Recommandation de Films

Romain Laurent, Félix Breton et Adnan Ben Mansour

Table des matières

1 Aspect théorique

1.1 Problème du Transport Optimal

On se place dans le cadre du transport optimal discret,
on se donne donc deux distributions sur μ sur $\mathcal{X}$ et ν sur $\mathcal{Y}$, et une fonction de coût $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$.

Notre objectif est de trouver une distribution γ sur $\mathcal{X} \times \mathcal{Y}$ dont les distributions marginales sont μ et ν, et on souhaite que γ minimise la quantité :

$$\mathbb{E}_{(X,Y)\sim\gamma}[c(X,Y)] = \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} c(x,y) * \gamma(x,y)$$

On appelle ça aussi le problème de Kantorovitch.

Cette formulation correspond à un problème d'optimisation linéaire avec contraintes linéaires. On peut donc trouver γ avec l'algorithme du simplexe ou des méthodes du point intérieur par exemple.

La distribution γ est alors le plan de transport optimal entre μ et ν, et $\mathbb{E}[c(X,Y)]$ est la distance de Wasserstein entre μ et ν.

Dans la suite on pose $\mathcal{X} = \{x_1, \dots, x_n\}$ et $\mathcal{Y} = \{y_1, \dots, y_m\}$.
Ce qui nous permet de reformuler le problème sous forme matricielle : $C_{i,j} = c(x_i, y_j)$, $\Gamma_{i,j} = \gamma(x_i, y_j)$, $\mu_i = \mu(x_i)$ et $\nu_j = \nu(y_j)$.
L'ensemble des contraintes s'écrit $\Pi(\mu,\nu) = \{\Gamma \in \mathcal{M}_{n,m}(\mathbb{R}^+) \mid \Gamma\mathbb{1}_m = \mu \wedge \Gamma^\top\mathbb{1}_n = \nu\}$ et on cherche à minimiser $\mathrm{tr}(\Gamma^\top C)$ pour $\Gamma \in \Pi(\mu,\nu)$.

1.2 Algorithme de Sinkhorn-Knopp

Afin d'obtenir un algorithme plus rapide, on peut penser à rajouter un terme de régularisation. L'idée va être de chercher à minimiser :

$$\mathbb{E}_{(X,Y)\sim\gamma}[c(X,Y)] - \frac{H(\gamma)}{\lambda}$$

avec les mêmes contraintes que dans le transport optimal.
Ici, λ est un réel strictement positif et H représente une fonction d'entropie définie par $-\sum \gamma(x,y)(\log(\gamma(x,y)) - 1)$.

L'intérêt est à la fois théorique et pratique.
En effet, théoriquement nous avons l'unicité de γ, ce qui n'était pas nécessairement le cas précédemment.
Mais de plus on sait que la matrice Γ s'écrit de la forme $\Gamma = diag(u)Kdiag(v)$ où $K = \exp[-\lambda C]$. Ce qui nous ramène à chercher deux vecteurs de dimensions n et m.

Preuve :
On rappelle qu'on cherche Γ dans $\Pi(\mu,\nu) = \left\{\Gamma \in \mathcal{M}_{n,m}(\mathbb{R}^+) \;\middle|\; \begin{array}{rcl} \Gamma\mathbb{1}_m & = & \mu \\ \Gamma^\top\mathbb{1}_n & = & \nu \end{array}\right\}$ qui minimise

$$f(\Gamma) = \sum_{(i,j)} \Gamma_{i,j}C_{i,j} + \frac{1}{\lambda}\Gamma_{i,j}\left[\log(\Gamma_{i,j}) - 1\right]$$

f est fortement convexe et l'ensemble des contraintes est convexe, on a donc l'unicité. Par ailleurs f est continue et l'ensemble des contraintes est compact donc on a aussi l'existence.
On peut écrire le lagrangien du problème $\mathcal{L}(\Gamma, \alpha, \beta) = f(\Gamma) - \alpha^\top(\Gamma \mathbb{1}_m - \mu) - \beta^\top(\Gamma^\top \mathbb{1}_n - \nu)$ et chercher Γ en écrivant les conditions du premier ordre :

$$C_{i,j} - 1/\lambda \log(\Gamma_{i,j}) - \alpha_i - \beta_j = 0$$

D'où $\Gamma = \exp[-\lambda * (C_{i,j} - \alpha_i - \beta_j)]$ qui correspond bien à la forme attendue.

L'algorithme de Sinkhorn-Knopp permet justement de résoudre cette nouvelle formulation.

1.3 Problème du Transport Optimal Inverse

On peut aussi vouloir, à partir de γ, μ et ν fixés, chercher une fonction de coût c qui redonne γ. C'est le problème du transport optimal inverse.

Sauf qu'en l'occurrence le problème est mal posé, on n'a pas l'unicité.
On va tout de même rajouter un terme de régularisation et réduire l'espace des fonctions de coût, en se limitant à un espace $\mathcal{C}$.
Dans les deux algorithmes que nous utiliserons on fait justement l'hypothèse que $\mathcal{C} = \{GAD \mid A \in \mathbb{R}^{p \times q}\}$.
Plus formellement nous allons minimiser $D_{\mathrm{KL}}(\Gamma \| \Gamma^c) + R(c)$ avec pour contrainte $c \in \mathcal{C}$.

Suivant l'hypothèse que l'on fait sur la matrice C à trouver, on prendra un terme de régularisation $R(c)$ différent.
Si on suppose la matrice de rang faible alors $R(C) = U\max(\mathrm{diag}(S - \zeta), 0)V$,
avec $C = USV$ la décomposition en valeur singulières de C.
Et si on suppose la matrice creuse alors $R(C) = \mathrm{sign}(C) \bigodot \max(|C| - \zeta, 0)$.

Pour cela nous ferons appel à l'algorithme RIOT.

1.4 ν optimal

Dans notre cas on peut aussi disposer de la matrice des coûts C et seulement de la distribution μ. On peut dans ce contexte vouloir chercher ν qui minimise la distance de Wasserstein régularisée entre μ et ν.
Notamment si $\mathcal{X} = \mathcal{Y}$, et C représente une distance dans $\mathcal{X}$ alors sans le terme de régularisation le ν optimal c'est μ lui-même. Mais plus λ tend vers 0, plus on va favoriser des distributions qui prennent plus de valeurs.

En réalité ce ν est donné par la formule : $K^\top(\mu/(K\mathbb{1}_m))$, avec $K = \exp[-\lambda C]$.

Preuve : On a toujours la même fonction $f(\Gamma)$ fortement convexe que pour le transport optimal régularisé, seulement cette fois les contraintes sont plus souples (on garde que la première contrainte) mais encore convexe, et donc on obtient que $\Gamma = diag(u)K$ par un raisonnement similaire.

Notre contrainte s'écrit $\Gamma \mathbb{1}_m = \mu = u \bigodot K\mathbb{1}_m$ donc $u = \mu/(K\mathbb{1}_m)$,
or $\nu = K^\top u$ d'où le résultat.

2 Estimer des vues

2.1 Formalisation du problème

Dans un premier temps on va faire abstraction des notes, et s'intéresser uniquement aux films vus. C'est-à-dire que pour un utilisateur donné, à partir des films qu'il a déjà vu, on aimerait estimer quels autres films il est susceptible d'avoir regardé.
Si on pose $\mathcal{X}$ l'espace des utilisateurs et $\mathcal{Y}$ l'espace des films, on peut imaginer l'existence d'une distribution sur $\mathcal{X} \times \mathcal{Y}$ qui décrit la probabilité qu'un utilisateur $x \in \mathcal{X}$ voit un film $y \in \mathcal{Y}$. Et supposer que les entrées dans la base de données suivent cette distribution (bien qu'en réalité si un film a déjà été vu, la probabilité qu'il soit vu de nouveau est moins importante).
Donc l'objectif qu'on se fixe, c'est qu'à partir d'un utilisateur x, et d'une collection de films qu'il a déjà vu, essayer de déterminer au mieux les autres films qu'il est susceptible d'avoir déjà vus.

2.2 Scores

Pour mesurer à quel point nos algorithmes sont efficaces, on peut regarder les K films ayant la plus forte probabilité d'être vus et calculer la proportion parmi ces films de ceux qu'il a effectivement vus.
On peut aussi prendre K égal au nombre de films vus au total par l'utilisateur en question.

Si on calcule ces scores sur un grand nombre d'utilisateurs, cela nous donne une métrique pour évaluer la fiabilité de notre algorithme.

On peut toujours comparer nos algorithmes à un algorithme naïf qui se contente de redonner les αN films pour lesquels il sait que l'utilisateur les a vus, et rajouter $(\alpha - 1)N$ films aléatoires.

2.3 Idée générale

On part d'une collection d'apprentissage $m = \{(x_1, y_1), \ldots, (x_\omega, y_\omega)\}$, qu'on peut aussi représenter comme une matrice creuse

$$M_{i,j} = \begin{cases} 1/\omega & \text{si} \quad (i,j) \in m \\ 0 & \text{sinon} \end{cases}$$

On considère que c'est un plan optimal pour une certaine matrice de coût C, qu'on peut obtenir avec la résolution du problème de transport optimal inverse pour la matrice d'apprentissage M.

Si maintenant on considère un nouvel utilisateur x, et $\{y_1, \ldots, y_\sigma\}$ une partie des films qu'il a déjà vu. On transforme cette ensemble en un vecteur ν_x,
pour chaque film y, on note μ_y le vecteur correspondant à la y-ème colonne de la matrice M,
le couple (x, y) a alors un coût donné par la distance de Wasserstein entre μ_y et ν_x en utilisant les coûts données par la matrice C.

En prenant les films associés aux coûts les moins élevés on peut proposer un certain nombre de films, et comparer ces propositions aux films que l'utilisateur x a déjà vu ou pas.

2.4 Aspect pratique

À présent on va décortiquer les étapes nécessaires à la mise en place d'un premier algorithme de référence.

On part de la matrice M décrite ci-dessus, on lui applique une décomposition en valeur singulières (SVD) et on retient uniquement les k vecteurs de valeurs singulières maximales. Cela nous donne une décomposition de la forme $M = Gdiag(s)D$ où $s \in \mathbb{R}^k$.

Ensuite on calcule C en utilisant l'algorithme RIOT sur M et la décomposition ci-dessus pour accélérer les calculs. Il suffit ensuite d'utiliser cette matrice C pour calculer les distances de Wasserstein comme indiqué dans l'article.

Malheureusement ces distances sont coûteuses, surtout si on veut évaluer notre algorithme sur un grand nombre de cas pour en mesurer le score.

2.5 Améliorations

2.5.1 Autres factorisations

À la place d'utiliser l'algorithme de décomposition en valeur singulière, on peut utiliser n'importe quelle factorisation de matrice. Notamment la décomposition en produit de matrices positives (NMF, pour non-negative matrix factorization).

2.5.2 Par régularisation

Si on arrive à construire une distance $D \in \mathbb{R}^{m \times m}$ sur les films qui soit suffisamment pertinente,
on peut chercher le vecteur ν qui minimise $W_{\lambda,D}(\nu_x, \nu)$, la distance de Wasserstein régularisée.

Et c'est avec ce vecteur qu'on prédira les films vus par l'utilisateur x, en regardant les films ayant le plus grand coefficient.

De nombreuses méthodes permettent de construire D. On peut par exemple comparer les colonnes dans la matrice M, par exemple avec $D_{i,j} = \|M_{\star,i} - M_{\star,j}\|^2$, ou sinon, on peut appliquer l'algorithme RIOT sur M et comparer les colonnes de C.

2.5.3 Agrégation des utilisateurs

Chacun des utilisateurs que nous avons pris pour entraîner notre modèle ne couvre qu'une petite partie du spectre des films existants. On peut alors penser à agréger les vues de plusieurs utilisateurs.

On partitionne nos utilisateurs en n' groupes, et on remplace tous les utilisateurs de chaque groupe par un utilisateur ayant pour profil la moyenne des profils des utilisateurs du groupe.

3 Estimer des notes

3.1 Formalisation du problème

Cette fois-ci on dispose de la matrice $R \in \mathbb{R}^{n \times m}$ des notes, les notes sont dans l'intervalle $]0, 5]$, un 0 représente donc l'absence de note.
Le problème ici c'est que même si on décide de renormaliser cette matrice R, aucune valeur par défaut pour décrire l'absence de note ne convient réellement. De plus, pour un utilisateur donné, les notes ne sont plus données qu'à un coefficient près. Il faut donc trouver une astuce pour contourner ces deux problèmes.

Notre objectif va être d'estimer les autres notes. Plus exactement, nous allons masquer une certaine partie des notes, et notre objectif sera d'estimer ces notes là au mieux, en calculant le RMSE entre les notes réelles, c'est à dire la partie non nulle de la matrice R, et les notes estimées associées.

3.2 Première méthode

On va rajouter à R un film de référence, et on va supposer que tous les utilisateurs donnent une note de 2.5 pour ce film. Ceci permet de contourner le problème de la normalisation.

Une fois que c'est fait on aimerait normaliser R pour obtenir R' et considérer que c'est un plan optimal pour une fonction de coût C que l'on pourrait calculer avec l'algorithme RIOT.
Sauf que c'est une mauvaise idée car ici les 0 auront un impact beaucoup trop important.
Donc au lieu de factoriser R' en calculant sa SVD, on va plutôt factoriser R' en GD manuellement, avec $G \in \mathbb{R}^{n \times k}$ et $D \in \mathbb{R}^{k \times m}$ de sorte à minimiser l'écart entre R' et GD sur l'ensemble des valeurs non nulles. Pour cela on va faire une descente de gradient.

Pour un nouvel utilisateur x on va de nouveau construire ν la distribution régularisée des notes, et on estimera la note attribuée au film y_j par $\dfrac{2.5\nu_j}{\nu_{m+1}}$. On peut éventuellement appliquer une fonction seuil pour se ramener à des notes comprises entre 0 et 5.

3.3 Deuxième méthode

On peut chercher à détecter si un utilisateur donné va mettre une note supérieure à 2.5 à un film ou non. Et on se retrouve avec un problème similaire à l'estimation des vues.

4 Résultats

Dans les deux cas, que ce soit pour l'estimation de vues ou l'estimation de notes, on a testé nos algorithmes sur deux bases de données différentes de MovieLens, la version avec 1 million d'entrées et la version avec 25 millions d'entrées.
Cependant nous n'avons pas pris tous les films, mais exclusivement les films qui ont été vus au moins un certain nombre de fois. L'objectif étant d'obtenir des matrices de notes $R_{i,j}$ pas trop creuses.

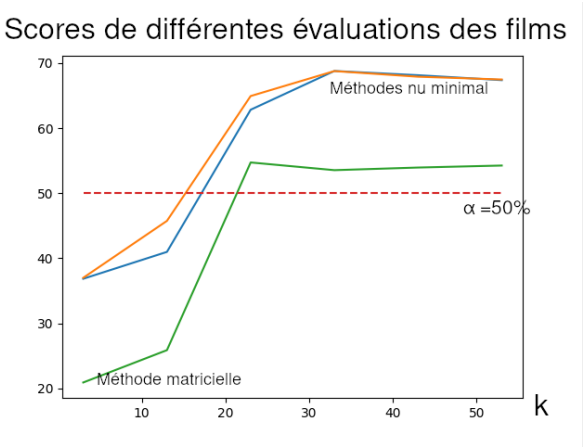
Une fois la matrice R obtenue (ou M dans le cas de l'estimation de vues), on découpe cette matrice en trois groupes : un groupe d'entraînement, un groupe de test et un groupe de validation. En effet nous avons plusieurs hyper-paramètres dans nos différents modèles qu'il sera nécessaire d'ajuster sans compromettre la fiabilité de nos résultats.

4.1 Estimation des vues

Premièrement on a regardé l'influence du paramètre k sur les algorithmes de base. On voit clairement que le score augmente avec k mais qu'il atteint rapidement une valeur limite pour un k autour de 30.

Ensuite on a comparé les manières de comparer les films entre eux pour un utilisateur donné.
k fixé, l'algorithme fixé, mais soit on calcul la distance de Wasserstein, soit on calcul le coût matriciel, soit on calcule le ν-optimal pour différents λ.

Cette fois-ci on compare NMF vs algoi sur la base du pourcentage de films cachés.



4.2 Estimation des notes

On compare simplement nos deux méthodes à l'algorithme naïf. On remarque alors que la seconde méthode n'est pas très efficace, cependant la première donne des résultats intéressants.

5 Annexes

5.1 algorithme de Sinkhorn-Knopp

```
# Algorithme de Sinkhorn-Knopp
def SK_algo ( mu, nu, C, l = 1. ) :
        K = np.exp ( - l * C )
        a = np.ones ( mu.shape )
        for i in range ( N ) :
                b = nu / ( K.T @ a )
                a = mu / ( K @ b )
        return np.diag ( a ) @ K @ np.diag ( b )
```

5.2 algorithme RIOT

```
# Algorithme RIOT (Regularized Inverse Optimal Transport)
def RIOT_algo ( pi, mu, nu, G, D, mode = 0 ) :
        eps = 1e-2
        gamma = 1e-5
        n, m = mu.shape[0], nu.shape[0]

        Gmp = np.linalg.pinv ( G )
        Dmp = np.linalg.pinv ( D )

        c = np.random.rand ( n, m )
        u = np.exp ( np.random.rand ( n ) / eps )
        v = np.exp ( np.random.rand ( m ) / eps )

        for i in range ( N ) :
                K = np.exp ( -c/eps )
                u = mu / ( K @ v )
                v = nu / ( K.T @ u )
                K = pi / ( np.outer ( u, v ) ) + 1e-9
                cX = - eps * Gmp @ np.log ( K ) @ Dmp
                if mode == 0 :
                        U, S, V = np.linalg.svd ( cX, full_matrices = False )
                        a = U @ np.maximum ( np.diag ( S - gamma ), 0 ) @ V
                elif mode == 1 :
                        a = np.sign ( cX ) * np.maximum ( np.abs ( cX ) - gamma, 0 )
                else :
                        print ( "RIOT mode not supported..." )
                        exit ( 1 )
                c = G @ a @ D
        return c, a
```

Références

[Li+18] Ruilin Li et al. “Learning to Recommend via Inverse Optimal Matching”. In : (fév. 2018).

[PC19] Gabriel Peyré et Marco Cuturi. “Computational Optimal Transport”. In : *Foundations and Trends in Machine Learning* 11.5-6 (2019), p. 355-607.

[Sun+20] Haodong Sun et al. *Learning Cost Functions for Optimal Transport*. 2020. arXiv : 2002.09650 [cs.LG].