LINFO1104 - TwitOz

Brieuc Dallemagne & Thomas Debelle

May 2023

1 Introduction

Dans le cadre du cours de *Concepts, paradigmes et sémantique des langages de programmation*, Il nous a été demandé de réaliser une application avec *interface graphique* utilisant un concept de base du *Natural Language Processing*, le **N-gramme**. Nous nous basons sur une base de donnée de tweet de Donald Trump [1] et nous devons prédire le mot suivant se basant sur les 2 précédents.

2 Exécution

Voici les différentes commandes utiles :

```
make                          %Compile le programme main.oz
make run                      %Lance et compile si besoin main.oz
make clean                    %Enleve .ozf, .ozp dans Pickle/Word et .txt dans User
make run TWEETS_FOLDER="path" %Avec path le chemin d'un dossier de fichier texte
```

3 Réalisation

3.1 Description

L'utilisateur écrit dans la boite de texte blanche son entrée et appuies sur le bouton *Result* et obtient son résultat dans la boite de texte noire en-dessous.

Il indique le N dans N-gramme dans l'input supérieure. Le programme retourne un résultat qui est celui d'un N-gramme si possible sinon un $(N-a)$-gramme. (a étant le premier arbre non vide).

Il peut également générer des phrases de X mots en utilisant le bouton *Infinity*. On indique le nombre de mots à rajouter à la suite de notre entrée dans la boite *Amount* et on appuie sur *Infinity*. Le résultat s'affiche dans la boite blanche et noire.

3.2 Fonctionnement et Discussion

Tout d'abord, nous lisons tous les fichiers en utilisant des Threads à faire. Nous "*nettoyons*" l'entrée en supprimant les espaces, les caractères spéciaux, ... et on met le tout dans une liste contenant des listes (une pour chaque fichier) de mots. Ainsi, nous possédons une liste de mot qui se suivent. Les mots gardent leur majsucule, la poncutation qui les suivent, ...

Ensuite, nous réalisons le N-gramme. Notre implémentation utilise des arbres *partiels* comme montré ci-contre. Donc on ne s'intéresse qu'au mot qui est composé dans le N-gramme. Si l'entrée utilisateur est "*Thank you this is*" et que nous sommes en 2-gramme, on ne retient que "*this is*" et on n'enregistre que les mots suivants ce bout de phrase. Cela permet d'utiliser qu'une faible portion de mémoire et ralentit moins la machine. On sauvegarde cela en Pickle ce qui sera plus détaillé au point 4.2.

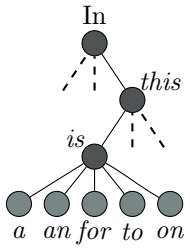

FIGURE 1 – Arbre partiel

1. Il était dit qu'il s'agissait de Elon Musk ce qui est faux au vu du contenu

Cela permet de fournir une réponse en moins d'une seconde (insérer vrai chiffre) pour une *nouvelle phrase* jamais rentrée par l'utilisateur et est quasi instantanée si l'utilisateur utilise un N-gramme déjà enregistré. Si nous avions opté pour charger le graphe N-gramme complet, cela nous aurait pris plusieurs minutes pour essayer avec tous les mots possibles. De plus nous aurions alloué une énorme quantité de mémoire pour une structure de donnée dont même pas 1% aurait été utile dans une utilisation normale. On peut néanmoins générer un graphe complet d'une profondeur 2 en cliquant sur le bouton Update Database.

Nous avons également créer un bouton *Randomizer* qui choisit au hasard parmi les mots suivants la phrase. Cela permet de générer de nouvelles phrases et ne pas tomber dans des boucles infinies.

4 extensions

4.1 * - Amélioration de l'interface graphique

Nous avons amélioré et rendu l'interface graphique plus *User friendly* et plus proche aux standards actuels. Nous possédons une barre d'outil permettant l'utilisateur de sauvegarder, générer le N-gramme, quitter, avoir des informations en plus, ... Nous avons également rajouter des boutons et des entrées pour simplifier l'utilisation du programme.

4.2 ** - Garder un historique de l'utilisateur

Nous sauvegardons des arbres partielles de N-grammes des entrées précédentes de l'utilisateur via des **Pickles** si cela est possible. Ainsi, on ne doit plus relire et recréer un N-gramme. Nous pouvons simplement le charger et être plus rapide. On peut également sauvegarde les sorties des utilisateurs dans un fichier *texte* dans le répertoire *User* via le raccourci *Control-S*.

4.3 ** - Généralisation de la formule du N-gramme

L'utilisateur peut spécifier le N du N-gramme sans que cela pose aucun soucis. (voir 3.1 pour plus d'informations sur le fonctionnement)

4.4 ** - Ajouter des bases de données custom

En appuyant sur le bouton "*update database*" on peut relancer le processus de lecture des fichiers. On peut donc mettre à jour la base données en modifiant les fichiers texte dans le dossier "*tweets*" ou en spécifiant une autre base de donnée dans le *makefile*. Pour préciser un dossier différent, réaliser la dernière ligne de la section 2.

4.5 *** - Optimisations de la formule du N-gramme

On ne charge qu'un arbre décisionnel *partiel*. Ainsi, on recrée des sous-arbres constamment mais moins gourmands en puissance de calcul, mémoire, ... Au plus l'utilisateur utilise notre programme au plus nous avons un arbre N-gramme complet. Cette façon d'aborder le problème est plus simple et un grand bonus pour l'utilisateur. Cette méthode d'optimisation est rendu très efficace via cette extension 4.2.

Références

[1] N-gramme — Wikipédia. https://fr.wikipedia.org/wiki/N-gramme.

[2] *n*-gram. https://en.wikipedia.org/w/index.php?title=N-gram&oldid=1143917543, March 2023. Page Version ID : 1143917543.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. O'Reilly, Beijing ; Cambridge [Mass.], 1st ed edition, 2009. OCLC : ocn301885973.

[4] Beranger Natanelic. NLP : Génération de texte par n-grams. https://bit.ly/3mNVGbJ, October 2020.