Étiquetage d'entités nommées avec HMM

PSTAL - TP 2 - Carlos Ramisch

L'objectif de ce TP est de développer et évaluer un système reconnaissance d'entités nommées avec un modèle de Markov caché (*hidden markov model*, dorénavant HMM). Il s'agit d'un système probabiliste et non pas d'un réseau de neurones. L'estimation de paramètres se fait à partir des comptes dans le corpus, sans apprentissage itératif par rétro-propagation. Nous n'utiliserons donc **pas** la bibliothèque `pytorch` cette fois-ci.

1 La reconnaissance d'entités nommées

Les entités nommées (dorénavant EN) sont des expressions linguistiques désignant des entités **uniques** dans le monde. Les exemples prototypiques sont les noms de personnes (célèbres ou non), de lieux, et d'organisations. La tâche consiste à "souligner" des mots adjacents dans le texte, et à leur attribuer une catégorie parmi :

- `PERS` – personne, p.ex. *Jul*, *Amy Winehouse*, *Jacques Chirac*
- `LOC` – lieu, p.ex. *La Belle de Mai*, *Chili*, *Ajaccio*
- `ORG` – organisation, p.ex. *Olympique de Marseille*, *ONU*, *Parti Socialiste*
- `PROD` – production (culturelle, artistique…), p.ex. *Wikipédia*, *Le Canard Enchaîné*, *Le Petit Prince*
- `EVE` – événement nommé, p.ex. *JO de Paris 2024*, *guerre d'Indochine*, *Coupe d'Europe*

Cette tâche possède trois défis principaux. Premièrement, l'**ambiguïté** concerne les homonymes (*Saint Laurent* est un fleuve et aussi une marque, `LOC` et `PROD`) et les métonymies (*Le Monde* est un journal écrit et aussi une rédaction, `PROD` et `ORG`). Deuxièmement, certaines EN possèdent des **variantes** dues à des acronymes ou à des ellipses (*États Unis*, *États Unis d'Amérique*, *USA*, *US*, *Amérique*). Troisièmement, il peut y avoir des **imbrications** (*JO de Paris* est un `EVE` qui contient la `LOC` *Paris*). Dans ce TP, le système modélise implicitement l'ambiguïté via des probabilités. Les variantes peuvent être traitées en extension (voir § 8). Les imbrications ne sont pas traitées car le corpus a été simplifié pour ne pas en contenir. [1]

2 Préparation des données : encodage BIO

Dans le corpus, les EN sont codées dans la 13ème (dernière) colonne, nommée `parseme:ne`, au format `parseme`. Les mots n'appartenant à aucune EN sont annotés avec une astérisque (`*`). Les mots appartenant à une EN sont numérotés avec un indice, et le premier mot contient l'étiquette de catégorie. L'exemple ci-dessous contient les EN *Le Petit Prince* (`PROD`), *Saint-Exupéry* (`PERS`) et *École Jules-Romains* (`ORG`) :

	Le	*Petit*	*Prince*	*de*	*Saint-Exupéry*	*est*	*entré*	*à*	*l'*	*École*	*Jules-Romains*
`parseme`	`1:PROD`	`1`	`1`	`*`	`2:PERS`	`*`	`*`	`*`	`*`	`3:ORG`	`3`
`BIO`	`B-PROD`	`I-PROD`	`I-PROD`	`O`	`B-PERS`	`O`	`O`	`O`	`O`	`B-ORG`	`I-ORG`

Le système développé affectera une étiquette par mot, comme pour les parties du discours (TP 1). C'est l'encodage des entrées qui permettra à cet étiqueteur d'effectuer une tâche de segmentation. Nous utiliserons l'encodage dit "BIO" pour *begin*, *inside* et *outside*, montré dans la dernière ligne de l'exemple ci-dessus. Les mots n'appartenant à aucune EN prennent l'étiquette `O` (*outside*). Les mots appartenant à une EN prennent l'étiquette `I-CAT` (*inside*), où `CAT` est la catégorie de l'entité (`PERS`, `LOC`, etc.). Cependant, si le mot est le premier (ou le seul) de l'EN, il prend l'étiquette `B-CAT` (*begin*). Nous vous recommandons d'utiliser la fonction `to_bio()` de la bibliothèque fournie `conllulib.py`. Elle prend en entrée une `TokenList` renvoyée par `CoNLLUReader.readConllu()` et renvoie la liste d'étiquettes BIO correspondante.

3 Entraînement : estimation simple

Le modèle est un HMM avec trois jeux de paramètres : la matrice d'émission E, la matrice de transition T, et les probabilités initiales π. Ces paramètres sont estimés en fonction des nombres d'occurrences des mots

1. Détails sur ces phénomènes sur le site du corpus Sequoia : https://deep-sequoia.inria.fr/, ensuite *PARSEME-FR guide*.

w_i (colonne `form`) et étiquettes t_i (colonne `parseme:ne` convertie en BIO). Votre programme devra parcourir le corpus encodé en BIO une seule fois et en extraire :

- $c(w_j, t_i)$ nombre d'occurrences de chaque paire mot-étiquette $\langle w_j, t_i \rangle$, p.ex. $c(Le, \texttt{B-PROD}) \to 1$
- $c(t_i, t_j)$ nombre d'occurrences adjacentes de chaque paire d'étiquettes $\langle t_i, t_j \rangle$, p.ex. : $c(\texttt{O}, \texttt{O}) \to 3$
- $c(t_i)$ nombre d'occurrences de chaque étiquette simple $\langle t_i \rangle$, p.ex. $c(\texttt{I-PROD}) \to 2$
- $c(\texttt{<s>}, t_i)$ nombre d'occurrences de chaque étiquette en début de phrase, p.ex. $c(\texttt{<s>}, \texttt{I-PROD}) \to 0$

Nous conseillons l'utilisation de dictionnaires plutôt que de matrices étant donné la nature creuse des comptes. Vous pouvez utiliser les structures `defaultdict` et `Counter` de la bibliothèque `collections`. Par exemple, $c(w_j, t_i)$ peut être un `defaultdict` qui associe chaque w_j à un `Counter` comptant tous les t_i vus avec w_j.

Une fois ces comptes obtenus, on pourrait estimer les paramètres du HMM comme suit :

$$E(t_i, w_j) = \frac{c(w_j, t_i)}{c(t_i)} \qquad T(t_i, t_j) = \frac{c(t_i, t_j)}{c(t_i)} \qquad \pi(t_i) = \frac{c(\texttt{<s>}, t_i)}{S}$$

Où $S = \sum_{t_i} c(\texttt{<s>}, t_i)$ est le nombre total de phrases du corpus. Cependant, pour éviter les problèmes de *underflow*, nous allons stocker plutôt les "moins log-probabilités", calculés comme suit :

$$\begin{aligned} -\log E(t_i, w_j) &= \log c(t_i) - \log c(w_j, t_i) \\ -\log T(t_i, t_j) &= \log c(t_i) - \log c(t_i, t_j) \\ -\log \pi(t_i) &= \log S - \log c(\texttt{<s>}, t_i) \end{aligned}$$

Notez que certains comptes peuvent être nuls, notamment lors de la prédiction. Pour éviter les erreurs, utilisez la fonction fournie `log_cap` qui renvoie `-9999.0` si l'argument vaut zéro, et le logarithme de base 10 sinon.

Pour sauvegarder les paramètres, nous utiliserons `pickle.dump` (sous-jacent et équivalent à `torch.save`). N'oubliez pas d'importer `pickle` et, comme pour les vocabulaires du TP1, convertir les `defaultdict` en `dict`.

4 Prédiction : algorithme de Viterbi

Les paramètres E, T et π sont chargés avec `pickle.load`. Vous lirez ensuite le corpus `.dev` phrase à phrase. Supposons une phase à étiqueter de longueur n, et un vocabulaire $V_t = \{t_1 \dots t_N\}$ contenant $|V_t| = N$ étiquettes différentes. Nous allons remplir un tableau $\delta(t_j, w_k) \in \mathbb{R}^{N \times n}$, où les lignes sont indexées par les étiquettes possibles t_j et les colonnes par les mots w_k de la phrase, avec $k = 1 \dots n$:

$$\begin{aligned} \text{Initialisation (colonne 1)} \quad & \delta(t_j, w_1) = -\log \pi(t_j) - \log E(t_j, w_1) && 1 \le j \le N \\ \text{Étape récursive} \quad & \delta(t_j, w_k) = \min_{1 \le i \le N} [\delta(t_i, w_{k-1}) - \log T(t_i, t_j)] - \log E(t_j, w_k) && 2 \le k \le n, 1 \le j \le N \end{aligned}$$

N'oubliez pas de stocker dans un tableau analogue $\psi(t_j, w_k)$ l'arg min correspondant à la valeur de i retenue pour le min ci-dessus. Vous retrouverez l'étiquetage optimal en parcourant la matrice ψ à l'envers. La dernière étiquette est $\hat{t}_n = \arg\min_{1 \le i \le N} \delta(t_i, w_n)$, et ensuite $\hat{t}_k = \psi(\hat{t}_{k+1}, w_{k+1})$ jusqu'à la première colonne.

Voici quelques astuces pour vous aider à implémenter l'algorithme.

- Initialisez $\delta(t_j, w_k)$ avec `Util.PSEUDOINF` à l'aide de la fonction `np.full`
- Attention : manipuler des "moins log-probabilités" implique de minimiser, et non pas maximiser !
- En pratique, il faut explicitement accéder aux probabilités lissées pour les OOV (voir § 5).

À la fin, pensez à reconvertir les étiquettes BIO vers le format `parseme` avec `from_bio` de `conllulib`. Cette fonction essaie de rattraper les problèmes, si par hasard les prédictions BIO sont incohérentes (voir documentation de la fonction). Vous verrez des "warnings" dans ce cas. Bon à savoir : si l'apprentissage avec lissage et la prédiction avec Viterbi n'ont pas de bug, il ne devrait pas y avoir de prédiction incohérente.

5 Entraînement : lissage

Les probabilités estimés de cette façon ne peuvent pas traiter des phrases contenant des mots hors vocabulaire (OOV), ce qui arrive souvent avec les EN. Pour contourner cette limitation, nous allons ajouter un facteur

de lissage $\alpha = 0.1$ à chaque nombre d'occurrences. Cela transforme le calcul des "moins log-probabilités" :

$$\begin{aligned} -\log E(t_i, w_j) &= \log\left[c(t_i) + |V_w|\alpha\right] - \log\left[c(w_j, t_i) + \alpha\right] \\ -\log T(t_i, t_j) &= \log\left[c(t_i) + |V_t|\alpha\right] - \log\left[c(t_i, t_j) + \alpha\right] \\ -\log \pi(t_i) &= \log(S + |V_t|\alpha) - \log\left[c(\texttt{<s>}, t_i) + \alpha\right] \end{aligned}$$

Ce type de lissage ne paraît pas très compliqué, à première vue, mais il cache un certain nombre de détails à régler. Premièrement, remarquez que V_w est le vocabulaire des mots et V_t est le vocabulaire des étiquettes : la taille de ces vocabulaires doit être obtenue (p.ex. $|V_w|$ est le nombre d'entrées dans le dictionnaire $c(w_j, t_i)$). Deuxièmement, quand $c(w_j, t_i) = 0$ mais $c(t_i) \neq 0$ la valeur de $-\log E(t_i, w_j)$ dépend de $c(t_i)$. Nous vous conseillons de stocker dans une clé spéciale de E (p.ex. `"<<<OOV>>>"`) les valeurs $\log\left[c(t_i) + |V_w|\alpha\right] - \log\left[0 + \alpha\right]$ de toutes les étiquettes t_i possibles, à utiliser dans Viterbi pour les OOV. De manière similaire, pensez à calculer et stocker dans T les probabilités de transition pour les paires dont $c(t_i, t_j) = 0$, et pareil pour $\pi(t_i)$.

6 Évaluation

L'évaluation des prédictions sera effectuée par le script fourni `lib/accuracy.py`. Ce script permet d'obtenir non seulement l'exactitude des étiquettes, mais aussi la précision, rappel et F-score de chaque catégorie d'EN, ainsi que la macro-moyenne (`macro-average`), la micro-moyenne (`Exact-nocat`) et la métrique floue (`Fuzzy-nocat`). Rappelez vous que pour cette tâche, l'*accuracy* n'est pas pertinente du tout : il faut l'ignorer ! Regardez plutôt les trois derniers F-scores. De plus, vous remarquerez que, même avec lissage, votre système ne dépassera pas les 60% de macro F-score. Il s'agit d'une tâche difficile, et le corpus d'entraînement n'est pas très grand. Vous pouvez essayer de faire mieux avec les extensions (§ 8).

7 Travail à effectuer

Vous devez, comme pour le TP précédent, écrire deux scripts différents : `train_ner.py` pour l'entraînement du modèle (Section 3), et `predict_ner.py` pour la prédiction des EN (Section 4). Nous vous recommandons de commencer par un système sans lissage, développé de bout en bout, puis avec l'ajout du lissage pour obtenir des performances correctes.

8 Extensions

Pour les deux extensions ci-dessous, il convient d'optimiser votre programme. Pour cela, dès l'entraînement, transformez les mots et les étiquettes en indices entiers contenus dans des vocabulaires V_w et V_t (`defaultdict`) comme dans le TP précédent.

Modèle d'émission neuronal Vous devez remplacer $E(t_i, w_j)$ par un modèle neuronal du type RNN, identique à celui utilisé pour la prédiction des POS. Ce modèle devra sera entraîné à part. Lors de la prédiction, on doit explicitement appeler le softmax et le $-\log$, puis appliquer Viterbi sur la $-\log$-distribution de probabilités résultante. Les paramètres π et T restent tels quels, estimés sur les comptes du corpus. Un tel modèle devrait mieux prendre en compte la variabilité, et donc obtenir de meilleurs résultats. De plus, vous pouvez y ajouter des traits tels que la casse (le mot commence par une majuscule) et la présence de traits d'union, par exemple.

Encodages et lissage Vous devez essayer d'améliorer le modèle en faisant varier (a) le type d'encodage des entrées, (b) le paramétrage du lissage, et (c) la forme des mots en entrée (bruts ou tout en minuscules). Sur le corpus de développement, testez plusieurs types d'encodage parmi IO, BIO, BIOES, avec la catégorie attachée au premier mot, ou à tous les mots. Modifiez le lissage pour avoir une valeur de lissage par paramètre α_E, α_T et α_π, et testez plusieurs valeurs pour chaque α, par exemple, allant de 0 à 1. Essayez de mettre les mots en entée tout en minuscules. Une fois que vous avez trouvé la meilleure combinaison d'encodage, lissage et casse sur le `dev` (p.ex. avec un *grid search*), vous pouvez donner les résultats sur le `test`. Analysez les erreurs restantes : peut-on identifier des caractéristiques communes aux entités mal identifiées ?