Model-Based Learning: Implémentation de l'algorithme LBVEM dans un package R

Gauthier Castro, Pierre Prablanc

Février 2019

Introduction

Dans ce rapport de projet, nous présentons d'abord brièvement le modèle des blocs latents. Dans une deuxième partie nous décrivons l'algorithme utilisé pour estimer les paramètres du modèle ainsi que les détails sur son implémentation en R. Enfin, nous testons l'algorithme sur plusieurs jeux de données ainsi que la capacité du critère de convergence à choisir le bon nombre de co-clusters.

1 Modèle des blocs latents

Dans le modèle des blocs latents (*Latent Block Model*, LBM), on cherche à effectuer un clustering simultané sur les individus et sur les variables. Ce modèle repose sur deux hypothèses:

- Les X_{ij} sont indépendants et identiquements distribués.
- Les Z (partitions en lignes) et W (partitions en colonnes) sont a priori indépendants.

LBM dans le cas gaussien est un modèle très parcimonieux car chaque cluster n'est représenté que par une moyenne, une variance et un poids par ligne et par colonne.

Algorithm 5.3 Gaussian LBVEM

input: x, g, m

initialization: $\widetilde{\mathbf{z}}$, $\widetilde{\mathbf{w}}$, $\pi_k = \frac{\widetilde{z}_{.k}}{n}$, $\rho_\ell = \frac{\widetilde{\mathrm{w}}_{.\ell}}{d}$, $\mu_{k\ell} = \frac{x^{\widetilde{\mathbf{z}}\widetilde{\mathbf{w}}}_{k\ell}}{\widetilde{z}_{.k}\widetilde{\mathrm{w}}_{.\ell}}$. $\sigma^2_{k\ell} = \frac{\sum_{ij}\widetilde{z}_{ik}\widetilde{\mathrm{w}}_{j\ell}x^2_{ij}}{\widetilde{z}_{.k}\widetilde{\mathrm{w}}_{.\ell}} - \mu^2_{k\ell}$

repeat

$x^{\widetilde{\mathbf{w}}}_{i\ell} = \frac{1}{\widetilde{\mathrm{w}}_{.\ell}}\sum_j \widetilde{\mathrm{w}}_{j\ell}x_{ij}$, $u^{\widetilde{\mathbf{w}}}_{i\ell} = \frac{1}{\widetilde{\mathrm{w}}_{.\ell}}\sum_j \widetilde{\mathrm{w}}_{j\ell}x^2_{ij}$

repeat

step 1. $\widetilde{z}_{ik} \propto \pi_k \exp\left(-\frac{1}{2}\sum_\ell \widetilde{\mathrm{w}}_{.\ell}\left(\log\sigma^2_{k\ell} + \frac{u^{\widetilde{\mathbf{w}}}_{i\ell} - 2\mu_{k\ell}x^{\widetilde{\mathbf{w}}}_{i\ell} + \mu^2_{k\ell}}{\sigma^2_{k\ell}}\right)\right)$

step 2. $\pi_k = \frac{\widetilde{z}_{.k}}{n}$, $\mu_{k\ell} = \frac{\sum_i \widetilde{z}_{ik}x^{\widetilde{\mathbf{w}}}_{i\ell}}{\widetilde{z}_{.k}}$, $\sigma^2_{k\ell} = \frac{\sum_i \widetilde{z}_{ik}u^{\widetilde{\mathbf{w}}}_{i\ell}}{\widetilde{z}_{.k}} - \mu^2_{k\ell}$

until convergence

$x^{\widetilde{\mathbf{z}}}_{kj} = \frac{1}{\widetilde{z}_{.k}}\sum_i \widetilde{z}_{ik}x_{ij}$, $v^{\widetilde{\mathbf{z}}}_{kj} = \frac{1}{\widetilde{z}_{.k}}\sum_i \widetilde{z}_{ik}x^2_{ij}$

repeat

step 3. $\widetilde{\mathrm{w}}_{j\ell} \propto \rho_\ell \exp\left(-\frac{1}{2}\sum_k \widetilde{z}_{.k}\left(\log\sigma^2_{k\ell} + \frac{v^{\widetilde{\mathbf{z}}}_{kj} - 2\mu_{k\ell}x^{\widetilde{\mathbf{z}}}_{kj} + \mu^2_{k\ell}}{\sigma^2_{k\ell}}\right)\right)$

step 4. $\rho_\ell = \frac{\widetilde{\mathrm{w}}_{.\ell}}{d}$, $\mu_{k\ell} = \frac{\sum_j \widetilde{\mathrm{w}}_{j\ell}x^{\widetilde{\mathbf{z}}}_{kj}}{\widetilde{\mathrm{w}}_{.\ell}}$, $\sigma^2_{k\ell} = \frac{\sum_j \widetilde{\mathrm{w}}_{j\ell}v^{\widetilde{\mathbf{z}}}_{kj}}{\widetilde{\mathrm{w}}_{.\ell}} - \mu^2_{k\ell}$

until convergence

until convergence

return $\boldsymbol{\pi}$, $\boldsymbol{\rho}$, $\boldsymbol{\alpha}$

Figure 1: Algorithme LBVEM, tiré de CITATION

2 Algorithme et détails d'implémentation

Pour estimer les paramètres du modèle LBM gaussien, on utilise l'algorithme Expectation Maximization Variationnel (VEM). VEM utilisé pour estimer les paramètre du modèle LBM est appelé LBVEM.

- La matrice X représente les données à traiter, g et m respectivement le nombre de clusters ligne et colonnes souhaités.
- Parmis les variables internes on remarque π_k et ρ_l représentant respectivement les proportions du bloc (k et l). On trouve aussi l'ensemble des paramètre de la loi normale à savoir μ_{kl} la moyenne du co-cluster kl et σ^2_{kl} la variance du co-cluster kl.
- Dans le calcul des partitions en ligne est en colonnes (étapes 1 et 3 de l'algorithme), le terme contenu dans l'exponentielle a tendence à

être très négatif. Lorsqu'une valeur est inférieure à -750 (environ), l'exponentielle de ce nombre est arrondie à 0. Si on se retrouve dans cette situation, l'algorithme ne peut plus fonctionner correctement. Pour palier à ce problème numérique, nous examinons la plus petite valeur contenue dans l'exponentielle. Si elle est inférieur à -100, alors on divise l'ensemble des valeurs dans l'exponentielle par la valeur minimale (en valeur absolue) puis on re-scale en multipliant par 100. Nous sommes conscients que nous modifions la dynamique des valeurs après applications l'exponentielle (puisqu'il s'agit d'une fonction non-linéaire). Il s'agit pourtant de la seule solution qui puisse mener à un résultat graphique satisfaisant sur le calcul des partitions.

- Afin de ne pas risquer une division par zéro, on rajoute une faible valeur dans les variances (1E-5).

3 Applications

3.1 Test et comparaison de l'implémentation

- Jeu de données artificiel

  Afin de tester la pertinence de notre implémentation, nous avons décidé de la comparer à la librarie *blockcluster*. On constate que les résultats de notre implémentation figure 5 sont similaires aux résultats figure 2 de *blockcluster*. Les seules différences sont au niveau de l'ordonnancement des clusters, la structure reste la même.

- Jeu de données réelles

  On teste l'algorithme sur MNIST. On utilise les 500 premiers éléments, en dimension 784. figure (3). On effectue un coclustering sur ces données avec 5 clusters lignes et 4 clusters colonnes. Nous n'avons pas pu comparer les résultats obtenus avec blockcluster. En effet aucune de nos tentatives (combinaisons de 2 à 12 clusters ligne et colonne) n'a abouti. On constate que sur ces données l'algorithme, on a du mal à trouver une structure malgré les différentes combinaisons étudiées. La seule observation qui puisse être fait est qu'il se dégagent deux zone distincts à droite et à gauche. Peut être qu'en faisant tourner l'algorithme sur l'emble du jeu de test aurait été plus concluant mais les temps de

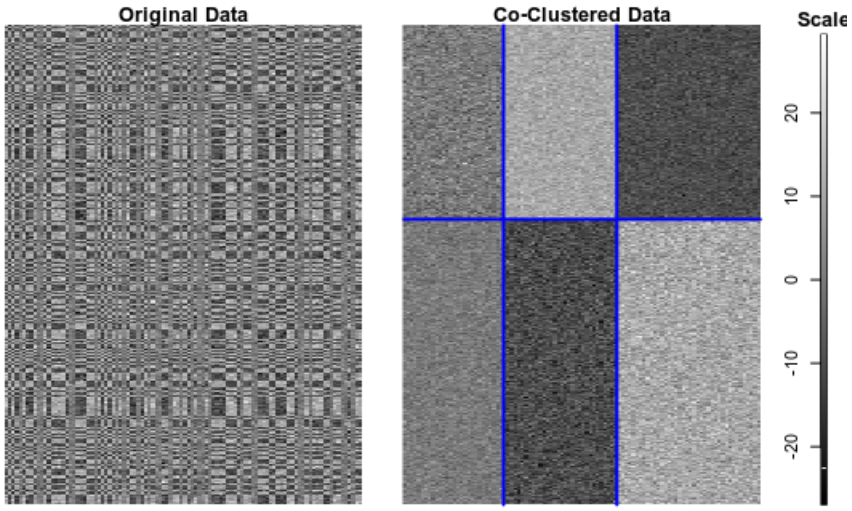

Figure 2: Résultats co-clustering avec 2 lignes 3 colonnes à l'aide de la librairie *blockcluster*.

calculs étaient très longs. Les données ne se portent peut être pas à ce genre d'algorithme.

3.2 Choix du nombre de co-clusters par critère ICL

Nous avons comparé trois modèles différents sur les données "gaussiandata" du package *blockcluster* avec les paramètres suivants:

- 2 lignes 2 colonnes
- 2 lignes 3 colonnes
- 3 lignes 2 colonnes

Nous avons noté que $ICL_{2,2} < ICL_{3,2} < ICL_{2,3}$. Les données sont effectivement mieux représentées par le découpage en cluster 2 lignes 3 colonnes. On voit notemment que sans les 3 colonnes, la matrice ne peut pas être efficacement rangée sans induire un certain mélange dans les cluster (apparence zebrée sur les figures 4 et 6). La figure 2 représente un modèle maximisant le critère ICL et présentant un résultat graphique satisfaisant au sens ou les blocs sont clairement séparés et semble relativement homogènes.

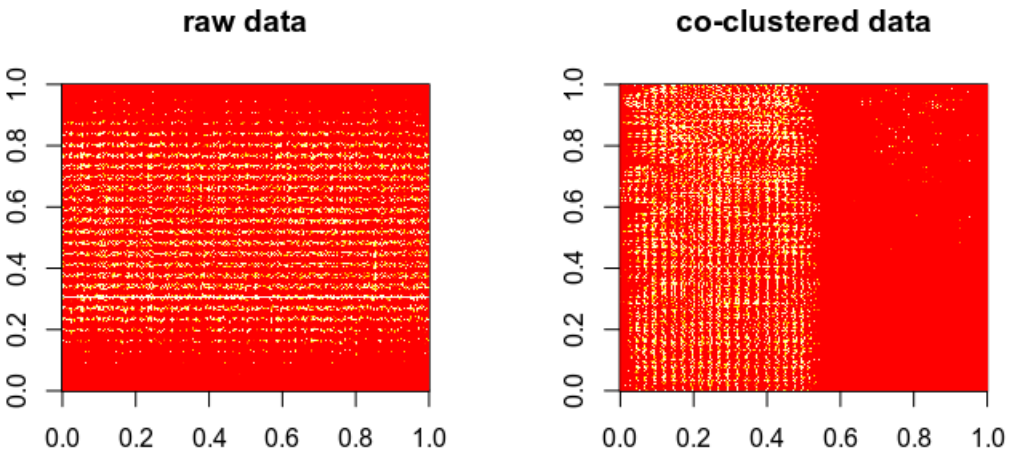

Figure 3: Résultats co-clustering sur MNIST.

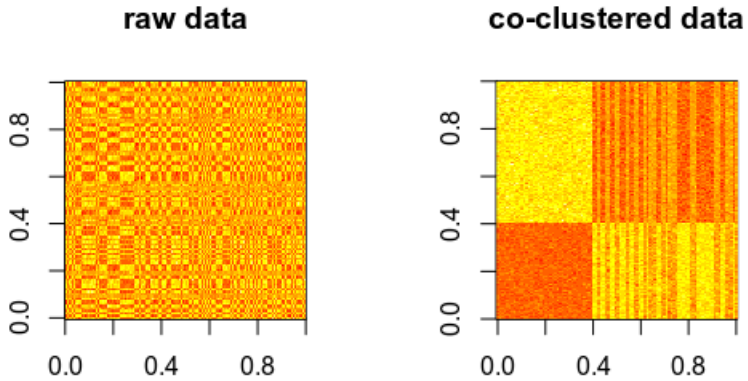

Figure 4: Résultats co-clustering avec 2 lignes 2 colonnes

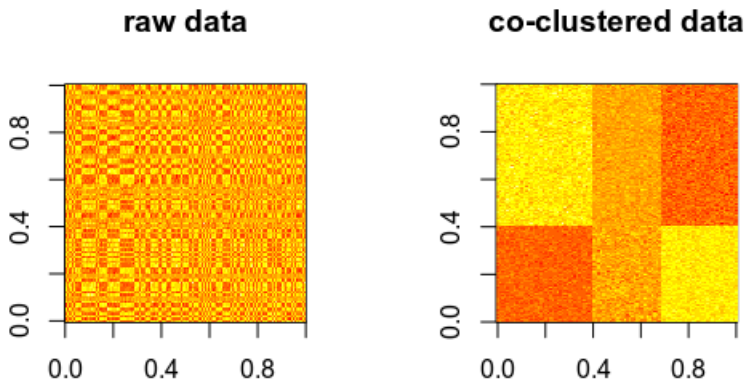

Figure 5: Résultats co-clustering avec 2 lignes 3 colonnes

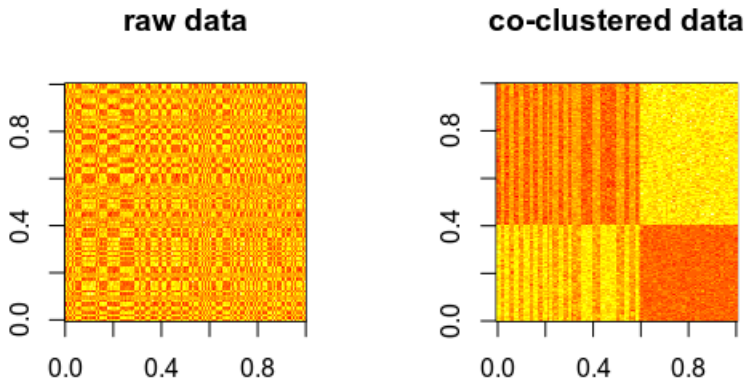

Figure 6: Résultats co-clustering avec 3 lignes 2 colonnes

Conclusion

Dans ce projet, nous avons présenté le modèle des blocs latents ainsi qu'un algorithme permettant d'estimer les paramètres du modèle. Nous avons montré que l'implémentation de l'algorithme peut souffrir de problèmes numériques et avons proposé une manière de contourner le problème tout en essayant de rester fidèle à l'algorithme initial. Nous avons également comparé les résultats de notre implémentation comparés à ceux du package blockcluster et obtenu des résultats très similaires. Des améliorations sur la visualisation des résultats pourraient être apportées ainsi que des améliorations sur l'implémentation. Notamment, pour éviter les cas où un cluster est "vide", nous pourrions relancer l'algorithme depuis une nouvelle initialisation. Nous avons également testé notre algorithme sur des données réelles. Cette partie semble encore délicate car les résultats n'ont été probants ni avec notre implémentation ni avec blockcluster.