Travail pratique 1

Expressions régulières, N-grammes et classification

Travail présenté à Luc Lamontagne
IFT-7022

Réalisés par :

Gabriel Rieger Junqueira
111 192 282

Vincent Gaudreault
111 267 663

Liv Nguekap
111 242 831

9 octobre 2020

Introduction

Ce travail consiste à compléter trois différentes tâches dans le but d'apprendre à analyser des textes grâce à des outils et des techniques de traitement de la langue naturelle. Entre autres, on travaille avec des expressions régulières, des modèles N-grammes et des techniques de classification de texte.

Tâche 1 – Expressions régulières

Cette tâche consiste à repérer des aliments et des quantités grâce à des expressions régulières. Le programme t1_extraction_ingredients.py effectue une lecture du fichier *ingredients.txt*. Avec l'aide de la bibliothèque *re* de Python, un découpage est ensuite effectué.

Exemple : get_ingredients("2 cuillères à café de poudre à pâte")
Le programme devrait retourner la paire : "2 cuillères à café", "poudre à pâte".

Puis, une lecture du fichier de solutions est effectuée pour comparer les résultats.

Résultats

Le score de performance représente le nombre d'items où une bonne détection a été effectuée.

Score en quantité : 83.22%
Score en ingrédients : 76.17%

Analyse

Trouver les expressions régulières qui capturent les quantités et ingrédients demandés était une tâche compliquée, surtout dû à l'inconsistance du formatage du fichier de solution. Par exemple, dans la ligne du fichier solution suivante, "1 cuillère à soupe (15 ml) fécule de maïs QUANTITE1 cuillère à soupe (15 ml)", le mot "quantité" n'est pas suivit des deux points comme dans les autres lignes. Aussi, dans d'autres cas, certaines valeurs de quantité d'ingrédients sont formatées différemment, et demandent par conséquent plus de temps pour pouvoir les gérer avec nos expressions régulières. Par exemple, pour la ligne du fichier solution "4 tranches de congre 150 g QUANTITE:4 tranches (150g) INGREDIENT:congre", l'espace entre le "150" et le "g" dans la quantité a été retiré. Donc, des variations de formatage similaires ont rendu l'accomplissement de la tâche vraiment complexe.

Tâche 2 – Modèles N-grammes

Pour cette tâche, quatre modèles de langue ont été construits pour compléter les proverbes. Trois d'entre eux proviennent de la bibliothèque NLTK tandis que le dernier a été tout simplement implémenté au complet en Python.

Au démarrage, le script effectue une lecture des proverbes d'entraînement. Une *tokenization* de chacun de ces proverbes est effectuée. Ce résultat permet ensuite de créer des N-grammes et le vocabulaire qui y est associé tout en prenant soin d'ajouter les caractères de début et de fin de phrase. Les différents modèles sont ensuite entraînés.

Après la lecture des proverbes incomplets de test, une probabilité logarithmique est calculée pour chacun des choix selon le modèle choisi. Le choix maximisant cette probabilité est sélectionnée et un score de perplexité est calculé pour le proverbe complet.
Exemple : phrase incomplète: aide-toi, le *** t'aidera
Choix pour ce proverbe : [médecin, **ciel**, professeur, whisky].

Une comparaison avec les solutions (listées en Python) est effectuée pour calculer le score de performance.

Pour ce qui est du développement du modèle bigramme, un compteur est utilisé pour conserver le vocabulaire. Pour le compte des N-grammes, un dictionnaire par défaut de dictionnaire par défaut d'entiers est utilisé. Lors de l'entraînement, une vérification d'existence de chaque mot est effectuée dans le vocabulaire. Si le mot est inexistant, il est alors remplacé par "UNK". Puis, le compte est incrémenté pour chaque contexte (premier dictionnaire) et mot (deuxième dictionnaire). Lors du calcul des probabilités logarithmiques, on vérifie aussi si le mot et le contexte existent dans le vocabulaire.

Résultats

Le tableau suivant montre la performance en test selon les différents modèles pour deux critères de sélection : une décision basée sur la probabilité logarithmique de chaque choix et une décision basée sur la perplexité pour l'entièreté du proverbe.

Modèle	**Performance pour décision en probabilité logarithmique (%)**	**Performance pour décision en perplexité (%)**
Unigrammes - NLTK	20	20
Bigrammes - NLTK	35	40
Trigrammes - NLTK	40	75
Bigrammes - Implémentation	35	40

Le tableau suivant montre la perplexité pour le premier et le dernier proverbe du fichier de test pour les différents modèles de langue avec une décision basée sur la probabilité logarithmique.

Modèle	**Proverbe**	**Perplexité**
Unigrammes - NLTK	Premier	434.13
Bigrammes - NLTK	Premier	1291.5
Trigrammes - NLTK	Premier	1803.7
Bigrammes - Implémentation	Premier	51.3
Unigrammes - NLTK	Dernier	194.7
Bigrammes - NLTK	Dernier	310.2
Trigrammes - NLTK	Dernier	888.0
Bigrammes - Implémentation	Dernier	32.9

Le tableau suivante montre la probabilité logarithmique pour le premier et le dernier proverbe du fichier de test pour les différents modèles de langue avec une décision basée sur la probabilité logarithmique.

Modèle	Proverbe	Probabilité logarithmique
Unigrammes - NLTK	Premier	-10.1
Bigrammes - NLTK	Premier	-9.8
Trigrammes - NLTK	Premier	-11.1
Bigrammes - Implémentation	Premier	-6.8
Unigrammes - NLTK	Dernier	-8.4
Bigrammes - NLTK	Dernier	-12.1
Trigrammes - NLTK	Dernier	-12.1
Bigrammes - Implémentation	Dernier	-8.4

Analyse

Notre implémentation du modèle bigramme arrive à obtenir la même performance que celui en NLTK. De ce point de vue, il ne doit pas y avoir d'erreurs d'implémentations.

D'une part, on peut clairement remarquer que les modèles trigrammes obtiennent un score plus performant que les modèles bigrammes ou unigrammes. C'est logique, puisqu'ils ont accès à un historique plus détaillé.

D'autre part, on peut aussi voir une amélioration significative des performances lors de l'utilisation d'une décision basée sur la perplexité pour le choix du meilleur terme. Malheureusement, nous n'arrivons pas à expliquer ce phénomène.

Tâche 3 – Classification

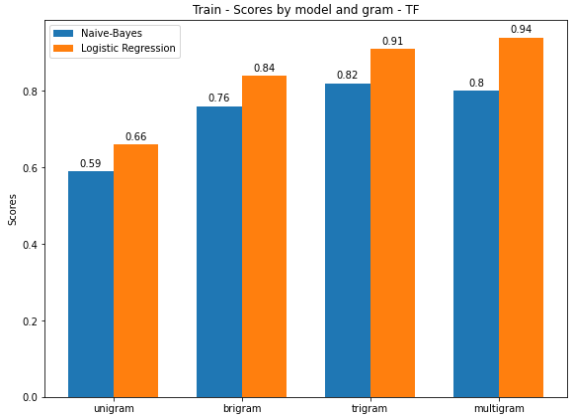
Le but de cette tâche est de déterminer la langue d'origine d'un nom. Par exemple, un nom débutant par Lamontagne devrait retourner "French".

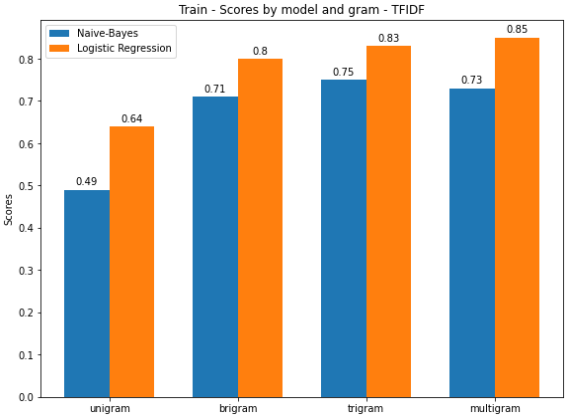
Tout d'abord, une *tokenization* au niveau des caractères était demandée. Ensuite, 16 modèles ont été entrainés pour répondre à toutes les possibilités à essayer (2 classificateurs, 4 modèles de langue et 2 types de poids). Les modèles de langue utilisés ont été unigrammes, bigrammes, trigrammes et multi-grammes. Les poids possibles testés ont été TF (compteur) et TF-IDF (fréquence relative). Les classificateurs utilisés ont été le bayésien naïf multinomial et la régression logistique.

Résultats

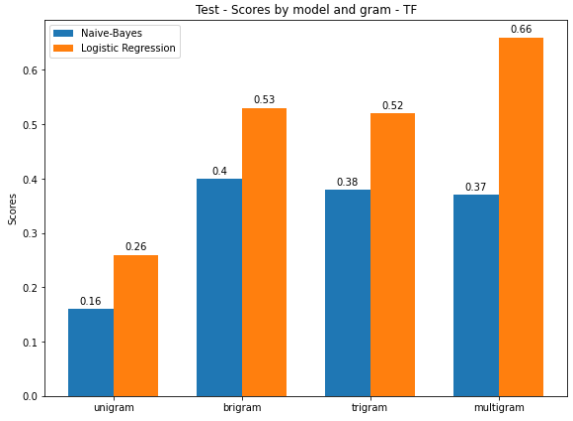
Les figures suivantes présentent les résultats des différents modèles en entraînement et en test.

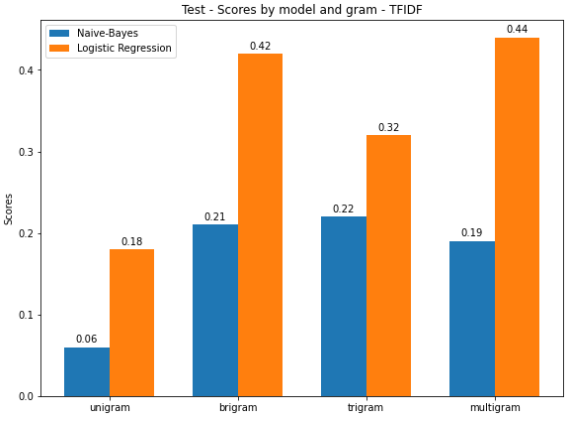
Scores en entraînement





Scores en test





Analyse

Il a été constaté que la régression logistique a une meilleure performance en entraînement et en test, que le modèle bayésien naïf multinomial dans toutes les modèles de langues et poids testés.

En général, le poids TF performe mieux que le poids TF-IDF. Pour les modèles de langue, les everygrams sont nettement supérieurs aux unigrammes et bigrammes. Ils sont aussi légèrement supérieurs aux trigrammes, quoi que ça soit plutôt négligeable.

Conclusion

La première tâche nous démontre la complexité d'écrire une expression régulière complète et performante. Une difficulté présente dans les trois tâches a été le fait de travailler avec des fonctions préétablies et l'interdiction de changer la signature des fonctions. Finalement, les techniques de traitement de la langue naturelle sont très puissantes malgré leur simplicité. Il ne faut pas oublier qu'aucune analyse sémantique ou syntaxique n'a été effectuée. Malgré cela, les résultats sont quand même très intéressants.