UCLouvain

École polytechnique de Louvain

LINFO1104 - Concepts des langages de programmation

Projet TwitOZ : un prédicteur de texte

Groupe 15

Auteurs :
Delsart Mathis
Kheirallah Cédric

Enseignant :
Van Roy Peter

Assistants :
Crochet Christophe
Sprockeels Damien
Wirtgen Thomas

2022-2023

1 Concept du projet

Le projet TwitOZ consistait en l'implémentation d'un prédicteur de texte codé dans le langage Oz et devait être capable de prédire le mot suivant l'input de l'utilisateur en se basant sur les N derniers mots de celui-ci.
L'implémentation de l'application a nécessité l'utilisation de multiples savoirs vus au cours LINFO1104 tels que les threads, les fonctions récursives terminales, les structures récursives telles que les arbres, les listes, etc.

2 Choix de l'implémentation

2.1 Structure de donnée

Afin de stocker les résultats, nous avons décidé d'implémenter un arbre binaire ordonné ayant comme clés les différents mots N-gramme et comme valeurs des sous-arbres binaires ordonnés. Ces sous-arbres ont comme clés des fréquences X et comme valeurs des listes de mots apparaissant X fois après le mot N-grammes.
La structure récursive de l'arbre binaire ordonné est la suivante :
obtree = obtree(key :Key value :Value tleft :obtree tright :obtree) | leaf.

Nous avons choisi de faire des arbres binaires ordonnés afin d'accéder facilement aux éléments avec une complexité en $O(\log n)$ et non en $O(n)$. De plus, pour chercher les mots de plus haute fréquence, il suffit de prendre la valeur la plus à droite possible dans l'arbre.

2.2 Parsing

Pour le parsing, chaque fichier est analysé de manière parallèle à l'aide de threads. Les étapes effectuées par les threads sont les suivantes :

- Les fichiers sont lus ligne par ligne pour éviter les mots en trop dans l'arbre
- Les caractères mal encodés (comme â/x80/x99 = ') tel que “ et ” sont remplacés par des espaces et ' par ' (on ne voit pas la différence sur Latex mais il y en a bien une).
- Tous les caractères spéciaux sont remplacés par un espace.
- Toutes les majuscules sont remplacés par des minuscules.
- Les chiffres sont gardés tels quels.
- Tous les espaces inutiles sont enlevés.

Ainsi une phrase comme "...I will tell you : "Yeah that's fun 99!"" donnera "i will tell you yeah that's fun 99".

3 Extensions implémentées

Nous avons trié les extensions dans l'ordre de facilité d'implémentation. Plus de détails sur l'implémentation du projet sont donnés dans le README.

3.1 Proposer plus d'un N-gramme à l'utilisateur (∗)

Dans cette extension, la prédiction propose tous les mots possibles de plus haute fréquence, ainsi que leur fréquence d'apparition et leur probabilité. Si vous entrez "i have been" en input, vous aurez la liste [so a] en sortie, ainsi que la fréquence 15 et la probabilité 0.4.

3.2 Généralisation de la formule du N-gramme (∗∗)

Notre généralisation de la formule du N-gramme fonctionne pour tout N >= 1. Pour l'effectuer, le programme parcourt tous les éléments d'une liste de mots et pour chaque élément, il leur applique une concaténation (avec comme délimiteur un espace) avec les N - 1 mots suivants. L'algorithme s'arrête évidemment après Length List - N + 1 itération(s) puisque, dans ce cas, il n'y a plus N éléments qui suivent.

3.3 Améliorer l'interface graphique existante (∗)

Nous avons rendu l'interface graphique plus agréable en y ajoutant des couleurs contrastées dans l'effet bleu/noir ressemblant aux couleurs de ChatGPT et une couleur de fond bleu ciel, couleur officielle de Twitter (vu que les données sur lesquelles nous avons travaillés sont des tweets). Nous y avons également ajouté une image du logo de Twitter avec le mot Oz à l'intérieur afin d'évoquer la coopération entre les tweets de la base de données et le code dans le langage de programmation Oz.
Nous avons également mis un titre et des boutons lisibles et réactifs à la souris passant dessus. Les actions de ces boutons sont expliquées dans les extensions ci-dessous.

3.4 Modifier le Makefile pour pouvoir spécifier les extensions (∗)

Nous avons amélioré le Makefile fourni de base afin de laisser à l'utilisateur le choix du N-gramme et des extensions qu'il souhaite utiliser pour ses prédictions ainsi qu'une commande d'aide pour avoir un aperçu de toutes les commandes et options disponibles. Toutes les commandes sont expliquées en détail dans le README.

3.5 Proposer la correction de mots dans une phrase déjà existante (∗ ∗ ∗)

Si cette extension est activée, l'utilisateur peut entrer un mot dans la boîte de correction adjacente et le programme relancera une prédiction sur chaque mot correspondant dans l'input en prenant en compte les N mots précédents ceux-ci (selon le N-gramme choisi par l'utilisateur). Par exemple : "I am going there so I must stop there". Une demande de correction sur "there" avec un bi-gramme donnera "to" pour le 1er "there" et "the" pour le 2ème "there". Si le mot est déjà correct ou qu'il ne trouve aucune autre prédiction, le programme le dira.

3.6 Proposition automatique (∗ ∗ ∗)

Si l'utilisateur choisit d'activer cette extension, la prédiction se fera automatiquement. Le bouton 'Predict' sera par conséquent inutilisable.
La prédiction automatique s'effectue par un thread toutes les 0.5 secondes en tâche de fond. L'extension a également été améliorée afin de compléter le mot que l'utilisateur est en train d'écrire.

Le principe est le suivant (avec un bi-gramme) :
Prenons la phrase "Hello, I have 'word'", la prédiction va d'abord essayer avec "have 'word'", si un mot est trouvé, alors il le prédit. Sinon, le prédicteur essaye avec "i have" en prenant en compte le préfixe du mot 'word'.
Par exemple : entrer "Hello, i have b" ou "Hello, i have bee" en input proposera "been" afin de compléter le dernier mot. Si vous entrer "Hello, i have be", le prédicteur proposera "able" car le bi-grammes "have be" existe dans la base de données.

3.7 Ajouter des bases de données custom (∗∗) + Garder un historique des inputs de l'utilisateur (∗∗)

Pour ces deux extensions, nous avons implémenté la possibilité pour l'utilisateur de modifier la base de données sur laquelle se base le programme de prédiction afin que les nouvelles données soient prises en compte lors des prochaines prédictions.
L'arbre est directement modifié sans devoir quitter l'application. A chaque exécution du programme, les données sont analysées comme les fichiers se trouvant dans le dosier spécifié par l'utilisateur.
Nous avons également offert la possibilité de supprimer l'historique directement depuis l'interface utilisateur (ou depuis le Makefile) pour ne plus prendre en compte les anciennes données dans les prochaines prédictions. Ce processus n'est pas immédiat, il faut quitter l'application et la relancer pour ne plus prédire les données présentes dans l'historique de l'utilisateur.
Pour offrir plus de flexibilité dans la gestion des fichiers, on peut également interagir avec des fichiers texte déjà présents sur l'ordinateur ou même en créer depuis TwitOZ.
En résumé, l'utilisateur peut :

- ✓ sauvegarder le texte écrit en input dans la base de données
- ✓ charger un fichier texte de son ordinateur et le sauvegarder dans la base de données
- ✓ sauvegarder le texte écrit en input dans un fichier texte sur l'ordinateur à l'emplacement de son choix
- ✓ charger le contenu d'un fichier présent sur l'ordinateur dans l'input de l'application
- ✓ supprimer l'historique de l'utilisateur qui est utilisé comme seconde base de donnée