Modèle génératif et modèle discriminant pour l'étiquetage morpho-syntaxique : Rendu

Pacôme Perrotin

Table des matières

0.1 Introduction

Ce projet a pour objectif de fournir un étiquetage morpho-syntaxique simple à partir d'un corpus complet. Ce travail est exécuté par le biais d'un Modèle Caché de Markov de forme bigramme, c'est à dire un algorithme présupposant l'affectation de catégories aux mots, en ne considérant que la successions des mots deux par deux. Ce HMM est implémenté de deux façon différentes, l'une par le biais d'un modèle génératif, l'autre grâce à un modèle discriminant.

Dans ce document nous présenterons l'implémentation de ce double problème sous de nombreux aspects. D'abord nous approcherons en plus de détails le modèle mathématique ici implémenté, puis nous aborderons l'implémentation elle-même. Une note sera adressée durant ces deux parties quant aux améliorations qui ont été ajoutées au modèle de base. Enfin les résultats obtenus en faisant varier les paramètres proposés seront détaillés et critiqués.

1 Modèle

Dans ce projet nous nous intéressons à l'étiquettage d'une séquence de mots. Cet étiquettage peut être représenté de la façon suivante :

Soit $S = \{S_1, \cdots, S_N\}$ un ensemble de catégories et $O = O_1, \cdots, O_T$ une séquence de mots. L'étiquettage estimé de cette séquence est calculé par :

$$\hat{Q} = arg \max_{Q \in S^T} P(O, Q)$$

Avec $P(O, Q)$ la probabilité d'observer la séquence O étiquetée par Q.

Le calcul en soit de cet étiquettage est pris en charge par l'algorithme de Viterbi. Cet algorithme présuppose l'existence des scores suivants :

- Pour chaque catégorie S_i, π_i est le score de l'apparition de la catégorie S_i au début d'une séquence observée.
- Pour chaque couple de catégorie (S_i, S_j), a_{ij} est le score de l'apparition successive de S_i et S_j n'importe où dans une séquence observée.
- Pour chaque mot V_j et chaque catégorie S_i, $b_i(j)$ est le score de l'affectation du mot V_j à la catégorie S_i n'importe où dans une séquence observée.

Derrière un score se cache un procédé qui donne un ordre d'importance numérique à un ensemble d'objet. Cet ordre d'importance peut-être opéré grâce à une distribution de probabilités, ou grâce à n'importe quelle autre méthode, par exemple un calcul de poids via un algorithme perceptron. Dans le reste de notre rapport, nous considérerons les scores comme devant être **minimisés** plutôt que maximisés afin d'obtenir un étiquetage optimal.

Notre approche du problème de l'étiquettage sera de d'abord calculer l'ensemble de ces scores suivant une certaine méthode grâce à un corpus d'apprentissage, puis d'appliquer l'algorithme de Viterbi pour calculer un ensemble d'étiquettes sur un corpus de test. Heureusement, avec l'énoncé de ce projet nous ont été fournis un corpus d'apprentissage et un corpus de test en bonnes formes et entièrement étiquetés, ce qui simplifie grandement le travail d'étiquettage automatique.

Reste à savoir comment extraire du corpus d'apprentissage les scores en question. Deux modèles différents de cette extraction seront abordés lors de ce projet : un modèle génératif basé sur un calcul de probabilités, et un modèle discriminant basé sur un algorithme de perceptron.

1.1 Modèle génératif

Ce modèle considère nos scores de la façon suivante :

— Scores initiaux :

$$\pi_i = -\ log\ P(q_1 = S_i) \approx -\ log\ \frac{C_\pi(S_i) + \alpha}{Nb_{sentences} + N \times \alpha}$$

— Scores de transition :

$$a_{ij} = \sum_{t=2}^{T} -\ log\ P(q_t = S_j \mid q_{t-1} = S_i) \approx -\ log\ \frac{C_a(S_i, S_j) + \alpha}{C(S_i) + N \times \alpha}$$

— Scores d'émission :

$$b_i(j) = \sum_{t=1}^{T} -\ log\ P(O_t = V_j \mid q_t = S_i) \approx -\ log\ \frac{C_b(S_i, V_j) + \alpha}{C(S_i) + K \times \alpha}$$

Avec, par ordre d'apparition :

— $C_\pi(S_i)$ = le nombre total de fois que S_i a été observée au début d'une sous-séquence.

— α = le paramètre de lissage du modèle

— $Nb_{sentences}$ = le nombre total de sous séquences dans le corpus d'apprentissage.

— N = le nombre d'étiquettes différentes.

— T = la longueur du corpus.

— $C_a(S_i, S_j)$ = le nombre total de fois que le bigramme (S_i, S_j) a été observé dans le corpus d'apprentissage.

— $C(S_i)$ = le nombre total de fois que l'étiquette S_i a été observée dans le corpus d'apprentissage.

— $C_b(S_i, V_j)$ = le nombre total de fois que l'étiquette S_i a coincidée avec l'observable V_j dans le corpus d'apprentissage.

— K = le nombre d'observables différents.

Ce premier modèle est aussi le modèle le plus simple : étant donné un certain corpus d'apprentissage étiqueté, il se contente d'en extraire plusieurs statistiques, qu'il transforme ensuite en score grâce à une simple fraction. On notera l'utilisation systématique d'un logarithmique et d'une négation : ces modifications sont purement d'ordre pratique et permettent de repartitionner les probabilitées obtenues dans l'ensemble $[0, 1]$ sur l'ensemble $[0, +\infty[$ avec $0 \to +\infty$, ce qui donne une plus grande précision sur les données en programmation. La négation permet de ne pas travailler avec des nombres négatifs ; c'est son usage qui oblige le reste de notre programme à travailler sur la **minimisation** des scores obtenus plutôt que sur leur maximisation.

Le paramètre de lissage α fait partie des améliorations ajoutées au modèle initial proposé dans l'énoncé. Ce paramètre a pour mission de donner une probabilité minimale à l'ensemble des évènements ; ainsi, aucun évènement n'a de probabilité nulle (et donc de score $+\infty$). Ce paramètre aura généralement une valeur comprise dans $[0, 1]$. Cette garantie minimale de probabilité se fait en ajoutant α à la partie supérieure de chaque fraction, puis en ajoutant $N \times \alpha$ ou $K \times \alpha$ dans la partie inférieure de chaque fraction, afin que la somme des comptes soit toujours équivalente.

Ce modèle est très simple, car il n'exige qu'une seule lecture du corpus d'apprentissage, sur lequel il effectue un calcul de complexité linéaire.

1.2 Modèle discriminant

Ce second modèle, de type discriminant, emploie une solution plus complexe algorithmiquement ; plutôt que d'extraire directement les scores du corpus, on approxime leurs valeurs grâce à un algorithme de perceptron. Le vecteur de poids du perceptron considéré contiendra donc l'ensemble de nos scores π_i, a_{ij} et $b_i(j)$.

Voici l'algorithme de perceptron utilisé par ce modèle, avec pour paramètre un nombre entier I dans $[1, +\infty[$:

1. Initialiser les poids π_i, a_{ij} et $b_i(j)$ à 0.
2. Faire les opérations suivantes I fois :
   - 2.1. Pour chaque phrase $O = O_1 \cdots O_T$ d'étiquettes réelles $Q = q_1 \cdots q_T$, faire :
     - 2.1.1. Calculer $\hat{Q} = \hat{q_1} \cdots \hat{q_T}$ grâce à l'algorithme de Viterbi sur π_i, a_{ij} et $b_i(j)$
     - 2.1.2. Si $Q \neq \hat{Q}$, faire :
       - 2.1.2.1. $\pi_i = \pi_i - \phi_{\pi_i}(Q) + \phi_{\pi_i}(\hat{Q})$
       - 2.1.2.2. $a_{ij} = a_{ij} - \sum_{t=2}^{T} \phi_{a_{ij}}(t, Q) + \sum_{t=2}^{T} \phi_{a_{ij}}(t, \hat{Q})$
       - 2.1.2.3. $b_i(j) = b_i(j) - \sum_{t=1}^{T} \phi_{b_i(j)}(t, Q) + \sum_{t=1}^{T} \phi_{b_i(j)}(t, \hat{Q})$
3. Renvoyer les poids π_i, a_{ij} et $b_i(j)$.

Avec, par ordre d'apparition :

— $\phi_{\pi_i}(Q) = 1$ *si* $q_1 = S_i$, 0 *sinon*

— $\phi_{a_{ij}}(t, Q) = 1$ *si* $q_{t-1} = S_i$ *et* $q_t = S_j$, 0 *sinon*

— $\phi_{b_i(j)}(t, Q) = 1$ *si* $q_t = S_i$ *et* $O_t = V_j$, 0 *sinon*

Ce perceptron diminue à chaque itération les poids correspondant à l'étiquettage réelle, et augmente les poids correspondant à l'étiquettage approximé. Cela résulte en un calcul bien plus complexe que pour le modèle précédent, qui cependant affecte bien les scores minimaux aux évènements les plus probables.

2 Implémentation

Notre implémentation a été réalisée en langage C, choisi pour son efficacité algorithmique. Le code qui a été produit dans le cadre de ce projet se divise en dix modules : data, evaluation, hmm, hmm init, log sum, main, parameters,

parser, perceptron et viterbi. Chacun de ces modules possède un fichier en extension .h et un fichier en extension .c qui lui correspond dans le dossier src. Afin de détailler notre implémentation, nous allons présenter ces modules un à un, dans l'ordre de complexité.

Il est à noter que chacun des fichier de ce projet dispose de commentaires qui présentent les différentes fonctions et expliquent le fonctionnement interne des fonctions les plus algorithmiques.

2.1 Module log sum

2.2 Module parameters

2.3 Module data

2.4 Module hmm

2.5 Module parser

2.6 Module viterbi

2.7 Module hmm init

2.8 Module perceptron

2.9 Module main

3 Résultats

3.1 Résultats via le modèle génératif

3.2 Résultats via le modèle discriminant

3.3 Conclusion