Université de Montréal

IFT6285 – Traitement automatique des langues naturelles

Devoir 1

---

par:

Eugénie Yockell
(20071932)

Date de remise: 18 septembre 2020

Nous avons un ensemble de fichiers de texte à analyser du corpus *1Bwc*. Nous cherchons à compter le nombre de mots contenu dans ce corpus. Nous avons d'abord itéré à travers les 99 fichiers de textes. Il suffit ensuite de lire le texte et le séparer en mot distinct dans une liste, soit la tokenisation. Compter la longueur de cette liste retourne le nombre de mots. Pour s'y faire, nous avons étudier différents tokenizers. Nous avons utilisé deux méthodes de la librairie `NLTK` `word_tokenise()` et `ToktokTokenizer().tokenize()`, ainsi qu'une méthode python, `split()`. `ToktokTokenizer()` est tokenizer plutôt simple où seulement le point final est tokenisé.

La méthode `word_tokenise()` sépare les mots selon différents séparateurs, tandis que la méthode `split()` sépare seulement par les espaces blancs. On se retrouve alors avec quelques problèmes. Par exemple, pour la phrase "players who took advantage of a quiet–and eventually sunny–afternoon on the day before". Avec `NLTK` on obtient,

['players', 'who', 'took', 'advantage', 'of', 'a', 'quiet, '–', 'and', 'eventually', 'sunny', '–', 'afternoon', 'on', 'the', 'day', 'before']

Tandis qu'avec `split()` on obtient,

['players', 'who', 'took', 'advantage', 'of', 'a', 'quiet–and', 'and', 'eventually', 'sunny–afternoon', 'on', 'the', 'day', 'before']

On recherche un tokenizer qui permet diviser les mots séparés par une ponctuation ou un espace blanc. Avec `ToktokTokenizer()`, la tokenisation est un peu plus sévère, par exemple pour la phrase "making him the Festival's", `ToktokTokenizer()` retourne,

['making', 'him', 'the', 'Festival', ' ' ', 's']

Tandis que `word_tokenise()` retourne,

['making', 'him', 'the', 'Festival', ' 's']

Les trois tokenizers génèrent un nombre total de mots différents décrit à la table 1.

	`split()`	`word_tokenise()`	`ToktokTokenizer()`
Mots totals	768 648 884	770 239 036	778 318 180

Table 1: Le nombre de mots total calculé dans les 99 tranches pour les trois différents tokenizers.

Le temps pris pour calculer le nombre total de mots selon le nombre de tranches considéré avec les trois méthodes est représenté à la figure 1. Comme on s'y attend, nous avons trois droites linéaires. En effet, chaque tranche contient environs le même nombre de mots et le temps pour compter les mots est constant pour une même grandeur de texte. Ces trois droites ont des pentes différentes où on remarque que `word_tokenise()` est très lent et prend près de 1h30 pour compter les mots dans 99 tranches. C'est pour cette raison qu'il était pertinent de chercher d'autre méthode de tokenisation. La méthode `split()` est très rapide, soit 2min18, mais n'était pas assez forte pour séparer les mots. Un bon compromis est la méthode `ToktokTokenizer()` qui tokenise bien les mots et prend 17min pour compter les mots dans 99 tranches.

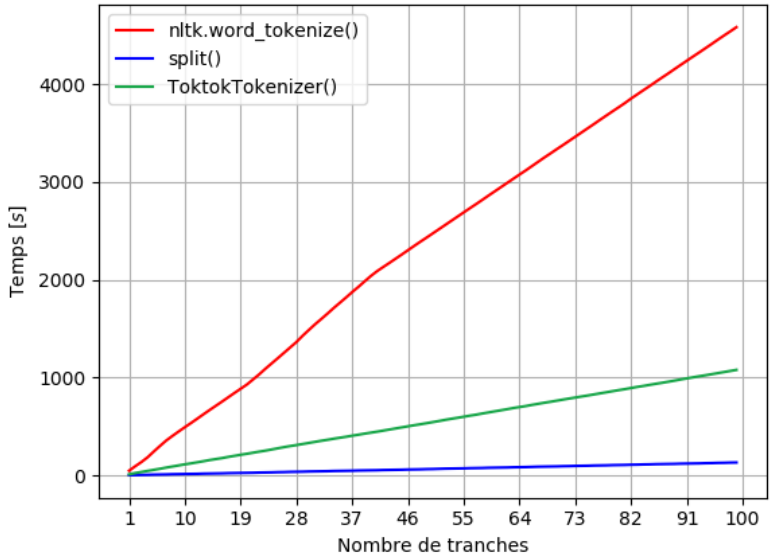

Figure 1: Le temps mis pour calculer le nombre total de mots selon le nombre de tranches considérés selon trois différentes méthodes. Nous obtenons trois droites linéaires de différentes pentes.

On s'intéresse maintenant aux différents types de mots dans le texte. Nous avons seulement appliqué un prétraitement lors de l'étude des types de mots, parce que c'est dans ce cas que les conséquences sont plus intéressantes. Le prétraitement effectué consiste tout d'abord à mettre tous les mots en minuscule. Par la suite, nous effectuons une lemmatisation qui ramène les mots vers une racine. De cette façon on peut ramener plusieurs mots sous un mot commun. Par exemple, *studying* et *studies* se ramène sous *study*. Comme montrer plus haut, l'outil de tokenisation créer des tokens de ponctuations qui ne sont pas des mots. Ces tokens de ponctuations ont été supprimer. Ils consistaient environs de 95 000 tokens par tranche. Nous avons aussi regroupé tous les tokens consistant uniquement de nombre sous le nom "NOMBRE" qui consiste environs de 94 000 tokens par tranche. Une fois que les mots ont été traité, il suffit d'extraire les mots uniques au fur et à mesure à l'aide de la fonction `unique()` de `pandas`. Tous les traitements sont fait à l'aide de la librairie `pandas` qui est facile d'utilisation. Le code se regroupe dans un total de 220 lignes pour faire les calculs et générés les différents graphiques. Les courbes sont générés à l'aide de la librairie python `matplotlib.pyplot`.

Comme on s'y attend le prétraitement fait diminuer le type de mots, car il permet de mieux regrouper les mots. On remarque ce phénomène sur la figure 2. La distance entre les deux courbes augmente avec le nombre de tranches. En effet, plus on considère de mots, plus le groupe non prétraité augmente rapidement comparé au groupe prétraité. On remarque que les courbes semblent tendre vers une asymptote. C'est avec un plus grand nombre de textes qu'on pourrait observer ce phénomène. C'est logique puisque les mots sont un ensemble fini.

Il est possible de visualiser sous une autre perspective l'évolution du nombre de mots uniques et le nombre de mots total selon le nombre de tranches à la figure 3. Le nombre de mots total augmente considérablement plus rapidement que le nombre de mots uniques. Notons aussi que chaque tranche contient environs 7 millions de mots, son augmentation est donc linéaire. Selon nos calculs, le corpus contient 778 318 180 mots au total et 1 932 611 mots uniques.

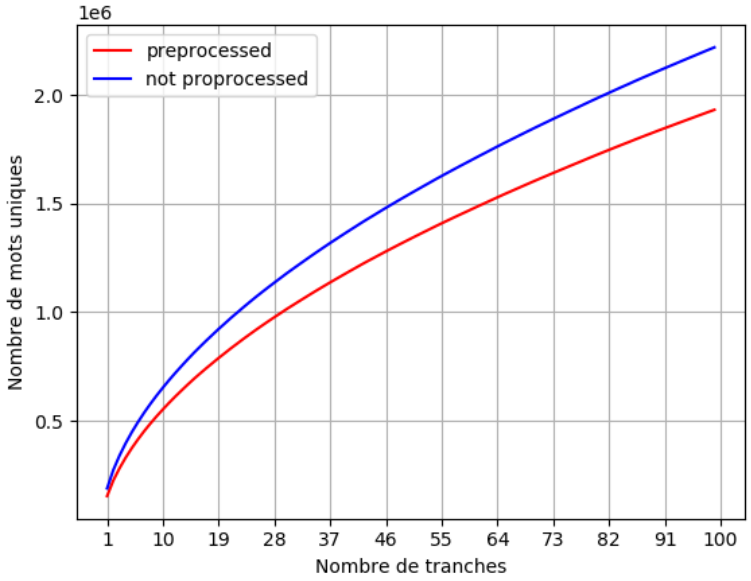

Figure 2: Nombre de mots uniques en fonction du nombre de tranches. On compare ici des mots qui ont été prétaités selon le processus définit plus haut avec des mots non prétaités. Dans les deux cas, les mots ont été tokenisé avec `ToktokTokenizer()`.

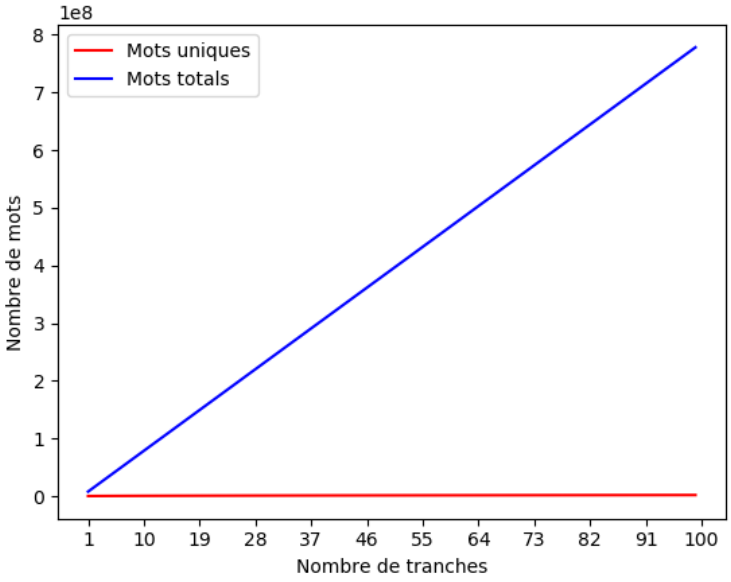

Figure 3: Comparaison de l'évolution du nombre de mots total et du nombre de mots uniques en fonction du nombre de tranches.

Bref, une des faiblesses des tokenizers est qu'aucun d'eux gère les *url*. À l'aide d'une rapide analyse on remarque qu'il y a beaucoup de liens *url* dans les textes. Regrouper tous ces *url* sous un même nom, ou extraire des informations des liens pourrait avoir un effet intéressant. De plus, il aurait été pertinent d'étudier plus en profondeur différents type de tokenizer avec la librairie spaCy par exemple. Il aurait aussi été intéressant de mesurer la différence entre la lemmatisation et le stemming.