Segmentation de phrases en japonais

Applications Multilingues (X9IT100)

Joseph Lark

15 décembre 2013

1 Introduction

Les phrases en japonais ne présentent pas de séparateur entre les termes comme l'espace en français ; la segmentation de ces phrases est donc un problème, d'autant plus qu'elles peuvent contenir des symboles de différents alphabets et que ces symboles sont en très grand nombre. On réalise ici quelques expériences sur la segmentation du japonais, à partir d'un corpus de phrases segmentées et non segmentées (corpus `knbc` de `nltk`), ainsi que quelques articles traitant du sujet.

On essaiera tout d'abord une approche naïve, puis l'implémentation d'une méthode expliquée dans un article et enfin on essaiera de modifier cette méthode en vue d'améliorer les résultats.

2 Approche naïve

Commencer par une approche naïve permet d'évaluer l'amélioration réelle qu'apporte la méthode suivante. Cela pourra aussi mettre en lumière certaines difficultés pour la suite.

On propose donc dans un premier temps un modèle de segmentation simple tel que pour un caractère donné c, on ait une probabilité $P(coupure|c)$ de terminer le token courant par ce caractère. En parcourant le corpus d'entraînement on a alors deux classes de caractères : ceux faisant majoritairement partie d'un token jusqu'à l'avant-dernier caractère, et ceux ayant une plus grande probabilité de terminer un token.

Précision	Rappel	F-mesure
71.47	76.61	73.95

Table 1 – Résultats de la méthode naïve (%)

Sur le corpus de test proposé, les résultats sont bien plus faibles que les résultats annoncés pour la méthode suivante (90%), ce qui était attendu. Cependant on peut considérer que ces performances ne sont pas ridicules étant donné la simplicité du modèle employé. Les principaux défauts ici sont qu'un caractère peut tout à fait être au début, au milieu ou à la fin d'un token et que certains caractères dans le corpus de test ne sont pas connus (18 sur 607 vus).

(Génération du modèle : `python mySegmenter_naive.py knbc-train.xml knbc-test.xml <outputfile>`)

3 Méthode de C. P. Papageorgiou

La méthode de l'article proposé utilise un HMM à deux états et des observations sur les bigrammes. Ainsi un bigramme peut soit être compris dans un token, soit être la frontière entre deux tokens. Cette méthode utilise le fait qu'un bigramme est plus représentatif d'une telle coupure qu'un caractère seul, et que le nombre de bigrammes apparaissant dans le japonais est bien inférieur au nombre de combinaisons possibles parmi tous les couples de caractères.

Précision	Rappel	F-mesure
83.92	96.60	89.81

Table 2 – Résultats de la méthode de C. P. Papageorgiou (%)

Le point faible de cette méthode est que l'on doit traiter de nombreux bigrammes non vus lors de l'apprentissage.

(Génération du modèle : `python mySegmenter_baseline.py knbc-train.xml knbc-test.xml <outputfile>`)

4 Modifications

4.1 Lexique de consonnes sourdes

Il existe en japonais des syllabes appelées consonnes sourdes et formées de deux caractères, créant des phonèmes qui n'existent pas avec un seul caractère. Ces syllabes peuvent être intéressantes pour la segmentation car elles forment des bigrammes fixes, dans le sens où un des caractères de la consonne sourde n'existe que dans ces bigrammes.
(*source : http ://fr.wikipedia.org/wiki/Japonais*).

L'idée est donc d'attribuer une probabilité de 1 pour l'état "continue le token" lorsque les deux caractères analysés forment une consonne sourde. Dans le cas contraire, le modèle est le même que dans la méthode précédente.

Précision	Rappel	F-mesure
84.00	96.03	89.62

Table 3 – Résultats de la méthode utilisant les consonnes sourdes (%)

Les résultats de cette modification ne sont pas concluants : on observe seulement un léger changement dans le compromis rappel-précision puisque vérifier la présence de consonne sourde rend la segmentation plus précise, au prix d'un rappel diminué.

4.2 Pondération pour les bigrammes inconnus

Un des points faibles de la méthode *baseline* est le grand nombre de bigrammes non vus dans le corpus. En effet sur 9710 occurrences de bigrammes en tout (2586 bigrammes uniques) dans le corpus test, on observe 4005 bigrammes inconnus du modèle (2155 bigrammes inconnus uniques). La technique utilisée jusqu'ici dans le cas où l'on a à classer un bigramme non vu est tout simplement de considérer un état par défaut (état "coupure entre tokens" dans le script utilisé). On propose ici de pallier ce problème en s'aidant des observations connues *similaires*.

On utilise plusieurs implémentation pour représenter cette notion de similarité. Ainsi on tente de déterminer l'état le plus probable pour un bigramme non vu B_{non-vu} en comptant le nombre de bigrammes connus B_{vu} parmi les observations de chacun des états tels que :

– le deuxième caractère de B_{non-vu} est le même que le deuxième caractère de B_{vu}
– le premier caractère de B_{non-vu} est le même que le premier caractère de B_{vu}
– un des deux caractères de B_{non-vu} est le même qu'un des deux caractères de B_{vu}

A partir des comptes obtenus pour chacun des états on calcule le rapport du minimum sur le maximum des deux valeurs, qui représente sur une échelle de 0 à 1 la complexité de la décision. On détermine un seuil s (on teste plusieurs valeurs) au delà duquel le rapport est trop équillibré et demande donc de trancher comme précédemment c'est à dire en attribuant un état par défaut. En dessous de ce seuil, l'état est déterminé par les comptes calculés.

	Précision	Rappel	F-mesure
$s = 0.5$	78.77	95.98	86.53
$s = 0.3$	82.92	95.57	89.23
$s = 0.1$	**83.97**	96.60	**89.85**

TABLE 4 – Résultats de la méthode utilisant la similarité : $B_{non-vu}[1] = B_{vu}[1]$

	Précision	Rappel	F-mesure
$s = 0.5$	78.05	96.61	86.34
$s = 0.3$	81.43	96.61	88.38
$s = 0.1$	83.74	96.61	89.72

TABLE 5 – Résultats de la méthode utilisant la similarité : $B_{non-vu}[0] = B_{vu}[0]$

	Précision	Rappel	F-mesure
$s = 0.5$	78.20	**96.64**	86.44
$s = 0.3$	82.37	96.61	88.94
$s = 0.1$	83.77	96.60	89.73

TABLE 6 – Résultats de la méthode utilisant la similarité : $B_{non-vu}[0,1] = B_{vu}[0,1]$

(Génération du modèle : `python mySegmenter_unseen.py knbc-train.xml knbc-test.xml <outputfile>`)

4.3 Blocs Unicode des différents alphabets

Le japonais utilise plusieurs alphabets et les lettres qui les composent sont réparties dans différents blocs Unicode. Les symboles vus sont classés parmi les blocs Hiragana, Katakana (alphabets japonais) et les blocs CJK Unified Ideographs, CJK Symbols and Punctuation (alphabet chinois).

On propose ici d'enrichir le modèle en comptant les changements d'états en fonction des changements d'alphabets, c'est à dire les occurrences de toutes les paires possibles de blocs Unicode dans un bigramme, pour chaque état.

Précision	Rappel	F-mesure
88.44	91.13	89.77

TABLE 7 – Résultats de la méthode utilisant les blocs Unicode

On peut remarquer que cette méthode change radicalement l'équillibre entre le rappel et la précision, car ici le système est beaucoup plus précis, au prix d'un rappel moindre. C'est en réalité un bon signe, dans le sens où obtenir plus de précision est ici plus significatif qu'obtenir plus de rappel – en effet en séparant tous les tokens on obtiendrait un rappel de 100%. Pour autant, le score de F-mesure reste le même.

(Génération du modèle : `python mySegmenter_unicode.py knbc-train.xml knbc-test.xml <outputfile>`)

4.4 Combinaison de méthodes

Afin d'obtenir de meilleurs résultats, on peut essayer de tirer le meilleur de chaque méthode en les combinant. Puisque le lexique de consonnes sourdes n'a pas apporté un changement significatif, on propose de ne combiner que la méthode par blocs Unicode et celle utilisant la pondération pour les bigrammes non vus.

On conserve donc le fonctionnement de la méthode des blocs Unicode, en rajoutant le calcul de pondération lorsque le bigramme n'a pas été rencontré dans le corpus d'entraînement. On utilise la version de la méthode par pondération donnant les meilleurs résultats, soit la similarité $B_{non-vu}[1] = B_{vu}[1]$ et un seuil $s = 0.1$.

Précision	Rappel	F-mesure
88.73	91.32	90.01

TABLE 8 – Résultats de la combinaison : blocs Unicode - similarité : $B_{non-vu}[1] = B_{vu}[1]$

La combinaison fonctionne, dans le sens où les deux calculs ne s'annulent pas et il en résulte un rappel et une précision plus élevés. Cependant cette hausse est très faible et ne permet pas d'émettre une conclusion franche sur l'utilité de l'une ou l'autre des méthodes.

(Génération du modèle : `python mySegmenter_combi.py knbc-train.xml knbc-test.xml <outputfile>`)

5 Autres travaux

L'article de C. P. Papageorgiou a été publié en 1994 et depuis de nombreux travaux ont été réalisés sur ce sujet. Cependant d'après ce que l'on a pu lire, les approches non supervisées comme celle présentée ici ou celle de R. K. Ando et L. Lee (*Mostly-Unsupervised Statistical Segmentation of Japanese : Applications to Kanji, 2000*) sont plutôt remplacées par des méthodes utilisant des patrons lexicaux (*T. Nakagawa : Chinese and Japanese word segmentation using word-level and character-level information [2004], Choi et al. : Word segmentation standard in Chinese, Japanese and Korean [2009]*).

L'inconvénient de ces nouvelles méthodes est qu'elles nécessitent des lexiques grammaticaux qui peuvent être volumineux, et surtout peuvent souffrir des erreurs d'étiquetage.

6 Conclusion

Que peut on retenir de ces expériences ? Tout d'abord qu'une méthode non supervisée relativement simple peut apporter des résultats corrects pour cette tâche de segmentation du japonais. Notamment, il paraît inutile de vérifier les bigrammes "fixes" (ici, les consonnes sourdes) car celles-ci sont déjà bien segmentées. Il existe peut être une méthode de pondération des bigrammes inconnus qui fonctionne mieux que celle présentée, mais cela requiert probablement

quelques connaissances en langue japonaise. Une méthode par n-grammes (à la manière de Ando et Lee,2000) a été testée ici mais celle-ci n'a donné que des résultats médiocres (F-mesure de l'ordre de 60%). Enfin, la méthode la plus prometteuse d'après ces expériences est de pondérer les probabilités d'observations avec les changements d'alphabet, cependant améliorer cette version demanderait encore une fois quelques bases en langue nippone.