Projet de linguistique informatique

Catégorisation morphosyntaxique

LI6

Le projet consiste à implanter un étiqueteur morphsyntaxique avec un modèle de Markov caché (Hmm).

Pour une phrase $\mathbf{w} = w_1 \dots w_n$ de n mots, la tâche d'étiquetage morphosyntaxique consiste à prédire la séquence $\mathbf{c} = c_1 \dots c_n$ de catégories de ces mots. Par exemple, la phrase *le chat mange une souris .* donnée en entrée sera typiquement catégorisée *D N V D N PONCT*.

On propose ici de modéliser la tâche de prédiction par un modèle de type Hmm bigramme qui pour rappel se formule comme suit:

$$P(\mathbf{w}, \mathbf{c}) = P(c_0) \prod_{i=1} P(w_i | c_i) P(c_i | c_{i-1})$$

La tâche de prédiction de la meilleure séquence de catégories consiste à évaluer la conditionnelle :

$$P(\mathbf{c}|\mathbf{w}) = \frac{P(c_0) \prod_{i=1} P(w_i | c_i) P(c_i | c_{i-1})}{P(w_1 \dots w_n)}$$

de manière à résoudre le problème suivant (où le dénominateur est constant et n'affecte pas le calcul du maximum):

$$\hat{\mathbf{c}} = \underset{\mathbf{c} \in GEN(\mathbf{w})}{\operatorname{argmax}} \; P(c_0) \prod_{i=1} P(w_i | c_i) P(c_i | c_{i-1})$$

ce qui revient à chercher la séquence $\hat{\mathbf{c}}$ de probabilité maximum parmi l'ensemble $GEN(\mathbf{w})$ de séquences de catégories possibles pour $\mathbf{w}$.

Partant de l'hypothèse minimaliste que chaque mot peut être étiqueté par une catégorie quelconque choisie dans un ensemble C de catégories, le nombre de séquences $c_1 \dots c_n$ à évaluer est $|C|^n$.

Nous avons vu en cours un algorithme de programmation dynamique (algorithme de Viterbi) qui permet de résoudre efficacement le problème par partage de calculs.

Le projet se divise en deux sous-tâches principales:

Estimation des probabilités du modèle

Prédiction d'une séquence de catégories pour une phrase donnée et évaluation de l'étiqueteur sur un jeu de données de test

1 Les donnees et l'estimation des probabilites

Les données sont issues du *French Treebank* un corpus (jeu de données) qui donne la catégorisation supposée exacte pour des extraits tirés du journal *Le Monde* dont voici un extrait :

word	lemma	cat	subcat	gen	num	pers	mood	tense
On	on	CL	suj	m	s	3	NA	NA
devrait	devoir	V	NA	NA	s	3	ind	cond
y	y	CL	obj	m	s	3	NA	NA
voir	voir	V	NA	NA	NA	NA	inf	NA
un	un	D	ind	m	s	NA	NA	NA
gage	gage	N	c	m	s	NA	NA	NA
précieux	précieux	A	qual	m	s	NA	NA	NA
.	.	PONCT	s	NA	NA	NA	NA	NA

Chaque ligne représente un mot du texte, alors que les colonnes représentent différentes informations sur le texte. Les colonnes qui vous concernent sont les colonnes 0 (`word`) et 2 (`cat`) qui encodent respectivement les mots et leurs catégories.

Exercice 1 (Segmentation en phrases) *Le premier exercice consiste à définir une fonction de signature* `Liste<Phrase> lireDonnees(fluxEntrant)` *où une phrase sera représentée par une liste de couples de mots et de catégories* `(mot,cat)`. *Les fins de phrases sont notées dans les données par une ligne dont le mot est un point et de catégorie* `Ponct`.

Exercice 2 (Decoupage des donnees) *On définit ici une fonction de signature* `(Liste<Phrase>,Liste<Phrase>) split(Liste<Phrase>)` *qui découpe le jeu de données en deux parties. La première sera utilisée pour estimer les probabilités (90% des données), la seconde pour tester la correction du modèle (10% des données).*

Les paramètres d'un modèle `Hmm` bigramme sont estimés en comptant dans les données. Les paramètres du modèle sont de la forme:

$$P(w_i | c_i) = \frac{\text{Compte}(w_i, c_i)}{\text{Compte}(c_i)}$$

$$P(c_i|c_{i-1}) = \frac{\text{Compte}(c_i, c_{i-1})}{\text{Compte}(c_{i-1})}$$

Il est conseillé ici de créer des objets de comptage de type `Compteur` (typiquement des objets de type dictionnaire ou des matrices) qui permettent de récupérer dans une liste de phrases les comptes suivants : Compte(c_i, c_{i-1}) Compte(w_i, c_i) (*comptes conjoints*) et Compte(c_i) (*comptes marginaux*).

Exercice 3 (Matrice de probabilites conditionnelles) *On propose de créer une classe représentant une distribution de probabilité conditionnelle dont le constructeur aura une signature du type* `CondProbs(Compteur comptes_conjoints,Compteur comptes_marginaux)` *et qui implémente une méthode publique de signature* `float cprob(float y, float x)` *qui permet de récupérer une probabilité de la forme* $P(Y = y|X = x)$.

Exercice 4 (Estimation des probabilites) *En utilisant les résultats des exercices précédents, définir une fonction de signature* `(CondProbs,CondProbs) estimation(Liste<Phrase>)` *qui renvoie deux matrices de probabilités conditionnelles à partir d'une liste de phrases. L'une encodera la distribution d'émission* $P(w_i|c_i)$ *l'autre, la distribution de transition* $P(c_i|c_{i-1})$.

2 Prediction : etiqueteur HMM

Exercice 5 (Hmm) *Créer un objet de type* `HmmTagger` *dont le constructeur sera typiquement de signature* `HmmTagger(CondProbs emissions,CondProbs transitions)`.

Exercice 6 (Prediction (Viterbi)) *Créer une méthode publique* `Liste<categories> predict(Liste<mots>)` *pour la classe* `HmmTagger`. *Il s'agit ici essentiellement d'implanter l'algorithme de Viterbi. Il est conseillé mais non obligatoire d'implanter le calcul dans une (ou plusieurs) matrices qui représentent le graphe acyclique orienté (*Dag*)*

Exercice 7 (Test de correction) *Créer une méthode publique* `float evalTagger(Liste<Phrase>)` *pour la classe* `HmmTagger` *qui évalue le pourcentage de correction du tagger. L'idée est de lui faire prédire les catégories pour chaque phrase passée en paramètre et de renvoyer la proportion de cas où le tagger prédit une catégorie identique à la référence.*

3 Problemes a regler

Comme pour la plupart des algorithmes probabilistes pour le traitement du langage, l'implantation d'un tagger Hmm demande de régler quelques problèmes pratiques:

Les produits de la forme $\prod_{i=1}^{n} P(x_i)$, avec $P(x_i)$ une probabilité epsilonesque, tendent à produire des problèmes d'arrondi (écrasement à 0). Pour les régler on utilise des log probabilités. Se rappeler que :

$$\log\left(\prod_{i=1}^{n} P(x_i)\right) = \sum_{i=1}^{n} \log(P(x_i))$$

et que la valeur de argmax n'est pas affectée par la transformation logarithmique.

Utiliser des log probabilités demande de formuler (1) une variante de l'algorithme de Viterbi et (2) faire en sorte que les matrices de probabilités conditionnelles (Exercices 3 et 4) renvoient des log probabilités.

Les produits de la forme $\prod_{i=1}^{n} P(x_i)$ sont aborbés par le moindre facteur nul (resp $\log(0) = \quad 1$). En traitement du langage lorsque certains mots ne sont pas vus lors de la phase d'estimation, on a des facteurs $P(w_i|c_i) = 0$. Il est habituel de garantir que les matrices de probabilités conditionnelles renvoient des probabilités $P(Y = y|X = x) > 0$ quels que soient les événements.

Pour régler cela on utilise des *méthodes de lissage*. La plus simple (ajouter 1 ou λ) consiste à supposer que tout compte dans les données vaut minimalement 1 (ou une valeur $\lambda > 0$). Utiliser cette méthode pose toutefois des difficultés pour la normalisation des lois de probabilité. Vous pouvez en inventer une qui vous paraît suffisante ou encore consulter la wikipedia à ce sujet (*lissage* ou *smoothing*). Ce problème est à traiter également lors de la résolution des exercices 3 et 4.

Débogage: une difficulté pratique dans l'implémentation d'algorithmes probabilistes est de garantir la correction de l'implantation. Vous pouvez comparer votre implantation, à une implantation dite "baseline" qui consiste à prédire la catégorie la plus fréquente pour chaque mot. Si vos résultats sont moins bons, il y a problème... À l'issue du projet vous pouvez espérer obtenir un tagger qui obtient 90-95% d'étiquetages corrects. Moins de 90% est signe de problèmes dans votre implémentation.

4 Evaluation qualitative

Exercice 8 *Evaluation qualitative Votre exercice terminé, observez les erreurs commises par votre tagger. Les erreurs qu'il commet sont-elles plutôt dûes à des cas qui demanderaient d'inclure des connaissances du monde dans le modèle (= d'avoir compris le texte en profondeur) ou révèlent-elles plutôt les faiblesses d'une méthode beaucoup trop grossière ? Faites quelques commentaires et donnez quelques exemples pour illustrer.*

5 Organisation

Le projet est à réaliser en binôme ou en trinôme (monômes interdits) dans le langage de votre choix et est à rendre par email à `bcrabbe@linguist.jussieu.fr` avant le jour de l'examen. Le rendu attendu est un code commenté (notamment paramètres et valeurs de retour des fonctions) ainsi qu'un README qui m'indique comment l'exécuter. Le code doit être divisé en deux sections qui correspondent aux deux premières sections du sujet et idéalement aux numéros d'exercices. À titre indicatif, le corrigé commenté du sujet fait environ 200 lignes en python.