# FIL-TP2: Calcul et utilisation d'un Modèle de Langage

Frederic Bechet, Carlos Ramisch

18 octobre 2016

## 1 Présentation

Etant donné un lexique de mots $\ell$, un *modèle de langage* permet d'attribuer une probabilité $P(W)$ à n'importe quelle séquence de $k$ mots $W = w_1 w_2 \dots w_k$. Ces modèles sont utilisés pour *générer* du texte et classer entre elles les séquences produites grâce aux probabilités $P(W)$. Ce type de modèle est très employé dans des applications telles que la Reconnaissance Automatique de la Parole ou la Traduction Automatique.

Les modèles de langage les plus fréquents sont les modèles ***n-gram***. Ces modèles représentent une séquence de mots comme un processus Markovien : les *Chaînes de Markov*. Cette représentation consiste à attribuer à chaque mot d'une séquence une probabilité dépendante de l'historique des mots précédents. Avec cette représentation, la probabilité de la séquence $W = w_1 w_2 \dots w_k$ devient :

$$P(W) = P(w_1 w_2 \dots w_k) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1 w_2) \tag{1}$$

$$\times P(w_4|w_1 w_2 w_3) \times \dots \times P(w_k|w_1 \dots w_{k-1}) \tag{2}$$

Dans les chaînes de Markov on se donne comme hypothèse qu'il est possible de restreindre l'historique à seulement quelques états précédents. On définit ainsi des *classes d'historiques*. La probabilité $P(w_i|h_i)$ avec $h_i = w_1 \dots w_{i-1}$ devient $P(w_i|\Phi(h_i))$.

Dans un modèle ***n-gram*** on utilise les $n-1$ mots précédents pour représenter l'historique. Ainsi, la fonction $\Phi$ devient :

$$\Phi(h_i) = \{w_{i-(n-1)} \dots w_{i-1}\} \tag{3}$$

Les trois types de modèles n-gram les plus employés sont les modèles unigrammes (*1-gram*), les modèles bigrammes (*2-gram*) et les modèles trigrammes (*3-gram*). Ils sont définis de la manière suivante :

- 1-gram : $P(w_i)$
- 2-gram : $P(w_i|w_{i-1})$
- 3-gram : $P(w_i|w_{i-2}w_{i-1})$

Par exemple, avec un modèle de langage bigramme, la probabilité de la séquence *W=il fait beau aujourd'hui* est :

$$P(\texttt{il fait beau aujourd'hui}) = P(\texttt{il}) \times P(\texttt{fait}|\texttt{il}) \tag{4}$$

$$\times P(\texttt{beau}|\texttt{fait}) \times P(\texttt{aujourd'hui}|\texttt{beau}) \tag{5}$$

Pour une séquence $W$ de longueur $N$, nous aurons :

$$P(W) = P(w_1) \times \prod_{i=2}^{N} P(w_i|w_{i-1}) \tag{6}$$

En conclusion un modèle ***n-gram*** est une chaîne de Markov d'ordre $n-1$.

## 2 Apprentissage d'un modèle de langage

Le moyen le plus simple d'entraîner un modèle de langage n-gram est d'utiliser l'estimateur du *maximum de vraisemblance* sur un corpus d'apprentissage. Cet estimateur, pour un modèle n-gram, est défini de la manière suivante :

$$P(w_i|w_{i-(n-1)} \dots w_{i-1}) = \frac{C(w_{i-(n-1)} \dots w_i)}{C(w_{i-(n-1)} \dots w_{i-1})} \tag{7}$$

où $C(w_i \dots w_{i+j})$ représente le nombre de fois où la séquence $w_i \dots w_{i+j}$ a été vue sur le corpus d'apprentissage.

Nous aurons les formulations suivantes pour les modèles unigrammes, bigrammes et trigrammes :

- 1-gram : $P(w_i) = \frac{C(w_i)}{N}$
- 2-gram : $P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$
- 3-gram : $P(w_i|w_{i-2}w_{i-1}) = \frac{C(w_{i-2}w_{i-1}w_i)}{C(w_{i-2}w_{i-1})}$

Avec $N$ correspondant au nombre de mots du corpus d'apprentissage.

# 3 Représentation des probabilités dans le domaine $-\log$

L'estimation de la probabilité d'une séquence de grande taille peut poser des problèmes de calcul pour une machine. En effet, chaque probabilité étant inférieure ou égale à 1, étant donné que la probabilité d'une séquence est le produit des probabilités de chaque n-gram, des problèmes de précision sur les très petits nombres peuvent se poser. Pour résoudre ce problème on utilise habituellement le domaine log pour représenter les probabilités (appelée *logprob*). Cela permet d'éviter tout dépassement en accélérant les calculs puisque les multiplications sont remplacées par des additions dans le domaine log : $\log a \times b = \log a + \log b$. Puisque le log d'un nombre entre 0 et 1 est négatif, on utilisera habituellement le domaine $-\log$.

Ainsi, avec un modèle de langage bigramme, la probabilité d'une séquence $W$ de longueur $N$ dans le domaine $-log$ sera :

$$P_{log}(W) = -\log P(w_1) + \sum_{i=2}^{N} -\log P(w_i|w_{i-1}) \tag{8}$$

# 4 Lissage dans un modèle de langage

L'estimateur par maximum de vraisemblance serait parfait si nous avions des corpus d'apprentissage infinis. En réalité ce n'est pas le cas, et nous tombons donc sur la malédiction des *évènements non vus* !! En effet, avec la formule précédente, tout n-gram absent du corpus d'apprentissage reçoit une probabilité nulle. Etant donné que dans une chaîne de Markov les probabilités de transitions sont multipliées les unes aux autres, il suffit qu'un seul évènement soit non vu dans une chaîne d'évènements de longueur quelconque pour que toute la chaîne reçoive une probabilité nulle.

De plus, dans le cas des mots inconnus, le dénominateur du quotient permettant de calculer la probabilité se retrouve à 0 ce qui rend le calcul impossible.

Pour répondre à ce problème, les modèles n-gram sont complétés par des mécanismes de *lissage* permettant d'éviter ces probabilités nulles en redistribuant une partie de la masse des probabilités des n-gram présents dans l'apprentissage vers ces évènements non vus.

Nous allons étudier une méthode de lissage dans ce TP : le lissage de *Laplace* par pseudo-compte..

Cette méthode, très simple, consiste à rajouter une quantité $\alpha$ à chaque compte de n-gram dans l'estimation des probabilités par maximum de vraisemblance.

Nous aurons les formulations suivantes pour les modèles unigrammes, bigrammes et trigrammes :

- 1-gram : $P^L(w_i) = \frac{C(w_i)+\alpha}{N+N\times\alpha}$
- 2-gram : $P^L(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)+\alpha}{C(w_{i-1}+N\times\alpha)}$
- 3-gram : $P^L(w_i|w_{i-2}w_{i-1}) = \frac{C(w_{i-2}w_{i-1}w_i)+\alpha}{C(w_{i-2}w_{i-1})+N\times\alpha}$

# 5 Evaluation d'un modèle de langage

Pour mesurer la qualité d'un modèle de langage indépendemment d'une tâche donnée, on utilise une mesure *objective* de la capacité d'un modèle à générer un corpus particulier. La mesure la plus communément utilisée est appelée mesure de la *perplexité* d'un modèle de langage.

D'un point de vue théorique cette mesure est reliée à la notion d'*entropie* telle qu'introduite par Shannon dans sa *Théorie de l'information*. Elle correspond à l'entropie croisée entre les probabilités observées sur le corpus de test et celles estimées par le modèle sur le corpus d'apprentissage.

D'un point de vue pratique, elle se calcule de la manière suivante :

Soit un corpus de test $T$ de $n$ mots ($T = w_1 w_2 \dots w_n$) et un modèle de langage $M$, la *perplexité* du modèle $M$ sur le corpus $T$ s'écrit $PP(T)$ et se calcule de la manière suivante :

$$PP(T) = 2^{LP(T)} \tag{9}$$

La quantité $LP(T)$ représente le *logprob* moyen de la chaîne de mots constituant le corpus $T$. Elle se calcule de la manière suivante :

$$LP(T) = -\frac{1}{n} \log P(T) = -\frac{1}{n} \sum_{i=1}^{n} \log P(w_i|\phi(h_i)) \tag{10}$$

avec $P(w_i|\phi(h_i))$ la probabilité n-gram du modèle $M$ sur le n-gram composé de l'historique $h_i$ et du mot $w_i$. Pour un modèle bigramme nous aurons : $P(w_i|\phi(h_i)) = P(w_i|w_{i-1})$

Avec la représentation des probabilités en domaine $-\log$ vue auparavant, et en utilisant le lissage de Laplace, nous aurons :

$$PP(T) = 2^{\frac{1}{n} \times P^L_{log}(W)} = 2^{\frac{1}{n} \times (-\log P(w_1) + \sum_{i=2}^{N} - \log P(w_i|w_{i-1}))} \tag{11}$$

# 6 Travail à effectuer

Vous avez deux programmes à écrire dans ce TP : un programme permettant d'estimer les comptes des n-gram nécessaires au calcul d'un modèle de langage et un programme implémentant le modèle à partir des comptes et calculant la perplexité de séquences de mots.

## 6.1 Calcul des comptes de n-gram et sauvegarde du modèle

En utilisant le tokenizeur et la traduction *mot→code* du TP précédent, calculez tous les comptes des unigrammes et des bigrammes d'un corpus donné en entrée de votre programme. La sortie de votre programme sera un fichier texte contenant l'ensemble de ces comptes selon le format suivant :

```
1-gram:
mot1 compte
mot1 compte
....
2-gram:
mot1 mot2 compte
mot1 mot2 compte
....
```

Par exemple, avec le corpus d'entraînement suivant :

```
1054 7815 4238 9297 6283
5831 1054 4554 688 9296 7960 8104 9297 757 8908 1203 6861 2624 4691 7802 4772 18 789 7815 4238 9297 6283
4691 7802 4771 4772 18 789 3311 9297 5964 2279 5818
5909
```

Vous devrez obtenir le fichier de comptes suivant :

```
1-gram:
18 2
688 1
757 1
789 2
1054 2
1203 1
2279 1
2624 1
3311 1
4238 2
4554 1
4691 2
4771 1
4772 2
5818 1
5831 1
```

```
5909 1
5964 1
6283 2
6861 1
7802 2
7815 2
7960 1
8104 1
8908 1
9296 1
9297 4
2-gram:
18 789 2
688 9296 1
757 8908 1
789 7815 1
789 3311 1
1054 7815 1
1054 4554 1
1203 6861 1
2279 5818 1
2624 4691 1
3311 9297 1
4238 9297 2
4554 688 1
4691 7802 2
4771 4772 1
4772 18 2
5831 1054 1
5964 2279 1
6861 2624 1
7802 4772 1
7802 4771 1
7815 4238 2
7960 8104 1
8104 9297 1
8908 1203 1
9296 7960 1
9297 6283 2
9297 757 1
9297 5964 1
```

## 6.2 Modèle de langage - perplexité

Vous allez maintenant écrire un programme ayant pour but de lire le fichier de comptes produit précédemment puis de calculer la perplexité d'un ensemble de phrases passé en paramètre grâce à un modèle de langage bigramme estimé sur le fichier de comptes.

Votre programme, une fois le fichier de compte lu, devra calculer des probabilités (exprimées en *-logprob*), pour n'importe quel unigramme ou bigramme. Pour cela vous utiliserez la méthode du maximum de vraisemblance avec un lissage de Laplace et une valeur de $\alpha$ à $0,1$.

Vous écrirez ensuite une fonction permettant de calculer la perplexité d'une phrase passée en paramètre en utilisant la formule présentée dans ce sujet de TP.

## 6.3 Extension : remettre une phrase dans l'ordre

Ce programme devra permettre de remettre en ordre une phrase dont les mots ont été donné *en vrac*. Pour cela votre programme devra énumérer toutes les permutations possibles de la phrase, calculer la perplexité de chacune d'entre elle, puis classer ces permutations par perplexité décroissante. La phrase ayant la perpelxité minimale sera celle considérée comme la meilleure par le modèle de langage.

Par exemple, si vous donnez à votre programme la phrase :

```
à vais Paris demain je
```

La permutation de plus faible perplexité devrait être :

```
demain je vais à Paris
```