudo apt-get install -y portaudio19-dev suivi de pip install PyAudio

*Note : Il est possible d’exécuter le programme sans bénéficier des fonctionnalités liées à la reconnaissance ou la lecture vocale. L’installation des packages cités dans cette dernière partie ne sont alors pas nécessaires. Se reporter à la section «****configuration du projet****» plus loin dans ce rapport.*

**Choix de Spacy**

Le système de question/réponse que nous avons conçu s’appuie principalement sur les outils apportés par la bibliothèque **Spacy**. Nous avons effectué le choix de Spacy pour plusieurs raisons. Tout d’abord nous souhaitions concevoir notre système en français et Spacy possède des modèles statistiques dans cette langue ainsi qu’un support natif du *Named-Entities Recognition (****NER****)* pour le français ce qui n’est pas le cas de ces principaux concurrents comme NLTK. De plus les algorithmes de tokenization utilisées dans la bibliothèque Spacy sont efficaces et rapides et la bibliothèque utilise une approche orientée objet facilement lisible. De plus certaines fonctionnalités tels qu’un support de la méthode d’apprentissage « word embedding » sont présentes avec Spacy alors qu’elles ne sont par exemple pas supportées par NLTK.

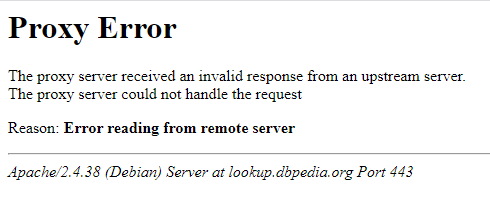
**Fonctionnement global du traitement question / réponse :**

**Cette introduction a pour but d’expliciter de manière générale le fonctionnement du programme que nous avons conçu avant d’entrer plus en détail dans le rôle de chaque fonction. Le rôle général du système de question / réponse que nous avons conçue consiste à prendre une question saisie par l’utilisateur en entrée et à afficher une réponse à cette question en sortie.**

**La 1ère étape consiste donc à extraire de la question posée les éléments qui peuvent nous permettre d’effectuer une requête capable d’afficher la réponse.** Pour cela nous nous sommes principalement appuyés sur des méthodes découlant d’une analyse lexicale (**tokenization**), d’un étiquetage morpho-syntaxique (**POS tagging** [part-of-speech tagging]) et sur une reconnaissance d'entités nommées (**NER** [named-entity recognition]). Ces différentes étapes nous permettent, par exemple, d’extraire les mots-clés les plus importants dans la question (fonction **get\_hotwords**). Afin d’identifier le type de questions posées et de récupérer les éléments nécessaires pour effectuer notre requête nous nous sommes également appuyés sur l’utilisation **d’expressions régulières associés à l’analyse NER** de la question (**fonction exp\_reg**). Cette étape nous permet d’identifier le type de recherche induite par la question (personne, lieu, date, etc) et de conserver les seuls mots nécessaires à l’exécution de la requête.

**La 2ème étape consiste à identifier la page dbpedia sur laquelle se trouve la réponse que l’on recherche.** Afin de pouvoir identifier précisément le nom de la page recherchée on se base sur le mot-clé principal identifié lors de l’étape précédente dans la question et on utilise l’API **DBpedia Lookup** qui nous renvoie le label précis de la page dbpedia correspondant à ce mot-clé (fonction **lookup\_keyword**). L’API nous permet ainsi d’effectuer une requête http renvoyant une page xml associé au mot-clé recherché et grâce à la bibliothèque *BeautifulSoup4* et au parser inclus dans la bibliothèque *Lxml*, on récupère le label exact nous permettant d’accéder à la page dbpedia contenant l’information recherchée. L’API fonctionnant en anglais, on utilise une étape de traduction intermédiaire sur le mot-clé de la question (français vers anglais).

**Note importante : *Nous avons malheureusement fait face à un problème inattendu à cette étape ayant fortement impacté le projet sur lequel nous travaillons. En effet nous utilisions depuis le début l’API suivante :*** [***https://lookup.dbpedia.org/api/search/KeywordSearch***](https://lookup.dbpedia.org/api/search/KeywordSearch) ***(***[***https://github.com/dbpedia/lookup***](https://github.com/dbpedia/lookup)***) qui semble avoir rencontré de forts dysfonctionnements à partir de fin décembre la rendant inutilisable.***



***En conséquence nous avons décidé de changer l’API utilisée :***

[***http://akswnc7.informatik.uni-leipzig.de/lookup/api/search***](http://akswnc7.informatik.uni-leipzig.de/lookup/api/search) ***(***[***https://github.com/dbpedia/dbpedia-lookup***](https://github.com/dbpedia/dbpedia-lookup)***)***

***Le programme continue avec cette nouvelle API de fonctionner globalement de la façon dont nous l’avions conçu mais cela a impacté la dernière ligne droite du projet ainsi que quelques recherches par mot-clé lors de cette étape qui s’effectuaient correctement avec l’API initialement utilisée mais pas avec celle-ci (par exemple le mot-clé « Marseille » de type ville renvoyé bien le label correspondant à la page dbpedia de la ville sur la première API mais pas sur la nouvelle, nous avons cependant implémenté de nouvelles solutions pour régler ou contourner ces problèmes lorsque cela été possible).***

**La 3ème étape consiste à exécuter une requête SPARQL sur la page dbpedia identifiée afin de récupérer la donnée correspondant à la réponse à la question.** Pour effectuer cette requête on s’appuie sur la page dbpedia identifié à l’étape précédente ainsi qu’aux autres mots-clés que l’on avait extrait de la question et qui nous permettent de savoir quelles données extraire de la page dbpedia. La requête elle-même est effectuée en se servant la fonction **« query »** directement via [l’interface dbpedia](https://dbpedia.org/sparql) et la formulation de la requête en se servant des mots-clés et de la page précédemment identifiée est effectuée par les fonctions « **requete\_dbpedia** » ou « **requete\_dbpedia\_multiple** » selon le type de réponse attendue (une seule réponse ou plusieurs réponses [par exemple si on cherche des créateurs pour un site web il peut y en avoir plusieurs mais si on cherche le roi d’un pays il n’y en a qu’un, cela dépend donc du type de la question]).

**Enfin la dernière étape consiste simplement à renvoyer le résultat de la requête qui devrait répondre à la question posée initialement par l’utilisateur.**

**Configuration du projet :**

Il est laissé à l’utilisateur du code de ce projet la possibilité d’effectuer quelques paramétrages afin d’activer ou de désactiver certaines fonctionnalités. Pour cela l’utilisateur doit se reporter à la fonction « **main** » du code contenu dans le fichier Python (à la fin) et configurer les différents booléens pré-configurés soit sur « **True** » soit sur « **False** » selon la configuration désirée. La liste des différents booléens configurables et des conséquences associées est la suivante :

**gui : *True*** pour activer **l’interface graphique*, False*** pour afficher les entrées et sorties dans **la** ***console*** de façon purement **textuelle** sans mise en forme

**vocal : *True*** pour activer **les fonctionnalités de reconnaissance et de lecture vocale*, False*** pour désactiver toutes fonctionnalités vocales (*Note : Il n’est pas possible d’activer les fonctionnalités vocales dans le cas où l’interface graphique est désactivée).*

**exemple\_questionsxml: *True*** pour afficher et répondre par défaut aux exemples de questions contenues dans le jeu de donnée « *questions.xml* » dès le début du programme sans action de l’utilisateur (lorsque les questions sont toutes traitées la main est donné à l’utilisateur afin qu’il puisse entrer ses propres questions)***,*** configurez ***False*** pour ne pas traiter ce jeu de donnée par défaut.

**exemple\_autres:** Même principe que ci-dessus mais pour des exemples autres que ceux du jeu de donnée « *question.xml* »

**Type de questions prises en compte dans notre programme :**

Question de type « qui est… (nom de personne » -> renvoi une description générale de la personne, « qui est le président / maire de … (nom de pays / ville) » -> renvoi nom du président, etc.

*Note : les réponses renvoyées correspondent aux données présentes dans la base dbpedia et ne sont pas forcément à jour.*

**Détail du programme par fonctions :**

**affichage\_reponse**

**reponse(*question*)**

**exp\_reg(*question*)**

**query(*q, epr*)**

**lookup\_keyword(*requete, type, translate=True*)**

**json\_load(*requete, predicate, entity\_of\_type*)**

**requete\_dbpedia[\_multiple](*requete, predicate, entity\_of\_type="dbo"*)/get\_abstract(*requete*)**

* Token : permet de tokenizer la phrase
* etc (pour chaque fonction).

**Difficultés rencontrées :**

Créer des expressions régulières ou autre qui sont assez générales pour prendre en compte un large spectre de questions.

**Evaluation de notre système :**

Recall, précision, F-measure (voir dernier TP)