Vincent FABIOUX
Nicolas MONTORO
Olivier NAPPERT

Polytech Paris Sud
ET4-Informatique

Traitement automatique des langues : Projet "Checkinbot"

2017-2018

Choix du Sujet

Initialement, nous étions tiraillés entre deux sujets, à savoir le projet “Chatbot” et le projet “Review”, qui consistait à donner une note à un commentaire en évaluant l’avis de l’auteur. D’une part, la majorité du groupe était pour le sujet “Chatbot”, et d’autre part en y réfléchissant bien il nous est apparu que ce dernier serait plus intéressant à mettre en place et plus simple techniquement (du moins, c’était notre préjugé lors de la sélection des sujets) que son concurrent. C’est ainsi que nous avons choisis ce projet, avec pour objectif de nous concentrer sur un aspect particulier et non de réaliser un Chatbot général de discussion.

Notre choix s’est porté vers la réservation de vols en ligne, le principe étant qu’un interlocuteur humain puisse demander au Chatbot, actuellement appelé le Checkinbot, s’il est possible de trouver un vol en lui fournissant certaines informations : lieux de départ et d’arrivée, dates, et même heures de décollage et d’atterrissage. Si le bot trouve un vol correspondant, il le signale à son interlocuteur qui peut alors confirmer sa réservation et quitter la session.

Sommaire

I) Structure du code

Dans un premier temps la question qui se posait était " Comment mettre en place un bot qui puisse répondre dynamiquement en fonction de l'entrée utilisateur ? ". Au cours du TP 1 et 2, nous avons étudié une approche probabiliste qui nous permettait de tenter de "deviner" si un mot appartenait à un certain lexique, en fonction de probabilités logarithmiques d'apparition dans le texte du mot en question. Cependant cette approche nous a semblé peu pratique et inadaptée à des interactions dynamiques nécessitant une réponse, ce pourquoi nous avons opté pour une autre option, à savoir l'approche des mots-clés. Cette approche consiste à indiquer au bot un ensemble de mots-clés censés apparaître régulièrement dans l'interaction pour qu'il tente de comprendre le sens d'une phrase. Elle est parfaitement adaptée à notre sujet, puisque le domaine d'utilisation du bot se réduit à quelque chose de très spécifique, la réservation de vols. Ainsi, voici le déroulement précis et concret des opérations que font notre bot lorsque nous lui adressons la parole :

1) Normalisation : normalize.py

Dans un premier temps, le bot va récupérer l'entrée utilisateur et la normaliser, pour réduire la complexité de compréhension du texte et s'appuyer sur certains standards. Par exemple, les dates entrées par l'utilisateur dans le système anglo-saxon seront toujours récupérées par le bot sous le même format après normalisation, quelque soit la nomenclature utilisée. Ces opérations sont réalisées dans le fichier normalize.py via la fonction normalize(), qui consiste donc en une pléthore d'expressions régulières visant à réduire la complexité d'interprétation.

2) Extraction des données : extract.py

Dans un deuxième temps, nous appelons la fonction extract() venant du fichier extract.py sur le texte récupéré. Cette fonction va notamment nous permettre de mettre les dates, heures et villes sous un format que nous pourrons utiliser par la suite, et consiste donc lui aussi en une avalanche d'expressions régulières. En effet, de nombreuses possibilités existent : Pour les heures, plusieurs formats sont possibles : 8 :00 :00 a.m, 8 :00 p.m, 8 p.m. De même l'utilisateur peut vouloir dire "morning", "afternoon" ou encore 8 :00 a.m - 9 :00 a.m, 8 :00 a.m / 9 :00 a.m, etc Pour les dates, plusieurs formats sont égalementpossibles : 25/03/1995, 25-03-1995, 25/03, 03/1995, 25 March 1995, 25th of March 1995, etc. Il peut également vouloir dire "in 5 days", in "5 months", "next Monday", Et pour finir les villes peuvent contenir des espaces, apostrophes, tirets, points, etc Une fois cette étape passée le texte est prêt à être interprété.

3)Interprétation des données : guess.py

Ensuite, l'interprétation réelle commence avec la fonction guess() du fichier guess.py. Cette fonction a pour but de tenter de "deviner" le sens d'une phrase entrée par l'utilisateur en se basant sur plusieurs outils : mots-clés comme évoqué précédemment, mais également en accordant une notion de poids à certains mots pour faire ressortir leur importance. Dans un

premier temps, guess() parcours donc le texte et remplace le texte par des tokens issus des mots-clés ; par exemple, les mots “Hello”, “Hi” et “Good morning” seront tous convertis en un token “HI”, ce qui facilitera l’analyse par la suite. Une fois cela fait, cette fonction attribue le rôle de chaque information (par exemple attribue à une date le rôle de “Date de départ”) selon les tokens précédents (plus le token est éloigné, plus son poids est faible). Elle remplie une structure de donnée “data” avec ces informations et les communique à la prochaine fonction, pour qu’elle réponde à l’utilisateur.

4) Intéraction avec l’utilisateur : answer.py

Dans un dernier temps, la structure de données “data” (dictionnaires avec les lieux de départ, lieux d’arrivée, champs pour les dates, etc.) est envoyée à une fonction answer() venant du fichier answer.py. Ellea pour but de répondre à l’utilisateur, en plusieurs temps. Pour commencer, elle se base sur un système de mot-clés similairement à guess(), afin de répondre à des envois de tokens comme “HI”, “YES”, ou “BYE” depuis la fonction précédente. Ensuite, elle rappelle à l’utilisateur les données qu’il a entré . Elle lui rappelle aussi les informations qui restent à indiquer le cas échéant, avec des formules qui divergent selon le nombre d’informations manquantes pour plus de convivialité. Enfin, si toutes les informations ont été indiquées (à minima : la ville de départ et d’arrivée souhaitées ainsi que les dates correspondants aux deux vols), elle appelle une fonction search() qui interroge la base de donnée pour savoir si un vol correspondant à ses exigences existe. Cette “base de donnée” est en réalité un fichier texte nommé “flights.txt” qui est généré dynamiquement à partir du fichier generate.py dans le répertoire data/. Ce fichier intègre tous les vols possibles, en les présentant sous la forme suivante : City1|City2|flight1_id|departure1_timestamp|arrival1_timestamp|flight2_id|...\n. Cela nous permet de facilement savoir si un résultat correspond à un souhait de l’utilisateur, et la lecture du fichier généré présente un double avantage : facilité de changer les vols générés(les villes se situant dans ”cities.txt”) et prend très peu de place en mémoire sans perdre trop de rapidité. Cette considération est peut être insignifiante à l’échelle de notre projet, mais si ce bot venait à être utilisé pour réserver des vols dans le monde entier, connectant plus de 1 000 000 de villes différentes entre elles, elle prendrait tout son intérêt. Quoi qu’il en soit, la fonction search() produit une réponse à partir des données du dictionnaire “data” et des vols disponibles, laissant à l’utilisateur deux choix : soit celui de changer une ou des informations, étape pendant laquelle le bot est évidemment là pour l’accompagner, soit celui d’accepter ou de refuser une réservation qui est proposée, dans le cas où il y’a une correspondance entre “data” et la base de donnée. Pour finir, il est possible pour l’utilisateur de quitter à tout moment ainsi que de dire “non” au moment de la proposition d’un vol pour changer les infos entrées. Cette répartition nous semble efficace, en séparant les parties du projet comme le traitement des données et leur génération.

Précisions : Si une personne demande de partir le 01 Mai et d’arriver le 04 Mai, nous lui proposons le premier vol qui lui garantit d’arriver le 04 au plus tard et donc possiblement avant. De plus si l’utilisateur dit qu’il veut partir le 01 Mai entre 12h et 14h, nous considérons qu’il est indisponible aux autres horaires ce jour là, et si aucun départ n’est prévu dans cet interval nous lui proposons un vol sur un jour suivant (même s’il ya un vol à 14h01). De la même manière il est possible de ne pas spécifier d’horaires de départ, d’arrivé ou les deux, nous proposons alors à l’utilisateur le premier vol disponible à partir de la date de départ spécifiée. Enfin, si l’utilisateur veut partir le 01 Mai à 14h, nous regardons tous les vols qui partent le 01 Mai entre 14h00 et 14h59, et lui proposons le cas échéant. Dans le cas ou aucun vol n’est trouvé entre 14h et 14h59, les prochains vols proposés seront le lendemain au minimum.

II) Utilisation de github et répartition du travail

La répartition du travail a été assez simple, dans le sens où chacun savait ce qu'il devait faire.

Vincent Fabioux : Fichier principal (checkinbot.py) avec description du projet, métadonnées, boucle principale, lien avec les autres fonctions et mode debug. README du projet pour le répertoire GitHub. Fonction d'interprétation des phrases de l'utilisateur (src/guess.py). Fonction de génération des données de vol (data/generate.py) pour générer le fichier data/flights.txt.

Nicolas Montoro : Fonction d'extraction des dates et lieux de villes (src/extract.py). Récupération de liste de villes pour data/cities.txt (fortement réduit pour réduire le nombre de possibilités). Tests initial d'une base de données pour stocker les vols, abandonné car trop dépendant de l'implémentation de la base de données (petites différences selon l'OS et instructions d'installations longues à mettre en place). Fonction de recherche dans la liste des vols (search() dans src/answer.py) pour trouver un vol correspondant aux informations entrées et transformation des dates / heures en un format lisible par l'utilisateur. Mise au propre du rapport à l'aide de Latex.

Olivier Nappert : Fonction de normalisation des phrases entrées (src/normalize.py). Aide à la définition des mots-clés dans la src/guess.py. Fonction de réponse à l'utilisateur (src/answer.py). Introduction, explication des différentes parties du programme et conclusion au format textuel pour préparer le rapport.

Le protocole git nous étant tous familier, il n'y a eu aucun soucis à l'utiliser une fois de plus pour réaliser ce projet. Cet outil se révèle toujours efficace quand il s'agit de mutualiser le code, de revenir sur des changements défectueux, ou tout simplement de documenter l'avancée du projet. La structure que nous avons au départ choisie nous a permis de chacun travailler de notre côté sans attendre les commits des autres membres.

Pour finir sur cette partie, la répartition du travail a été équitable et le groupe a fonctionné dans une bonne dynamique tout au long de la réalisation du projet. L'amitié sous-jacente à la constitution du groupe conjugué à l'utilisation d'outils comme TeamSpeak nous ont assuré une communication efficace, un relai clair des informations et une progression rapide dans l'avancement du projet.

III) Difficultés rencontrées

La première difficultée que nous avons rencontrée est bien sûre liée à l'outil dans lequel le bot est réalisé, c'est à dire Python. En effet, nous n'avons jamais utilisé Python depuis le début de notre cycle ingénieur, nous n'étions donc pas familier avec sa syntaxe, ses structures de données ou ses règles d'indentation très peu permissives. Cela a toutefois été l'occasion de découvrir le langage, ses forces et ses faiblesses, mais il est clair que passer d'une utilisation intensive du C et du Java au Python n'est pas intuitive. Tout particulièrement, l'utilisation des boucles est pratique mais peu intuitive lorsque l'on est habitué à parcourir nos structures de données à la main. Enfin, l'utilisation intensive des expressions régulières pour répondre à nos besoins nous ont donné du fil à retordre, et il aura fallu beaucoup se renseigner sur ces dernières pour réellement maîtriser leur utilisation.

Au delà de ces considérations, la méconnaissance du langage était également un frein dans l'implémentation d'algorithmes plus ou moins complexes, qui sont aisés à implémenter par exemple en C mais qui le sont moins en Python (venant par exemple du fait qu'une phrase soit une chaîne de caractère en C terminant par ‘
0’ alors qu'en Python les phrases sont des Objets String et possèdent tout un lot de méthodes pour les manipuler).

Ensuite comme évoqué précédemment, l'idée initiale était d'utiliser une base de donnée avec PostGreSQL et de l'interroger pour obtenir les résultats correspondants aux données entrées par l'utilisateur. Nous avons cependant choisis de bifurquer vers la génération d'un fichier texte dont nous avons choisi le format des données, pour deux principales raisons qui sont la facilité de mise en place (donc le temps) et la complexité d'utilisation de l'outil, en considérant que nous travaillons dans un langage nouveau. Alors que les TP 1 et 2 furent l'occasion de découvrir comment ouvrir, lire et écrire dans un fichier texte, il nous semblait plus délicat d'interagir avec un gestionnaire de base de données et pour ces raisons nous nous sommes contentés de la génération d'un fichier texte. Il est cependant à souligner que nous aurions pu décider d'implémenter “en dur” quelques vols et ne plus s'en soucier pour les besoins de l'exercice, mais dans une optique où l'amélioration future du projet est un axe de développement et où la propreté du travail est primordiale, nous avons préféré nous axer sur une génération dynamique et modulable.

Enfin, la dernière difficulté majeure rencontrée fut le format à choisir pour les données : listes de mots ou mots séparés par des espaces ? Dates et heures séparés ou regroupés ? Stockées sous forme d'un timestamp ou d'un string ? A chaque fois les données étaient réutilisées à plusieurs endroits mais de différentes façon. C'est le problème lorsqu'on utilise des fonctions déjà fournies (regex, algorithmes de parcours, formateurs de dates) : elles prennent en entrée des structures bien définies et contraignantes, et cela nous a obligé à convertir nos données dans d'autres formats dans beaucoup d'endroits du programme. Pour l'analyse de simples phrases en local, la perte en performances est négligeable. Cependant, si on lançait notre script sur un serveur et que beaucoup d'utilisateurs se connectaient en même temps, les ralentissements seraient conséquents.

Avancement et Conclusion

Pour conclure sur la réalisation de ce projet et son avancement, nous voulons d'abord dire que la découverte de Python et des expressions régulières dans le cadre du projet fut intéressante, que ce soit d'un point de vue scolaire ou d'un point de vue personnel. Python est un langage très utilisé dans l'industrie et les applications pratiques sont souvent le meilleur moyen de s'approprier une technologie, c'est donc un plus que ce projet nous a apporté par rapport aux autres.

Il reste quelques bugs au niveau de l'extraction d'informations et encore plein de cas non traités (par exemple la vérification de la validité des dates) que nous n'avons pas pu résoudre par contrainte de temps et du fait du très grand nombre de cas possibles de formats. Ce projet a un haut potentiel d'évolution puisqu'il serait possible d'ajouter plein d'autres critères pour le choix d'un vol : fourchette de prix, nombre d'étapes, classe de voyage, distance de l'aéroport, etc.

Finalement, nous avons trouvé le projet intéressant en soi, pas seulement de par l'utilisation de Python mais surtout au regard des fonctionnalités implémentées. Nous comprenons maintenant mieux comment un Chatbot fait pour traiter les demandes qui lui parviennent, et de manière plus générale ce fut l'occasion d'appréhender le domaine du traitement et de l'interprétation d'information avec une forte composante humaine. L'aboutissement du projet aura posé les bases d'une compréhension technique supérieure aussi bien du Python que des Regex et des Chatbot ; en cela nous ne sommes pas déçus de notre choix.