Projet Algorithmique et Programmation 2

Enseignent : Kim Gerdes
Membre : Xi Rong, Yunbei Zhang
Université Sorbonne Nouvelle

I. Introduction :

Ce projet est pour l'objectif de détecter la langue des fichiers d'un répertoire donné à l'aide d'un programme écrit en python3, et il sera capable de détecter la langue avec des algorithmes différents, et aussi donner aux utilisateurs la taux de réussite de chaque algorithme, il produit enfin un fichier au format tsv pour visualiser les résultats. Pour l'évaluation de notre programme, nous avons utilisé la mesure de précision.

II. Explication :

Dans cette partie nous allons expliquer notre projet basé sur le programme. Le programme principal compose de 8 algorithmes, dont un est ajouté après ce que nous avons vu en cours, il consiste aussi à détecter la langue. Dans un premier temps nous avons écrit une fonction *systemeEcriture* qui prends en entrée un texte et elle retourne enfin un "charactor set" ; En suite, nous avons pris une série de fonctions pour comparer la distance de mots ou de n-grammes ; d'ailleurs la fonctions *nb10mots* prend un texte à l'entrée et il est capable de sortir les 10 mots les plus fréquents de ce texte. Toutes les fonctions au-dessous sont vus en cours, et nous les avons écrits dans une liste qui s'appelle distanceAlgos. Par la suite, il y a une fonction *creerDicoTrain* qui prend en paramètre la liste que nous venons de décrire (distanceAlgos) et le nomDossierTrain("textesLangues" dans ce cas), et pour sortir un dictionnaire à multi dimension : { nom de l'algo --> { langue --> { token --> fréquence relative } } }. On peut intégrer cette fonction dans *lePlusProche*, qui prend chaque bout de texte à l'entée, l'algorithme et un dictionnaire { langue --> { token --> fréquence relative } }, il produit le code de langue ayant la fréquence minimale. La fonction *longueurQualite* consiste à calculer la taux de réussite de chaque algorithme, sa paramètre nbCar est un entier aléatoire. Puis pour former un tableau tsv, nous avons écrit une fonction *testSystematique* qui prend une liste d'entiers homogènes ou un longueur en entrée, il retourne un tableau de tsv, chaque longueur correspond aux taux de réussite de chaque algorithme(y compris l'algo ajouté).

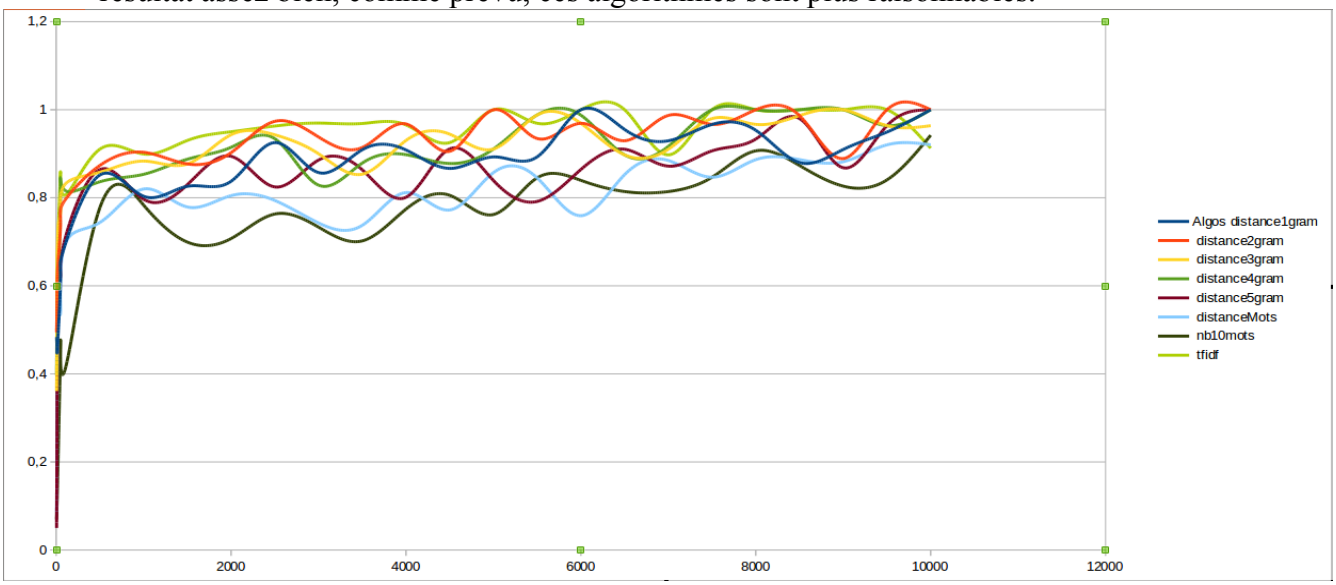
III. Ajout

En suivant l'objectif de détecter la langue des fichiers inconnus, nous avons ajouté un algorithme en plus. L'idée c'est d'utiliser la méthode de machine learning, puisque nous avons eu la connaissance de fouilles de textes et le TF-ID. Nous avons d'abord enlever tous les espaces, tabulations, les retours à la ligne. Après nous avons divisé le texte en plusieurs bouts, chaque bouts comporte 500 caractères. Pour chaque bout de texte, nous transformons chaque bout de texte en un vecteur, et l'associons au code de leur langue. Pour avoir une paire de code et bout de texte, eg, zip1 = *[(bout1, code1), (bout2, code1) .., (boutN, code1)]*. Après nous avons formé un dictionnaire d'ensemble d'entrainement, *trainingTexts = {'code1':zip1, 'code2':zip2...}*. À la base de ce dictionnaire, avec la connaissance de TF-IDF, nous avons pu transformer chaque bout de texte à un vecteur en important la méthode TfidfVectorizer du module sklearn. Nous utilisons l'ensemble de [vecteur, code] et avec l'aide d'un Classificateur-SGD qui est aussi dans le module sklearn pour entraîner notre modèle. Nous avons intégré cette fonction de *tfidf* dans la liste d'Algos afin de l'intégrer dans la fonction *longueurQualite* et pour qu'il apparaisse aussi dans le tableau tsv comme le résultat.

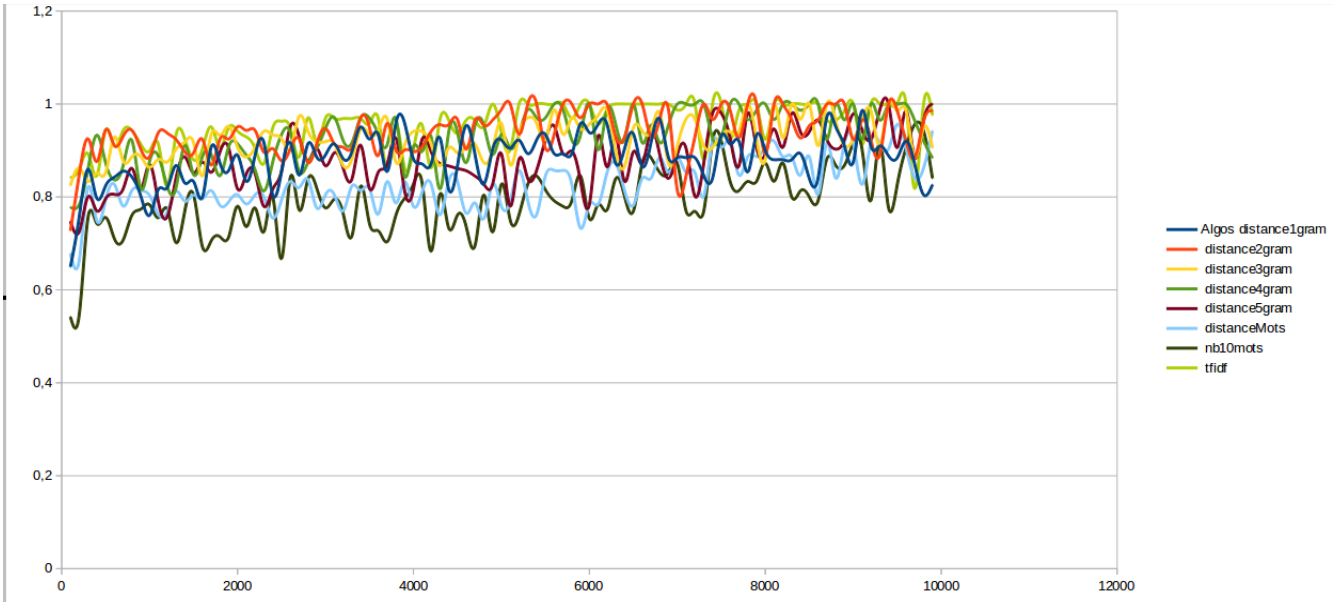
V. Analyse

Après avoir exécuté le programme, nous obtenons un fichier tsv, nous pouvons construire un graphe avec ce tableau. Dans le graphe 1, comme ce que nous avons attendu, nous voyons globalement que plus la longueur du texte est grande, plus la courbe se monte. Les deux graphes sont tous les résultats de notre programme sauf que dans le deuxième nous avons utilisé la

longueur de bout donné par le professeur, il est plutôt lent en temp de calcul et nous pouvons voir dans ce graphe 2, les courbes sont plus zigzag. Il pointe sur le fait que plus l'échantillon est grand, plus il est présentatif. Quand la longueur de texte est inférieur à 500, la taux de réussite n'est pas très optimal. Après quand la longueur atteint à 5000, toutes les courbes ont une tendance d'être stables. À souligné que, l'algorithme *nb10mots* et *distancesMots* sont moins efficaces lors que notre algorithme *tfidf* est le meilleur, *nb10mots* est moins efficace car il dépend de la taille de texte, si le texte est court, les 10 mots les plus fréquents sont moins représentatifs, aussi elle utilise *split()* pour découper des textes en mots, cependant pour les langues comme chinois, il n'est pas capable d'en découper en mots, c'est le même problème qui se pose sur la fonction *distancesMots*. Par contre, les séries de fonction N-grammes ont eu un résultat assez bien, comme prévu, ces algorithmes sont plus raisonnables.



Graphe (1)



Graphe (2)

VI. Difficulté

Dans cette partie nous allons parler des difficultés que nous avons rencontrées au cours de la réalisation de nos idées. D'abord, nous avons essayé de découper le texte par le retour à la ligne (\n), mais l'écart de la longueur de chaque ligne est trop grand, par exemple une ligne peut comprendre 500 mots lorsque des autres comportent 2 mots. Avec ce moyen, le taux de réussite n'est pas optimal, en conséquence, nous avons fixé une valeur de 500, c'est à dire, chaque bout de texte comporte 500 caractères. Ensuite, nous avons eu un autre problème lors que la transformation de texte en vecteurs, c'est le problème d'encodage, pour le résoudre, nous avons trouvé dans la documentation de TfidfVectorizer une paramètre *decode_error = 'ignore'*. L'autre problème qui se pose est le classificateur, d'abord nous avons essayé d'autres classificateurs qui ne sont pas assez bien que celui d'actuel, le SGD-Classifier (Stochastic Gradient Descent).