eFrei

PARIS PANTHÉON-ASSAS UNIVERSITÉ

MY FIRST PYCHAT BOY

PROJET SEMESTRIEL PROGRAMMATION EN PYTHON

Axel Monsarrat
Yanis Amara
L1-BDX
2023

Introduction

Le projet que nous dévoilons s'articule autour de l'analyse de texte, mettant en lumière une méthode spécifique pour élaborer un système de réponse intelligente. Notre initiative vise à explorer en profondeur les fondements du traitement de texte tout en offrant un aperçu pratique d'une approche clé pour développer des chatbots et des intelligences artificielles génératives, à l'instar du chatGPT.

L'objectif principal de ce projet est de présenter une méthode pragmatique fondée sur l'analyse des occurrences des mots dans un corpus pour générer des réponses intelligentes. Plutôt que de se focaliser sur la manipulation directe de réseaux de neurones, cette approche se concentre sur l'utilisation astucieuse de la fréquence des mots pour répondre à des questions.

Nous aspirons à dévoiler un processus détaillé, débutant par le prétraitement des données et se déployant à travers la création d'une matrice TF-IDF, jusqu'à la sélection finale de la réponse appropriée. Chaque étape de ce cheminement méthodique est conçue pour permettre au chatbot de comprendre et de répondre de manière intelligente aux questions posées, en se basant sur la similitude entre la question soumise et les informations contenues dans le corpus.

En mettant en lumière ces différentes phases, nous visons à offrir un aperçu pratique de la façon dont un système de réponse intelligente peut être construit sans nécessairement recourir à des architectures neuronales complexes. Cette démarche souligne l'importance de la fréquence des mots dans le processus de génération de réponses pertinentes, ouvrant ainsi des perspectives fascinantes pour la conception de chatbots et d'IA conversationnelles.

Les fonctionnalités principales

Ces fonctions dit « principales » sont les fonctions directement utilisable via l'interface de notre Pychat Bot

- Affiche les mots les moins importants dans le corpus de documents
- Affiche les mots les plus importants dans le corpus de documents
- Indique les mots les plus répétés par le président Chirac
- Indique les noms des présidents qui ont parlé de la «Nation» et celui qui l'a répété le plus de fois
  - Variabilisation de la recherche de mot
- Indique les noms des présidents qui ont parlé du climat et/ou de l'écologie
- Indique le discours ayant le plus de rapport avec une question donnée
- Indique le mot ayant le TF-IDF le plus élevé en rapport avec une question donnée
- Génère une réponse en lien avec une question donnée

Les fonctionnalités secondaires

Ces fonctions ne sont pas directement utilisables par l'utilisateur via l'interface mais elles sont essentiel pour le bon déroulement des fonctions principales

- Nettoient les fichiers discours en chaîne de caractère composé de lettres et d'espaces
- Dupliquent les documents d'un répertoire 1 vers un répertoire 2
- Extraient les noms des présidents à partir des noms des fichiers texte fournis
- Associent à chaque président un prénom
- Affichent la liste des noms des présidents
- Génèrent une matrice TF et un dictionnaire IDF
- Créent une matrice TF-IDF
- « Tokenisation » d'une question
- Recherchent un mot donné dans un corpus donné
- Calculent un vecteur TF-IDF pour les termes d'une question donnée

Présentation technique

Noms et prénoms

La première partie de notre projet s'occupe de la création d'un dictionnaire contenant les noms et prénoms des présidents. L'algorithme vient parcourir le nom des fichiers du dossier « speeches » pour venir récupérer les noms de famille (en supprimant les doublons) et les stocker en temps que clé dans un dictionnaire (la valeur associée est un int allant de 1 à 6). Nous avons ensuite créé un dictionnaire en dur contenant les prénoms (en prenant soin de mettre le nom et prénom d'une même personne avec la même valeur associée entre les deux dictionnaires). Il nous restait plus qu'à fusionner ces dictionnaires par rapport aux valeurs.
Nous avons décidé d'utiliser des dictionnaires car de cette manière l'association des noms et prénoms était plus sûr.

Nettoyage des textes

Il nous a ensuite fallu traiter les fichiers textes pour ne garder seulement que des lettres et des espaces. Nous avons donc créé une fonction qui, dans un premier temps, duplique le texte initial dans un nouveau dossier nommé « cleaned ». Elle convertit ensuite tout le texte en minuscule puis supprime toutes les formes de ponctuation.
Nous avons eu des difficultés sur le texte chirac1 car dans le début du texte est inscrit un « - » suivi d'un « » sauf que nous remplacions les « - » par des « » pour que des mots comme « elle-même » soit nettoyé en « elle même » et non « ellemême ». Nous nettoyons aussi les « » par des « » donc dans notre fichier nettoyé était inscrit deux « » à la suite. Finalement, nous avons compris que ce problème n'en était pas un et nous avons pu avancer sur le projet.

Matrice TF-IDF

La première étape de la réalisation de cette matrice est de calculer le TF (Term Frequency) de chaque fichier, soit le nombre de fois qu'un mot apparaît dans un texte. Nous avons réalisé cela grâce à une boucle assez simple qui compte le nombre d'occuration de chaque mot et qui renvoie un dictionnaire avec en clé tous les mots du texte et en valeur son occuration respective.
La deuxième étape est de calculer le IDF (Inverse Document Frequency), pour cela notre fonction parcourt tous les documents du corpus puis tous les mots du document en incrémentant un score IDF à chaque mot. Il applique ensuite une formule mathématique en fonction du corpus pour récupérer le score IDF du mot. Il nous renvoie finalement un dictionnaire suivant la même logique que notre dictionnaire TF.
La dernière étape de la création de matrice est la fusion de ces dictionnaires (1 dico TF pour chacun des 8 fichiers et 1 dico IDF global). La matrice est finalement composée de 8 dictionnaires composés des mêmes clefs (tous les mots de tous les textes) avec un score TF-IDF qui varie selon les fichiers.
Ici la fusion des différents dictionnaires nous a posé le plus de problème, car nous n'arrivions pas à avoir la logique derrière cette manipulation. Notre professeur a pu nous expliquer cette étape et nous avons pu avancer dans le projet.

Fonctions développées

Nous avons ensuite développé différentes fonctions qui utilisent différents éléments déjà créés. Par exemple les fonctions qui nous renvoient les mots plus ou moins importants ne font que parcourir la matrice TF-IDF et cherchent le/les mot(s) avec le plus gros ou petit score TF-IDF.
Les fonctions qui nous renvoient les mots les plus répétés ne font que parcourir les différents dictionnaires TF en quête du mot ayant la plus grosse valeur (soit le mot ayant le plus d'occurence). A noter qu'auparavant, nous avons soustrait la liste des mots les moins importants aux différents textes pour éviter de trouver les mêmes mots les plus répétés.
Nous avons eu du mal au niveau du changement de « variable » car, en début de projet, toutes les questions étaient en fonction des différents textes et non des présidents. Ici la difficulté a été de fusionner les différents discours appartenant au même président en un seul.

Menu

Le menu a été une étape plutôt simple à réaliser. Il a fallu intégrer les fonctions réalisées auparavant à une condition python du type if elif. Nous présentons dans un premier temps toutes les fonctions disponibles avec un numéro associé à chaque fonction. Il suffit à l'utilisateur d'entrer le numéro correspondant à la fonction.
Par exemple si la fonction 2 correspond aux mots les plus importants alors l'utilisateur entre « 2 » qui va être stocké dans une variable puis l'algorithme va lancer la boucle au niveau du « elif reponse == 2 : » et effectuer le code qui précède cette condition.

Traitement de la question

Dans cette partie il est question d'effectuer les mêmes traitements que nous avons réalisé auparavant sur les fichiers textes mais cette fois sur une question posée par l'utilisateur. Nous devons entre autres nettoyer la question pour ne garder que les mots et des espaces puis calculer leurs scores TF et leurs scores TF-IDF.
Ces opérations ont été plutôt rapide à effectuer car elles gardes le même fonctionnement que les traitements des textes. Nous avons donc pu réutiliser les fonctions TF, IDF et TF-IDF définis plus tôt.

Analyse de la question

Après avoir créé les différents vecteurs TF-IDF des mots de la question, il ne reste plus qu'à comparer leurs vecteurs avec ceux des mots du corpus grâce à une nouvelle formule mathématique mesurant le cosinus de l'angle entre deux vecteurs A et B dans un espace multidimensionnel. Nous pouvons ensuite en déduire quel texte présente le plus de similitude avec notre question en calculant la similarité du vecteur de la question avec chacun des vecteurs du document puis retourne le nom du document correspondant à la valeur de similarité la plus élevée
Notre difficulté ici a été de transformer notre matrice TF-IDF (en dictionnaire) sous forme de vecteur de dimension M associé à chaque document. Pour ce faire nous avons du initialiser une nouvelle matrice et parcourir l'ancienne pour remplir la nouvelle en fonction des clés tout en gardant les valeurs (soit les score TF-IDF).

Résultat

Matrice

Voici notre première matrice sous forme de dictionnaires.
Soit 1981 colonnes représentant les 1981 mots avec leur score TF-IDF associé et 8 lignes soit les 8 fichiers avec leur noms au dessus.

8 lignes pour les 8 fichiers

```
Nomination_Chirac1.txt
{'meilleur': 0.6020599913279624, 'législatif': 0.9030899869919435, 'voulu': 0.9030899869919435, 'réalit
Nomination_Chirac2.txt
{'meilleur': 0.0, 'législatif': 0.0, 'voulu': 0.0, 'réalité': 0.0, 'égaux': 0.0, 'elle': 0.753895310709
Nomination_Mitterrand2.txt
{'meilleur': 0.0, 'législatif': 0.0, 'voulu': 0.0, 'réalité': 0.0, 'égaux': 0.0, 'elle': 0.173975840933
Nomination_Sarkozy.txt
{'meilleur': 0.6020599913279624, 'législatif': 0.0, 'voulu': 0.0, 'réalité': 0.0, 'égaux': 0.602059991
Nomination_Mitterrand1.txt
{'meilleur': 0.0, 'législatif': 0.0, 'voulu': 0.0, 'réalité': 0.0, 'égaux': 0.0, 'elle': 0.173975840933
Nomination_Hollande.txt
{'meilleur': 0.0, 'législatif': 0.0, 'voulu': 0.0, 'réalité': 0.0, 'égaux': 0.0, 'elle': 0.521927522799
Nomination_Macron.txt
{'meilleur': 0.0, 'législatif': 0.0, 'voulu': 0.0, 'réalité': 0.0, 'égaux': 0.0, 'elle': 0.463935575821
Nomination_Giscard dEstaing.txt
{'meilleur': 0.0, 'législatif': 0.0, 'voulu': 0.0, 'réalité': 0.0, 'égaux': 0.0, 'elle': 0.0, 'sens': 0
```

1681 colonnes pour les 1681 mots
avec 1 élément = 1 mot : son TF-IDF

Voici notre seconde matrice qui représente comme demander avec N lignes et M colonnes, où N = nombre de documents dans le corpus (8 dans notre cas) et M = nombre de mots dans le corpus (1681 dans notre cas)

8 lignes pour les 8 fichiers

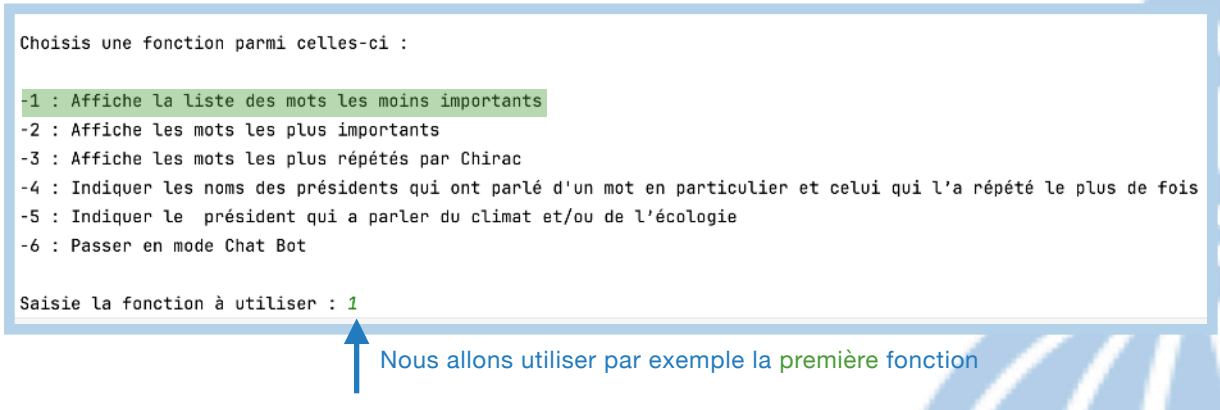
```
Nomination_Chirac1.txt
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.6020599913279624, 0.0, 0.0, 0
Nomination_Chirac2.txt
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
Nomination_Mitterrand2.txt
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9030899869919435, 0.0, 0.0, 0.0, 0.0, 0.0, 0.90308998
Nomination_Sarkozy.txt
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.2041199826559248, 0.0, 0.0, 0.0, 0
Nomination_Mitterrand1.txt
[0.0, 0.9030899869919435, 0.9030899869919435, 0.9030899869919435, 0.9030899869919435, 0.90308998699194
Nomination_Hollande.txt
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.6020599913279624, 0.0, 0.0, 0.0, 0
Nomination_Macron.txt
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.806179973983887, 0.0, 0.0, 0.
Nomination_Giscard dEstaing.txt
[0.9030899869919435, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9030899869919435, 0.0, 0.9030899869919435, 0.0, 0.0, 0
```

1681 colonnes pour les 1681 mots
avec 1 élément = 1 TF-IDF

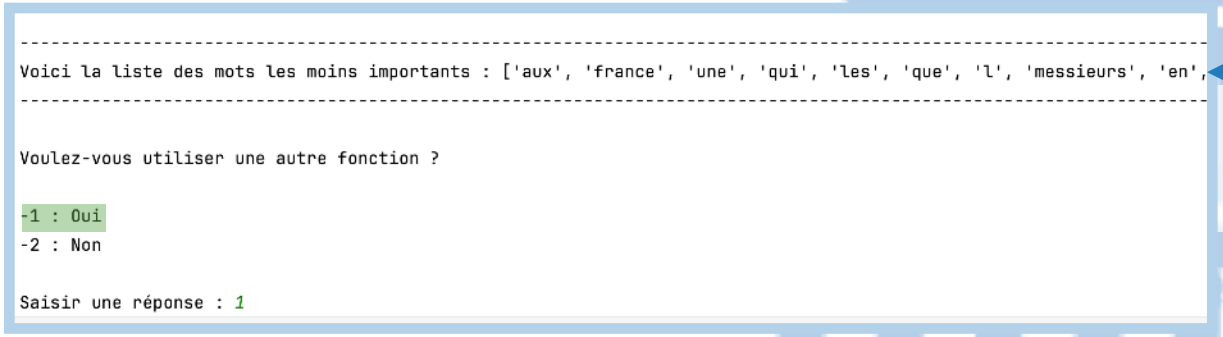
Interface utilisateur

Au lancement du programme nous devons demander a l'utilisateur dans quel dossier veut-il utiliser le chatbot.

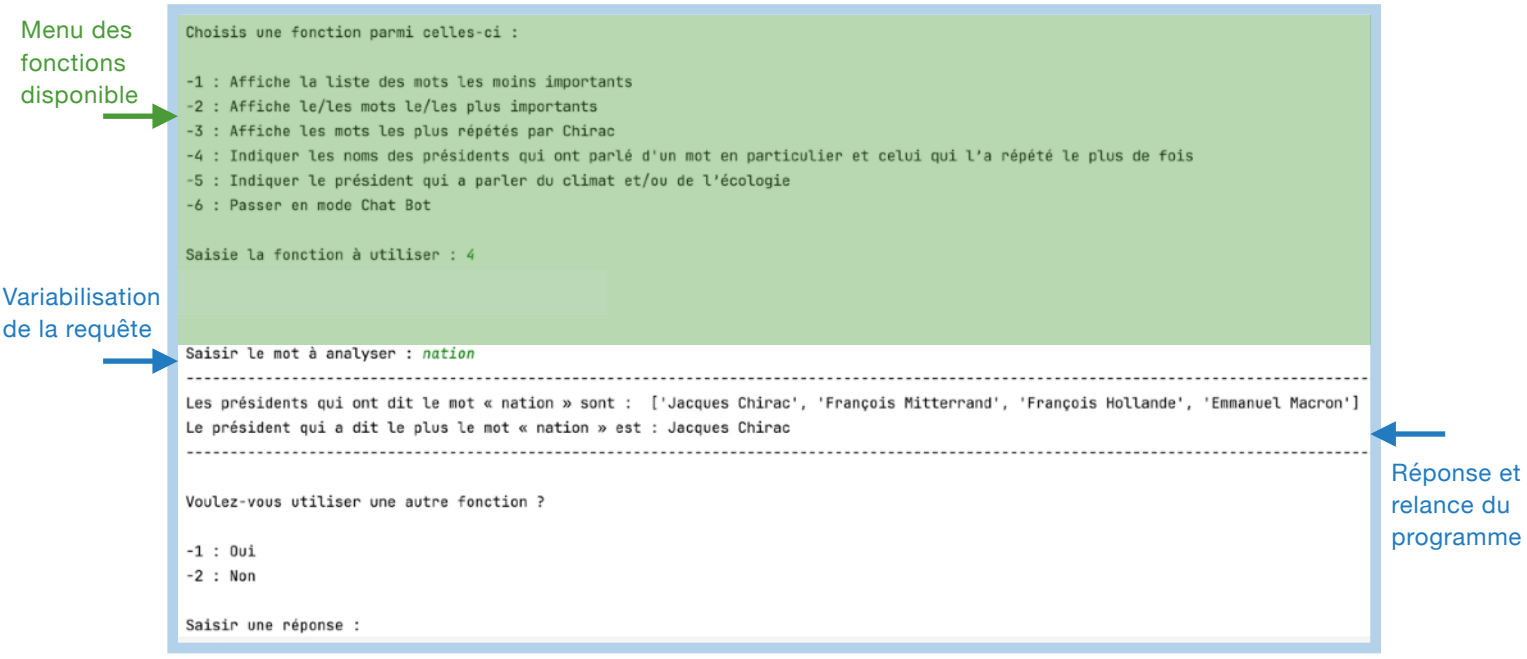
Nous proposons ensuite a l'utilisateur les différentes fonctions implémentée a utiliser sur les différents fichiers du répertoire souhaité.



L'interface nous renvoie alors la réponse désirée et nous demande si nous voulons réutiliser une autre fonction.



Autre exemple d'utilisation sur une fonction variabilisée.



Conclusion

Ce projet nous a permis de maîtriser des concepts clés tels que la création de matrices TF-IDF et l'évaluation de similarité. L'utilisation de PyCharm IDE nous a offert un environnement de développement efficace, tandis que GitHub Cloud a facilité la gestion collaborative du code.

Nous avons amélioré notre collaboration et notre communication en répartissant les tâches de manière efficace et en résolvant des problèmes ensemble. PyCharm IDE et GitHub Cloud ont renforcé notre capacité à travailler harmonieusement en équipe.

La gestion du temps a été cruciale. Nous avons appris à évaluer nos progrès, à ajuster nos priorités et à rester flexibles face aux imprévus, renforçant notre capacité à respecter les délais et à s'adapter aux changements,

FIN.