Traitement Automatique du Texte en Intelligence Artificielle

DENOS Alexandre
ALESSANDRO Kévin

Table des matières

Introduction

Dans le cadre du projet de Traitement automatique du texte en Intelligence Artificielle nous avons choisi de concevoir un générateur de phrase de Gordon Ramsay.

Gordon Ramsay est un grand chef cuisinier et restaurateur britannique. Il a ouvert de nombreux restaurants dans le monde. Il est également présentateur ou mis en vedette dans plusieurs émissions télévisées comme Cauchemar en Cuisine, MasterChef, ou The F Word.

Dans la culture populaire, il est connu pour sa personnalité assez perfectionniste, ainsi que son caractère très critique et facilement irritable.

L'idée est donc de capturer ce caractère unique à travers les phrases générées par notre projet.
C'est un projet qu'on a trouvé original et qui n'était pas exactement présent sur internet.

Gordon Ramsay

Description des tâches

Pour aboutit le projet s'est vu être divisé en différentes étapes de réalisation, chacune demandant un effort chronophage, un effort de réflexion, ou les deux. Dans l'ordre de leur réalisation, ces tâches sont :

- Rassembler des données : nous avons collecté du texte représentant des phrases que Gordon Ramsay a prononcé dans des émissions, en particulier l'émission *Kitchen's Nightmare*. Pour cela, nous avons cherché le script de l'émission sur Internet.
- Formatter les données : une fois un script obtenu, nous avons dû filtrer les phrases qui étaient liées à notre projet – c'est à dire les phrases que Gordon a prononcé (par opposition aux phrases des autres intervenants). Certains scripts censuraient des mots ; pour conserver le naturel nous avons dû trouver (avec d'autres sources) quels étaient ces mots. Enfin, les phrases de Gordon ont été inscrites dans notre base de données (un fichier txt).
- Parseur : convertir la base de données dite « brutes » (le texte) en une base de donnée « utilisables ». Dans notre cas, une donnée que notre langage de programmation (Python) sait manipuler.
- Algorithme : le cœur principal de la génération de phrases. Grâce aux données manipulables créées par le parseur, on produit et affiche les phrases Gordon Ramsay-esque.

Conception et Implémentation de l'algorithme

Le procédé de *computation* se divise en deux parties, le parseur et le générateur. Les instructions sont, comme mentionné plus haut, exécutées en Python.

Le parseur :

Il s'agit de transformer du texte, c'est à dire une potentielle très grande string en input, en une liste de citations.
Une ligne de texte dans notre base de données en .txt correspond à une « citation ». Une citation (*quote*) dans notre projet est un ensemble de 1 ou plusieurs phrases, liées sémantiquement entre elles. Un retour à la ligne (\n) est présent dans le fichier .txt dès que on jugeait qu'une phrase changeait suffisamment le contexte, terminant le quote.
Chaque citation est composée de phrases, qui elles-mêmes sont composées de *morphemes*.

Dans notre programme, le caractère « espace » définit quand un morphème se finit. La ponctuation peut faire donc partie d'un morphème si elle est collée à un mot. Un mot avec ou sans majuscule n'est pas considéré comme le même morphème.

Dans l'exemple suivant :

To be, or not to be: that is the question.

Les morphemes sont « To » « be, » « or » « not » « to » « be : » « that » « is » « the » « question. » . Chaque morphème n'est présent qu'une fois ici. « To » et « to » sont considérés comme différents. « be, » et « be : » sont également deux morphèmes différents.

La base de donnée dite utilisable par l'algorithme est une structure de donnée de type « dictionnaire ».
Les clefs sont des tuples (word1, word2) représentant deux morphèmes word1 et word2 avec word2 étant le morphème succèdant word1 (avec un espace entre les 2). Il s'agit tout simplement d'une paire de 2 mots successifs.

La valeur de cette clef est une liste comme suit :

[x, [word3, a], [word4, b], [word5, c], [word6, d]....]

Cette liste signifie que word3 succède word2 (qui lui même succède word1) dans a contextes, selon les informations de la base de données.

Et que word4 succède word2 dans b contextes, word5 succède dans c contextes, etc. La valeur x au premier index est juste la somme a+b+c+d+...

Exemple concret issu de notre programme :

("That", "is") = [3, ["a", 2], ["disgusting", 1]]

Cette entrée du dictionnaire représente l'information que la séquence « That is », dans les textes sources, a à 2 reprises été suivie du morphème « a » et 1 fois suivie du morphème « disgusting ». Un total de 3 contextes. Aussi, il y a une valeur de string spéciale : « NoneType », qui représente la fin d'un quote.

("in", "there?") = [3, ["It's", 1] ["NoneType", 1] ["That", 1]

« in there ? » a été suivi avec une nouvelle phrase qui commence par le morphème "It's" 1 fois ou par "That" 1 fois, et aussi à 1 occasion, "in there ?" n'est suivi par rien – ce qui est considéré dans notre projet comme la fin d'une citation.

Un autre cas spécial : les quotes à 1 morphème :

("Lovely.", "NoneType") = [1, [« NoneType », 1]]

Cela signifie que Gordon a prononcé « Lovely. » en 1 quote (==> que nous avons jugé que ce Lovely. n'était pas sémantiquement lié à la précédente ou suivante phrase de Gordon à ce moment).
Lovely n'est donc suivi de rien (une fin de quote).

Le générateur

Une fois le dictionnaire généré, le programme va générer les n messages demandés par l'utilisateur. Un message mesure au minimum 120 caractères, c'est une valeur purement arbitraire qu'on peut changer sans vraiment altérer le principe de l'algorithme.

Tant que le message ne fait pas 120 caractères au moins, un nouveau quote est demandé à être généré. Un quote se termine quand le morphème « NoneType » est rencontré.

Pour commencer l'écriture, le programme sélectionne un tuple de mots (word1, word2) dans le dictionnaire, complètement au hasard, MAIS parmi les word1 qui peuvent commencer une phrase (c'est-à-dire les word1 commençant par une majuscule). Ainsi word1 et word2 sont les 2 premiers mots de la phrase générée.

Par la suite, le programme garde en mémoire les 2 dernières morphèmes écrits, et à chaque étape, va choisir aléatoirement mais basé selon le taux d'apparition le prochain mot à écrire.

Dans l'exemple plus haut :

("That", "is") = [3, ["a", 2], ["disgusting", 1]]

Si les 2 derniers morphèmes en mémoires sont « That » et « Is », il y a 2/3 chance que le prochain mot sera « a », et 1/3 que il sera « disgusting ».
Il s'agit d'une chaîne de Markov de mémoire 2. En effet ce procédé détermine le mot n à partir des mots n-1 et n-2, et avec un respect envers leur taux d'apparition.
Et ainsi le programme génère des mots et forme un message. La condition d'arrêt est quand le programme choisit avec l'aléatoire de terminer le quote. Si le message fait bien au moins 120 caractères, il est alors fini et printé.

Jeux de données

Comme défini dans la description des tâches, les données ont été manuellement cherchées et formattées à partir de scripts d'émissions télévisées trouvés sur Internet. Il y a environ 17kb de données dans la version du projet active actuelle.

Résultats

Les tâches ont bien pû être menée à bout et le projet est « fini » par rapport aux objectifs ciblés. Les messages sont bien générés selon les règles de la chaîne de Markov et le dictionnaire des sets de donnée est correctement parsé avec les données brutes actuelles.

Comme il s'agit d'évènement aléatoire, tous les messages ne sont pas évidemment aussi travaillés ou uniques.

Taper debug au lieu d'un nombre à l'exécution du programme permet de voir la black box de l'algorithme pendant la génération du message.

Erreurs et imprévus

Il n'y a ni bug ni résultat qui va à l'encontre de notre concept et implémentation.

Evolutions du projet

La chaîne de Markov d'ordre 2 est assez puissante compte tenu de la simplicité qu'elle représente, elle retourne des résultats variés et tout de même un peu cohérent, dans notre programme, comme le Panthéon dans le repository git peut témoigner.
Ceci dit, nous sommes conscients que la taille du jeu de donnée est très faible dans notre projet : il y a très peu de données en source, ce qui fait que la variance de la génération n'est pas si diversifiée.

L'ordre de la chaîne de Markov pourrait lui également être augmenté à 3 ou 4 mots par exemple : mais ce niveau de précision, qui augmenterait considérablement le niveau de structure des phrases (le mot n'est décidé que selon les 4 mots précédents) doit être implémenté en parallèle d'un grand jeu de données. Il faudrait donc surtout collecter plus de données source.