TP1 - Modèles de langages markoviens

IAAA - TLNL

26 juillet 2023

1 Objectif

L'objectif de ce projet est de programmer un modèle de langage markovien, d'en calculer les performances et de s'en servir pour générer du texte. Ces trois étapes seront réalisées sous la forme de trois programmes différents, appelés dans ce document `ngram`, `perplexite` et `genere`.

2 Préparation des données

Pour estimer les probabilités d'un modèle de langage markovien il suffit de disposer de texte brut, comme vous pourrez en trouver ici. Cependant, ce texte doit être décomposé en mots, opération que l'on appelle segmentation ou encore *tokenization*. En français, les mots sont généralement séparés les uns des autres par des espaces et l'opération de segmentation est en général triviale. Mais les conventions typographiques compliquent un peu cette opération, en particulier pour les caractères non alphabétiques. Voici quelques cas :

— Certains caractères de ponctuation sont collés au mots précédents et doivent en être séparés :
`L'as-tu sorti du frigo?` → `L'as-tu sorti du frigo ?`

— De même l'apostrophe sépare généralement deux mots, mais elle est collée au mot précédent :
`L'as-tu sorti du frigo ?` → `L' as-tu sorti du frigo?`

— Même chose pour le tiret :
`L' as-tu sorti du frigo ?` → `L' as tu sorti du frigo ?`

La segmentation s'occupe aussi de transformer certains mots. Voici quelque cas :

— La première lettre du premier mot d'une phrase est en majuscule et doit être transformée en minuscule :
`L' as tu sorti du frigo ?` → `l' as tu sorti du frigo ?`

— Cerains mots ont fusionné et doivent être séparés :
`l' as tu sorti du frigo ?` → `l' as tu sorti de le frigo ?`

Comme souvent avec la langue naturelle, toutes ces règles admettent des exceptions. La segmentation n'est pas l'objet de ce TP, vous pourrez utiliser le segmenteur élémentaire qui se trouve là.

3 Calcul de probabilités

Les modèles de langage markoviens manipulent des probabilités conditionnelles, appelées *probabilités n-grammes* de la forme $P(m_i|m_{i-n+1},\ldots,m_{i-1})$ où n est *l'ordre* du modèle [1]. Les probabilités qui nous intéressent sont donc les probabilités du mot m_i sachant les $n-1$ *mots qui le précèdent.*

- Lorsque n vaut 1, on parle d'un modèle d'ordre un ou d'un modèle unigramme. Dans ce cas particulier, les probabilités n-grammes ne sont pas des probabilités conditionnelles, elles sont de la forme $P(m_i)$.
- Lorsque n vaut 2, on parle d'un modèle d'ordre deux ou d'un modèle bigramme. Dans ce cas les probabilités conditionnelles sont de la forme $P(m_i|m_{i-1})$
- Lorsque n vaut 3, on parle d'un modèle d'ordre trois ou d'un modèle trigramme. Dans ce cas les probabilités conditionnelles sont de la forme $P(m_i|m_{i-2},m_{i-1})$

Les probabilités n-grammes permettent de calculer la probabilité d'une séquence $m_1 \ldots m_N$ grâce à la règle de multiplication et à l'hypothèse d'indépendance de Markov :

$$P(m_1 \ldots m_N) = \prod_{i=n}^{N} P(m_i|m_{i-n+1},\ldots,m_{i-1}) \tag{1}$$

L'application de la formule 1 pose un problème pour les valeurs de $n > 1$. En effet, pour $n = 2$, le premier terme du produit de l'équation 1 est $P(m_2|m_1)$, mais la probabilité que le mot m_1 soit le premier mot de la phrase n'est pas prise en compte par le modèle. Lorsque $n = 3$, ce sont les probabilités que m_1 soit le premier mot de la phrase et m_2 le second qui ne sont pas prises en compte. C'est la raison pour laquelle on ajoute en début de phrase $n-1$ symboles particuliers (par exemple le symbole `<s>`) et en fin de phrase $n-1$ symboles particuliers (par exemple `</s>`). Avec cette astuce, pour $n = 2$, le premier terme du produit devient $P(m_1|\texttt{<s>})$ qui est la probabilité que le mot m_1 soit le premier mot d'une phrase.

Comme on peut l'observer dans la formule 1, le calcul de $P(m_1 \ldots m_N)$ effectue N produits de probabilités, ce qui peut aboutir à des valeurs trop petites, impossibles à représenter en mémoire (underflow). C'est la raison pour laquelle on manipule généralement des logarithmes de probabilités (appelés logprobs et notés LP). L'autre effet bénéfique du passage aux probabilités est que les produits sont transformés en sommes et les exposants en produits, ce qui simplifie la dérivation des formules. La formule 1 devient donc :

$$LP(m_1 \ldots m_N) = \sum_{i=n}^{N} LP(m_i|m_{i-n+1},\ldots,m_{i-1}) \tag{2}$$

1. Attention, le terme probabilité n-grammes est un peu trompeur, il ne s'agit pas de la probabilité d'une suite de n mots ($P(m_{i-n+1},\ldots,m_i)$) mais bien de la probabilité conditionnelle $P(m_i|m_{i-n+1},\ldots,m_{i-1})$.

L'estimation des probabilités n-grammes par maximum de vraisemblance (noté ML) pose le problème des probabilités nulles associées aux n-grammes non observés en corpus. Une méthode simple pour remédier à ce problème consiste à réaliser une estimation avec lissage de Laplace :

Ordre	Probabilité ML	Probabilité Laplace
$P(m_i)$	$\frac{C(m_i)}{N}$	$\frac{C(m_i)+1}{N+\|V\|}$
$P(m_i\|m_{i-1})$	$\frac{C(m_{i-1}m_i)}{C(m_{i-1})}$	$\frac{C(m_{i-1}m_i)+1}{C(m_{i-1})+\|V\|}$
$P(m_i\|m_{i-2}m_{i-1})$	$\frac{C(m_{i-2},m_{i-1},m_i)}{C(m_{i-2},m_{i-1})}$	$\frac{C(m_{i-2},m_{i-1},m_i)+1}{C(m_{i-2},m_{i-1})+\|V\|}$

Dans ces formules, V est le vocabulaire de unigrammes, bigrammes et trigrammes présents dans le corpus sur lequel les probabilités ont été estimées. Lors du calcul la perplexité (Sec. 4) et de la génération des phrases (Sec. 5), ces formules garantissent qu'aucune probabilité ne sera nulle, même si les comptes $C(\cdot)$ du numérateur et/ou du dénominateur le sont.

Les probabilités n-grammes sont calculées à l'aide du programme `ngrams` que vous devez réaliser. Ce dernier prend en entrée un fichier texte segmenté et produit un fichier dans lequel sont stockées les probabilités unigrammes, bigrammes et trigrammes. Le format de ce fichier est le suivant :

```
#unigrammes 13023
<s> 2.572588161852979
le 4.056275795765149
24 11.73321294158323
...
crayon 12.426360122143175
#bigrammes 76698
<s> <s> 0.6931471805599453
<s> le 4.15667847378479
...
au crayon 7.00669522683704
#trigrammes 148549
<s> <s> le 3.4635312932248454
<s> le 24 5.697093486505405
...
écrit au crayon 0.0
```

Le fichier est composé de trois parties matérialisées par les mots clefs `#unigrammes`, `#bigrammes` et `#trigrammes`. Ces mots clefs sont suivis par le nombre d'unigrammes, bigrammes et trigrammes *différents* extraits du texte ($|V|$). Chaque partie comporte autant de lignes que le nombre de n-grammes différents indiqués. Chaque ligne est composée d'un n-gramme suivi de l'opposé de sa logprob.

4 Perplexité

Une manière de d'évaluer la qualité d'un modèle de langage M étant donné un texte T consiste à calculer la probabilité qu'attribue M à T, que l'on notera

$P_M(T)$. Plus cette probabilité est élevée, plus on peut dire que M modélise bien T.[2] Pour comparer deux modèles M_1 et M_2, il suffit alors de comparer les deux probabilités $P_{M_1}(T)$ et $P_{M_2}(T)$. En pratique, on utilise plutôt une métrique appelée perplexité, qui est une version normalisée de la probabilité et permet de comparer des textes de longueurs différentes. La perplexité se calcule de la manière suivante :

$$PP(T) = P(T)^{-\frac{1}{N}} = P(m_1, \ldots, m_N)^{-\frac{1}{N}} = (\prod_{i=n}^{N} P(m_i|m_{i-n+1}, \ldots, m_{i-1}))^{-\frac{1}{N}}$$

Etant donné que l'on manipule des logprob, on utilisera le logarithme de la perplexité :

$$LPP(T) = -\frac{1}{N}LP(T) = -\frac{1}{N}LP(m_1, \ldots, m_N) = -\frac{1}{N}\sum_{i=n}^{N} LP(m_i|m_{i-n+1}, \ldots, m_{i-1})$$

5 Génération de phrases

Il est possible de générer des phrases à l'aide d'un modèle n-grammes. On part pour cela d'une amorce (aussi appelé *prompt*), qui est `<s><s>` dans le cas d'un modèle trigramme, `<s>` dans le cas d'un modèle bigramme, et rien du tout dans le cas d'un modèle unigramme. Cette amorce permet de calculer la distribution de probabilité du premier mot de la phrase ($P(m|$`<s><s>`$)$ pour un modèle trigramme, $P(m|$`<s>`$)$ pour un modèle bigramme et $P(m)$ pour un modèle unigramme).

Il est alors possible d'effectuer un tirage aléatoire dans cette distribution de probabilité afin de choisir le premier mot m_1 de la phrase. m_1 est alors intégré à l'amorce, qui devient `<s>`m_1 dans le cas d'un trigramme, m_1 dans le cas d'un bigramme, et rien du tout dans le cas d'un unigramme. La nouvelle amorce permet de calculer la distribution de probabilité du deuxième mot de la phrase et de choisir ce dernier par tirage aléatoire. Le processus continue ainsi jusqu'à ce que le symbole de fin de phrase soit généré ou que la phrase ait atteint une valeur limite que l'on aura fixée.

6 Ce qu'il faut faire

1. Réaliser le programme `ngram.py` qui prend en entrée un texte segmenté et produit en sortie un fichier de n-grams dans le format décrit dans la section 3.

2. T est un texte quelconque, à ne pas confondre avec le texte qui a été utilisé pour estimer les probabilités de M.

2. Réaliser le programme `perplexite.py` qui prend en entrée un texte segmenté T et un modèle n-gramme et produit la log perplexité du texte T calculée avec un modèle uni-gramme, bi-gramme et tri-gramme.
3. Réaliser le programme `genere.py` qui prend en entrée un modèle n-gramme et un entier valant `1`, `2` ou `3` et génère du texte à l'aide d'un modèle uni-gramme, bi-gramme ou tri-gramme.

7 Quelques pistes à creuser

— Étudier l'ordre du modèle - est-ce qu'on améliore la perplexité ? La fluidité du texte généré ? Quelles astuces pour ne pas faire exploser la taille du modèle ?
— Étudier d'autres techniques de lissage, interpolation, etc. - est-ce que c'est mieux sur des phrases avec des OOV ?
— Peut-on faire un tel modèle pour les caractères ? Générer des mots plutôt que des phrases ?
— ...