Séminaire de Modélisation Statistique

Complétion de Matrices Binaires

Romain AVOUAC et Bruno BJAÏ
sous la supervision de Vincent COTTET
ENSAE

2017-2018

Introduction

Ce projet, réalisé dans le cadre des séminaires de modélisation statistique de la deuxième année d'ENSAE, vise à implémenter un algorithme de complétion de matrices. Ces méthodes cherchent à compléter des matrices, potentiellement de très grande dimension, à partir d'un nombre restreint de ses entrées observées. Pour ce projet, nous nous intéressons plus particulièrement aux matrices binaires, c'est-à-dire aux matrices composées uniquement de -1 et de 1. Ce genre de méthodes de complétion trouvent des applications dans des domaines variés, la plus connue étant représentée par les algorithmes de recommandations (voir *Netflix challenge*).

Les méthodes de complétions de matrices reposent sur l'hypothèse de faible rang. Pour un algorithme de recommandation, cette hypothèse pourrait s'interpréter comme le fait que chaque spectateur n'est qu'une "combinaison linéaire" d'un faible nombre d'utilisateurs canoniques.

Le rang n'étant pas une fonction convexe, une contrainte est imposée sur la norme nucléaire de la matrice, c'est-à-dire la somme de ses valeurs singulières. La programme d'optimisation en résultant est donné par:

$$\min_M \frac{1}{n} \sum_{i=0}^{n} l(Y_i, \langle X_i, M \rangle) + \lambda \|M\|_*$$

avec

- M la matrice complète (inconnue) ;
- Y_i la matrice contenant seulement les entrées observée de M ;
- X_i la matrice masque, qui ne contient que des 0, à l'exception de son i-ème élément égal à 1, notons que $\langle X_i, M \rangle = M_i$;
- l une fonction de perte ;
- $\|.\|_*$ l'opérateur donnant la norme nucléaire de la matrice ;
- λ le paramètre de pénalisation du critère de norme.

Nous optons pour une fonction de perte l de type logistique, adaptée au cas binaire que nous étudions. La fonction objectif précédente étant convexe, le programme possède une solution. Même si cette dernière ne peut être déterminée de façon analytique, nous pouvons implémenter des méthodes numériques. Après revue de la littérature sur le sujet, nous avons choisi l'algorithme du gradient proximal.

Algorithme du gradient proximal

L'algorithme du gradient proximal peut être implémenté dans les cas où la fonction objectif à minimiser est de la forme :

$$f(z) = g(z) + h(z)$$

avec g convexe et différentiable, et h convexe (mais pas forcément différentiable).

Ici, nous avons opté pour une fonction de perte logistique et une pénalisation par la norme nucléaire, soit:

$$g(M) = \sum_{i=1}^{n} log(1 + exp(-Y_i M_i))$$

et

$$h(M) = \lambda \|M\|_*$$

Gradient de g

Le gradient de la fonction g est donné par:

$$\begin{aligned} \frac{\partial g(M)}{\partial M} &= \sum_{i=1}^{n} \frac{\partial log(1 + exp(-Y_i M_i))}{\partial M} \\ &= \sum_{i=1}^{n} \frac{1}{1 + exp(-Y_i M_i)} \frac{\partial exp(-Y_i M_i))}{\partial M} \\ &= -\sum_{i=1}^{n} \frac{X_i Y_i exp(-Y_i M_i)}{1 + exp(-Y_i M_i)} \end{aligned} \tag{1}$$

Soft-Thresholding

On calcule la matrice de soft-thresholding, qui va nous donner le gradient proximal de la fonction h:

$$\begin{aligned} M &= U \Sigma V^T \\ (\Sigma_{\lambda t})_{ii} &= \max\{\Sigma_{ii} - \lambda, 0\} \\ S_{\lambda t} &= U \Sigma_{\lambda t} V^T \end{aligned} \tag{2}$$

On a $prox_t(B) = S_{\lambda t}(B)$. L'opération de soft-thresholding permet d'annuler les valeurs singulières de la matrice inférieures à λ, c'est une façon d'imposer une contrainte sur le rang (et la norme nucléaire). Le soft-thresholding nécessite de calculer à chaque itération de l'algorithme la décomposition en valeurs

singulières de la matrice, ce qui représente l'élement le plus coûteux en temps de calcul de l'algorithme. Pour limiter ce temps, nous avons utilisé la *fast truncated singular value decomposition* du package irlba, qui permet de limiter très fortement le temps mis par l'algorithme pour converger sans modifier les résultats de manière significative.

Algorithme

L'algorithme itère l'opération suivante:

$$\begin{aligned} M^{(k+1)} &= S_{\lambda t}\left(M^{(k)} - t(\nabla g(M))\right) \\ &= S_{\lambda t}\left(M^{(k)} + t(\sum_{i=1}^{n} \frac{Y_i X_i exp(-Y_i M_i^{(k)})}{1 + exp(-Y_i M_i^{(k)})}\right) \end{aligned} \tag{3}$$

Détermination empirique du λ optimal

Notre algorithme dépend de deux paramètres: t et λ. La littérature suggère que $t = 1$ est une valeur adéquate dans le cadre de la complétion de matrices. Il existe une valeur théorique optimale pour λ, dépendant des paramètres de notre modèle. Toutefois, certains de ces paramètres étant inconnus, nous devons procéder par validation croisée pour estimer la valeur de λ la plus adaptée.

Maximisation du pouvoir de prédiction

La validation croisée étant assez couteuse en calcul, nous avons procédé en deux étapes. Une première approche a consisté à tester notre algorithme avec plusieurs valeur de λ et à comparer ses performances pour déterminer une approximation de la valeur optimale du paramètre.

Pour se faire, nous divisons la matrice que nous cherchons à compléter en deux parties. Une partie (composée de 80 % des entrées observée) nous sert de base d'apprentissage, sur laquelle nous appliquons notre algorithme, et la seconde partie (environ 20 % des entrées observées) est utilisée pour tester le pouvoir prédictif de nos résultat.

A partir d'une partition aléatoire, nous appliquons l'algorithme avec une série de valeurs de λ, et sélectionnons la valeur qui nous permet d'obtenir le taux de prédiction maximal sur notre base de test. Cette méthode permet d'obtenir une valeur approchée de la valeur optimale du paramètre.

Validation croisée

Nous avons également appliqué une validation croisée afin de rendre le choix du lambda optimal plus robuste. Le principe de la validation croisée est d'implémenter notre algorithme sur un sous-échantillon

de notre base de test, en utilisant différentes valeurs de λ (proches des celles obtenues précédemment), et de sélectionner celle qui donne le meilleur taux de prédiction.

Plus précisément:

1. On sélectionne une série de valeur de λ à tester, λ_1, ..., λ_m ;
2. On scinde aléatoirement notre base en K blocs ;
3. Pour chaque bloc k de 1 à K, on calcule notre matrice complète à partir de notre base à laquelle est retirée le bloc k, puis on calcule l'erreur moyenne sur les K blocs:

$$\frac{1}{K}\sum_{k=1}^{K}\left(\sum_{i=1}^{n} log(1+exp(Y_i M_i^k)) + \lambda \left\|M^k\right\|_*\right)$$

4. On répète l'opération précédente pour chaque valeur de λ et on sélectionne celle qui nous donne l'erreur moyenne (sur tous les blocs) la plus faible.

Applications

Test sur données simulées

Une fois notre algorithme terminé, nous avons testé ses performances. Avant de nous confronter à des données réelles, nous avons effectué un premier test sur des données simulées. Pour cela, nous avons généré aléatoirement une matrice binaire de dimension 100x100, dont chaque entrée est le signe du produit de deux gaussiennes centrées réduites. Une telle matrice peut être assimilée à une matrice de faible rang, car deux de ses valeurs propres se détachent nettement.

Pour se rapprocher des cas réalistes de complétion de matrice, nous retirons aléatoirement 80 % des entrées de cette matrice, que nous complétons ensuite à partir de notre algorithme. La validation croisée nous donne une valeur optimale de λ égale à 0.0022. Notons que, après plusieurs essais, λ ne semble pas pouvoir excéder la valeur 0.0032 (auquel cas toutes les valeurs singulières de notre matrices sont annulées par l'algorithme, et ce dernier ne peut donc pas converger). Le λ optimal nous donne un taux de prédiction de 91,9 % (Figure 1).

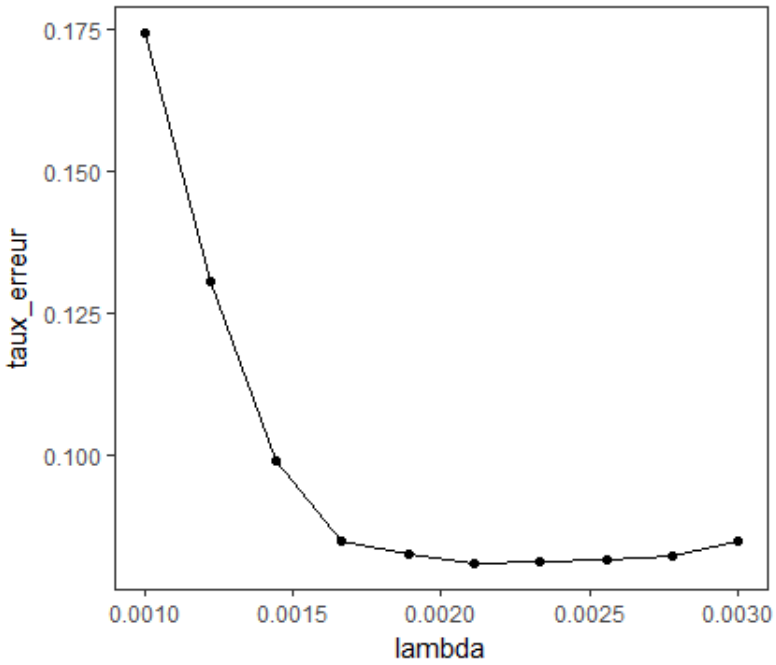

Figure 1: Taux d'erreur en fonction de λ sur une matrice 100*100 simulée

Données Movielens

Nous avons ensuite testé notre algorithme sur des données réelles, issues de la base Movielens 100K. Cette base regroupe des notes données par des utilisateurs à un grand nombre de films. Évidemment, chaque utilisateur n'en ayant vu et noté qu'un très faible nombre, la matrice des notes comporte un grand nombre de valeurs manquantes et se prête donc très bien à l'exercice de la complétion.

Les notes attribuées sont des entiers entre 1 et 5. Pour obtenir une matrice binaire, nous transformons les "bonnes notes" (4 ou 5) en 1 et les "mauvaises" notes en -1. Les notes inobservées sont fixées à 0.

Nous utilisons comme input une matrice d'environ 100 000 entrées observées (943 utilisateurs pour 1682 films). Nous appliquons l'algorithme pour plusieurs sous-matrices de différentes tailles de la matrice complète (pour les obtenir, on échantillonne aléatoirement un nombre n d'individus au sein des données observées). Puis nous calculons le pouvoir prédictif de l'algorithme sur la base de test. Les résultats sont présentés dans la Table 1. Figure 2 représente la validation croisée opérée sur la base complète (943 utilisateurs), pour laquelle on obtient un taux de prédiction de 71.5 %.

Table 1: Résultats de la validation croisée sur movielens 100K (10 valeurs de lambda)

Taille de la matrice	λ^*	Taux de prédiction	Temps de calcul
50*1058	0.00179	64.7 %	19 secondes
500*1595	0.0005	68.5 %	1 minute 30 secondes
943*1682	0.00013	71.5 %	8 minutes 28 secondes

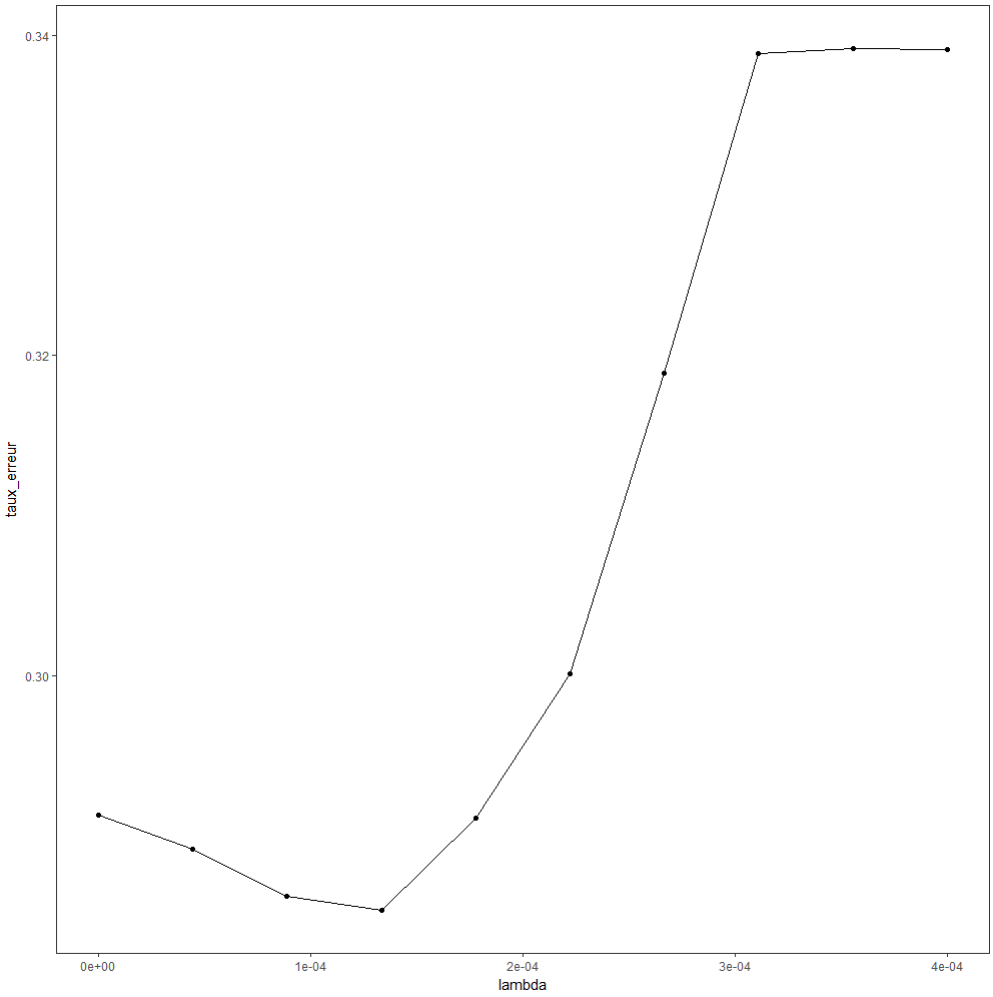

Figure 2: Taux d'erreur en fonction de λ sur la base movielens 100K (10 valeurs de lambda)