Avancement Stabilisation MixtComp

Vincent KUBICKI

8 août 2017

Table des matières

Résumé

Ce document décrit les modifications effectuées à ce jour par rapport à la feuille de route présentée dans le document "Stabilisation de MixtComp" daté du 23 juin 2017.

1 Initialisation sans séparation des données fonctionnelles

Pour rappel, on parle ici de l'initialisation des variables latentes uniquement. L'initialisation des paramètres est l'objet de la section 2.

Dans l'ancienne initialisation des données fonctionnelles, les domaines d'appartenance à chaque sous-régression sont connexes. C'est-à-dire que tous une série de points qui se suivent sont assignés à une sous-régression, puis une autre série à une autre sous-régression et ainsi de suite. On tire les points de séparation entre les domaines des sous-régressions, et on assigne les labels de sous-régression dans ces blocs. Cela résulte en une séparation complète des sous-régressions, qui pose problème pour le modèle de régression logistique utilisé dans l'estimation des paramètres α du modèle. Le paramètre α intervient dans la probabilité d'appartenance à une sous-régression, quand on connaît uniquement la valeur de x et la classe.

L'initialisation a en conséquence été modifiée pour que les transitions entre les sous-régressions se fassent en douceur. La séparation n'est plus complète.

2 Initialisation avec un représentant par classe

Pour rappel, l'ancienne initialisation des paramètres (et non des variables latentes) consistait à assigner une classe au hasard à chaque individu, pour ensuite effectuer une estimation de paramètres par maximum de vraisemblance. Or, quand le nombre d'individus augmente, les paramètres estimés dans chaque classe tendent à devenir similaires aux paramètres qui sont estimés sur tout l'échantillon. Et, par conséquent, les paramètres estimés tendent à devenir identiques entre toutes les classes. Ce qui pose problème si on veut mettre en place un système de relance pour éviter les dégénérescences, car on va initialiser l'algorithme de façon peu variée, ignorant des initialisations potentiellement intéressantes.

La solution proposée est donc de choisir un individu au hasard par classe (que l'on identifiera par la suite comme le représentant de la classe). Cet individu est bien sûr identique pour toutes les variables. Chaque variable utilise comme elle le souhaite la valeur fournie par cet individu pour en déduire ses valeurs de paramètres. La séquence d'initialisation devient donc :

- Initialisation des données, notamment l'imputation initiale des valeurs manquantes et des variables latentes.
- Initialisation des paramètres, en utilisant un représentant par classe.
- Lancement de la première itération SEM.

Les initialisations individuelles pour chaque modèles diffèrent un peu de ce qui a été proposé dans la documentation initiale. Il s'agit à présent de :

- Loi normale : la valeur de μ est la valeur de l'observation. Pour l'écart-type, on calcule l'écart-type sur tout l'échantillon et on le divise par le nombre de classes.
- Loi multinomiale : on fixe toutes les proportions à 1 / nombre de classes, puis on ajoute 1 à la proportion de la modalité du représentant de la classe. On normalise pour obtenir une loi de probabilité.
- Loi de Poisson : la valeur du représentant est utilisée pour déterminer λ. Si cette valeur est 0, la valeur 0.5 est assignée, pour obtenir une distribution très concentrée en 0, mais qui n'assigne pas de probabilité nulle aux observations non nulles.
- Rangs : la valeur du représentant est utilisée pour déterminer le rang central. La dispersion vaut 0.5 / nombre de classes.
- Ordinal : la valeur du représentant est utilisée pour déterminer le mode. La dispersion vaut 1 / nombre de classes.
- Données fonctionnelles : une étape M est effectuée en utilisant uniquement le représentant comme observation dans chaque classe.

L'idée générale est que pour les modèles ayant une notion de dispersion dans leurs paramètres, elle devienne de plus en plus faible à mesure que le nombre de classes augmente. A noter qu'à part pour le modèle sur les données fonctionnelles, il serait rapide d'ajuster la façon dont l'individu représentatif est traduit en paramètres.

A noter qu'on peut ignorer l'information du représentant et effectuer une étape M comme avant. Par exemple, il y a encore quelques problèmes à résoudre pour le modèle sur les données fonctionnelles. Il suffit de modifier une ligne de code dans le modèle pour effectuer l'initialisation en utilisant toutes les observations de la classe et non plus uniquement le représentant. Dans ce cas, on revient à l'ancien système d'initialisation pour ce modèle uniquement. Le nouveau système intègre l'ancien.

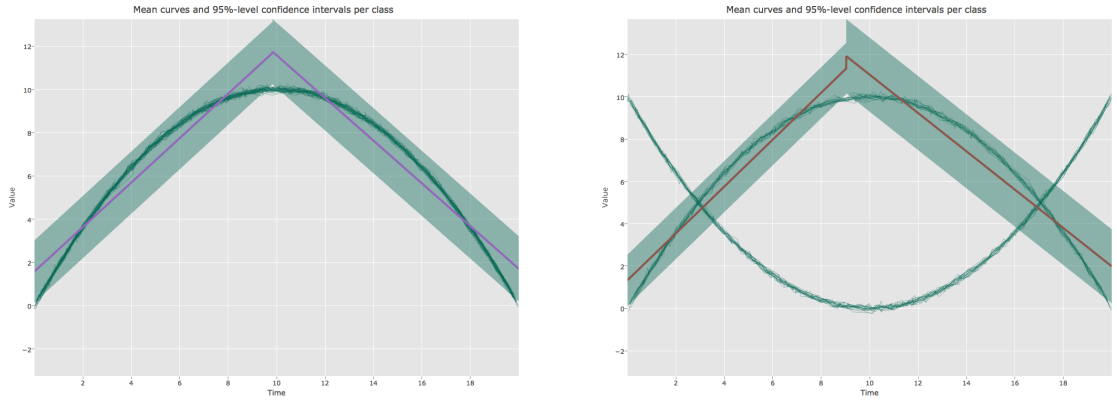
	1	2
1	25	9
2	0	66

(a) Ancienne initialisation

	1	2
1	0	27
2	45	28

(b) Nouvelle initialisation

TABLE 1 – Comparaison des matrices de confusion en apprentissage



(a) Initialisation traditionnelle

(b) Nouvelle initialisation

FIGURE 1 – Comparaison des lois estimées pour la première classe.

3 Problèmes avec l'initialisation utilisant un individu par classe

Dans les essais effectués, on a relevé des problèmes d'initialisation avec les données fonctionnelles.

Pour le moment le critère principal est simplement de regarder la matrice de confusion. On a 100 individus, et on s'attend à un maximum d'une dizaine de mauvais classements en utilisant l'ancienne méthode. Sur la table 1, on peut trouver les matrices de confusion avec l'ancienne et la nouvelle méthode d'initialisation. On voit nettement que le classement est plus mauvais avec la nouvelle méthode d'initialisation. On va essayer de comprendre d'où cela vient.

Sur la figure 1, on peut voir les modèles estimés avec l'ancienne et la nouvelle initialisation. Dans la génération de données, on a utilisé deux classes, et donc on a demandé à MixtComp d'estimer deux classes. Pour simplifier la figure, le modèle estimé pour une classe uniquement a été représenté, ainsi que les observations qui a l'issue du run (c'est-à-dire après le Gibbs) ont été assignés à la classe 1. On peut voir que dans le cas de l'ancienne initialisation, toutes les observations proviennent de la même classe du jeu de données, alors que pour la nouvelle initialisation, il y a un mélange des observations issues des deux classes. Il est étonnant qu'il y ait de telles différences dans les imputations de classes, car les deux modèles estimés semblent proches graphiquement.

On peut aussi s'intéresser aux différentes vraisemblances, pour essayer de détecter des comportements particuliers. Ces vraisemblances sont regroupée dans la table 2. Pour les

	log-vraisemblance complétée	log-vraisemblance observée
ancienne initialisation	4537.205	4537.205
nouvelle initialisation	-931201.5	961.0851

TABLE 2 – Comparaison des log-vraisemblances

log-vraisemblances observées, on voit dans la table qu'on avait 4537.205 avec l'ancienne initialisation, et 961.0851 avec la nouvelle initialisation. Ce qui est plutôt cohérent avec les erreurs de classement observées avec la nouvelle initialisation. Une meilleure vraisemblance observée est ainsi associée à un nombre inférieur d'erreurs de classement.

On regarde maintenant les vraisemblances complétées (uniquement selon la classe, on a bien marginalisé sur les autres variables latentes). Pour l'ancienne initialisation, la vraisemblance complétée est égale à la vraisemblance observée. Par contre, pour la nouvelle initialisation, la vraisemblance complétée est très inférieure à la vraisemblance observée. Tout se passe comme si les individus n'étaient pas assignés à la classe la plus probable sachant leur valeur observée. On va pousser les analyses pour comprendre quelle est la cause de cette différence. Par exemple, en comparant les t_{ik} obtenus par tirage des t_{ik} obtenus par calcul direct. Les résultats seront l'objet d'une prochaine communication.

4 Initialisations multiples

Comme on est capables d'initialiser de façons variées le calcul (ou du moins on sera capables quand les bugs évoqués dans la section 3 seront corrigés), il devient plus intéressant de gérer les dégénérescences en relançant le calcul.

Ces modifications ont été implémentées. A présent, les deux systèmes suivants ont été complètement éliminés de MixtComp (dans la branche de développement, bien sûr) :

- estimateur biaisé : les paramètres sont bornés pour ne jamais être estimés sur le bord. Par exemple, si l'écart-type d'une loi normale devait être estimé à 0, il sera estimé à 1e-8. Un des problèmes de ce système est que si une classe se vide, même si on fixe sa proportion dans π à 0, on a aucun individu pour faire les estimations dans les différentes variables. Cela conduit à un crash.
- échantillonneur de Gibbs : tous les tirages sont effectués en supposant qu'on ne peut pas tirer une valeur qui entrainerait une estimation d'un paramètre sur le bord. Ce système était extrêmement lent à l'usage, et jamais utilisé en pratique.

En plus de fournir une alternative plutôt robustes aux deux solutions évoquées ci-dessus, un bénéfice majeur des initialisations multiples est la simplification du code. Lors de l'ancienne initialisation par exemple, il faut savoir si on utilise un estimateur biaisé ou l'échantillonneur de Gibbs. Dans le cas où on utilise l'échantillonneur de Gibbs, il faut savoir si il est activé ou pas (est-ce que l'estimation a déjà dégénéré ou pas ?). Il faut aussi vérifier que l'initialisation des données permet d'effectuer la première étape M, et procéder à des relances jusqu'à ce que ce soit le cas. De plus, dans le cœur du code de chaque modèle, il faut pour l'échantilloneur de Gibbs déterminer pour chaque tirage de chaque variable latente si des valeurs sont interdites ou pas. Cela introduit une complexité distribuée un peu partout dans le code du modèle.

Cependant, contrairement aux deux implémentations précédentes, il n'est pas garanti qu'au moins un run aboutisse. Si aucun run n'aboutit, un message d'erreur détaillé est transmis à l'utilisateur. Il faut tenir compte de cette possibilité, notamment dans Massiccc qui est très automatisé.

5 Divers

Le code a été transféré sur Gitlab et l'intégration continue a été configurée sur le serveur Inria. Par contre, comme l'algorithme est stochastique, il n'est pas possible d'obtenir des résultats reproductibles. On ne peut pas fixer la graine du générateur aléatoire car il y a de nombreux générateurs dans le code. Il faut mettre en place une solution un peu plus compliquée pour que tous les générateurs soient initialisés différemment, mais de façon reproductible. On peut imaginer utiliser un compteur par exemple. Comme ce n'est pas prioritaire cette implémentation a été repoussée à plus tard.

6 A faire

Il n'y a pour le moment pas de vérifications effectuées sur les représentants utilisés pour initialiser chaque classe. Or, rien ne garanti qu'un individu tiré au hasard puisse initialiser une classe. Par exemple, pour les données fonctionnelles, il est possible qu'une observation n'ait pas assez de points pour initialiser toutes les sous-régressions. Une solution serait dans ce cas d'effectuer un nouveau tirage des individus représentatifs, jusqu'à soit arriver à initialiser, soit sortir un message expliquant le problème.

Avec le code de détection de dégénérescence on a des lancements multiples, mais on ne conserve que le premier run qui fonctionne, ce qui simplifie l'implémentation. On a commencé à regarder comment gérer des lancements multiples à l'intérieur de MixtComp, et ne conserver que le meilleur résultat (en terme de vraisemblance observée). Le problème est que dans un run MixtComp on exporte les paramètres et les statistiques sur ces paramètres. Les modèles n'ont pas été écrits dans cette optique de sauvegarde, et il faudrait les modifier individuellement.

De façon générale, si il est nécessaire de faire du coclustering dans MixtComp, il faudra repenser et unifier la gestion des paramètres entre les différentes variables décrites par le même modèle. Ca pourrait être l'occasion de mieux gérer les sauvegardes / restaurations. Il semble contre-productif d'implémenter une première version de mise en cache maintenant.

Pour résumer, le calcul est relancé automatiquement en cas de dégénérescence, mais n'est pas relancé pour sélectionner un résultat sur un critère de performance, ni par exemple pour tester plusieurs nombre de classes. Dans l'immédiat cela doit être fait depuis la fonction appelant MixtComp.

7 Conclusion

Une part importante des modifications proposées dans le document "Stabilisation de MixtComp" ont été implémentées. Il reste des modifications à effectuer sur l'initialisation

des paramètres pour les données fonctionnelles, mais il y a des pistes à explorer dès à présent pour comprendre.