IFT6285 (Devoir) - Université de Montréal

COLELLA Charles

September 24, 2021

1 Introduction

Dans ce document nous allons entrainer un model de langue, de type Kneser&Ney à l'aide du 1B-word corpus[1] mis à disposition. Ce corpus d'entrainement est composé de 99 tranches différentes. Pour entrainer ce model, nous utiliserons la librairie **KenLM** [3] qui permet de réduire le temps et le les coûts de mémoire.

1.1 Lissage Kneser-Ney

Le lissage de Kneser-Ney est une méthode utilisée pour calculer la distribution de probabilité de n-grammes dans un document en fonction de leur historique.

Etant donné $c : \omega_1 \in \Omega \times \omega_2 \in \Omega \longrightarrow N$ une fonction définie sur l'ensemble des mots, qui exprime nombre d'occurrences du mot ω_1 suivi du mot ω_2 dans le corpus et $\lambda_{\omega_{i-1}}$ un facteur de normalisation de la probabilité p_{KN} évaluée avec $\omega_i \in \Omega$.
La probabilité d'un bigramme, s'écrit comme suit :

$$p_{KN}(\omega_i|\omega_{i-1}) = \frac{\max(c(\omega_{i-1},\omega_i) - \delta, 0)}{\sum_{\omega'} c(\omega_{i-1},\omega')} + \lambda_{\omega_{i-1}} p_{KN}(\omega_i) \tag{1}$$

On remarque le parametre δ, qui représente une valeur d'actualisation des fréquences des bigrammes observés en entrainement afin de correspondre aux fréquences observés lors des tests (1991 - Church & Gale).
On calcul donc d'abord la fréquence du bigramme dans le corpus (avec une constante d'actualisation) puis on ajoute la probabilité p_{KN} d'observer ω_i dans un contexte inconnu, qui est estimé comme la fréquence d'apparition après tout autre mot :

$$p_{KN}(w_i) = \frac{|\{\omega' : 0 < c(\omega', \omega_i)\}|}{|\{(\omega', \omega'') : 0 < c(\omega', \omega'')\}|} \tag{2}$$

L'avantage de cette méthode se trouve donc en la considération des mots possibles qui précédent un unigramme lors de la définition de sa probabilité.

1.2 Mesure de la perplexité

Soit $(\omega_n)_{n\in N}$ une combinaison de mots appartenant à l'ensemble P des phrases. On définit la perplexité PP par :

$$\forall k \in N, \quad PP(\omega_1\omega_2...\omega_k) = p(\omega_1\omega_2...\omega_k)^{-\frac{1}{k}} \tag{3}$$

Durant la suite de ce document, nous allons utiliser la fonction **perplexity**[2] de **KenLM**[3]. Etant donné un score $S = log_{10}(p)$ avec p la probabilité associée à une suite de mots de longueur $k \in N$, l'utilisation de cette fonction revient à calculer :

$$PP(\omega_1\omega_2...\omega_k) = (10^{S(\omega_1\omega_2...\omega_k)})^{-\frac{1}{k}} = 10^{-\frac{S(\omega_1\omega_2...\omega_k)}{k}} \tag{4}$$

[1] https://github.com/ciprian-chelba/1-billion-word-language-modeling-benchmark
[2] https://github.com/kpu/kenlm/blob/master/python/kenlm.pyx#L182
[3] https://kheafield.com/code/kenlm/

2 Question (a)

2.1 Mise en place

KenLM est entrainé avec un model bigramme sur les 9 premieres tranches du repertoire d'entrainement, prétraités au préalable, et dont le model est enrengistré dans un fichier *data.arpa* puis dans *data.binary* (le format binaire se charge beaucoup plus rapidement et offre plus de flexibilité). Ensuite le model est appliqué sur les 1000 premieres phrases, prétraités au préalable, de la tranche de test et les resultats enregistrés dans le fichier *data.txt*.
Le code est affiché ci-dessous :

execution.sh

```
#!/bin/bash
python search.py $training_dir -n 9 -p|$kenLM_dir/lmplz -o 2 > ./data.arpa
$kenLM_dir/build_binary data.arpa data.binary
python computing.py $test_file ./data.binary -p -n 1000 -r "data.txt"
```

Il est a noter que **search.py** et **computing.py** sont deux programmes écrits en python pour l'entrainement et le test de notre model de langue.

2.2 Résultats

Les résultats obtenus sont représentés par la moyenne, le min et le max de perplexité obtenus en utilisant la fonction **kenlm.perplexity** [2] présentée en introduction.

	Sans prétraitrement	Avec prétraitrement
Moyenne	412.933	339.687
Maximum	21410.839	15752.395
Minimum	28.735	18.588

Le prétraitement consiste à mettre, autant pour l'entrainement que pour le test, le texte en minuscule, remplacer les URL par __URL__, les nombres par __NUMBER__, les emails par __E_MAIl__ et les numéros de téléphone par __PHONE__. Cette tâche à pu être effectuée avec la librairie **cleantext**[4].

3 Question (b)

3.1 Mise en place

Pour analyser les performances du model de langue, un entrainement sur un nombre de tranches $i \in [\![1, 99]\!]$ du 1B-word corpus a été établi. Chaque model entrainé sur les i premieres tranche est ensuite confronté au texte final pour être comparé aux autres.

script_compare.sh

```
#!/bin/bash
for i in $(eval echo {1..$number_training_files}
do
    train $i # on entraine notre model sur les i premieres tranches
    compute  # on l'applique a la tranche de test
done
```

[4] https://pypi.org/project/clean-text/

3.2 Résultats

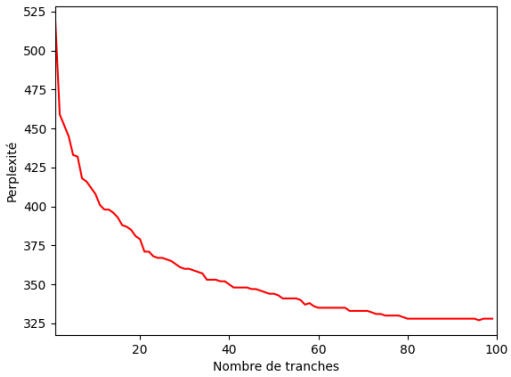

Figure 1: Moyenne perplexité

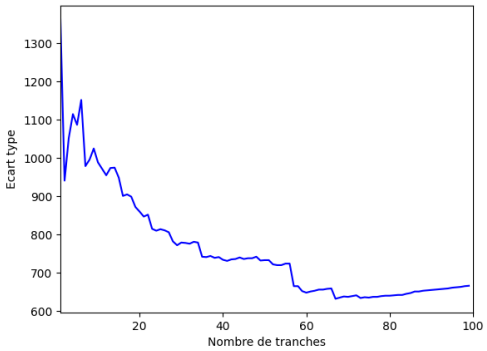

Figure 2: Ecart-type perplexité

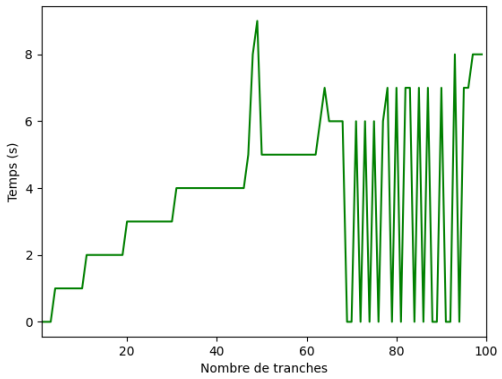

Figure 3: Temps de traitement de la tranche de test

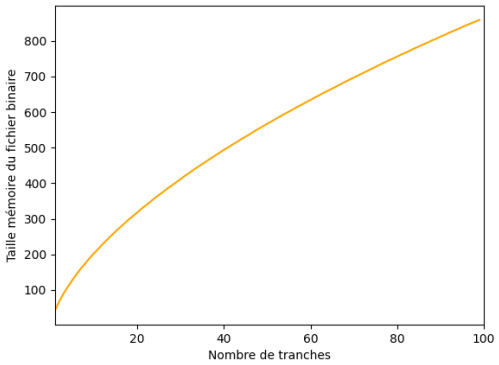

Figure 4: Taille du fichier binaire contenant le modele

La perplexité d'une phrase pris indépendamment étant très sensible à la distribution de probabilités, la figure 2 met en avant quelques disfluences de l'écart type par rapport aux nombres de tranches d'entrainement (n'excédant pas 15%).

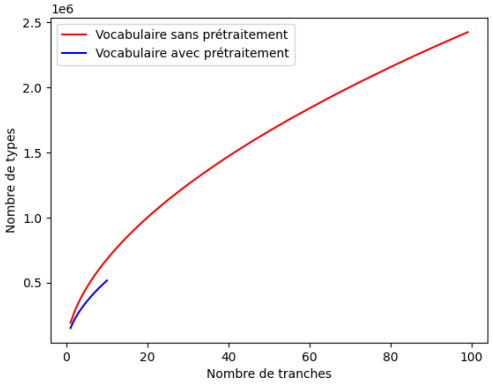

Figure 5: Nombre de types ($\times 10^6$) du corpus d'entrainement

On constate sur la figure 1 une diminution de plus en plus lente de la perplexité en fonction du nombre de tranches étudié lors de l'entrainement. Les figures 1 et 4 semblent fortement corrélées à la figure 5 representant le nombre de types étudié.

4 Question (c)

4.1 Mise en place

Afin de mettre en place le model le plus adapté à notre tranche de test, trois entrainements différents ont été testés sur les 10 premieres tranches du repertoire d'entrainement :

- Entrainement sur un texte **sans prétraitement** en utilisant des **bigrammes**
- Entrainement sur un texte **avec prétraitement** en utilisant des **bigrammes**
- Entrainement sur un texte **sans prétraitement** en utilisant des **trigrammes**

L'entrainement sur un texte prétraité avec des trigrammes n'a pas pu être réalisé du fait du manque de performances de la machine utilisé.

4.2 Résultats

Les même données que précédemment ont été récoltés. Cependant, par soucis de performances[5], seules les dix premieres tranches ont pu être comparées.

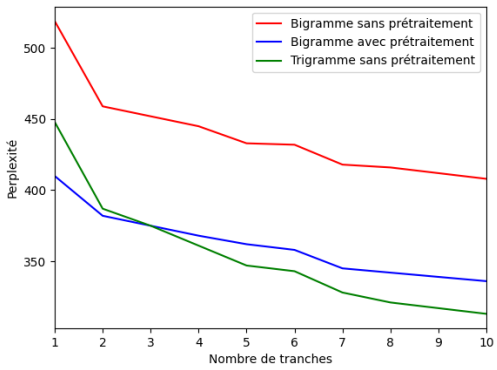

Figure 6: Moyenne perplexité

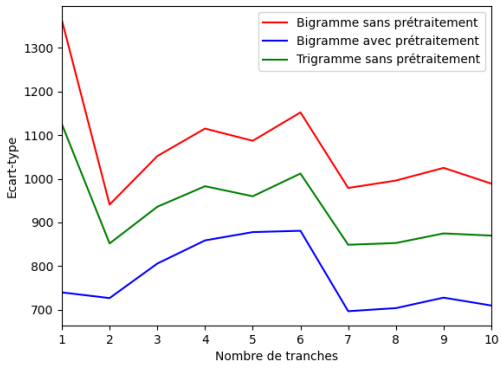

Figure 7: Ecart-type perplexité

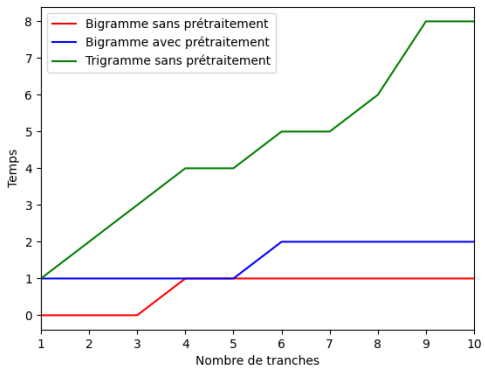

Figure 8: Temps de traitement de la tranche de test

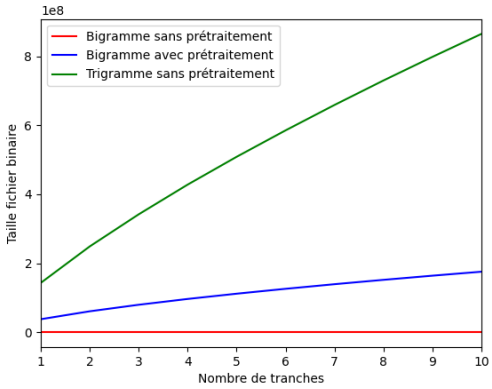

Figure 9: Taille du fichier binaire contenant le modele

Sans grosse suprise, le minimum de perplexité est atteint pour le modele utilisant des trigrammes sur la figure 6. On peut supposer que la perplexité diminurait encore en utilisant un modele d'ordre supérieur. La figure 7 met en évidence l'impact du prétraitement, qui engendre une normalisation des mots, sur l'écart-type de la perplexité. Il toutefois important de noter, relativement aux figures 8 et 9 qu'en augmentant l'ordre du modele, la taille du fichier en mémoire ainsi que le temps de calcul augmente drastiquement ce qui rend le traitement beaucoup plus compliqué a effectuer.

[5]Les programmes ont été éxecutés sur une machine du DIRO de l'Université de Montréal