Projet TC4

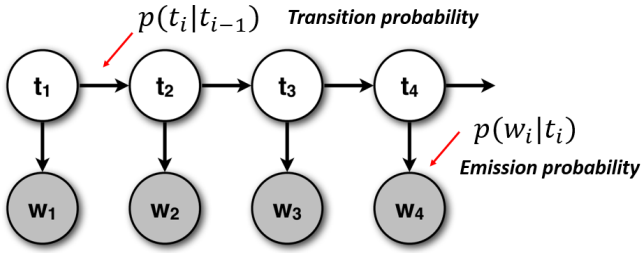
YAKOUBI Yacine & MADJI Youcef

January 2019

1 Introduction

L'objectif de ce projet est de concevoir un modèle pour corriger les fautes de frappe dans les textes sans dictionnaires en utilisant un modéle de markov à états cachés (HMM). Étant donnée, une séquence de lettres réellement tapées, le probléme consiste à reconstruire la séquence de lettre voulue.

Figure 1: Hidden Markov Model



Les trois types de fautes de frappe que nous allons traiter dans ce projet sont :

- **Substitution :** Une lettre sera remplacée par une autre lettre pour avoir le mot correcte.
- **Insertion :** Une lettre est inserée en plus au mot voulue.
- **Suppression :** Il manque une lettre pour avoir le mot voulue.

2 Modèle de Markov Cachè

La modélisation est faite par une chaîne de Markov discreète.

- **Un état Si**, est la lettre qui aurait dû être tapée,
- **Une observation Oi**, est la lettre qui a vraiment été tapée.

L'ensemble des états possibles et l'ensemble des observations possibles sont tout les deux de taille 27; 26 lettres et la balise **unk** pour les caractéres inconnus. Le problème est donc de reconstruire, à partir d'une sèquence d'observations, la sèquence d'états cacheés à l'origine de celle-ci. C'est-à-dire, à partir d'un texte avec des fautes de frappe, retrouver le texte voulu en premier lieu.

2.1 HMM de premier ordre

Pour implementer le HMM1, nous avons repris le code du TP et nous avons impleémenté l'algorithme de Viterbi. Cet algorithme a pour but de trouver la sèquence d'états la plus probable ayant produit une sèquence mesurée.

Dataset	Pourcentage avant correction	Pourcentage aprés correction
Test10	10.17	7.58
Test20	19.40	14.48

Avec le HMM1, on obtient donc une amélioration de 30% en moyenne, c'est-à-dire que le modèle corrige un tiers des fautes de frappe.

2.2 HMM de second ordre

Dans un HMM de second ordre, la probabilité d'apparition d'une lettre depend des deux lettres précédentes. Plus formellement, ce qui signifie que la probabilité d'un état suivant dépend de l'état actuel et de l'état précédent.

Dataset	Pourcentage avant correction	Pourcentage aprés correction
Test10	10.17	4.65
Test20	19.40	9.26

Le HMM2 corrige 55% des erreurs en moyenne, c'est à dire que le modèle corrige un peu plus de la moitié des fautes de frappe. On voit que l'ordre 2 est nettement plus efficace.

2.3 Insertion

Dans le cas de l'insertion, nous avons opté à changer le corpus car nous n'avions pas eu le besoin de modifier le modèle; Nous avons procédé à ce changement en ajoutant une lettre au hasard avec la balise **ins** pour signifier que cette lettre a été inserée. Donc l'ensemble des états possibles est actuellement de 28 de taille. A titre d'exemple : [('g', 'g'), ('o', 'o'), ('u', 'u'), ('f', 'f'), ('d', '**ins**'), ('r', 'r'), ('e', 'e')]

Nous avons entrainé et testé de modèle sur ce nouveau corpus avec le caractère **ins**. Les résultats obtenus sont les suivants :

On remarque que le HMM2 obtient une amélioration moyenne de 40% lorsqu'il s'agit des fautes de frappe d'insertion de caractère.

2.4 Suppression

Hélas, pour la suppression nous ne pouvons pas utiliser la même technique qu'auparavant. Dans le cas présent, il est nécessaire de toucher completement à la structure de données et la modifier. Nous modifions la structure des données pour coupler les lettres par deux au lieu d'avoir des couples (état, observation) pour chaque lettre. Nous couplons les lettres dans leur ordre d'arrivée. Ainsi, pour le mot de longueur impair, sa dernière lettre reste seule. Par exemple :

[(t,t), (r, r), (a, a), (i, i), (n, n)] $\Rightarrow [(tr, tr), (ai, ai), (n, n)]$

ce qui nous emmene donc à modèliser la suppression d'un caractère ainsi: [(tr, tr), (a, ai), (t, n)] Dans l'exemple ci-dessous le 'i' a été supprimé.

Cette solution nous permet donc de corrigé la supprission de catactère, mais elle a aussi ses inconvénient. Parmi ses inconvénient on trouve :

- **Le nombre d'états possibles augmente énormément,** passant de 27 à $(27 * 26 + 1)$, soit 703.
- **Certaines suppressions ne sont pas possible.**Par exemple,dans le cas des mots de longueur impair, la suppression de la dernière lettre est impossible.

Voici les résultats obtenu:

Dataset	Pourcentage avant correction	Pourcentage aprés correction
Test10	8.02	2.77
Test20	16.75	5.10

Cependant, l'algorithme met beaucoup de temps mais les résultats sont intéréssants !

3 Apprentissage non supervisé

3.1 Algorithme de Baum Welch

Nous avons essayé d'implémenter l'algorithme de Baum-Welch [1]. Il permet d'estimer les paramètres d'un HMM. L'algorithme de Baum-Welch est un algorithme itératif, qui permet d'estimer les paramètres du modèle qui maximisent la probabilité d'une séquence d'observations. Il converge vers un maximum local. Malhereusement, nous n'avons pas terminé l'implementation de ce dernier, nous nous pouvons donc pas fournir des résultats sur l'effficacité de cette méthode.

3.1.1 Initialisation

Nous initialisons tout d'abord les valeurs des paramètres du modèle de Markov : $\theta = \Pi$, A, B. L'initialisation peut etre aléatoire par défaut. Pour accélérer la convergence de l'algorithme est nécessaire de connaitre les distributions à priori des probabilités. Une fois l'initialisation faite, l'algorithme suit une itération des trois parties suivantes jusqu'à ce qu'un critère de convergence ou d'arrêt soit atteint.

3.1.2 Etape Forward

Dans cette étape nous calculons $\alpha_i(t) = P(Y_1 = y_1, ..., Y_t = y_t, X_t = i|\theta$). Pour cela on procède par boucle sur sur les lettres du texte comme suit :

$\alpha_i(1)$ =$\pi_i b_i(y1)\alpha$(t + 1) = $b_i(y_{t+1})\sum \alpha_j(t)a_{ji}$

3.1.3 Etape Backward

Nous poursuivons après l'étape Forward par parcourir le texte dans le sens contraire, pour calculer la probabilité de la séquence $Y_{t+1}...Y_T sachant X_t$:$\beta_i(t) = Y_{t+1} = y_{t+1}, ..., Y_T = y_T|X_t = i,\theta)$. Elle est calculée d'une manière similaire à l'étape Forward.

3.1.4 Mise à jour des paramètres

La mise à jour des paramètres du modèle de Markov est faite dès que les grandeurs α et β sont calculées. Pour ce faire on introduit les deux grandeurs : $\gamma = P(X_t|Y) et \xi = P(X_t, X_{t+1}|Y)$.

Le γ est calculé comme suit :

$$\begin{aligned}\gamma_t(i) &= P(X_t = i|Y, \theta) \\ &= \frac{P(X_t = i, Y|\theta)}{P(Y|\theta)} \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}\end{aligned}$$

Et ξ, on le calcule de la manière suivante :

$$\begin{aligned}\xi_t(i,j) &= P(X_t = i, X_{t+1} = j|Y, \theta) \\ &= \frac{P(X_t = i, X_{t+1} = j, Y|\theta)}{P(Y|\theta)} \\ &= \frac{\alpha_t(i)a_{ij}b_j(y_{t+1})\beta_{t+1}(j)}{P(Y|\theta)} \\ &= \frac{\alpha_t(i)a_{ij}b_j(y_{t+1})\beta_{t+1}(j)}{\sum_{k=1}^{N}\sum_{l=1}^{N} \alpha_t(k)a_{kl}b_l(y_{t+1})\beta_{t+1}(l)}\end{aligned}$$

aij est la probabilité de transition de l'état i vers l'état j et bj(y) représente la probabilité d'observer y à l'état j, tandis que les valeurs αt(i) et βt(i) sont calculées en utilisant l'algorithme forward-backward. Une fois qu'on dispose de ces deux grandeurs, nous mettons à jour les paramètres du modèle de Markov de la manière suivante:

$$\pi = \gamma(1)$$
$$A = \frac{\sum_{t=1}^{T} \xi(t)}{\sum_{t=1}^{T} \gamma(t)}$$
$$B(t') = \frac{\sum_{t=1}^{T} \mathbb{1}_{y_t = t'} \gamma(t)}{\sum_{t=1}^{T} \gamma(t)}$$

4 Conclusion

En effectuant ce travail, nous avons pu créer un bon correcteur de typos, il pourra être amélioré encore plus en prenant en compte de beaucoup d'autres paramétres. On trouve d'autres types de fautes de frappes courantes comme par exemple l'omission d'un espace entre deux mots ou le contraire. Il est aussi intéressant d'augmenter l'ordre du HMM, de modifier la taille du vocabulaire ou d'utiliser un réseau de neurones récurrent pour avoir de meilleurs performances.

References

[1] Wikipedia article on Baum-Welch algorithm.URL: https://en.wikipedia.org/-wiki/BaumWelch_algorithm.

[2] Github source code.URL:https://github.com/amaltarghi/TC4-Second-Order-HMM-for-typos-correction-

[3]Wolfram article https://www.wolfram.com/mathematica/new-in-10/enhanced-random-processes/perform-typo-correction-without-a-dictionary.html