Traitement automatique des langages: Translittération de la langue Japonaise.

Taillandier Valentin

Sommaire

Sommaire

Quatre alphabets

Kanji	猫, 寿司, 神, 髪, 紙
Hiragana	ねこ, すし, かみ
Katakana	ネコ, スシ, カミ

Une phrase en japonais

コーヒーを飲みたいですか？

Quatre alphabets

Kanji	猫, 寿司, 神, 髪, 紙
Hiragana	ねこ, すし, かみ
Katakana	ネコ, スシ, カミ

Une phrase en japonais

コーヒーを飲みたいですか？

Rōmaji	neko, sushi, kami

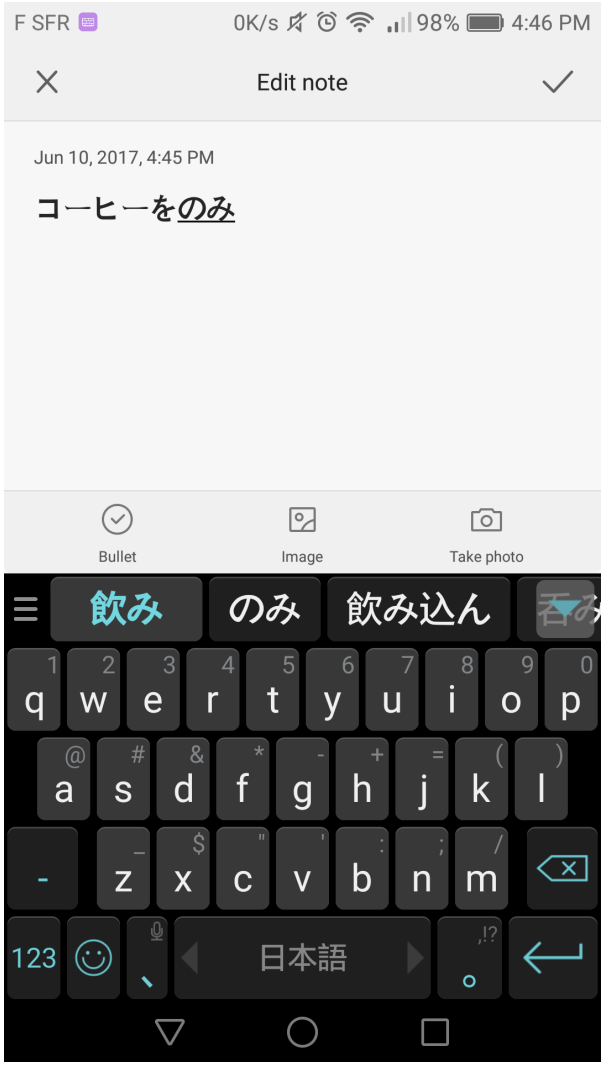
La méthode *Hepburn-shiki (ANSI Z39.11-1972)*

- Introduite en 1887 par James Curtis Hepburn.
- Nous parlerons de *Rōmaji* pour décrire les mots romanisés et de *Kanji* pour décrire les idéogrammes originaux.
- La romanisation du japonais a rendu possible la création de logiciels d'aide à la saisie *(IME, input method editor)* et donc l'utilisation des claviers d'ordinateurs par les Japonais.

Un problème de phonétique

- La romanisation du Japonais ne tient compte que de la phonétique.
- Pour un *Rōmaji*, nous pouvons associer plusieurs *Kanji* possibles.
- Une simple concordance (dictionnaire) ne suffit pas, il faut prendre en compte le sens de la phrase dans laquelle le mot se trouve.

Un logiciel d'aide à la saisie



Comment prendre en compte le sens de la phrase ?

Une chaîne de caractères

$$w_0\, w_1\, w_2 \ldots w_{n-1}\, w_n$$

- On veut convertir en *kanji* le mot w_n initialement écrit en *rōmaji*.
- On suppose que l'utilisateur a déjà saisi les mots précédents w_n.
- On admet alors que l'on peut convertir w_n à partir de cette information.

Finalité du projet

Une fonction

Soit R et K les ensembles des mots respectivement écrits en *rōmaji* et *kanji* :

$$\begin{array}{cccc} f & : & R \times K^\star & \to K \\ & & (w_n, w_0 ... w_{n-1}) & \mapsto \quad {w_n}' \end{array}$$

Sommaire

Un automate

Un HMM est défini par un quintuplet (S, Σ, T, G, π), où :

- S est un ensemble de N états,
- Σ est un alphabet de M symboles,
- $T = S \times S \rightarrow [0, 1]$ est la matrice de transition, indiquant les probabilités $P(s \rightarrow s')$,
- $G = S \times \Sigma \rightarrow [0, 1]$ est la matrice de génération, indiquant les probabilités $P(w|s)$,
- $\pi : S \rightarrow [0, 1]$ est un vecteur de probabilités initiales de visites.

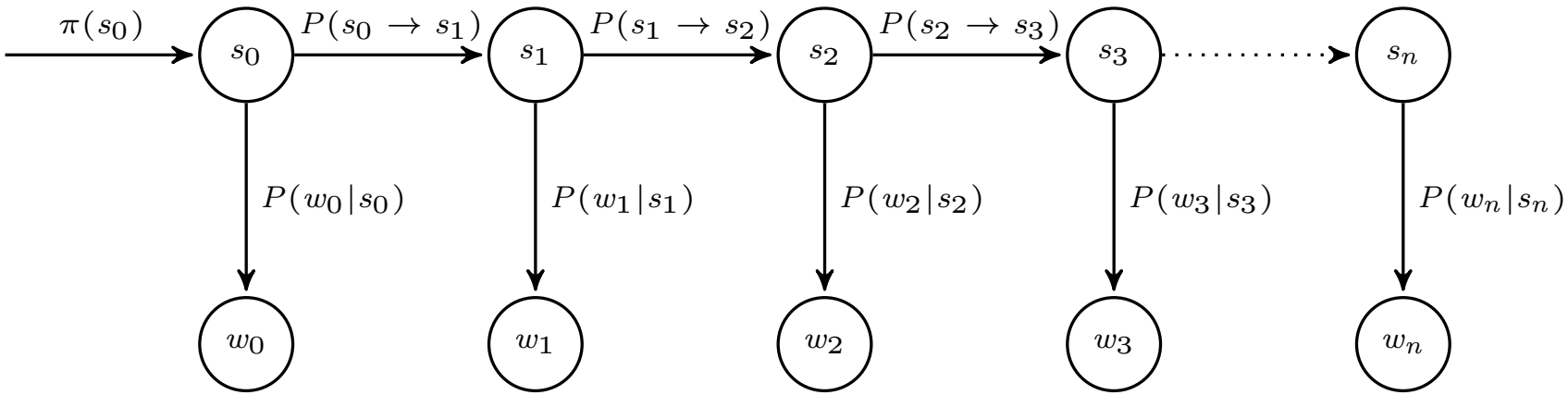
Génération d'une phrase

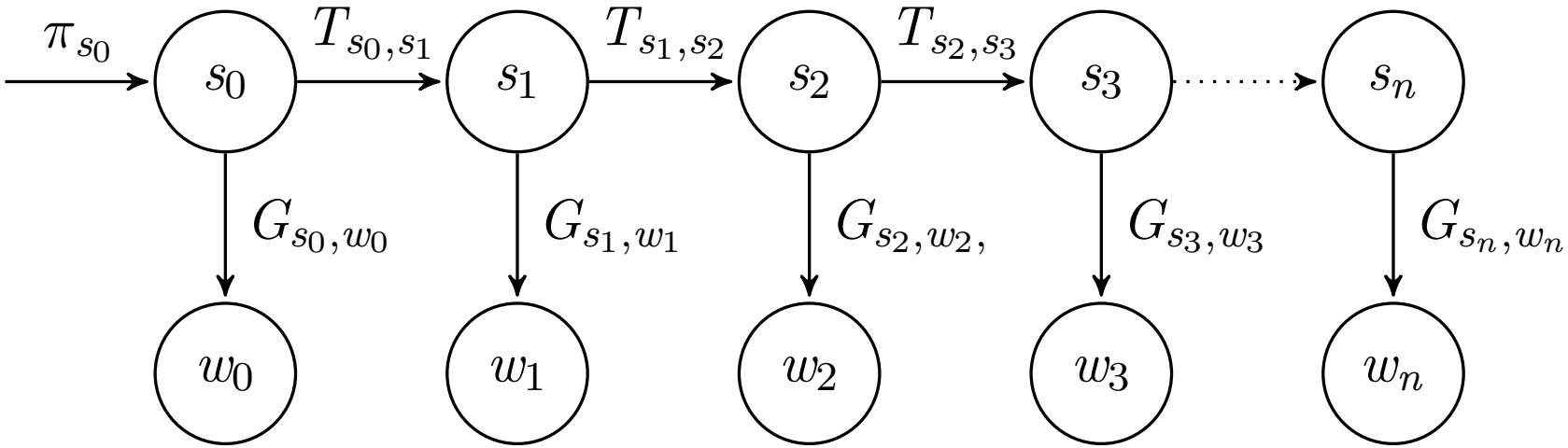
- La probabilité de générer la phrase $W = w_0 ... w_n$ en passant par le chemin $C = s_0 ... s_n$ est :

$$P(W|C) = \pi(s_0) P(w_0|s_0) \prod_{i=1}^{n} P(s_{i-1} \rightarrow s_i) P(w_i|s_i)$$

- La probabilité de générer la phrase $W = w_0 ... w_n$ est :

$$P(W) = \sum_{C} P(W|C)$$


$\pi(s_0)$
s_0
$P(s_0 \rightarrow s_1)$
s_1
$P(s_1 \rightarrow s_2)$
s_2
$P(s_2 \rightarrow s_3)$
s_3
s_n
$P(w_0|s_0)$
$P(w_1|s_1)$
$P(w_2|s_2)$
$P(w_3|s_3)$
$P(w_n|s_n)$
w_0
w_1
w_2
w_3
w_n


π_{s_0}
s_0
T_{s_0,s_1}
s_1
T_{s_1,s_2}
s_2
T_{s_2,s_3}
s_3
s_n
G_{s_0,w_0}
G_{s_1,w_1}
$G_{s_2,w_2,}$
G_{s_3,w_3}
G_{s_n,w_n}
w_0
w_1
w_2
w_3
w_n

Sommaire

Algorithme Forward-backward

- On suppose que l'on veut calculer la probabilité de générer la phrase $W = w_0 ... w_n$ via le HMM.
- Une idée naïve consiste à utiliser directement la formule précédente : $P(W) = \sum_C P(W|C)$.
- Le nombre de chemins possibles est en $\mathcal{O}(N^{n+1})$ et pour chaque chemin on a $\mathcal{O}(n+1)$ calculs.

Algorithme Forward

- Soit $\alpha_t(s) = P(w_0 ... w_t, s_t = s)$.
- On calcule $\alpha_t(s)$ par induction :

Procédé

- Initialisation :

$$\alpha_0(s) = \pi(s).P(w_0|s)$$

- Induction :

$$\alpha_t(s) = \left(\sum_{s' \in S} \alpha_{t-1}(s').P(s' \rightarrow s) \right) P(w_t|s)$$

- L'algorithme Forward retourne la matrice $(\alpha_t(s))_{t \in [\![0,n]\!], s \in S}$.

Algorithme Backward

- Soit $\beta_t(s) = P(w_{t+1}...w_n, s_t = s)$.
- L'algorithme Backward retourne la matrice $(\beta_t(s))_{t \in [\![0,n]\!], s \in S}$.

Retour sur le problème

- Pour calculer $P(W = w_0 ... w_n)$ on utilise la formule :

$$P(W = w_0 ... w_n) = \sum_{s \in S} \alpha_n(s)$$

- Ou bien :

$$P(W = w_0 ... w_n) = \sum_{s \in S} \pi(s)\beta_0(s)$$

- Le calcul se fait en $\mathcal{O}((n+1)N^2)$

Algorithme de Baum-Welch

- Il s'agit d'un algorithme d'entrainement qui permet de réestimer (T, G, π) à partir d'un corpus $\mathfrak{W}$.
- Supposons que $\mathfrak{W}$ ne comporte qu'une seule phrase W.
- On définit :

$$\begin{aligned}\xi_t(s, s') &= \frac{P(s_t = s, s_{t+1} = s', W)}{P(W)} \\ &= \frac{\alpha_t(s) P(s \to s') P(w_{t+1}|s') \beta_{t+1}(s')}{P(W)}\end{aligned}$$

et

$$\gamma_t(s) = \sum_{s' \in S} \xi_t(s, s')$$

Réestimation

$$\pi'(s) = \gamma_0(s)$$

$$P'(s \to s') = \frac{\sum_{t=0}^{n-1} \xi_t(s, s')}{\sum_{t=0}^{n-1} \gamma_t(s)}$$

$$P'(w|s) = \frac{\sum_{t=0, w_t=w}^{n} \gamma_t(s)}{\sum_{t=0}^{n} \gamma_t(s)}$$

- Si le corpus comporte plusieurs phrases, alors on somme numérateurs et dénominateurs pour chaque phrase.
- On réestime itérativement les matrices (T, G, π) .

Convergence

- Baum a démontré qu'à chaque itération $H' = H$ ou alors $P(\mathfrak{W}|H') > P(\mathfrak{W}|H)$.
- Dans tout les cas, l'algorithme converge vers un optimum local H_∞.
- L'automate H_∞ dépend donc de H_0.

Algorithme de Brown

- Cet algorithme permet de classer les mots d'un corpus et de créer une relation d'équivalence.
- L'idée est de placer des mots similaires dans une même catégorie (cluster).
- Cela nous permettra de construire H_0.

Algorithme de Brown

- Soit C un clustering de K.
- Pour tout w de K, on note $c(w)$ le cluster dans lequel se trouve w.
- On définit la qualité $Q(C)$ de la façon suivant :

$$Q(C) = \frac{1}{n} \log P(\mathfrak{W}|C)$$

avec :

$$P(\mathfrak{W}|C) = \prod_{W \in \mathfrak{W}} P(w_i|c(w_i)) \prod_{i=1}^{|W|} P(c(w_{i-1}) \rightarrow c(w_i)).P(w_i|c(w_i))$$

Algorithme de Brown

- Soit n le nombre de mots dans $\mathfrak{W}$ compté avec répétitions.
- Soit $n(c)$ le nombre de fois qu'un mot de c apparait dans le corpus.
- Soit $n(c, c')$ le nombre de fois que la transition $c \rightarrow c'$ a lieu.
- Après simplification dans la formule précédente, on a :

$$Q(C) = \sum_{c,c'} \frac{n(c,c')}{n} \log \frac{n(c,c')n}{n(c)n(c')} + k$$

- Le but est de trouver C qui maximise $Q(C)$.

- On définit pour tout couple de clusters (c, c') :

$$w(c, c') = \begin{cases} \frac{n(c, c')}{n} \log \frac{n(c, c')n}{n(c)n(c')} + \frac{n(c', c)}{n} \log \frac{n(c', c)n}{n(c')n(c)} & si\ c \neq c' \\ \frac{n(c, c)}{n} \log \frac{n(c, c)n}{n(c)n(c)} & si\ c = c' \end{cases}$$

- Avec $C' = C - \{c, c'\} + \{c \cup c'\}$, on définit :

$$\forall c \neq c' : L(c, c') = \sum_{d \in C'} w(c \cup c', d) - \sum_{d \in C} (w(c, d) + w(c', d))$$

Algorithme

Initialisation Chaque mot appartient à son propre cluster.
La matrice des w et des L est calculée pour chaque couple de clusters.

Itérations À chaque itération, le couple (c, c') ayant le plus grand $L(c, c')$ est fusionné.
Les matrices des w et des L sont recalculées pour le nouveau clustering obtenu.

Compléxité

- À l'initialisation, chaque $L(c, c')$ est calculé en $\mathcal{O}(k)$:
  - En sommant pour chaque pair, on a une complexité en $\mathcal{O}(k^3)$.
- À chaque itération, on cherche le couple qui maximise $L(c, c')$, cela fait une complexité en $\mathcal{O}(k^2)$.
- Après la fusion de deux clusters, il faut mettre à jour la matrice des L :
  - Lorsque c ou c' est impliqué dans la fusion, $L(c, c')$ est recalculé complètement en $\mathcal{O}(k)$, soit une complexité en $\mathcal{O}(k^2)$.
  - Les autres $L(c, c')$ sont mis à jour en $\mathcal{O}(1)$ à partir de leur valeurs précédentes, soit une complexité en $\mathcal{O}(k^2)$.
- Il y au plus k fusions, donc au plus k itérations, cela fait un algorithme en $\mathcal{O}(k^3)$ (Contre $\mathcal{O}(k^5)$ sans la gestion d'une matrice L).

Sommaire

Le corpus

- Il s'agit d'un dump de Wikipedia/jp au format xml.
- J'ai codé une fonction Python permettant d'extraire le premier million de phrases.
- Pour un poids de 247 Mo, et 200 000 mots différents.

Sommaire

- J'ai implémenté les algorithmes en créant deux classes, Corpus et Hmm.
- À chaque instant, on peut sauvegarder le travail déjà effectué.
- Le package réalisé est pré-compilé en C avec Cython.

Traitement du corpus

- A nécessité plus de 24h de calculs pour 500 clusters.
- Les clusters sont très bien construits.

Traitement du HMM

- Un entrainement de Baum-Welch demande 1 jour et 6 heures.
- Après une dizaine d'entrainements, le HMM semble avoir convergé.
- La reconnaissance du corpus par le HMM est meilleure avec un automate issue de l'algorithme de Brown.

Sommaire

Principe

- Il s'agit d'une fonction prenant un mot w_n en $Rōmaji$ et $w_0 ... w_{n-1}$, les mots qui lui précède en $Kanji$.
- Les traductions possibles de w_n en $Kanji$ sont extraites via un dictionnaire.
- Pour chaque traduction possible, on obtient une phrase.
- La probabilité de génération de chaque phrase est étudiée à partir du HMM.
- Enfin, les traductions possibles sont classées par vraisemblance décroissante.
- Ce classement est retourné par la fonction et forme une liste de suggestions pour l'utilisateur.

Résultats

- On compare les résultats de la fonction précédente à celle d'une fonction qui renvoie des résultats classés aléatoirement.
- Le logiciel d'aide à la saisie fonctionne bien sur des exemples tirés du corpus.
- Les résultats sont aussi corrects sur d'autres exemples simples.

Sommaire

Bibliographie

1 J. C. HEPBURN, A.M., M.D. : A Japanese and English dictionnary (1888) : https ://archive.org/details/japaneseenglishe00hepbuoft

2 A. A. Markov : An Example of Statistical Investigation of the Text Eugene Onegin. Concerning the Connection of Samples in Chain (1913) : https ://doi.org/10.1017/S0269889706001074

3 Leonard E. Baum et al. : A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains (1970) : http ://projecteuclid.org/euclid.aoms/1177697196

4 Percy Liang : Semi-Supervised Learning for Natural Language (2005) :https ://wwwcs.stanford.edu/ pliang/papers/meng-thesis.pdf

5 Peter F. Brown et al. : Class-Based n-gram Models of Natural Language (1992) : http ://acl-arc.comp.nus.edu.sg/archives/acl-arc-090501d3/data/pdf/anthology-PDF/J/J92/J92-4003.pdf

Remerciements

- M. Damien Nouvel, maître de conférences à l'INALCO, pour ses nombreux conseils.
- Mlle. Chiaki Sakaguchi, institutrice du Japonais à l'Université de Bourgogne, pour ses conseils linguistiques.