Projet TC4

Eléonore Bartenlian, Adrien Pavao, Hoël Plantec

January 2018

1 Introduction

L'objectif de ce projet était de concevoir un modèle pour corriger les fautes de frappe dans des textes sans utiliser de dictionnaire.

Le problème a été modélisé par un modèle de Markov à états cachés (HMM). Étant donnée une séquence d'observations (c'est-à-dire des lettres réellement tapées), le problème consistait à reconstruire la séquence d'états cachés (c'est-à-dire la séquence de lettres voulue).

Table 1: Major Types of Query Spelling Errors

Type		Example	Correction
In-Word	Insertion	esspresso	espresso
	Deletion	vollyball	volleyball
	Substitution	comtemplate	contemplate
	Mis-use	capital hill	capitol hill
Cross-Word	Concatenation	intermilan	inter milan
	Splitting	power point	powerpoint

Figure 1: Différents types de typos [1]

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

- **Substitution :** Une lettre est remplacée par une autre.
  Exemple : "r**a**dis" → "r**z**dis"
- **Insertion :** Il y a une lettre en trop.
  Exemple : "carotte" → "carot**h**te"
- **Suppression :** Il manque une lettre.
  Exemple : "p**o**ireau" → "pireau"

2 Les données

Les données ont été générées partir d'un document texte (*The Unabomber Manifesto*). Les ponctuations et autres caractères spéciaux ont été remplacées par des espaces blancs et les lettres ont été converties en minuscules.

Des fautes de frappes ont ensuite été ajoutées artificiellement : chaque lettre a 10% (ou 20% selon le fichier) de chance d'être changée aléatoirement par une autre lettre proche sur un clavier ordinaire.

3 Modèle de Markov Caché

La modélisation est faite par une chaîne de Markov discrète.

- **Un état** S_i est la lettre correcte qui aurait dû être tapée.
- **Une observation** O_i est la lettre qui a vraiment été tapée.

L'ensemble des états possibles et l'ensemble des observations possibles sont tous deux de taille 27 (26 lettres et la balise $< unk >$ pour les caractères inconnus).

Le problème est donc de reconstruire, à partir d'une séquence d'observations, la séquence d'états cachés 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.

3.1 HMM de premier ordre

Nous avons repris le code du TP 2 et nous avons implémenté l'algorithme de Viterbi. Cet algorithme a pour but de trouver la séquence d'états la plus probable ayant produit une séquence mesurée.

Pourcentage de typos	Après correction
10.18	7.58
19.41	14.48

Figure 2: Résultats avec un HMM d'ordre 1

Avec le HMM d'ordre 1, on obtient donc une amélioration de 25 % en moyenne, c'est-à-dire que le modèle corrige un quart des fautes de frappe.

3.2 HMM de second ordre

Dans le cas d'un HMM de second ordre, plutôt que la probabilité d'apparition d'une lettre ne dépende que de la précédente, elle va dépendre des deux précédentes. Plus formellement, cela signifie que la probabilité d'un état suivant dépend de l'état actuel et de l'état précédent.

Pourcentage de typos	Après correction
10.18	4.66
19.41	9.26

Figure 3: Résultats avec un HMM d'ordre 2

Le HMM d'ordre 2 corrige 53% 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.

3.3 Insertion

Pour l'insertion, nous n'avons pas eu besoin de modifier notre modèle. Nous avons simplement changé le corpus : nous avons rajouté une lettre au hasard et rajouté la balise $< ins >$ pour signifier que cette lettre a été insérée. L'ensemble des états possibles est donc à présent de taille 28. Cela donne, par exemple :

```
[('t', 't'), ('o', 'o'), ('m', 'm'), ('z', '<ins>'), ('a', 'a'), ('t', 't'), ('e', 'e')]
```

Il nous a suffit d'entraîner et de tester le modèle sur ce nouveau corpus avec le nouveau caractère $< ins >$. Nous avons obtenu les résultats suivants :

Pourcentage de typos	Après correction
10.31	6.37
20.12	10.89

Figure 4: Résultats avec un HMM d'ordre 2, insertions uniquement

Avec des insertions et des substitution (10% + 10% et 20% + 20%) :

Pourcentage de typos	Après correction
18.77	12.79
35.43	26.10

Figure 5: Résultats avec un HMM d'ordre 2, insertions et substitutions

Lorsque les fautes de frappe sont uniquement des insertions de caractères, le HMM d'ordre 2 obtient une amélioration moyenne de 42%. Il est donc un peu moins efficace lorsqu'il s'agit de corriger des insertions que pour des substitutions.

Lorsque le texte contient des insertions et des substitutions, le modèle corrige 29 % des erreurs. On remarque que la correction des typos est moins bonnes lorsque l'on combine ces deux types de typos. Il faut tout de même souligner que dans cet exemple, on traite des textes avec beaucoup de typos (18.77% et 35.43%) tandis habituellement on se situe entre 10% et 20%. Cela contribue sans doute a baisser un peu le score.

3.4 Suppression

Pour la suppression, nous ne pouvons malheureusement pas utiliser la même technique. Pour corriger une insertion, on pouvait ramener le cas à une substitution tandis que pour prendre en compte les suppressions il est nécessaire de modifier la structure de données.

Plutôt que d'avoir des couples (état, observation) pour chaque lettre, on pourrait modifier la structure des données afin de coupler les lettres par deux. Ainsi, on pourrait représenter la suppression d'un caractère. On couple les lettres dans leur ordre d'arrivée. Ainsi, pour les mots de longueur impair, la dernière lettre reste seule.

Exemple :

$$[(n, n), (a, a), (v, v), (e, e), (t, t)]$$

devient

$$[(na, na), (ve, ve), (t, t)]$$

On peut donc modéliser la suppression d'un caractère ainsi :

$$[(na, na), (v, ve), (t, t)]$$

Ici, le 'e' a été supprimé.

Cette nouvelle représentation permet donc au modèle de corriger les suppressions de caractères, cependant elle a certains défauts :

- Certaines suppressions ne sont pas possible, par exemple la suppression simultanée des deux premières lettres d'un mot. Deux lettres côtes à côtes peuvent être supprimées ensemble si et seulement si la première est à une position impaire du mot (en commençant les indices par 0). Dans le cas des mots de longueur impair, la suppression de la dernière lettre est impossible.

- Le nombre d'états possibles augmente énormément, passant de 27 à $(27 \times 26 + 1)$, soit 703. En effet, le vocabulaire n'est plus constituée seulement des lettres de l'alphabet mais aussi des paires de lettres. Cela a pour conséquences d'augmenter grandement la complexité de l'algorithme.

En revanche, cette méthode à l'avantage de ne s'appuyer que sur une modification de la représentation des données. Ainsi, le modèle de Markov reste identique. Voici les résultats obtenus :

Pourcentage de typos	Après correction
8.28	2.85
16.09	4.46

Figure 6: Résultats avec un HMM d'ordre 2, suppressions uniquement

Avec des suppressions et des substitutions :

Pourcentage de typos	Après correction
24.04	5.53
43.01	11.31

Figure 7: Résultats avec un HMM d'ordre 2, suppressions et substitutions

Le calcul met beaucoup de temps, mais les résultats sont très bons. Lorsque les fautes de frappe sont uniquement les suppressions présentées ci-dessus, 69% d'entre elles sont corrigées. Il s'agit du meilleur score que nous ayons obtenu dans la correction d'un seul type d'erreur à la fois. Le fait d'avoir un plus gros vocabulaire semble améliorer les résultats, au pris d'une grande complexité algorithmique.

L'algorithme corrige 75% des erreurs lorsqu'on trouve des substitutions et des suppressions dans les données. Les substitutions peuvent subvenir sur n'importe quelle lettre avec une probabilité de 0.1 ou 0.2, tandis que les suppressions surviennent sur des paires de lettres avec ces mêmes probabilités. Il faut donc prendre cela en compte en observant les résultats, car il y a deux fois plus de substitutions que de suppressions. On pourrait corriger cela en doublant la probabilité des suppressions.

De plus, la mise en couple influence le calcul du taux d'erreur, augmentant celui-ci. Quoiqu'il en soit, le fait d'avoir un vocabulaire constitué des lettres et des paires de lettres améliore grandement les résultats, c'est donc une bonne idée pour un correcteur d'erreurs dans des textes.

On notera également qu'il est possible que le fait que certaines suppressions ne soient pas possible influe positivement les résultats.

Pour traiter les problèmes d'insertion et de suppression, on peut également utiliser une des techniques décrites dans [2] ou [3].

4 Apprentissage non supervisé

4.1 Algorithme de Baum Welch

Nous avons implémenté l'algorithme de Baum-Welch [4]. Il permet d'estimer les paramètres d'un HMM ; c'est un cas particulier d'une généralisation de l'algorithme espérance-maximisation (EM). 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. Malheureusement des bugs persistent dans notre algorithme et la mise à jour des paramètres du modèle de Markov ne s'effectue pas correctement, nous ne pouvons donc pas fournir de résultats sur l'efficacité de cette méthode.

Initialisation

On commence par initialiser les valeur des paramètres du modèle de Markov : $\theta = \pi, A, B$. On peut prendre une initialisation aléatoire par défaut, mais une connaissance a priori sur les distributions de probabilités peut être utile pour accélérer la convergence de l'algorithme.

Une fois cette initialisation effectuée, l'algorithme suit une itération des trois parties suivantes jusqu'à ce qu'un critère de convergence (ou d'arrêt) soit atteint.

Etape Forward

L'étape Forward consiste à calculer $\alpha_i(t) = P(Y_1 = y_1, ..., Y_t = y_t, X_t = i|\theta)$. Pour ce faire on procède par boucle sur les étapes de temps (donc dans notre cas sur les lettres du texte) :

$$\begin{aligned}\alpha_i(1) &= \pi_i b_i(y_1) \\ \alpha_i(t+1) &= b_i(y_{t+1}) \sum_{j=1}^{N} \alpha_j(t) a_{ji}\end{aligned}$$

Etape Backward

Après l'étape Forward, le but est de parcourir le texte dans l'autre sens, afin de calculer la probabilité de la séquence $Y_{t+1}...Y_T$ sachant X_t: $\beta_i(t) = Y_{t+1} = y_{t+1}, ..., Y_T = y_T|X_t = i, \theta)$. Ce calcul se fait de manière similaire à l'étape précédente :

$$\begin{aligned}\beta_i(T) &= 1 \\ \beta_i(t) &= \sum_{j=1}^{N} \beta_j(t+1) a_{ij} b_j(y_t + 1)\end{aligned}$$

Mise à jour des paramètres

Une fois les grandeurs α et β calculées, il convient de mettre à jour les paramètres du modèle de Markov. Pour cela on introduit deux autres grandeurs : γ et ξ, respectivement $P(X_t|Y)$ et $P(X_t, X_{t+1}|Y)$.

Concernant γ, on peut le calculer de la manière suivante :

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

Pour ξ, on procède ensuite de la manière suivante :

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

Avec a_{ij} la probabilité de transition de l'état i vers l'état j et $b_j(y)$ la probabilité d'observer y lorsque l'on est dans l'état j, et où les valeurs $\alpha_t(i)$ et $\beta_t(i)$ se calculent avec l'algorithme forward-backward.

Dès lors que l'on dispose de ces deux grandeurs, on peut mettre à jour les paramètres du modèle de Markov :

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

4.2 Usage d'un dictionnaire

D'autres méthodes sont possibles afin d'effectuer un apprentissage non-supervisé pour estimer la distribution d'états la plus probable. On peut par exemple créer un dictionnaire de mots sur les données d'apprentissage (avec leur fréquence d'apparition), puis de s'en servir pour la prédiction. Une idée d'utilisation peut être d'observer, pour chaque observation du modèle de Markov, les mots de même longueur présents dans le dictionnaire ; il convient ensuite d'en calculer la distance à l'observation considérée, puis de sélectionner le mot le plus vraisemblable (en tenant en compte la distance ainsi que la fréquence, qui agira alors comme un terme de régularisation).

Pour la distance, on peut s'intéresser à la distance entre deux lettres sur un clavier azerty (ou qwerty selon la langue), puis de combiner ces distances en considérant que les erreurs de frappes sont iid. La distance entre deux mots sera alors le produit des distances entre leurs lettres.

Remarquons que cette hypothèse d'indépendance est probablement fausse (si on décale une première lettre sur la gauche, il est probablement plus vraisemblable que les lettres suivantes soient également décalées dans la même direction).

En outre, le dictionnaire appris sur cette méthode est également biaisé car il contient les erreurs de frappe de l'utilisateur sur les données d'entraînement, ainsi une erreur récurrente peut amener la création d'erreurs par le modèle. On peut, afin de parer cette éventualité, utiliser un dictionnaire extérieur (supposé sans fautes, et qui fournirait des fréquences d'apparition de mots potentiellement plus fiables car calculées sur un corpus de textes plus grand).

Il est également intéressant de noter que ce modèle tel quel ne permet pas de gérer les problèmes d'insertion ou de suppression de caractères, uniquement les substitutions. On peut alors élargir le modèle en modifiant la distance entre les mots pour prendre en compte les mots de nombres différents de lettres. Dans ce cas, il vaudrait pour calculer la vraisemblance de substitution d'un mot à l'autre considérer ce changement de nombres de lettres en considérant tout mapping possible des lettres du mot substituant aux lettres de l'observation.

La méthode peut bien entendu être étendue en utilisant des bigrammes ou trigrammes de mots afin d'obtenir des portions de phrases ayant une cohérence (au sens des données d'apprentissage).

5 Conclusion

Bien que rudimentaire, nous avons pu créer un correcteur de typos qui fonctionne relativement bien. Cependant, il y aurait encore beaucoup de paramètres à prendre en compte si nous voulions l'améliorer : en effet, en plus du fait que toutes les fautes d'insertion et de suppression ne sont pas forcément bien traitées, il existe d'autres types de fautes de frappes qui sont courantes telles que l'inversion de deux lettres dans un mot, l'omission d'un espace entre deux mots qui vont se retrouver collés ou, au contraire, un mot qui coupé en deux par un espace.

De plus, il pourrait être intéressant de jouer sur la taille du vocabulaire ou d'augmenter l'ordre du HMM afin d'obtenir de meilleurs performances. L'utilisation d'un réseau de neurones récurrent pourrait être une piste également.

References

[1] Huizhong Duan Yanen Li and ChengXiang Zhai. "A Generalized Hidden Markov Model with Discriminative Training for Query Spelling Correction". In: *SIGIR* (2012). URL: http://sifaka.cs.uiuc.edu/czhai/pub/sigir12-spelling.pdf.

[2] Vahid Noroozi. "Probabilistic insertion, deletion and substitution error correction using Markov inference in next generation sequencing read". In: *Graduate Theses and Dissertations* (2016). URL: http://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=6104&context=etd.

[3] ChengXiang Zhai Yanen Li Huizhong Duan. "CloudSpeller: Spelling Correction for Search Queries by Using a Unified Hidden Markov Model with Web-scale Resources". In: *WWW'12 - Proceedings of the 21st Annual Conference on World Wide Web Companion* (2012). URL: http://times.cs.uiuc.edu/czhai/pub/speller-2011.pdf.

[4] *Wikipedia article on Baum-Welch algorithm*. URL: https://en.wikipedia.org/wiki/Baum-Welch_algorithm.