Université de Montréal

IFT6285 – Traitement automatique des langues naturelles

Devoir 4

---

par:

Eugénie Yockell
(20071932)
Bassirou Ndao
(0803389)

Date de remise: 9 octobre 2020

Nous avons implémenté un programme Python qui permet d'étudier les n-grammes du corpus `1Bwc` à l'aide de l'API `Phraser` de la librairie `gensim`.

Nous avons mesuré le nombre de n-grammes en fonction du nombre de phrases digérées à la figure 1. On remarque que le nombre de n-grammes augmente avec le nombre de phrases digérées et que le nombre de n-grammes augmente plus lentement selon leurs ordres. La différence de quantité de n-grammes diminue exponentiellement selon leur ordre. En effet, à 6 millions de phrase, nous avons 1.6 fois moins de trigrammes que de bigrammes et 4.4 fois moins de 4-grammes que de trigrammes. On peut interpoler les courbes des graphiques et supposer que ces différences deviennent de plus en plus importantes selon le nombre de phrases traitées. Plus on traite de phrases, plus la différence entre le nombre de n-grammes selon leurs ordres est grande. On s'attend à ce résultat, car trouver des n-grammes de grand ordre est plus rare et difficile.

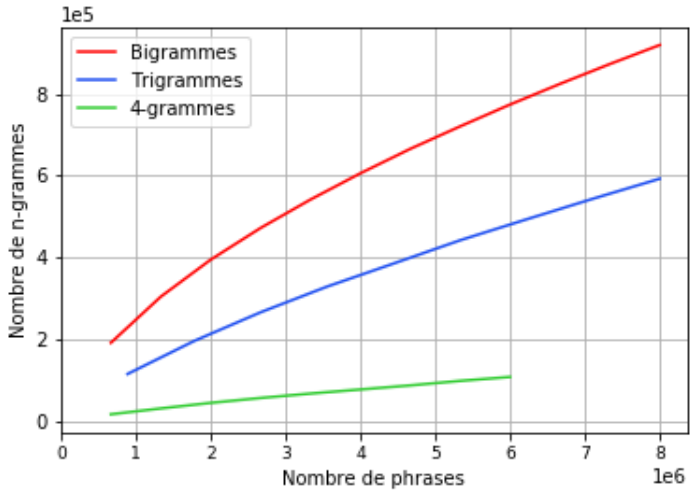

Figure 1: Le nombre de bigrammes, trigrammes et 4-grammes selon le nombre de phrases étudiées par `Phraser`.

Nous avons définit les 10 meilleurs bigrammes sur le corpus coupé `1bshort.tar.gz` de 2.7 millions de phrases. Par défaut, nous avons un compte minimal de n-grammes de 5 et un seuil de score 10. Nous avons aussi tenté de nettoyer le texte en supprimant les ponctuations exceptées les tiret '-' et les apostrophes qui sont des parties importantes des mots. Nous avons aussi tout mis en minuscule. On peut étudier la différence des 10 meilleurs bigrammes d'un texte brut et nettoyé à la table 1. On trouve beaucoup de noms propres. Avec le texte nettoyé nous avons le bigramme "week-in week-out" qui est la collocation la plus intéressante et représente une expression anglaise. Par contre, le bigramme "neo-geek self-loving" n'est pas une expression et semble seulement correspondre à une situation bien particulière qui est étudié plus loin. On remarque aussi que tous les scores sont plus bas pour un texte nettoyé. Une hypothèse qui peut expliquer ce résultat est que le score est calculé de la façon suivante,

$$score(w_i, w_j) = \frac{size\_vocab(count(w_i w_j) - min\_count)}{count(w_i)count(w_j)}$$

où *size_vocab* est la taille du vocabulaire et *min_count* est la paramètre de compte minimum d'un n-grammes. En plaçant les mots en minuscule, au dénominateur, on retrouve un plus grand compte d'un certain mots. De plus, la longueur totale du vocabulaire est diminuer parce qu'on retire des ponctuations. Ainsi, le score est plus petit.

Bigramme	Score	Bigramme nettoyé	Score nettoyé
Jomana Karadsheh	491 245.45	jomana karadsheh	490 951.20
Ekho Moskvy	487 185.57	neo-geek self-loving	486 893.75
neo-geek self-loving	487 185.57	ekho moskvy	486 893.75
NUSA DUA	485 180.69	lese majeste	477 313.66
rio tinto	485 180.69	fisnik abrashi	460 266.75
BATON ROUGE	477 599.74	policia judiciaria	460 266.75
BOCA RATON	465 084.45	week-in week-out	460 266.75
Fisnik Abrashi	460 542.61	somdev devvarman	446 319.27
VINCE McMAHON	460 542.61	tathiana garbin	446 319.27
Policia Judiciaria	460 542.61	evonne goolagong	436 401.06

Table 1: Les 10 meilleurs bigrammes sur `1bshort.tar.gz` de 2.7M de phrases brute et nettoyées en utilisant `Phraser` avec les options par défaut. Les bigrammes communs sont colorés semblablement.

Il serait intéressant d'observer les contextes des deux meilleurs bigrammes et trigrammes sur le corpus coupé de 2.7M de phrases nettoyées. Le premier bigramme "jomana karadsheh" est le nom d'une journaliste et se retrouve 10 fois dans le texte.

> cnn 's arwa damon **jomana karadsheh** and youssif basil contributed to this report
> cnn 's **jomana karadsheh** and mohammed jamjoom in baghdad hada ...
> ... raja razek daniela berretta and **jomana karadsheh** contributed to this report
> cnn 's **jomana karadsheh** and mohammed jamjoom contributed to this report

Le deuxième bigramme le plus populaire est "neo-geek self-loving", il se retrouve 11 fois dans le texte. Les contextes dans lequel il apparaît sont exactement les mêmes. Les différences sont dans le voisinage plus éloignées du bigramme.

> ... galley slaves sporadically updated **neo-geek self-loving** kausfiles sporadically updated ...

Dans les trigrammes on obtient encore principalement des noms propres. Il est intéressant d'observer que dans les 10 meilleurs trigrammes on obtient "governor zhou xiaochuan" et "spokesman chun hae-sung" qui permet de définir le status des personnes. Le trigramme le plus populaire est "updated neo-geek self-loving" qui apparaît encore 11 fois dans les mêmes contextes que le bigramme. Ce résultat n'est pas surprenant, car le bigramme "neo-geek self-loving" apparaît dans des contextes presque identique. Le deuxième trigramme le plus populaire est "largest atom smasher" qui apparaît 10 fois. On pourrait supposer qu'un 4-grammes populaire contiendrait "world 's largest atom smasher". Ci-dessous se trouve le contexte du trigramme.

> ... success of the world 's **largest atom smasher** on its opening day was more surprising ...
> ... works at the world 's **largest atom smasher** and is suspected of al-qaida ...
> geneva - the world 's **largest atom smasher** conducted its first experiments ...
> geneva ap - the world 's **largest atom smasher** has recorded its first high-energy ...

Nous avons aussi étudier les 4-grammes. Dans les 10 meilleurs 4-grammes, nous avons l'expression anglophone "let bygones be bygones", "millions fatally wounded tories", "beltway cranky conservatives unite", "cbcategory information technology cbindustry" et "indefinite delivery indefinite quantity". Ces 4-grammes contiennent d'intéressantes collocations.

Nous avons mesuré le temps nécessaire pour traiter différents nombre de phrases à la figure 2. À noter qu'avec les trigrammes, on obtenait une erreur de mémoire lorsqu'on prenait plus de 8 millions de phrases. Pour généré les 4-grammes nous avons alors seulement étudier 6 millions de phrases. On remarque que le temps évolue plus rapidement pour les modèles de plus grands ordres. On remarque aussi que le temps de calcul augmente de plus en plus rapidement de manière exponentielle selon l'ordre des n-grammes. On s'attendait à ce résultat, car les modèles `Phrases` contient aussi les n-grammes d'ordre inférieur.

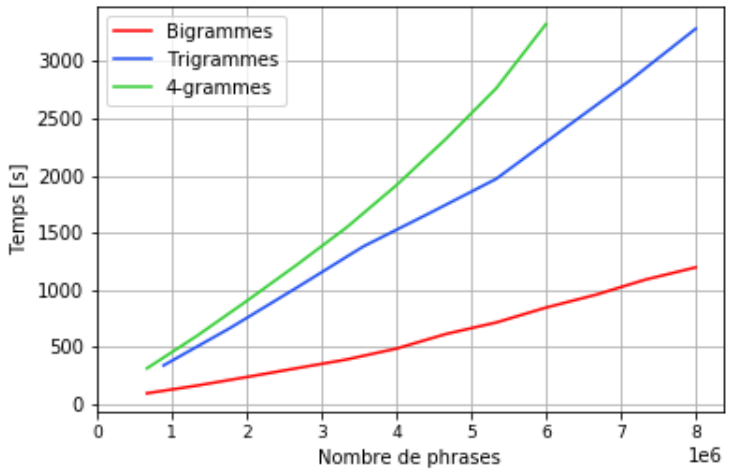

Figure 2: Évolution du temps nécessaire pour générer les n-grammes en fonction du nombre de phrases.

Bref, un des problèmes importants auquel on fait face est la limitation dans les phrases qu'il est possible d'analyser. En effet, les n-grammes qu'on obtient sont rarement des collocations relevant d'une expression, mais plutôt de noms. Avec un plus grand nombre de phrases, on aurait la chance de détecter des expressions, car elles sont plus rares. De plus, permettre de traiter un plus grand nombre de phrases aiderait à trouver des n-grammes plus intéressant que des noms.

L'API `Phraser` utilise une méthode de scoring basé sur l'information mutuelle. Cette méthode très populaire a tout de même un défaut lorsqu'il s'agit de traiter les évènements rares. La méthode a tendance a surestimer le score des évènements rares. Une amélioration possible serait de définir nous-mêmes une fonction de score. Une fonction de score basée sur la vraisemblance ou le score d'entropie peuvent être pertinentes dans le contexte de notre devoir. La vraisemblance permettrait d'être moins fragile lorsqu'il s'agit de traiter les évènements rares. Le score d'entropie quand à lui permettrait de faire une présélection des séquences candidats.

D'autre part, nous avons fait face à la difficulté de donner les 10 bigrammes/trigrammes les plus saillants. En effet, plusieurs n-grammes avait des scores très similaires, ce qui nous empêchait de les discriminer correctement. De plus, pour trouver des n-grammes d'ordre supérieur il faut faire une récursion d'estimation. Cela ajoute de la complexité d'implémentations, du temps et des ressources de calcul.

En résumé, dans cet exercice, nous nous sommes familiarisés avec l'API `Phraser` à travers l'analyse de multimots. Nous avons extrait des expressions contigües ainsi que certaines collocations. Mais principalement nous avons extrait beaucoup de noms propres. Ainsi cette exercice serait un moyen de pouvoir identifier et d'isoler les noms dans un corpus qui, dans le cas des trigrammes et 4-grammes, est souvent accompagné d'un titre ou d'une fonction/occupation.