Projet3 – Chaîne de traitement pour Google News

I - Introduction

Le projet a pour but de lire les flux RSS de Google News et d'en déduire des mots-clés permettant au post rédigé de se positionner dans le cluster de la catégorie d'articles visée.
Ainsi, l'application permet de rédiger des posts basés sur les mots-clés et de les publier.

II - Fonctionnalités

- Il est possible de choisir la catégorie de Google News et les mots-clés sont mis à jour.
- La possibilité de générer les mots-clés à partir de l'article entier et non pas seulement à partir du sommaire proposé par Google News.
- Le site supporte les news en anglais et français.
- Rédaction de posts et leur publication.
- Des images sont proposées en fonction des mots clés (pour la création d'article)
- Ensuite nous pouvons visualiser les articles sur la page d'accueil.

III - Spécificités techniques

Le projet a été réalisé en Python (3.x) et Flask pour la partie application web, les différentes données sont stockées dans une base de données MySql.
Le front-end a été réalité en HTML5 / Css (Bootstrap) / Jquery.

La première étape de la chaîne de traitement a été la lecture du flux RSS Google News et l'extraction du cluster de données sur lequel nous devions alors nous positionner en calculant les mots-clés dans la seconde partie du traitement.

La lecture et analyse du flux RSS a été effectué avec l'aide de la bibliothèque "feedparser" (https://github.com/kurtmckee/feedparser), permettant de parcourir les nodes du flux de la catégorie et d'accéder aux informations qui nous intéressent

pour la constitution du cluster, pour lesquelles nous proposons soit une extraction “simple” ou “avancée”:

- Extraction “simple”: ici le titre et le “sommaire” proposé par Google News de chaque article sont extraits.

- Extraction avancée: l’url de chaque article est récupérée, puis son titre et contenu complets sont extraits à partir de la page d’origine, grâce à la bibliothèque “newspaper” (https://github.com/codelucas/newspaper).

Ainsi, la seconde solution permet d’avoir potentiellement des résultats finaux , mais le coût en terme de performance nous a donné l’idée de laisser l’utilisateur choisir la solution qu’il souhaite utiliser.

Ces informations sont alors “nettoyées” des symboles html qu’elles pourraient contenir, et sont passées ensuite à l’étape de lemmatisation.

Pour la lemmatisation nous avons opté pour TreeTagger qui est bien plus rapide que nltk et gère plusieurs langues, dans le cadre de ce projet seul l’anglais et le français sont gérés mais il suffit de rajouter les fichiers de la langue que l’on souhaite pour qu’une nouvelle langue soit prise en compte.
TreeTagger est téléchargeable sous forme de binaire, nous en avons donc pris 2: un pour linux(x32) et un pour linux(x64). Le chemin vers le bon binaire est géré automatiquement. Afin d’utiliser TreeTagger nous avons récupéré un wrapper sur internet mais celui-ci était en python2. Après un coup de 2to3 le script ne fonctionnait plus (Problème avec les unicode et bytes), mais après quelques temps les problèmes ont été résolus.
Une fois le wrapper terminé, on a ajouté une petite classe permettant de manipuler plus simplement les valeurs de retour de TreeTagger.
Pour lemmatiser notre texte, rien de plus simple: On créée un objet TreeTagger avec la langue que l’on souhaite puis on applique la fonction TagText. Une fois le texte taggé nous supprimons tout les stop_words et la ponctuation (Afin de ne pas fausser les résultats).
Nous pouvons enfin construire nos n-grams lemmatisés et compté le nombre d'occurrences. Pour chaque document nous comptons le nombre d'occurrences d’un n-gram et s’il apparaît dans le document. Puis nous appliquons le tf*idf (Qui n’est d’ailleurs pas tout a fait bon dans le cours idf = log(n / nx) + 1, mais j’ai vu un peu partout que le +1 était dans le log).
Maintenant que nos tfidf sont calculés, on trie les résultats dans l’ordre décroissant et on garde seulement les 3 premiers n-grams.
Nous séparons les titres du contenus afin d’avoir une suggestion pour les titres et pour le contenus bien différencié.

Pour améliorer les performances, les mot-clés sont cachés afin que si les articles n’ont pas changés, les mot-clés ne soient pas recalculés.

Au niveau de l’interface, vu que la lemmatisation et le calcule des tfidf peut être long, nous avons affiché une animation afin de faire patienter l’utilisateur.

La dernière étape consiste à requêter des images au travers de l’api de recherche “Bing” en utilisant les mots-clés les mieux classés.