Vraisemblance conditionnelle pour apprentissage de paramètres

François Laurent[†] FRANCOIS.LAURENT@INSA-ROUEN.FR
[†]Etudiant master 2 IGIS GI, Université de Rouen

Résumé

Ce document est un rapport délivré au terme d'un projet qui a consisté à implémenter l'algorithme présenté par [1]. Cet algorithme est un mécanisme d'apprentissage de paramètres d'un réseau bayésien destiné à la classification, dans lequel les variables sont en général toutes observées sauf une, que l'on appelle classe. L'outil au centre de cette méthode est la vraisemblance conditionnelle (ou classifiante) et le nom donné par [1] à l'algorithme est "régression logisitique étendue" (*Extended Logistic Regression*), abrégé en ELR. Le cas de l'apprentissage à partir de données incomplètes n'a pas été abordé.

1 Introduction

Sans reprendre en détail l'article cité ([1]), il s'agit dans un premier temps de saisir quels sont les éléments clés de l'algorithme ELR (*Extended Logistic Regression*). Ce dernier visant à l'apprentissage de paramètres $\theta_{d_j|\mathbf{f}_j}$[1] d'un réseau bayésien, c'est-à-dire des tables de probabilités conditionnelles, la structure G est donnée et fixée. Soit K le nombre de nœuds de G, et soit $B =< G, \Theta >$ le réseau bayésien en tant que graphe aux arcs paramétrés.

Pour l'apprentissage, on dispose d'une base d'exemples, sans donnée manquante, que l'on note S. Pour chaque exemple d'indice i[2], on désigne par c_i la valeur de la variable classe C, et par $\mathbf{e}_i$ les valeurs des autres variables ($\mathbf{E}$). Ainsi, pour N exemples, $S = \{< \mathbf{e}_i, c_i > \forall i \in [\![1, N]\!]\}$.

Deux grandeurs principales sont à introduire. Il s'agit de la log-vraisemblance conditionnelle empirique, ici déterminée sur l'ensemble de la base

$$LCL^S(\Theta) = \frac{1}{N}\sum_{i=1}^{N} \log(\mathrm{P}(C = c_i/\mathbf{E} = \mathbf{e}_i, B))$$

et de son gradient

$$(\frac{\partial LCL^S(\Theta)}{\partial \beta_{d_j|\mathbf{f}_j}})_{j\in[\![1,K]\!]}$$

avec

$$\frac{\partial LCL^S(\Theta)}{\partial \beta_{d_j|\mathbf{f}_j}} = \sum_{i=1}^{N} \frac{\partial LCL^{<\mathbf{e}_i,c_i>}(\Theta)}{\partial \beta_{d_j|\mathbf{f}_j}}$$

pour toute valeur d_j du nœud d'indice j, toute configuration $\mathbf{f}_j$ des parents de ce nœud, et tout j.

Les $\beta_{d_j|\mathbf{f}_j}$ sont des "équivalents logistiques" des $\theta_{d_j|\mathbf{f}_j}$ ($\theta_{d_j|\mathbf{f}_j} = \frac{e^{\beta}_{d_j|\mathbf{f}_j}}{\sum_{d'_j} e^{\beta}_{d'_j|\mathbf{f}_j}}$). Ce changement de variable est opéré pour échapper aux contraintes posées sur les $\theta_{d_j|\mathbf{f}_j}$ ($\theta_{d_j|\mathbf{f}_j} > 0$ et $\sum_{d_j} \theta_{d_j|\mathbf{f}_j} = 1 \forall j$) et se permettre d'utiliser alors un algorithme d'optimisation (non contrainte) tel qu'une méthode de descente de gradient.

Une quantité encore à expliciter est

$$\frac{\partial LCL^{<\mathbf{e}_i,c_i>}(\Theta)}{\partial \beta_{d_j|\mathbf{f}_j}} = [\mathrm{P}(d_j, \mathbf{f}_j|\mathbf{e}_i, c_i, B) - \mathrm{P}(d_j, \mathbf{f}_j|\mathbf{e}_i, B)] - \theta_{d_j|\mathbf{f}_j}[\mathrm{P}(\mathbf{f}_j|\mathbf{e}_i, c_i, B) - \mathrm{P}(\mathbf{f}_j|\mathbf{e}_i, B)].$$

La démonstration de cette formule est à retrouver dans [1].

L'algorithme repose sur l'idée suivante : si l'on initialise les paramètres Θ du réseau B à l'aide du maximum de vraisemblance[3] (MV), on peut calculer dans un premier temps et pour une première fois la log-vraisemblance

[1] d_j désigne une valeur du nœud d'indice j et $\mathbf{f}_j$ la configuration des parents de ce nœud.

[2] Pour rendre les notations plus accessibles, les exemples sont indicés par la lettre i et les variables par la lettre j.

[3] Rechercher le maximum de vraisemblance pour en retenir l'argument Θ^{ML} est une opération relativement simple et rapide : il s'agit du calcul de fréquences des différentes probabilités conditionnelles.

conditionnelle et son gradient. On initialise les paramètres logistiques par $\beta^{(0)}_{d_j|\mathbf{f}_j} = \ln(\theta^{ML}_{d_j|\mathbf{f}_j})$, ce qui n'est qu'une solution possible parmi d'autres.

[1] propose de produire une première solution par une recherche ligne (*line search* en anglais) avant d'appliquer une descente de gradient (conjugué) à partir de cette solution pour estimer les meilleurs paramètres $(\beta_{d_j|\mathbf{f}_j})_{j\in[\![1,K]\!]}$. A noter que cette méthode ne permet pas de décomposer le calcul sur les cliques du réseau ; la descente implique la log-vraisemblance conditionnelle globale.

2 Première implémentation

Matlab a été retenu en tant que langage, et *Bayes Net Toolbox*[4] (BNT) en boîte à outils pour travailler sur les réseaux bayésiens. L'intégration dans cette boîte à outils étant à envisager, un premier lot de contraintes est apparu pour l'homogénéisation du code :

- Les noms de fonctions suivent une nomenclature implicite.
- Les données sont à présentés (ici plutôt à recevoir) suivant un certain format (matrice dans laquelle les exemples sont disposés en colonnes).
- Les fonctions d'optimisation sont rendues disponibles par l'intégration de la boîte à outils NetLab[5] dans la BNT, ce qui n'autorise que peu la réutilisation de fonctions équivalentes en provenance d'autres projets.

Il s'agit de produire une fonction dont le prototype réduit à l'essentiel serait le suivant :

```
bnet = learn_params_elr( bnet, data, cnode )
```

avec `bnet` l'objet réseau bayésien, `data` les données d'apprentissage et `cnode` l'indice du nœud classe (C).

La structure de cette fonction est, comme décrite précédemment :

1. si besoin, initialiser les CPT de `bnet`,
2. apprendre les paramètres Θ^{ML} de bnet à l'aide du maximum de vraisemblance,
3. extraire les paramètres et initialiser β par $\beta^{(0)}_{d_j|\mathbf{f}_j} = \ln(\theta^{ML}_{d_j|\mathbf{f}_j})$[6],
4. calculer $LCL^{(0)}$ et $\nabla LCL^{(0)}$,
5. obtenir β^{LS} par minimisation le long de $\nabla LCL^{(0)}$ depuis $\beta^{(0)}$,
6. calculer β^{CG} par gradient conjugué depuis β^{LS},
7. convertir β^{CG} en θ^{opt},
8. mettre à jour `bnet` et le retourner.

Les fonctions de minimisation sur ligne et de descentes de gradient sont fournies par NetLab :

- `linemin` pour la recherche suivant une direction,
- `conjgrad` (gradient conjugué) ou `scg` (gradient conjugué à facteur d'échelle) pour la descente de gradient.

Elles impliquent l'écriture de fonctions indépendantes pour le calcul de la log-vraisemblance conditionnelle (empirique) et de son gradient, fonctions dont les noms sont passés en argument pour évaluation. Il s'agit de `LCL` et `LCLgrad`.

Comme on cherche à maximiser la vraisemblance conditionnelle, on minimise l'opposé de la log-vraisemblance conditionnelle empirique. A noter que `linemin` est mal documentée, son premier argument en sortie n'étant pas l'*argmin* mais le facteur à appliquer au gradient pour obtenir le déplacement optimal sur la ligne.

Une autre difficulté rencontrée a été la mise en vecteur des paramètres (tant les probabilités conditionnelles que les paramètres logistiques). Des fonctions ont été ajoutées pour cette tâche :

- `bnet_pak` pour l'internalisation d'un vecteur Θ,
- `bnet_unpak` pour l'extraction des probabilités conditionnelles Θ,
- aucune fonction pour la transformation des probabilités conditionnelles en paramètres logistiques, l'opération étant unique et simple (item 3),
- et `beta2theta` pour convertir les paramètres logistiques en probabilités conditionnelles.

3 Tentatives d'optimisation

Tous les essais suivants ont été menés sur une base de 300 exemples en apprentissage, 100 en test, avec 4 variables à 6 valeurs possibles et 1 variable classe binaire. La structure utilisée était celle d'un classifieur naïf de Bayes (la classe "explique" indépendamment chacune des variables observées).

La partie la plus coûteuse en temps de calcul, dans l'algorithme, est l'étape de minimisation par descente de gradient. Un premier axe d'amélioration est donc d'essayer différentes fonctions. Cela a été réalisé par la

[4] Ecrite par Kevin Murphy.

[5] Ecrite par Ian Nabney et Christopher Bishop.

[6] Certains $\theta^{ML}_{d_j|\mathbf{f}_j}$ peuvent être nuls ; il faut donc prévoir d'ajouter à ces paramètres là une unité de précision numérique, telle que 10^{-8} par exemple.

comparaison de `conjgrad` et `scg`. Le résultat est sans appel, car la qualité de la solution proposée par `conjgrad` dépend du nombre maximal d'itérations autorisé, à moins de régler directement la tolérance - l'erreur maximale attendue - sur la fonction minimisée. Pour des résultats équivalents en classification, `conjgrad` est ainsi cinq fois plus long que `scg`. Ultérieurement à ces évaluations, il a été remarqué que `conjgrad` réalise l'étape de minimisation suivant une direction, et donc que le calcul par son aide a été inutilement alourdi.

La seconde étape la plus laborieuse est ensuite la descente en ligne, qui précède la descente de gradient. Une tentative d'optimisation a été menée sur `linemin` car c'était à l'origine l'étape la plus longue. Elle en devenait interminable sur des bases plus volumineuses. La cause provenait d'un biais de conception dans cette fonction, celle-ci ne réalisant la recherche d'un minimum que sur une demi-droite.

Si l'on pose les notations suivantes pour la mise à jour des paramètres : $\beta^{(k+1)}_{d_j|\mathbf{f}_j} = \beta^{(k)}_{d_j|\mathbf{f}_j} + \alpha^{(k)} \times \nabla LCL^S$, `linemin` cherche dans un premier temps une fenêtre $[\alpha_{min}, \alpha_{max}]$ par une *golden search* à l'intérieur de l'intervalle $[0, 1]$ (fonction `minbrack`). Il n'y a aucune raison pour que le gradient soit, comme cela est sous-tendu par ce choix, négatif sur au moins une composante, sauf peut-être dans l'application première de cette fonction (réseaux de neurones). Le cas où la fonction à optimiser (LCL^S) est strictement monotone et croissante sur le segment $[0, 1]$ (pour α) s'est présenté et `minbrack` n'en finissait pas d'approcher le singleton $\{0\}$ (toujours pour α).

Au final, une fois ce problème corrigé, l'idée était de réduire de nombre d'appels imbriqués. De plus, ces appels se faisaient par des évaluations de chaînes de caractères. L'espoir est apparu d'obtenir un gain de temps en spécialisant `linemin` et ses sous-fonctions. Malheureusement, ce gain est très faible, pas significatif du fait des conditions d'évaluation[7], et du coup peu intéressant au regard de la duplication de code qu'il entraîne.

Ont été comparés les algorithmes avec et sans étape préliminaire de recherche ligne. Le gain en performance de l'ajout n'apparaît que de manière très minime sur la vraisemblance (non conditionnelle), et pas sur les résultats en classification. En revanche, le temps de calcul est augmenté de 20%, cette valeur étant probablement dépendante du problème. Une fois de plus, la solution qui vise à la croissance du code n'est pas convaincante, ici malgré les assertions de [1].

Ces comparaisons sont à reproduire sur des problèmes autres, afin de les valider.

Une dernière tentative d'optimisation a porté sur la traduction des $\beta_{d_j|\mathbf{f}_j}$ vers leurs $\theta_{d_j|\mathbf{f}_j}$ correspondants (fonction `beta2theta`). L'intérêt de cette fonction est de retrouver les composantes du vecteur des $\beta_{d_j|\mathbf{f}_j}$ qui partagent un même conditionnement ($\mathbf{f}_j$) pour ensuite appliquer la formule présentée bien avant. Plutôt que de recalculer les positions de ces composantes, pour chaque nœud et chaque configuration des parents de ce nœd, l'idée a été émise de sauver ces relations dans une matrice diagonale par blocs constituée de 0 et de 1. Cette matrice est creuse (*sparse*) et un type de représentation adapté est offert dans Matlab.

Le code a été adapté de manière à l'augmenter le moins que possible. Aucune fonction n'a été ajoutée. En termes de performances, les résultats en vraisemblance et en classification sont strictement identiques à la version précédente, ce qui valide l'implémentation de l'idée. En revanche, une fois encore, les gains en temps de calcul sont décevants, ce même avec la structure plus complexe représentée figure 1

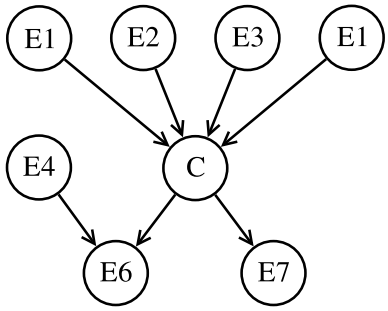

FIG. 1 – Structure générique de test.

4 Tests et résultats

L'algorithme utilisé est donc ELR tel qu'implémenté, sans l'étape de recherche ligne, et sans aucune des optimisations proposées dans la section précédente mise à part l'utilisation de `scg` plutôt que `conjgrad`. La structure choisie est systématiquement celle d'un classifieur naïf de Bayes, structure pas nécessairement adéquate suivant les bases.

[7] Le calcul n'était pas souvent le seul processus utilisateur à occuper la machine, et cette occupation n'était évidemment pas homogène.

Les valeurs entre parenthèses sont les résultats du maximum vraisemblance. Les valeurs de temps ne sont pas toutes données avec la même précision, notamment celles correspondant au maximum de vraisemblance. Les résultats en classification sont des taux de bonnes classifications.

Enfin, la machine mise à contribution est le serveur de calcul de l'antenne INSA du LITIS. Celle-ci menait d'autres calculs et il est donc possible que les conditions influençant le temps de calcul n'est pas été toutes les mêmes d'une base à une autre.

base	temps (s.)	vraisemblance	classification (%)
sens de supériorité	-	+ (- en absolu)	+
abaloneD	17690 (0,082)	-21,21 (-20,54)	22,22 (23,28)
australianD	2977 (0,0207)	-16,54 (-16,31)	81,38 (83,79)
carD	3685 (0,0115)	-8,35 (-7,88)	89,93 (84,03)
contrasepD	4029 (0,015)	-10,92 (-10,91)	47,35 (43,34)
diabetesD	1676 (0,0119)	-15,66 (-15,73)	77,17 (76,36)
germanD	7246 (0,0347)	-22,63 (-22,59)	75 (73,25)
heartD	910 (0,0171)	-16,61 (-16,91)	80 (79,17)
letterD	144430 (0,17)	-33,01 (-31,88)	82,22 (74,84)
nurseryD	39348 (0,043)	-10,14 (-9,78)	92,62 (91,32)
penD	48360 (0,083)	-28,78 (-27,46)	91,37 (83,19)
segmentD	11637 (0,034)	-30,38 (-30,76)	95,27 (88,90)
taeD	195 (0,0043)	-21,46 (-30,12)	31,37 (33,33)

Tab. 1 – Résultats sur des bases de la boîte à outils BNT-SLP.

ELR est censée offrir de meilleurs résultats en classification, et on peut le vérifier dans le tableau 1 à quelques exceptions près. A noter également une capacité de généralisation peu médiocre pour la vraisemblance. L'exemple de complexité réduite[8] utilisé au cours du développement permettait à ELR d'être en moyenne meilleur que le MV pour la vraisemblance sur l'ensemble de test.

Pour validation, des résultats qui laissent penser que l'algorithme fonctionne avec la structure complexe de la section précédente, et donc probablement avec toute structure sont présentés dans le tableau 2.

	temps (s.)	vraisemblance	classification (%)
sens de supériorité	-	+ (- en absolu)	+
	805 (0,0504)	-4,20 (-4,63)	96 (96)

Tab. 2 – Résultat sur une base destinée à évaluer une structure voulue générique.

5 A faire

Les différentes étapes de minimisation se font à l'origine par lots, c'est-à-dire qu'à chaque évaluation de la log-vraisemblance conditionnelle ou de son gradient, la totalité de la base est utilisée, ce qui implique un grand nombre d'inférences à chaque itération. L'alternative de traitements en ligne, qui appliquent la descente de gradient sur la log-vraisemblance conditionnelle localement à un exemple, en itérant exemple après exemple, a été implémentée. Malheureusement, les temps de calcul explosent, et une unique itération sur tous les exemples semble déjà ne pas avoir de fin comparativement à la version en lots. Il doit donc y avoir des optimisations à réaliser, mais peut-être plus au niveau de la *Bayes Net Toolbox*. Une utilisation des fonctions Matlab de quantification des temps de calcul pour chaque fonction appelée montre une lourdeur des étapes d'inférence.

Enfin, de manière générale, des tests sont à multiplier, en particulier ceux qui impliqueraient des structures autres que celle d'un classifier naïf de Bayes.

Références

[1] Russell Greiner, Xiaoyuan Su, Bin Shen, and Wei Zhou. Structural extension to logistic regression : Discriminative parameter learning of belief net classifiers. *Machine Learning*, 59(3) :297–322, 2005.

[8] Il s'agit plutôt d'une famille de bases car les données sont générées aléatoirement, la distinction entre les classes étant rendue possible par l'ajout d'un biais.