Stabilisation de MixtComp

Vincent KUBICKI

23 juin 2017

Table des matières

Résumé

Plusieurs projets InriaTech récents ont utilisé le modèle de données fonctionnelles. Or, ce modèle semble souvent activer certains chemins d'exécution assez rares pour les autres modèles de MixtComp, qui amènent à des plantages. Il existe actuellement une méthode implémentée dans MixtComp pour éviter ces chemins d'exécution, mais elle est extrêmement lente, complexe dans le code, et qui n'est jamais utilisée en pratique (pas même sur MASSICCC). L'idée générale de ce document est de décrire les modifications à apporter à MixtComp pour supprimer ces problème.

1 Organisation

Il y a quatre modifications principales. On se fixe une fenêtre de base de 3 mois pour les effectuer et les valider.

Ces modifications sont dans l'ordre :

1. Initialisation en utilisant un individu par classe

2. Lancement multiples, sans détection de dégénerescence, en ne gardant que le meilleur run
3. Détection de la dégénerescence
4. Suppression du Gibbs et de tout le code rendu inutile par les modifications précédentes

L'implémentation sera effectuée par étape avec pour objectif d'avoir un code fonctionnel et directement utilisable dans les contrats à chaque étape.

2 Initialisation avec un individu par classe

2.1 Problème

L'initialisation actuelle de MixtComp est basée sur une partition aléatoire en utilisant une loi uniforme. Ce tirage correspond à une étape S du SEM, en fixant des probabilités uniformes. Le problème de cette initialisation est que les lois estimées dans chaque classe convergent vers les lois estimées sur l'échantillon complet quand le nombre d'observations augmentent. Les initialisations sont donc quasiment identiques dans chaque classe.

2.2 Solution

On va sélectionner un individu pour générer chaque classe.

Les initialisations deviennent, modèle par modèle (i_k désigne l'individu assigné à la classe k pour cette initialisation et N le nombre d'observations dans l'échantillon) :

— **Loi normale** : la valeur observée pour i_k est utilisée pour initialiser μ. Pour σ, on calcule dans un premier temps l'écart-type sur tout l'échantillon σ_s et on en déduit : $\sigma = \frac{\sigma_s}{N}$

— **Loi multinomiale** : à la modalité observée de i_k, on associe une probabilité de 1, et aux autres modalités une probabilité de $1/N$, puis on normalise

— **Loi de Poisson** : Valeur de i_k à laquelle on ajoute $\frac{1}{N}$ pour éviter le cas $\lambda = 0$

— **Rang** : Le rang central est celui de i_k et on fixe une valeur arbitraire pour π

— **Fonctionnel** : on fait comme si i_k était le seul individu de la classe, et on fait une estimation des paramètres. A noter qu'il faut que les sous-regressions soient équi-répartie sur cette observation.

— **Ordinal** : l'individu i_k fournit le mode, et la dispersion est fixée arbitrairement

A noter que cette initialisation est effectuée après la complétion initiale des données (qui utilise des tirages uniformes puisqu'on a pas encore estimé de loi). Cela permet d'avoir des valeurs pour chaque variable de i_k qui ne sont jamais partiellement observées.

D'autres solutions plus complexes peuvent être envisagées. Par exemple on pourrait partitionner les observations, introduire un ordre dans chaque partie, et permettre à chaque modèle de sélectionner autant d'observations qu'il veut pour s'initialiser. Par exemple, un modèle de Poisson pourrait prendre la première valeur, une loi normale les deux premières valeurs, etc... On attendra que le besoin se présente pour implémenter cette solution plus complexe.

A noter que même si il y a des seuils ou des valeurs qui semblent arbitraires, il s'agit simplement de l'initialisation. On ne perd pas de propriétés du SEM en les utilisant. De plus il faut rester simple dans ces initialisations.

3 Lancements multiples

3.1 Problème

Le SEM est sujet à deux sources d'aléas. La première, que l'on retrouve dans l'EM, est liée à l'initialisation. La seconde, qui lui est propre, est liée au caractère stochastique de l'algorithme. Ces deux sources d'aléas, combinées au caractère non convexe de l'estimation par maximum de vraisemblance pour un modèle de mélange engendrent une variabilité dans les résultats. Idéalement, il faut relancer le SEM plusieurs fois et comparer les résultats obtenus.

3.2 Solution

Comme chaque jeu de paramètre est propre à chaque modèle, il est nécessaire que la gestion des paramètres soit déléguée à chaque modèle. Il faut donc que dans un modèle il y ait une option de mise en cache / récupération du cache des paramètres. On fait tourner les SEM (sans les Gibbs) plusieurs fois. A l'issue de chaque lancement n on a estimé $\hat{\theta}_n$ et la vraisemblance (observée) de l'échantillon associée. On ne conserve en cache que le jeu de paramètres maximisant cette vraisemblance. On lance le Gibbs avec le meilleur jeu de paramètres, et on exporte les résultats exactement comme dans le cas actuel.

Ce fonctionnement est transparent pour l'utilisateur. Quand on parle de lancements multiples, il s'agit de lancements de chaînes SEM en interne. L'utilisateur lui ne lance MixtComp qu'une seule fois. Et, comme on améliore par ailleurs les initialisations, il est possible que l'algorithme soit tout aussi efficace qu'avant en utilisant cependant un nombre plus faible d'itérations pour chacun des lancements.

4 Détection de la dégénérescence

4.1 Problème

Les dégénérescences limitent le nombre de chaînes SEM qui arrivent à leur terme. Elles ont tendance à être plus fréquentes quand on augmente le nombre de classes ce qui limite la complexité des modèles que l'on peut estimer. La combinaison d'initialisations plus efficaces avec les relances multiples permet de mettre en place des méthodes plus efficaces que les méthodes actuelles :

— Fixer des bornes inférieures aux paramètres estimés pour ne pas être sur le bord de l'espace. Le problème de cette méthode est qu'elle introduit des seuils arbitraires, et que si une classe se vide on observe de toute façon des plantages. Elle n'est utile que dans les dégénérescences de type toute les valeurs identiques pour une loi normale,

ou toutes les valeurs à 0 pour une loi de Poisson. Le code résultant n'est pas très complexe.

— Utiliser des lois conditionnelles lors des tirages. Il s'agit d'effectuer le même tirage que pour le cas standard, excepté qu'on conditionne en interdisant des résultats de tirages qui produiraient une estimation sur le bord de l'espace des paramètres. Par exemple, pour une loi catégorielle, on est obligé de tirer une modalité l'individu actuel est le seul où elle est observée. interdit de tirer une modalité . Le calcul devient très long car il y a énormément de tests à effectuer. De plus, comme les observations ne sont plus indépendantes, la parallélisation ne fonctionne plus. Le code est devenu beaucoup plus complexe après l'introduction de cette modification.

On peut distinguer les dégénérescences qui se traduisent par une vraisemblance bornée de celles qui se traduisent par une vraisemblance non bornée. On peut relancer la chaîne SEM uniquement dans le cas des dégénerescence entraînant une vraisemblance non bornée. En effet, une dégénerescence avec vraisemblance non bornée présente pour seul problème d'être un état absorbant dans la chaîne, mais par contre il n'y a pas de problèmes numériques (pendant l'apprentissage en tout cas), on peut calculer les t_{ik} normalement par exemple. Comme on effectue plusieurs lancements de MixtComp, cela ne pose pas de problème insurmontable que certaines chaînes soient dans un état absorbant, puisque la vraisemblance reste le critère de sélection de la meilleure chaîne.

4.2 Solution

On peut ne relancer une chaîne SEM que dans les cas où la dégénerescence induit une vraisemblance non bornée.

L'avantage est de rejeter moins de chaînes, parce qu'il n'y a que les lois normale et fonctionnelles avec régressions peuvent induire une dégénerescence avec vraisemblance non bornée. Cela permet sur un jeu de donnée de pouvoir tester avec plus de classes par exemple.

Le principal problème de cette approche apparaît par contre en prédiction. Si on a une dégénérescence, il faut décider de ce qu'il faut faire pour ne pas avoir de probabilités nulles en prédiction. Deux solutions semblent se dégager :

— soit refuser un jeu de données en prédiction quand on a une condition particulière, par exemple qu'on observe une modalité qu'on observait pas en apprentissage.

— l'autre solution est de normaliser les paramètres, ce qui ressemble un peu à la problématique évoquée en 2.2, puisqu'on ne voulait pas initialiser la chaîne par un modèle dégénéré.

Les détections étant déjà implémentées, il s'agit de les faire évoluer et de les utiliser différemment de la façon actuelle.

5 Divers

— Les vraisemblances observées sont utilisées à plusieurs endroit du code et sont longues à calculer. Actuellement la mise en cache est déléguée aux modèles, mais il serait plus performant et simple d'effectuer la mise en cache à l'échelle du composeur.

— On peut calculer directement les t_{ik} au lieu de les tirer, puisqu'on calcule les probabilités observées pour tous les modèles. C'est simple à effectuer et permet des calculs

plus précis sur les sorties visuelles.