Projet de Traitement Automatique de la Langue: Étiquetage Morpho-Syntaxique

1ère partie: Modèle Génératif

Luc Giffon

18 octobre 2016

L'étiquetage Morpho-syntaxique consiste en l'attribution d'une étiquete morpho-syntaxique (Nom, Verbe, Adjectif, etc.), un état, à chaque mot d'une phrase, une séquence d'observations. Le but de cette première partie de projet était de concevoir un programme Python capable de concevoir un HMM à partir des fréquences des *n-grammes* d'états et des émissions d'observations dans un corpus d'entrainement. Ce HMM devait ensuite être utilisé par l'algorithme de Viterbi afin de déterminer les états qui auraient généré les observations contenues dans un corpus de test. On fera varier la taille du corpus d'apprentissage et les hyper-paramètres du modèle pour observer leur impact sur la performance de généralisation du modèle sur le corpus de test.

1 Travail réalisé

Le travail réalisé a consisté en le développement d'un programme Python exécutable en ligne de commande permettant :

- de construire un modèle génératif, c'est-à-dire un modèle basé sur les fréquences des $n-grammes$ d'états et sur les fréquences d'émission des observations par les états.
- d'évaluer la séquence d'états ayant le plus probablement généré une séquence d'observations suivant l'algorithme de Viterbi

1.1 Installation et exécution du programme

Une fois l'archive zip décompressée ou le dépot git (https://github.com/lucgiffon/morpho_tagger) cloné, allez dans le répertoire racine et installez les modules Python requis : `pip install -r requirements.txt`.

Une aide à l'exécution est disponible en executant `tagger.py -h`.

Les fichiers de corpus / vocabulaire devraient être formattés comme les échantillons disponibles dans le répertoire `data/`.

Example 1.1. Construire un modèle : $n = 2, \alpha = 1.0, interpolation = (1.0, 0.0)$

```
python tagger.py train data/ftb.train.encoded  model.pickle --obs=data/
    voc_observables.txt --state=data/voc_etats.txt --n=2 --alpha=1.0 --
    interpolation=(1.0,0.0)
```

Example 1.2. Tester un modèle :

```
python tagger.py test data/ftb.test.encoded  model.pickle
```

Example 1.3. Faire une prédiction sur une phrase :

```
python tagger.py predict "Je fabrique des chaises ." model.pickle --encode-obs=
    data/voc_observables.txt  --decode-state=data/voc_etats.txt
```

Remarque 1.1. Les mots présents dans le corpus de test ou la séquence à évaluer doivent être connus du modèle. C'est-à-dire qu'ils doivent apparaitre dans le vocabulaire des observables fourni lors de l'entrainement du modèle.

1.2 Architecture du programme

Le programme comporte une seule classe appelée `NGramModel` qui est initialisée avec les hyper-paramètres du modèle :

- `n` est l'entier correspondant à la taille des $n-grammes$ considérés
- `alpha` est un flottant correspondant au α utilisé dans le cadre du lissage $add-\alpha$
- `interpolation` est un tuple de n flottants qui somment à 1, utilisé dans le cadre du lissage par interpolation

D'autre part, le modèle possède les attributs qui correspondent à la matrice de transition a et à la matrice d'émission b.

Parmis les méthodes principales de la classe, il y a :

- `gen_training` qui réalise l'entrainement génératif du modèle en comptant les occurences de chaque $n-gram$ et de chaque émissions d'observation dans un corpus d'entrainement (passé en argument)
- `viterbir` qui, à partir d'une séquence d'observations formant une phrase, prédit la séquence d'états ayant le plus probablement généré cette phrase.
- `test` qui décompose un corpus de test en sous-séquences, les donne à analyser par `viterbir`, compare les résultats obtenus à ceux fournis avec le corpus de test et retourne le pourcentage d'erreur.

1.3 Lissages

Le *lissage* permet de considérer efficacement les $n-grammes$ et emissions absents du corpus d'entrainement.

Par défault : D'une part le lissage de Laplace ($\alpha = 1$) est appliqué car sans lui, le programme sera incapable de réaliser la moindre prédiction. D'autre part, le lissage par interpolation est nulifié (les coefficients sont tous égaux à 0, sauf le premier, égal à 1) : c'est-à-dire que seul les $n-grammes$ d'ordre le plus élevé sont pris en compte.

Il est possible de spécifier un autre α : soit en passant par l'interface en ligne de commande à l'aide de l'option `--alpha` soit directement dans le code source, en modifiant l'attribut `alpha` de l'objet `NGramModel`. De la même façon, on peut spécifier d'autres coefficients d'interpolation, à la condition que leur nombre soit égal à n et qu'ils somment à 1.

1.4 Une implémentation récursive

Bien que l'énoncé du projet ne demande qu'une implémentation pour les modèles $2-grammes$, j'ai décidé d'écrire une code généralisable qui pour tout n, produit le modèle associé et est capable de l'utiliser dans un algorithme dit `viterbi-like`.

Cette idée a été assez complexe à mettre en oeuvre car elle demandait beaucoup de discipline quant-à l'utilisation de boucles récursives à chaque fois qu'un calcul dépendant de n (taille des $n-grammes$) était fait. C'est à dire assez souvent, en fait.

Malgré tout, ce n'est pas la construction du modèle qui a été réellement problématique mais l'implémentation de l'algorithme de Viterbi. En effet, le cas des $2-grammes$ est un cas particulièrement pratique qui permet de s'exempter de certains problèmes, notamment

en ce qui concerne le *backpointer* qui permet de reconstituer la séquence d'états à partir des scores obtenus. Cependant, ce problème a été résolu grâce à un cours où Michael Collins propose une adaptation de l'algorithme de Viterbi aux modèles $3-grammes$ et qui m'a permit de mieux comprendre le fonctionnement global de Viterbi et de le généraliser aux modèles $n-grammes$.

2 Résultats obtenus

Dans cette section, nous allons voir comment la variation des hyper-paramètres influence la performance de généralisation du modèle sur le corpus de test. Les hyper-paramètres considérés sont les suivants :

— La taille des $n-grammes$
— La taille du corpus d'apprentissage
— La valeur de α
— La valeur des coefficients d'interpolation

2.1 Influence de la taille des n-grammes

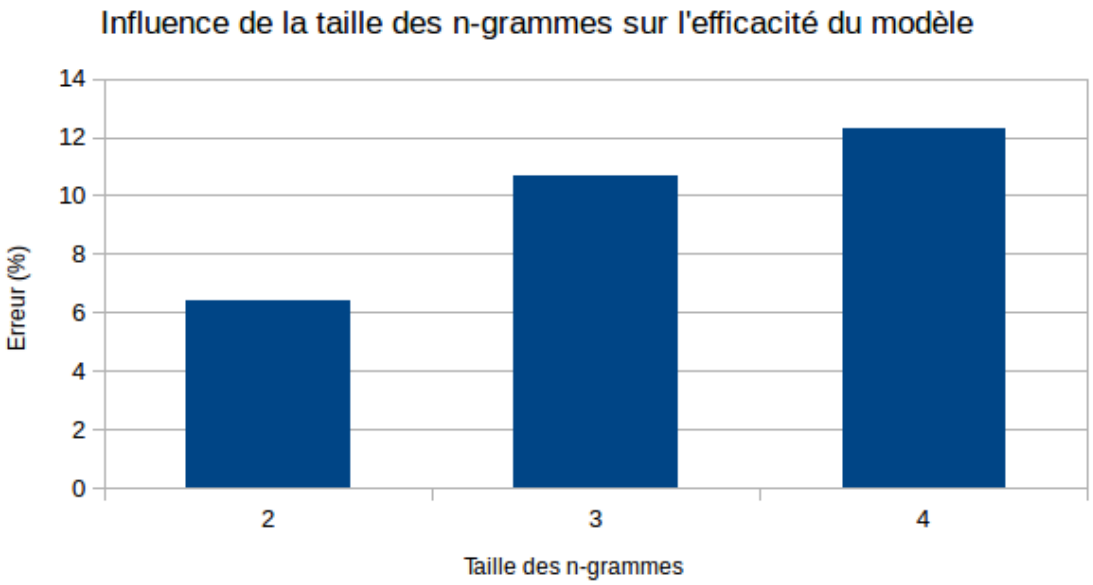

FIGURE 1 – Histogramme de l'influence de la taille des n-grammes sur l'efficacité du modèle

On voit que l'efficacité du modèle diminue avec l'augmentation de la taille des $n-grammes$ considérés. On peut supposer que cela est dû au fait que plus les $n-grammes$ considérés sont grands, moins ils sont fréquents et peut-être que le corpus d'entrainement est trop court pour que l'on puisse supposer qu'il soit un bon représentant de l'univers des possibles $n-grammes$ $(n>2)$.

Néanmoins, je pense que mon programme est bugué. En effet, après avoir observé une réduction de l'efficacité pour les $3-grammes$ par rapport aux $2-grammes$, on pouvait s'attendre à une explosion du taux d'erreur pour les $4-grammes$... or, ce n'est pas le cas : la différence entre le taux d'erreur pour les $3-grammes$ et les $4-grammes$ est même plus basse que celle entre les $2-grammes$ et les $3-grammes$.

2.2 Influence de la taille du corpus d'apprentissage

Dans cette section, nous allons étudier l'influence de la taille du corpus d'apprentissage sur l'efficacité des modèles $n-grammes$. Dans chacun des cas présentés, la valeur de α choisie était de 1.0 et aucun coefficient d'interpolation n'est appliqué.

2.2.1 Sur les modèles 2-grammes

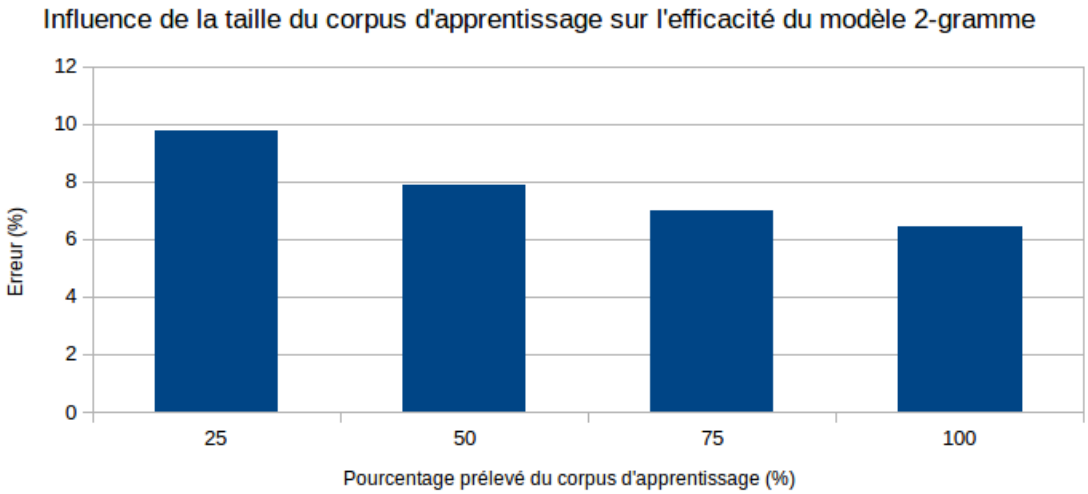

FIGURE 2 – Histogramme de l'influence de la taille du corpus d'apprentissage sur l'efficacité du modèle 2-gramme

Le constat est trivial : plus la taille du corpus d'apprentissage augmente, plus le taux d'erreur diminue. Cependant, la variation du taux d'erreur semble s'amoindrir avec l'augmentation de la taille du corpus d'apprentissage. Il faudrait essayer avec plus de tailles différentes pour le confirmer.

2.2.2 Sur les modèles 3-grammes

Comme pour les $2-grammes$, le constat est trivial : plus la taille du corpus d'apprentissage augmente, plus le taux d'erreur diminue.

2.3 Influence de la valeur α

Dans cette section, nous allons étudier l'influence de la valeur de α sur l'efficacité des modèles $n-grammes$. Dans chacun aucun coefficient d'interpolation n'est appliqué.

2.3.1 Sur les modèles 2-grammes

On voit clairement que plus α diminue, plus l'efficacité du modèle augmente. Cependant, et bien que cela n'apparaisse pas sur le graphe, j'ai expérimenté le fait que cette augmentation de l'efficacité ne continue pas toujours de diminuer jusqu'à $\alpha = 0$.

2.4 Influence des coefficients d'interpolation

Dans cette section, nous allons étudier l'influence des coefficients d'interpolation sur l'efficacité des modèles $n-grammes$. Dans chacun des cas présentés, la valeur de α choisie était de 1.0.

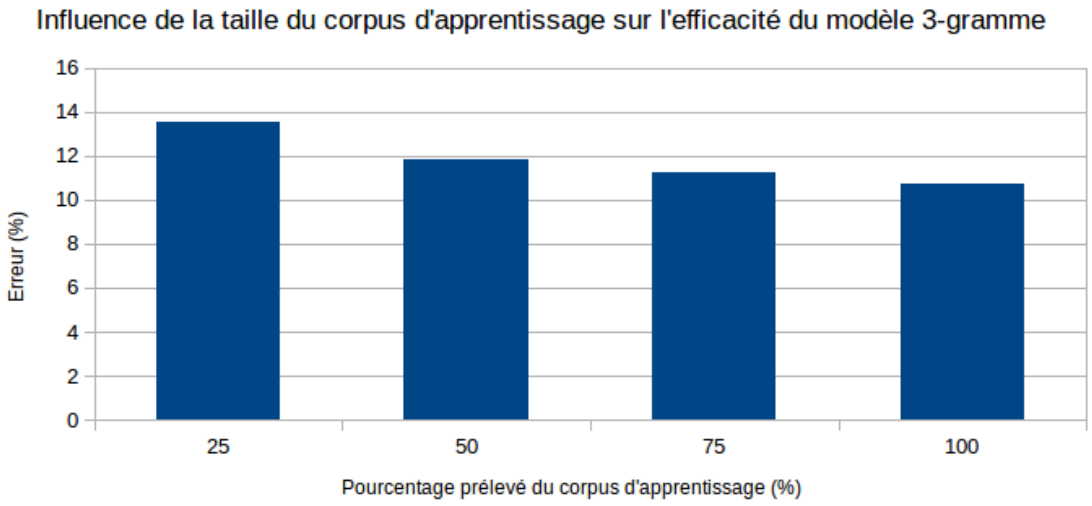

FIGURE 3 – Histogramme de l'influence de la taille du corpus d'apprentissage sur l'efficacité du modèle 3-gramme

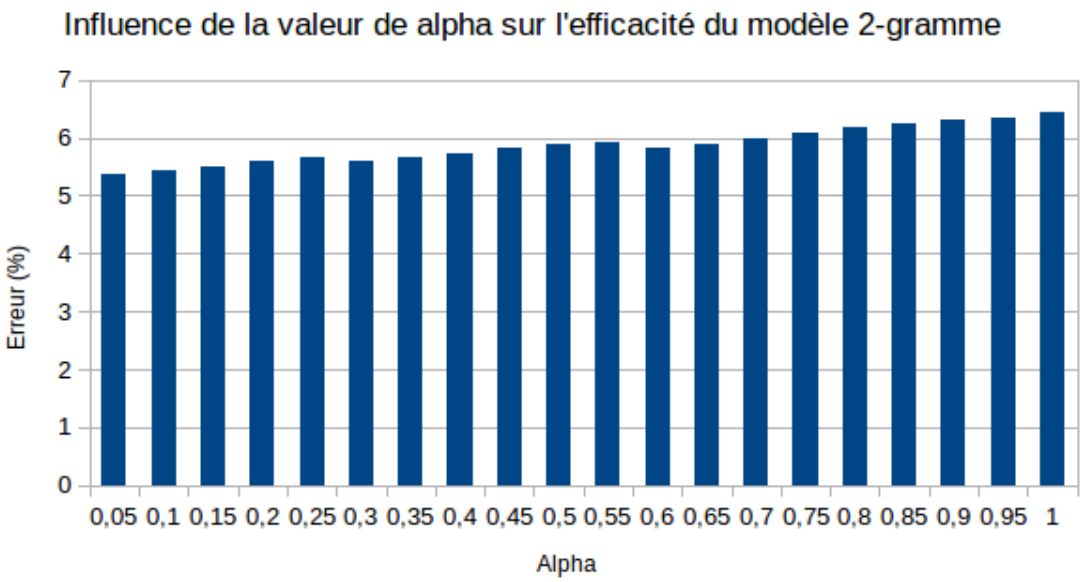

FIGURE 4 – Histogramme de l'influence de la valeur de α sur l'efficacité du modèle 2-gramme

Sur les graphiques, vous pourrez voir la légende "Première valeur d'interpolation". Par soucis de simplicité d'implémentation (et de temps), les coefficients d'interpolation ont été généré comme suis :

- Selection d'une valeur d'interpolation pour le $n-gramme$ d'ordre le plus élevé
- Obtention de la valeur d'interpolation pour le $n-gramme$ d'ordre inférieur par soustraction de (environ) 1 par la valeur obtenue précédemment.
- S'il reste des $n-grammes$ d'ordre inférieur, leur coefficient d'interpolation est ajusté à (environ) 0.

2.4.1 Sur les modèles 2-grammes

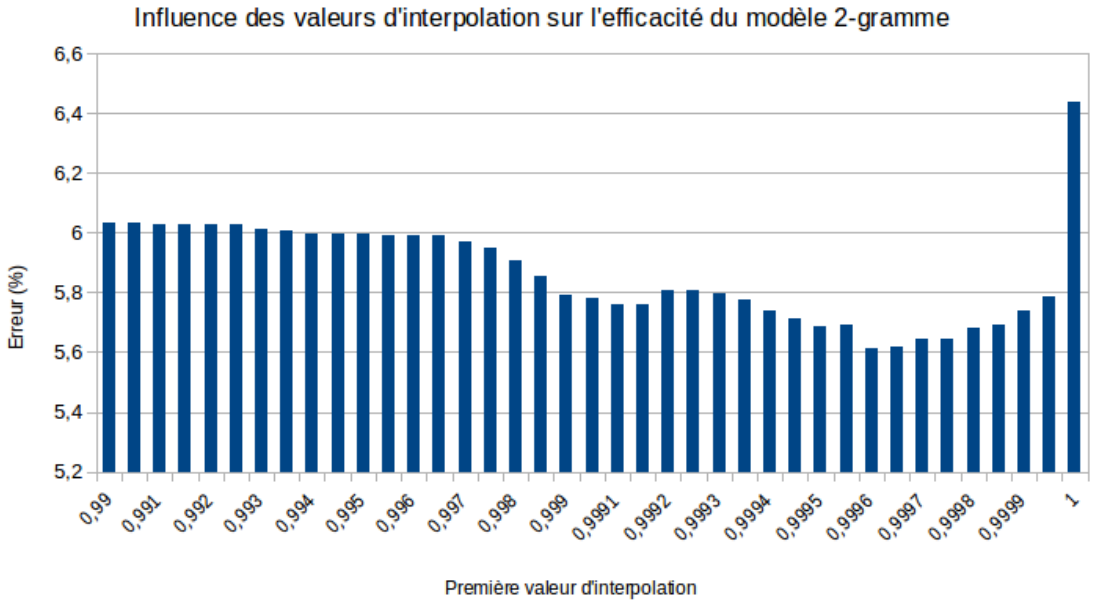

FIGURE 5 – Histogramme de l'influence des valeurs d'interpolation sur l'efficacité du modèle 2-gramme

— D'une part, on voit clairement que l'utilisation de coefficients d'interpolation joue un rôle important dans l'amélioration des résultats lorsque α est égal à 1.0.
— D'autre part, si vous regardez attentivement la légende, vous pourrez constater que les coefficients d'interpolation ne varient pas de la même façon entre $\lambda_1 = 0.99...0.997$ $\lambda_1 = 0.997...1.0$. Pourtant, la réduction du taux d'erreur est bien plus flagrante dans le second interval que dans le premier.

Il y a fort a parier qu'en continuant de réduire λ_1 en dessous de 0.99, on n'obtiendrait pas de meilleur résultat et on pourrait en déduire que dans le cas du modèle $2-gramme$, il vaut mieux attribuer un poids très bas aux probabilités des $1-grammes$... sans les négliger pour autant !

3 Conclusion

Par manque de temps, de nombreux objectifs n'ont pas pu être accomplis et le seront peut-être pour le rendu de la seconde partie, parmis eux :

— Implémentation de l'algorithme EM pour l'apprentissage semi-supervisé
— Plus de tests pour confirmer les suspicions (influence de la taille du corpus d'apprentissage, influence de α, ...) évoquées dans le compte-rendu
— Vérification de l'algorithme de Viterbi pour confirmer/infirmer le bug relatif aux $4-grammes$
— Détermination des meilleurs hyper-paramètres sur un corpus de developpement afin d'obtenir le meilleur score possible sur le corpus de test (sans tricher)

A part ça, désolé pour le compte-rendu baclé. J'ai sous-estimé le temps nécessaire à sa rédaction.