Rapport Projet TAL

Azemard Thomas - Jardet Quentin

05/05/2018

1 Introduction

Les objectifs de ce projet étaient multiples. Bien évidemment, le premier objectif était d'appliquer les notions vues en cours sur un exemple concret : comment interagir avec un utilisateur, comment reconnaître les informations importantes dans ses messages et comment y répondre de manière automatique (et aussi intelligente que possible). Mais c'est aussi l'occasion de découvrir d'autres technologies (ou de s'y pencher un peu plus) : Github et Python. Enfin c'est un projet court qui permet de voir comment mener un projet entier, en groupe, de l'idée initiale au rendu final.

2 Le Cuistobot dans ses différents modes

Notre bot est séparé en 3 modes. Ils correspondent respectivement à :
— Mode 1 : Réponse simple.
— Mode 2 : Bot psychologue.
— Mode 3 : Recherche d'un plat.

Avant d'expliquer l'avancement de chacun des modes, nous allons parler brièvement de l'organisation du code. Nous avons une fonction principale (main) qui prendra une ligne de commande et qui exécutera le bot dans un mode spécifique. Ainsi, les modes sont représentés dans une fonction du fichier *chatBot.py*, mais le mode 3 fait aussi appel à des fonctions du fichier *mode3methods.py*.

2.1 Mode 1 : Réponse simple

Le mode 1 est entièrement situé dans la fonction `discussion mode 1`. Son mode de fonctionnement est assez simple. Il possède une liste de réponses banales pouvant servir au bot pour fournir à l'utilisateur un message, même s'il n'a pas compris son message.

L'état actuel du mode permet d'éviter que **le bot ne répète pas deux fois la même chose**. Les réponses sont choisies aléatoirement à chaque passage dans la fonction. Les améliorations visibles pour ce mode sont assez facile à constater :

Augmenter le panel de réponse que possède le bot. En effet, la liste possède 5 éléments, ce qui est assez peu. Un utilisateur humain peut rapidement comprendre que le bot se répète. Pour résoudre ce problème la première solution serait d'augmenter la capacité de cette liste.

La seconde solution serait d'**éviter au maximum d'avoir recours à ce mode**, car plus le bot utilisera ce mode plus il y a de chance que l'utilisateur s'aperçoive d'une répétition. En effet, comme il représente un "*mode de dernier recours*", donc plus nous allons avoir un bot performant, moins il devra utiliser ce mode.

2.2 Mode 2 : Bot psychologue

Le mode 1 est entièrement situé dans la fonction `discussion mode 2`. Son fonctionnement est plus complexe que le premier mode, mais reste assez simple. Actuellement, le bot connait six thèmes de discussion :
— La santé
— La famille

— L'argent
— La fatigue
— Le jeu hearthstone (un jeu de carte)
— Saluer quelqu'un

Si l'utilisateur parle d'un de ces thèmes, le bot utilisera une réponse sous forme de question assez générale sur ce thème. Comme pour le mode précédent, le bot ne répondra pas deux fois la même chose. Pour que le bot reconnaisse un thème, nous utilisons une liste de mots pour chaque thème : Par exemple, "euros" et "gold" appartiennent au thème de l'argent. Ainsi, si le bot trouve un mot appartenant à une liste, il répondra en fonction. Si l'utilisateur envoie un message pouvant être associé à plusieurs thèmes, alors il prendra le thème qui apparait le plus de fois dans le message.

Par exemple : "Hello, I like gold and silver.", nous avons "hello" pour le thème 6 et "gold","silver" pour le thème de l'argent. Le bot constatera que le thème de l'argent apparait plus de fois. Il utilisera alors le thème de l'argent.

Pour améliorer ce second mode, nous avons diverses possibilités :

Augmenter le nombre de thème que le bot peut reconnaitre. En faisant cela, il sera plus probable que la réponse donnée par le bot ne soit pas hors-sujet. De plus, si le bot ne reconnait aucun des thèmes dans le message, celui-ci passera en mode 1, il aura encore moins de chance de fournir une réponse adaptée.

Augmenter le nombre de réponse possible pour un thème. En effet, nous avons actuellement deux réponses par thèmes, donc si l'utilisateur envoie trois messages sur un même thème, il pourra constater que le bot se répète. Pour éviter ce problème, il suffit d'augmenter le nombre de réponses possibles.

Augmenter le nombre de mots dans une liste d'un thème. Cette amélioration est assez discutable : si nous augmentons le nombre de mots par liste, il y a plus de chance qu'un message soit reconnu. Mais, il y a aussi plus de chance que le message soit mal interprété et que la réponse soit complètement hors-sujet. Si nous ajoutons, par exemple, le mot "golden" au thème de l'argent et que l'utilisateur envoie le message "Golden Globes are interesting.". Alors le bot répondra sur le thème de l'argent (ce qui est assez hors-sujet), il faut donc que les mots ajoutés aux listes soient précis et qu'ils aient peu de double-sens.

2.3 Mode 3 : Recherche d'un plat

2.3.1 Présentation et répartition des tâches

La première question (et peut-être la plus compliquée) fut de savoir ce qu'allait faire notre bot. Notre objectif était de choisir un sujet permettant d'avoir un bot entièrement fonctionnel, sur un sujet relativement réaliste et aussi intéressant que possible. Parmi nos quelques idées (un bot permettant de renvoyer les informations depuis un wiki défini par exemple), on a choisi de partir sur un chatbot de cuisine, dont la fonction principale serait de proposer des recettes.

La première étape du projet fut de définir le fonctionnement global de notre chatbot. Pendant que Quentin reprenait nos TPs pour réaliser les modes 1 et 2 (comme vu précédemment), Thomas s'est occupé de trouver une base de données contenant suffisamment de recettes pour pouvoir proposer un bot de qualité. La meilleure base trouvée se trouvait sur le site Kaggle.com, un site dédié justement à la data science qui appartient à Google. Cette base contient plus de 20000 recettes issues du site Epicurious.com, avec toutes les informations imaginables associées (ingrédients, calories, gras...).

Cette base se divise en deux fichiers : un fichier .csv contenant la liste des plats avec leurs ingrédients et autres données simples (nombre de calories, appartenance à une catégorie comme vegan par exemple). Le second fichier (en .json) contient les informations détaillées des plats : les quantités précises des ingrédients, les recettes complètes, la description du plat. Enfin, cette base est en anglais ce qui a conditionné le reste de notre bot qui doit donc être entièrement en anglais.

A partir de la structure de cette base, on pouvait prévoir le fonctionnement du bot. Il faudra interagir avec l'utilisateur pour obtenir une liste d'aliments (ou directement un plat). Ensuite il faudra chercher dans le .csv les plats correspondants et les proposer à l'utilisateur. Enfin on devra aller récupérer les informations supplémentaires dans le fichier .json toujours pour les donner à l'utilisateur.

2.3.2 Partie dialogue

Ce mode est l'axe principal de notre Cuistobot. L'objectif est qu'il puisse fournir les informations nécessaires à l'utilisateur pour qu'il puisse cuisiner un repas. Pour cela, nous avons séparé en trois étapes :

— La collecte des informations
— La recherche dans la base de données
— La transmission des informations

La collecte d'information, dans cette partie nous allons poser des questions à l'utilisateur afin de pouvoir effectuer une recherche en fonction de sa requête. Pour rendre cette étape la plus intuitive pour l'utilisateur, le bot utilisera un dictionnaire de mots symbolisant de potentiels critères de recherche. En effet, celui-ci pourra alors cibler la requête sans être influencé par la structure des messages de l'utilisateur.

Le bot peut aussi reconnaitre un mot de son dictionnaire même si celui-ci est dans une autre forme (conjugué, pluriel, etc.). Ainsi, le critère *apple* peut aussi être sous sa forme plurielle *apples*.

L'utilisateur n'aura donc pas à se soucier de respecter un langage ou des restrictions sur son message, pour qu'il soit compris par le bot. Si nous prenons, par exemple : "*apple, bananas, cheese.*" et "*I want an apple and bananas, with a lot of CHEESE.*". Dans ces deux exemples, les critères seront apple,cheese et banana.

Lorsque l'utilisateur finit la description de son plat, le bot pourra intervenir pour effectuer un commentaire sur la liste d'informations qu'il a collecté. Nous avons mis en place cette partie, car sinon le bot effectuera des recherches incohérentes (par exemple, des recherches sans aucun critère).

La recherche dans la base de données, cette étape est assez transparente pour l'utilisateur, car il n'y a pas d'échange avec le bot. En effet, le bot informera du résultat de sa recherche et effectuera un commentaire. Selon le nombre de critère, nous pouvons facilement atteindre des résultats avec plusieurs dizaines d'éléments, ce qui devient assez lourd à manipuler dans un chatbot. C'est pourquoi la quantité de résultats sera toujours d'une taille raisonnable (maximum 5).

Encore une fois, à la fin de cette étape le bot fera un commentaire sur le résultat et si celui-ci lui semble incohérent (aucun résultat par exemple), il proposera de retenter la recherche. Cela permet d'arreter une recherche en cours de route et de corriger l'erreur (si ce n'était pas le cas, nous devrions effectuer une recherche complète.).

La transmission des informations est la dernière partie du mode 3. Le bot possède les différents plats qui sont compatibles avec les informations de l'étape 1. Le bot pourra alors proposer de choisir un plat et donnera les informations que l'utilisateur souhaite. Le Cuistobot peut donner 5 informations sur un plat : la recette, une description rapide, la liste des ingrédients, la valeur en calorie et la masse de sel dans le plat. Pour chaque question, l'utilisateur peut répondre de différentes manières. Ainsi, nous avons essayé d'anticiper pour que malgré cela, le bot comprenne le choix de l'utilisateur.

Par exemple, si l'utilisateur doit choisir dans une liste de plat celui qu'il préfère, il peut répondre de manière simple "*1*" pour le premier. Ou encore "*The first one.*", ou "*I want the meat toast*" (avec *meat toast* le nom du plat). Ainsi en faisant tester notre bot par de vraies personnes, nous avons constaté qu'elles pouvaient se retrouver bloquées, car le bot ne comprenait pas leur réponse. C'est pourquoi nous avons inclus diverses manières de répondre aux questions en essayant d'anticiper.

Une fois les informations en sa possession, l'utilisateur peut consulter d'autres plats de la recherche ou bien décider de parler d'autres choses.

Si nous regardons l'objectif primaire de ce mode (pouvoir rechercher des plats), alors celui-ci est atteint. Mais nous avons aussi intégré une liberté dans les possibilités de répondre au Cuistobot. En effet, nous avons ajouté différentes possibilités de répondre aux questions du bot. Ainsi, le dialogue avec le bot sera plus proche d'un dialogue avec un être humain, car nous n'aurons pas à respecter une syntaxe particulière dans les messages.

2.3.3 Partie base de données

Une fois la base de données trouvée, il a fallu permettre son exploitation ce qui fut plus compliqué que prévu. Pour celà, j'ai utilisé les bibliothèques csv et json qui sont intégrées de base dans Python. Ces bibliothèques

permettent de parcourir facilement ces types de fichiers pour extraire l'information voulue.

La première difficulté est arrivée très vite : le fichier .json n'était pas reconnu par la bibliothèque de Python, et donc inutilisable. Sans aucune autre information que "format invalide", il a été nécessaire de se pencher sur le fichier à la main. Sans surprise, un fichier de plusieurs Mégaoctets écrit sur une seule ligne n'est pas le plus pratique à lire par un humain ! Au final après avoir regardé des fichiers json valides, la solution s'est révélée plutôt simple à mettre en place : il a suffit de rajouter une balise englobant l'ensemble des données avec une nouvelle clé (ici "meal").

Après avoir extrait les informations élémentaires du .csv (toutes les catégories disponibles), il a fallu s'atteler à la tâche principale : assurer le lien entre les données du .csv et celles du .json. La base de données choisie s'est révélée avoir un défaut majeur : elle a été créée dans le but d'être exploitée pour faire de l'analyse de données. Par conséquent ce n'est pas sa structure et sa fiabilité qui sont importantes, mais plus le volume de données. Pour reformuler, là où on s'intéresse à un élément (un plat) en particulier, le data scientist s'intéresse aux tendances et aux statistiques qu'on peut dégager de l'étude de l'ensemble des éléments.

Pour nous, cette différence qui peut sembler uniquement sémantique a eu des conséquences fortes, compliquant les choses. Tout d'abord, les plats n'ont pas d'identifiant unique qui aurait permis de retrouver facilement le plat dans chacun des fichiers. De plus certains plats ont le même nom mêmes s'ils ont des recettes et parfois des ingrédients différents (par exemple certaines personnes mettent de l'ail dans la salade alors que...), donc on ne peut pas se baser non plus sur le nom si on veut éviter tous les problèmes potentiels.

Les deux solutions possibles restantes étaient de soit utiliser une clé multiple (le nom du plat et le nombre de calories), soit utiliser la position dans la liste puisque c'est théoriquement la même dans chaque fichier. La deuxième solution est beaucoup moins coûteuse en ressource lors de la recherche, et plus simple de manière générale. C'est donc la solution choisie.

La dernière étape restante est d'assurer la correspondance entre les deux fichiers, et la conservation de l'ordre. Et ici encore, on est confronté à 2 problèmes majeurs : certains plats sont présents dans le .csv mais pas dans le .json (et inversement), et le .json contient des éléments vides. Les causes des problèmes étant parfois contradictoires, il est très difficile de régler tout ça automatiquement (est ce que le décalage est dû au fichier .csv ou au fichier .json ? Est ce que c'est un élément manquant ou un élément en trop ?).
Il a donc fallu créer un script affichant la liste des éléments (titres des plats) de chaque fichier cote à cote, indiquant les différences, et modifier à la main en fonction de la cause du décalage. La modification est rapide et facile à réaliser en ouvrant le .csv avec un tableur : il suffit de supprimer une ligne n'ayant pas de correspondance dans le .json, ou d'ajouter une ligne vide au bon endroit si on se décale dans l'autre sens. Puis on relance le script, et on recommence au décalage suivant.

Au final, cette base a demandé beaucoup de travail pour être exploitable. Mais une fois ce travail réalisé, on a un résultat assurant l'absence de bugs dus à une mauvaise correspondance. De plus les fonctions pour récupérer les informations voulues sont très simples à écrire, et aussi rapides que possible à l'exécution.

2.4 Améliorations possibles

Notre choix d'avoir une tranche verticale entièrement fonctionnelle fait qu'on a un bot complet, permettant de remplir totalement sa fonctionnalité première (la recherche de plat par ses aliments). Cependant il y a deux axes majeurs pour l'améliorer.

Tout d'abord, on peut **améliorer la partie langage et interaction avec l'utilisateur** : plus de phrases de réponses, plus de feedback, plus d'informations de manière générale. Le but de ces améliorations serait de rendre le bot plus "humain" et réaliste

Le second axe d'amélioration est d'**exploiter les catégories inutilisées du fichier .csv**. En effet on peut rajouter des fonctions de tri en fonction du nombre de calories, ou de la quantité de gras ou de sel. On peut aussi utiliser les catégories sur l'origine du plat quand on la connaît (même si ce sont majoritairement différents

états américains ce qui nous intéresse peu). Enfin on a des catégories plus libres comme repas de fête, repas pour le 4 juillet, repas pour le “Bastille Day”, repas pour un anniversaire etc qu’on peut regrouper et exploiter pour toujours donner plus d’options de recherche à l’utilisateur suivant ses désirs.