Rapport du projet Twit-Oz – Groupe T

Anton Lamotte - 43031800 — Aurélien Larue - 17611800

Informations pratiques

Objectif. L'objectif de ce programme est de prédire, lors de l'écriture d'un texte, le mot suivant, à-partir des mots précédents, moyennant l'analyse d'une base de données de fichiers texte. En l'occurrence, la base de données sont 208 fichiers contenant chacun jusqu'à 100 tweets de Donald Trump.

1 Fonctionnement du programme

Comme conseillé dans l'énoncé du projet, notre programme se divise en 3 : la lecture des fichiers donnés, le parsing des lignes et des mots se trouvant dans ces fichiers, et l'ajout à un dictionnaire selon le principe des 2-grams. Un nombre N de threads sont assignés à la lecture des fichiers, qui renvoient chacun les lignes qu'ils lisent vers un port (unique à chaque thread), dont le stream est récupéré par les N threads de parsing. À chaque thread de lecture correspond un port, dont le stream correspond à un thread de parsing. À leur tour, les N threads de parsing renvoient les lignes formatées et divisées en mots vers un port, dont le stream est récupéré par l'unique thread d'écriture. Celui-ci, finalement, écrit les 2-gram dans le dictionnaire.

Le dictionnaire est constitué comme suit : [je # [suis # [grand # 4 petit # 3] mange # [des # 1 une # 3]] tu # [es # [grand # 2]]], ce qui signifie que la suite de mots "je suis" est suivi 4 fois par le mot "grand", 3 fois par le mot "petit", etc.

Pour connaître le mot le plus fréquent après une suite de mots Word1 et Word2, il suffit de prendre le subdictionnaire Word1 dans le dictionnaire global, ensuite de prendre le subsubdictionnaire Word2 dans le subdictionnaire Word1, et finalement de regarder quel est le mot le plus fréquent.

2 Choix de conception

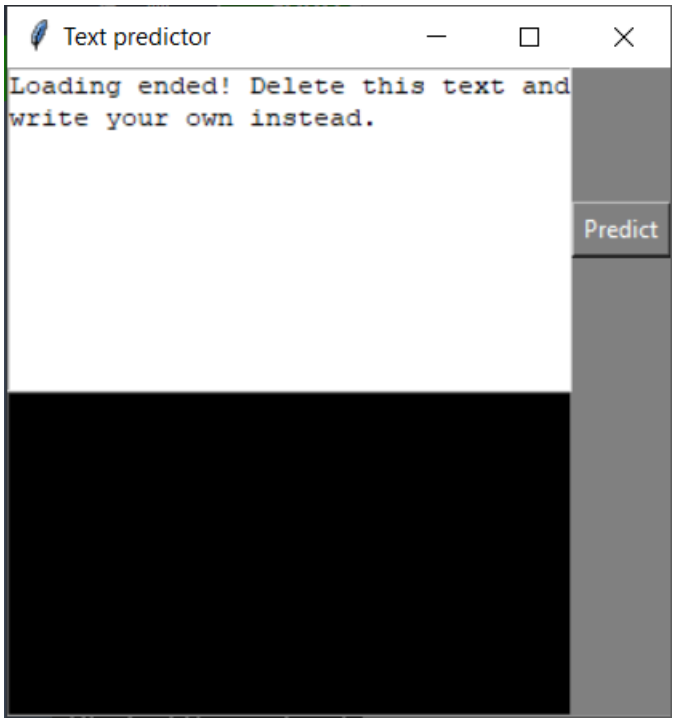
Lors de l'écriture de ce programme, nous avons plusieurs fois été confrontés à des choix de conception. En voici un bref récapitulatif.

1. *Suppression de la ponctuation.* Dans un premier temps, nous avions décidé de ne supprimer que la ponctuation que nous considérions « inutile », telle que les points, les points d'exclamation et d'interrogation, et les virgules, car ils ne sont pas d'une grande utilité, si aucune analyse syntaxique n'est effectuée sur les mots plus pour évaluer leur nécessité ou importance, chose qui n'était pas demandée dans ce projet. Par la suite, nous avons été confrontés à la difficulté de repérer les apostrophes et guillemets, car ceux fournis dans les tweets ne sont pas des apostrophes et guillemets classiques, mais des caractères non-ASCII non reconnus par Oz,

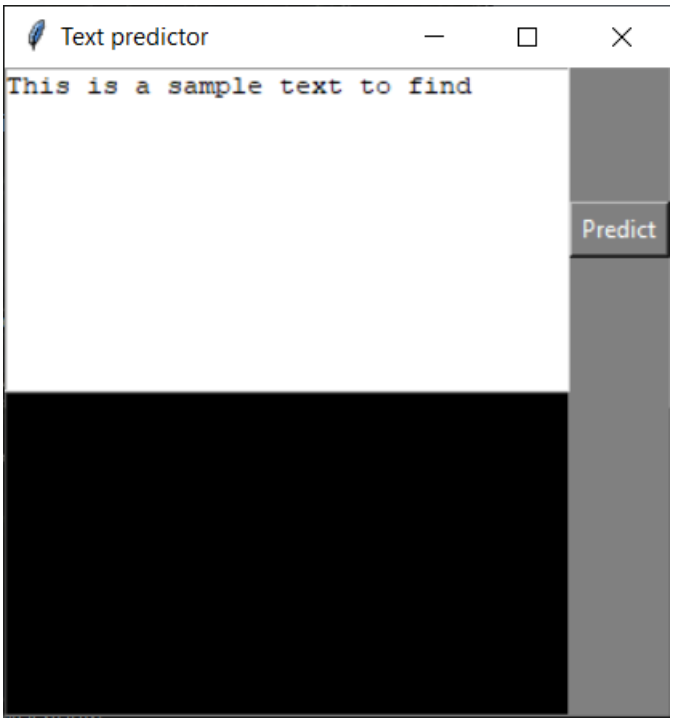
ce qui compliquait grandement leur traitement. Nous avons donc pris la décision de supprimer tout bonnement tout ce qui ne serait pas une lettre majuscule ou minuscule, ou un espace. Ce choix de conception concerne à la fois la base de données et le texte introduit par l'utilisateur, même s'il lui est toujours possible d'introduire de la ponctuation et des apostrophes (qui ne seront alors pas prises en compte).

2. *Gestion des majuscules.* Dans une même optique, nous avons décidé de ne pas prêter attention aux majuscules ou minuscules d'une phrase, bien que cela eût pu être pertinent, car il est évident que savoir quels mots se retrouvent fréquemment en début de phrase aurait eu une influence sur la prédiction des mots. Néanmoins, sachant qu'on trouve, dans les tweets de Donald Trump, de nombreuses majuscules à divers endroits, pas nécessairement situés en début de phrase, et parfois même, des mots entièrement écrits en majuscules. Comme expliqué précédemment, étant donné que nous ne prenons pas les points en compte, il aurait été difficile de déterminer quelle majuscule correspond à un début de phrase, et quelle autre majuscule n'était qu'une emphase apportée par Mr Trump à son tweet.
3. *Gestion des espaces et passages à la ligne.* Avec pour objectif une facilité d'emploi de l'application par l'utilisateur, nous avons également pris la décision d'effacer tout double espace ou passage à la ligne. En d'autres termes, un utilisateur distrait, ayant appuyé deux (voire trois !) fois sur sa barre espace, aura la même prédiction que s'il n'avait appuyé une fois.
4. *Nombre de threads par fonction.* Dans une première version du programme, la partie de lecture était exagérément plus lente que les deux autres parties. (25 secondes, contre 4 et 3 secondes, lorsque tournées séparément) Nous avions donc décidé de créer plusieurs threads de lectures, qui renverraient les lignes lues vers un port dont l'unique thread de parsing récupérerait le stream. Après une amélioration qui mena à une accélération fulgurante de la fonction de lecture (réduite à moins de 2 secondes), nous nous sommes rendus compte de la nécessité de mettre plusieurs threads sur la fonction de parsing. Étant donné qu'il est très facile, en Oz, de réunir plusieurs streams, grâce aux ports, mais beaucoup plus laborieux d'en diviser un, nous nous sommes vus dans l'obligation de créer autant de threads de parsing que de threads de lecture, comme recommandé dans l'énoncé du projet. Lors du lancement du programme, il y a donc un nombre égal de threads de lecture et de parsing qui tournent, renvoyant leur résultat à l'unique thread d'écriture, de façon que le programme se termine immédiatement après que les threads de parsing ont terminé leur travail, car le thread d'écriture est toujours plus rapide que ceux-ci.
5. *Nombre de threads.* Le nombre de threads par défaut est 4, car nous avons supposé que la probabilité que quelqu'un fasse tourner ce programme sur un ordinateur disposant de moins de 4 coeurs est faible. Il est toutefois possible de changer ce nombre générique de threads en changeant la variable NbOfThreads du programme.
6. *Suppression du fichier Reader.oz.* Étant donné que nous avons retravaillé la fonction de lecture de fichier fournie avec l'énoncé du projet et que celle-ci envoie désormais directement ses résultats à un port, nous avons préféré l'inclure de le fichier main.oz, rendant ainsi l'existence du fichier Reader.oz inutile, ce pourquoi nous l'avons supprimé.

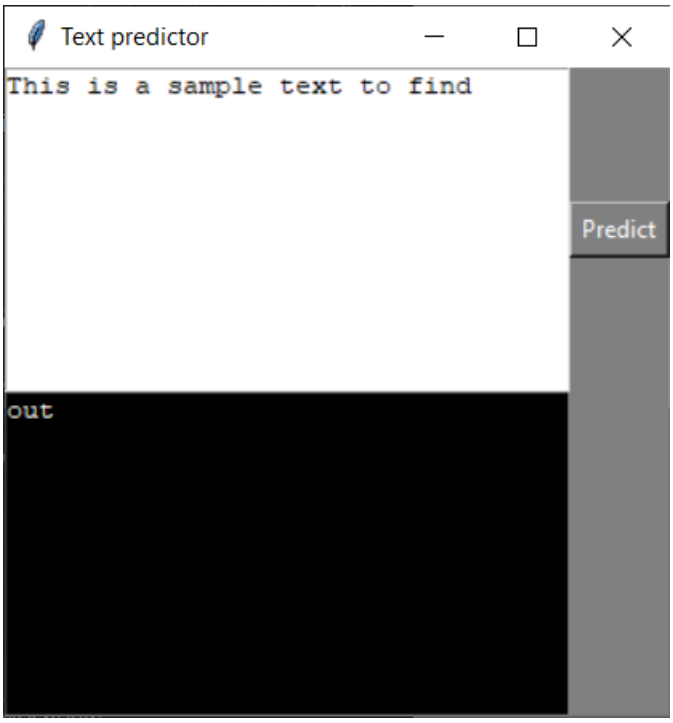
3 Utilisation du programme



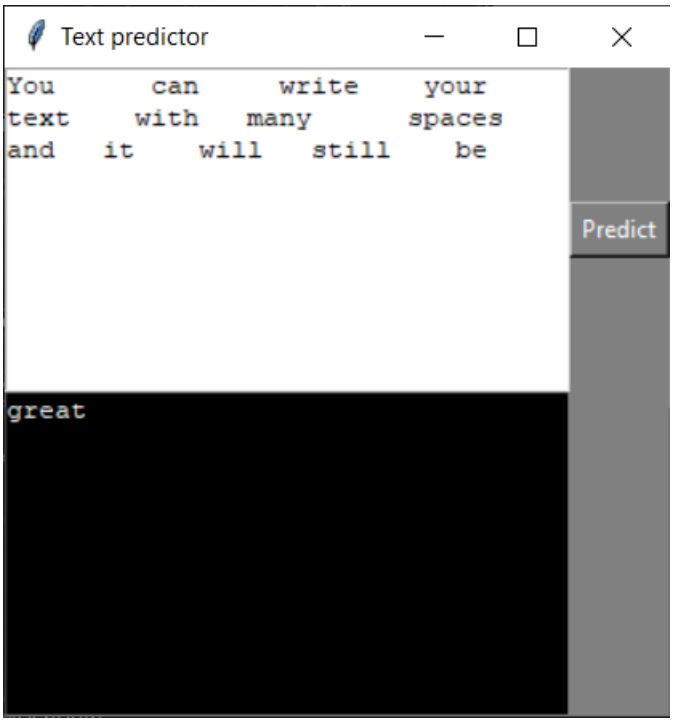
1. Une fois le programme compilé et lancé (cf. Readme, disponible à l'adresse suivante : `https://github.com/inmisericors/Twit-Oz/blob/master/README.md`), attendez que le programme ait fini de charger (c'est-à-dire, de lire les fichiers, de les parser et de les réécrire dans le dictionnaire de 2-grams), vous pouvez effacer et écrire un texte dans la zone blanche.



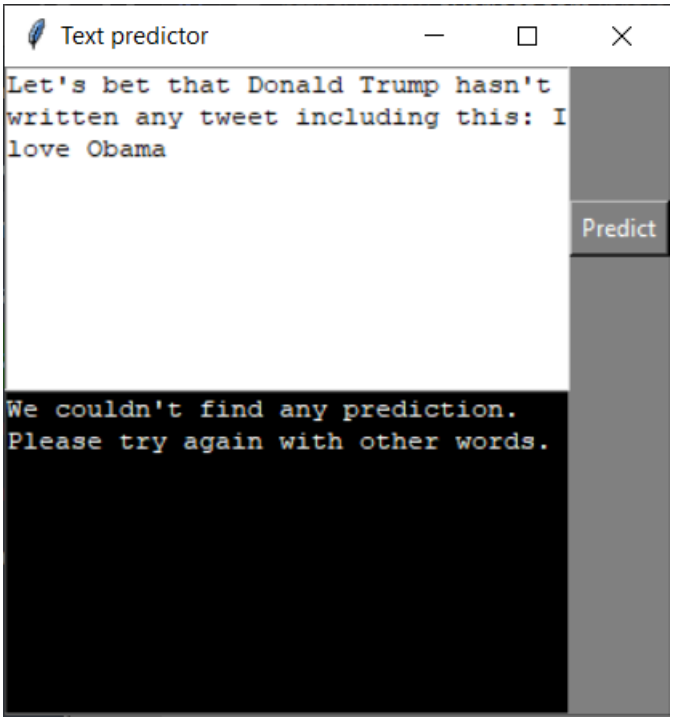
2. Ensuite, appuyez sur le bouton « Predict » pour obtenir votre prédiction.



3. L'application fonctionne même si vous insérez trop d'espaces.

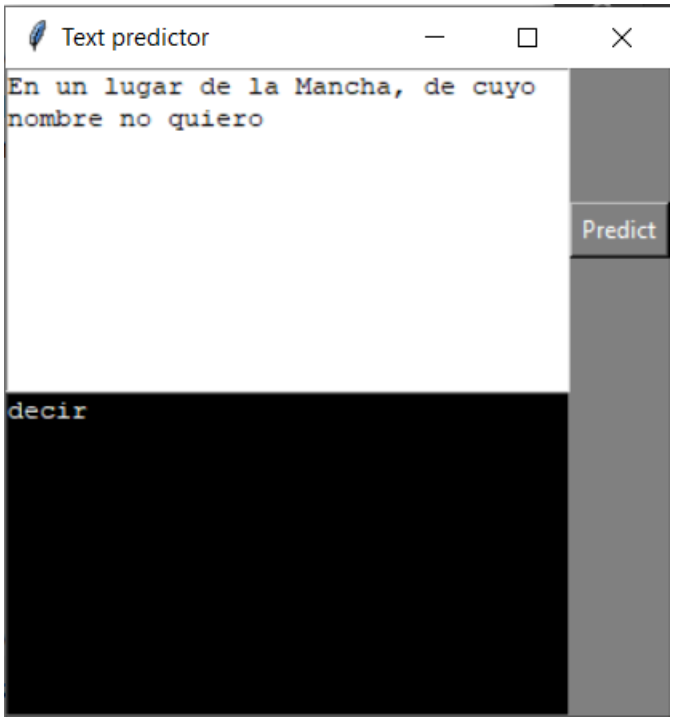


4. Finalement, l'application vous avertira si la base de données fournie ne lui permet pas de prédire un mot sur base des mots que vous avez entrés.



Bonus

Nous avons trouvé, sur internet, une version de Don Quijote en 2000 lignes au format .txt, que nous avons inclus comme 209e fichier texte dans le dossier tweets. Il suffit de changer le dernier argument de la fonction {LaunchThreads} de 208 à 209 pour le prendre en compte ! (Évidemment, les threads ne sont plus d'une grande utilité pour ce fichier, étant donné qu'il fait à lui tout seul 2000 lignes, mais nous n'avons pas pris le temps de subdiviser le fichier en plusieurs).



Dommage, les 2-gram n'auront pas été suffisants pour prédire l'incipit du roman le plus vendu de l'Histoire ! (« En un lugar de la Mancha, de cuyo nombre no quiero acordarme »)