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

2nd partie: Modèle Discriminant

Luc Giffon

15 novembre 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 seconde partie de projet était de concevoir un Perceptron en Python à entrainer sur un corpus d'entrainement. L'entrainement de ce HMM devait être fait grâce à l'algorithme de Viterbi développé dans la partie précédente. 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 une amélioration du programme existant afin de permettre à l'utilisateur de choisir entre un HMM basé sur les fréquences des observations ou bien basé sur l'entrainement itératif du perceptron.

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`. (attention, les commandes d'utilisation ont changé depuis la partie 1)

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

Example 1.1. Construire un modèle : $nombred'itrations = 2$

```
python tagger.py disc-train data/ftb.train.encoded  model.pickle --obs=data/
    voc_observables.txt --state=data/voc_etats.txt --iteration=2
```

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 :

— `number_iteration` est l'entier correspondant au nombre d'itérations souhaité.

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 :

— `disc_training` qui réalise l'entrainement discriminant du modèle grâce à l'algorithme du perceptron.

— `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.

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 :

— Le nombre d'itérations

— La taille du corpus d'apprentissage

2.1 Influence du nombre d'itérations

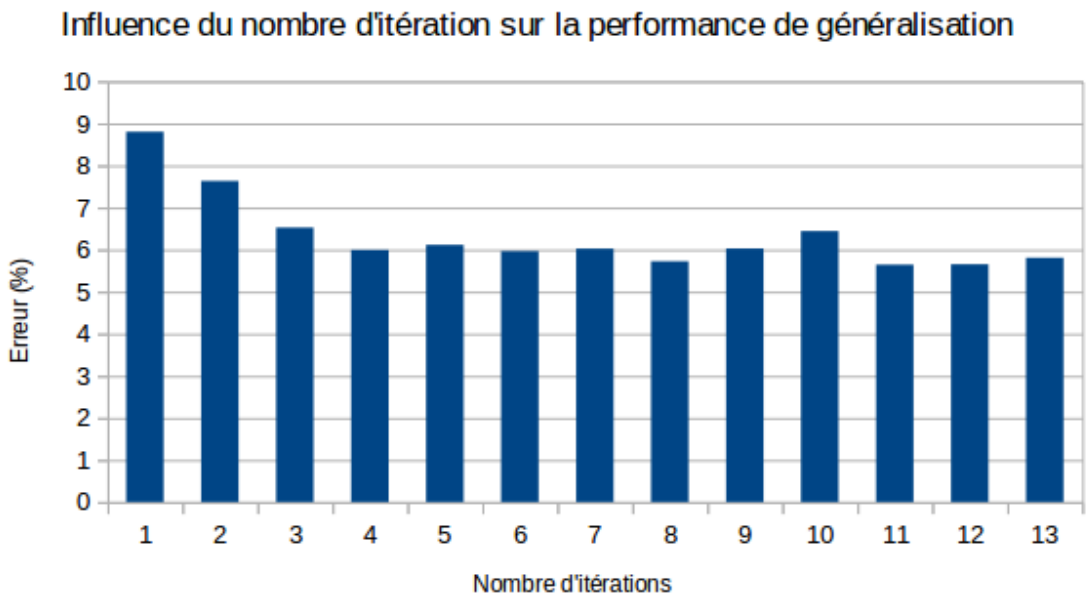

FIGURE 1 – Histogramme de l'influence du nombre d'itérations sur l'efficacité du modèle

On observe une nette réduction du pourcentage d'erreur entre 1 et 3 itérations (de 8.80% à 6.52%) puis une sorte de stabilisation. Je m'attendais à observer une ré-augmentation du pourcentage à partir d'un certain point à cause d'un éventuel phénomène de sur-apprentissage mais ça n'a pas été le cas. Je pense que c'est dû au nombre d'itération

qui est trop bas pour voir cela apparaitre. Je n'ai pas pu faire plus d'expérimentation car l'algorithme est très lent.

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, le nombre d'itérations est de 3, compte tenu des résultats obtenus dans la sous-section précédente.

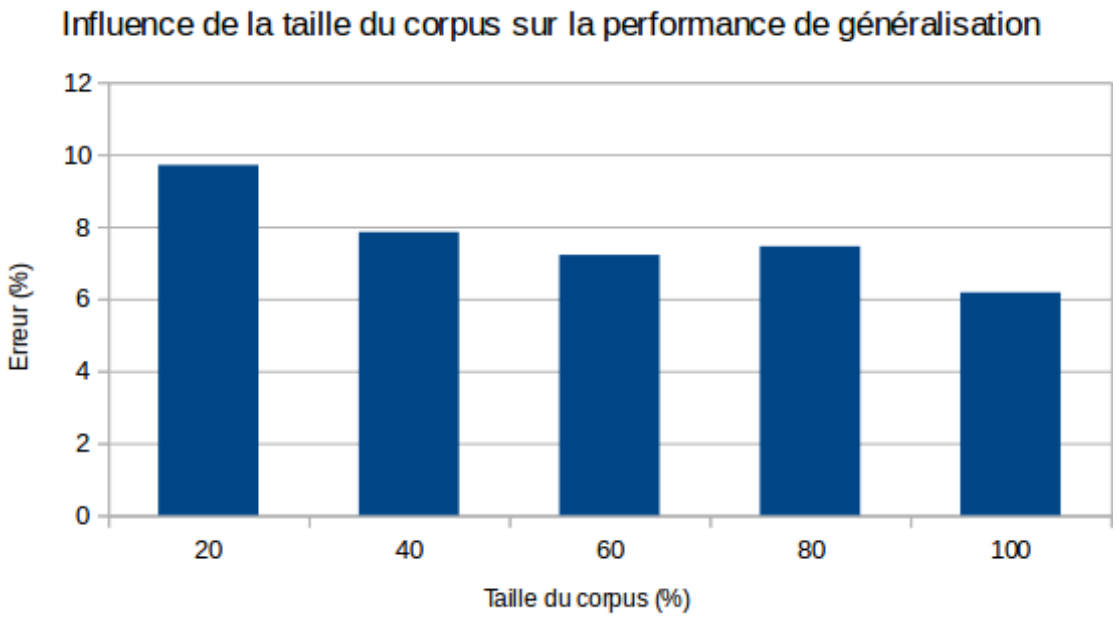

FIGURE 2 – Histogramme de l'influence de la taille du corpus sur l'efficacité du modèle

On observe que l'efficacité augmente globalement avec la taille du corpus d'apprentissage jusqu'à 80% du corpus total.

2.3 Couplage de l'algorithme génératif et discriminant

L'algorithme du perceptron a été executé à partir d'un modèle génératif pour vérifier si on pouvait obtenir de meilleur résultat : alpha=1 | nombre d'itérations = 1 | pas d'interpolation.

Avec ces paramètres, on obtient 7.20% d'erreur, ce qui est moins bon que les résultats obtenus sans perceptron (6.44%). On peut expliquer cette diminution de l'efficacité par un phénomène de sur-apprentissage du corpus d'entrainement.

3 Conclusion

Les objectifs fixés en conclusion de la première partie n'ont pas été remplis.

Les résultats obtenus dans cette partie ne me semblent pas pertinents en ce sens que l'algorithme du perceptron admet un caractère aléatoire dû au mélange des séquences du corpus d'entrainement. Il aurait donc fallût faire, pour chaque expérience, un plus grand nombre d'essai et en extraire les moyennes afin de conclure de façon plus tranchée.