Projet

12 avril 2017

Résumé

A l'ère du numérique, la collecte de données d'une multitude d'utilisateurs peut devenir très intéressante pour leur faire des suggestions personnalisées de tout type. C'est une chose qu'on rencontre souvent : recommandations d'amis sur Facebook, suggestions d'achats sur Internet, etc.

Nous avons choisi de nous intéresser à la recommandation automatique de films. En effet la recommandation automatique d'un film obéit à plusieurs critères Ajout. Comment concevoir un programme informatique qui prenne en compte tous ces critères pour produire une recommandation personnalisée pertinente ? Les méthodes proposées lors de ce projet consistent à estimer les notes que mettrait l'utilisateur, afin de trouver les films les plus accordés à ses goûts. Cela se fait en regardant les prédictions des notes les plus hautes. Deux méthodes sont proposées. Pour la première, on suppose qu'il existe une relation linéaire entre l'ensemble des notes données par un utilisateur et la note qu'on cherche à prévoir. Cette méthode consiste à trouver cette relation. Pour ce faire nous utilisons un algorithme, qui sera explicitée par la suite. Nous proposons aussi un second modèle que nous comparons au premier. Dans celui ci, nous supposons que chaque film possède des caractéristiques et chaque utilisateur des préférences, toutes deux liées linéairement. La deuxième méthode consiste donc à déterminer ces préférences et caractéristiques à l'aide d'une utilisation plus avancée de la descente du gradient.

1 Notre approche du problème

1.1 Nos données

1.1.1 Extraction des données

Notre projet nous a forcément mener à posséder des données sur des films déjà vu et noté par quelques utilisateurs. Cela nous à guider vers une grande base de données appelé " movielens " qui est un site communautaire de recommandation de films où les utilisateurs du site notent des films de 1 à 5. Plusieurs jeux de données étaient disponibles et différé entre eux selon leur taille, nous avons choisi de travailler avec une base de données de 670 utilisateurs et 9125 films. On a donc extrait 2 fichiers : l'un contenant les 9125 films avec leurs titres et un numéro attribué , l'autre avec les notes des utilisateurs qui était avaient chacun un identifiants, le fichier était du type : identifiant de l'utilisateur, id du film qu'il a noté, note. Ces 2 fichiers étaient donc peu pratiques pour commencer à faire quelque chose avec, nous avons donc créer une fonction

tableau_des_notes() qui permet de ranger toutes ces notes dans un tableau numpy 9125*670 avec en lignes les utilisateurs, et en colonnes les films. Quand un utilisateur n'a pas vu un film donc qu'il ne l'a pas noté, on insère un " Nan "(Not a Number) qui est un " symbole " facile à traiter. On appellera ce tableau Y tout au long du projet.

(image de Y)

Pour continuer notre projet a bien, nous avons étudié nos données pour mieux les traiter plus tard.

1.1.2 Analyse des données

Dans la logique des choses, notre tableau Y étant bien trop grand pour en tirer des conclusions juste à l'œil, nous avons créer des fonctions permettant de faire des sortes de statistiques sur nos données. Nous avons créer quelques fonctions pour étudier nos données : l'une permettant de compter combien de films chaque utilisateur a vu (cf nbre_de_films_vu_par_utilisateur), et l'autre comptant combien de fois avait été noté chaque film (cf nbre_de_notes_par_film). De ces fonctions, nous avons pu tirer des graphiques nous montrant comment été répartis nos données.

On a remarquer que la grande majorité des films posséder entre 0 et 40 notes(Plus de 8000 sur 9000). Le nombre de notes le plus récurrents étant 2 notes par film. Mais il y a tout de même quelque film qui ont été beaucoup vu sachant que celui qui avait le plus de note en avait 339.

(histogramme nbre de note associés au film)

Dans l'autre sens, en moyenne les utilisateurs ont noté ... films, et la majorité est répartis entre blabla et babla.

(autre histogramme)

conclusion sur les données ce qu'on peut en tirer ... Notre tableau présente 98,4 % de NaN. Cela nous fait donc plus de 6 millions de notes à prédire... Dans la prochaine partie, nous verrons donc la méthode permettant de prédire toutes ces notes à partir de très peu de données.

1.2 Parler de la factorisation de Y : partie de Anthony lors de la présentation orale des POQ

Expliquer cela avec un exemple simple et des figures de Yf, theta et X

2 Implémentation de l'algorithme

2.1 Introduction de la fonction de coût (et pourquoi on l'a introduite) avec la norme de frobenius

Dire que notre but c de minimiser cette fonction en trouvant theta et X qui représentent le mieux possible nos données Explications de Yassine aux POQ sur le fait qu'on fixe aléatoirement theta et X et on les bouges l'un apres l autre plein de fois en faisant diminuer J au fur et a mesure

2.2 C la que l on parle du principe de la descente du gradient avec les dérivées

partielles et tt ça : explication de la fonction étape du gradient et descente du gradient non optimisées (sans le gradient) : ce qu'on faisait avant que Valentin nous montre le gradient et toutes ses supers formules au tableau qu on avait pris en photo Fig avec un bol comme dans le mooc

3 optimisation

3.1 Notations (on pourra mettre cette partie autre part)

À remplir et dire aussi que l'on represente par des matrices colonnes les vecteurs et la notation Y y et X x et $y_{.,j}$ preciser qu on travaille avec des matrices réelles

3.2 Ancien algorithme

Une étape :
Faire simultanément :
Pour tout j $\in$ {1, 2, ..., nf} :
Pour tout k $\in$ {1, 2, ..., n} :
Affecter à $x_{j,k}$ la valeur $x_{j,k} - \alpha \frac{\partial J}{\partial x_{j,k}}$
Faire simultanément :
Pour tout i $\in$ {1, 2, ..., nu} :
Pour tout k $\in$ {1, 2, ..., n} :
Affecter à $\theta_{i,k}$ la valeur $\theta_{i,k} - \alpha \frac{\partial J}{\partial \theta_{i,k}}$

Cet algorithme possède deux fois deux boucles imbriquées ce qui le rend très lent : 30 secondes par étapes avec 10 caractéristiques sachant qu'il faut au moins 500 étapes pour arriver à une approximation de Y correcte. C'est pourquoi nous voulons l'optimiser.

3.3 Utilisation du gradient

Sachant que $\forall j \in \{1, 2, ..., nf\}$ x_j est une matrice de dimension $1 * n$, nous pouvons remarquer que faire simultanément :

Pour tout j $\in$ {1, 2, ..., nf} :
Pour tout k $\in$ {1, 2, ..., n} :
Affecter à $x_{j,k}$ la valeur $x_{j,k} - \alpha \frac{\partial J}{\partial x_{j,k}}(\theta, X)$

reviens à faire simultanément :

Pour tout j $\in$ {1, 2, ..., nu} :
$x_j := x_j - \alpha \left(\frac{\partial J}{\partial x_{j,1}}(\theta, X) \quad \frac{\partial J}{\partial x_{j,2}}(\theta, X) \quad \cdots \quad \frac{\partial J}{\partial x_{j,n}}(\theta, X) \right)$

Or, $\left(\frac{\partial J}{\partial x_{j,1}}(\theta, X) \quad \frac{\partial J}{\partial x_{j,2}}(\theta, X) \quad \cdots \quad \frac{\partial J}{\partial x_{j,n}}(\theta, X) \right) = (\nabla_{x_j^T} J(\theta, X))^T$

Et il n'y a ainsi plus de boucles imbriquées. Nous nous posons maintenant la question comment faire une fonction renvoyant rapidement $(\nabla_{x_j^T} J(\theta, X))^T$ afin de pouvoir utiliser cette astuce.

Nous avons :

$$\begin{aligned} J(\theta, X) &= \frac{1}{2} \sum_{\substack{i,j \\ y_{i,j} \neq nan}} (\theta_i x_j^T - y_{i,j})^2 \\ &= \frac{1}{2} \sum_{k=1}^{nf} \|\tilde{\theta} x_k^T - \tilde{y}_{.,k}\|^2 \end{aligned}$$

Qu'est ce que $\tilde{y}_{.,k}$ et $\tilde{\theta}$? En fait $\tilde{y}_{.,k}$ est le vecteur $y_{.,k}$ auquel nous avons enlevé les composantes nan : par exemple si $y_{.,k} = \begin{pmatrix} 2 \\ 3 \\ nan \\ 5 \\ nan \end{pmatrix}$ alors $\tilde{y}_{.,k} = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$. Et si nous avons pour cela retiré les lignes $l_1, l_2, \cdots$ et lu de y, nous retirons les mêmes lignes de θ pour former $\tilde{\theta}$. D'où $\tilde{\theta}$ dépend de $y_{.,k}$ et donc de k.

Et alors, nous en déduisons, $\forall j \in \{1, 2, ..., nf\}$:

$$\nabla_{x_j^T} J(\theta, X) = \begin{pmatrix} \frac{\partial \sum_{k=1}^{nf} \frac{1}{2} \|\tilde{\theta} x_k^T - \tilde{y}_{.,k}\|^2}{\partial x_{j,1}^T} \\ \frac{\partial \sum_{k=1}^{nf} \frac{1}{2} \|\tilde{\theta} x_k^T - \tilde{y}_{.,k}\|^2}{\partial x_{j,2}^T} \\ \vdots \\ \frac{\partial \sum_{k=1}^{nf} \frac{1}{2} \|\tilde{\theta} x_k^T - \tilde{y}_{.,k}\|^2}{\partial x_{j,n}^T} \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{nf} \frac{1}{2} \frac{\partial \|\tilde{\theta} x_k^T - \tilde{y}_{.,k}\|^2}{\partial x_{j,1}^T} \\ \sum_{k=1}^{nf} \frac{1}{2} \frac{\partial \|\tilde{\theta} x_k^T - \tilde{y}_{.,k}\|^2}{\partial x_{j,2}^T} \\ \vdots \\ \sum_{k=1}^{nf} \frac{1}{2} \frac{\partial \|\tilde{\theta} x_k^T - \tilde{y}_{.,k}\|^2}{\partial x_{j,n}^T} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{2} \frac{\partial \|\tilde{\theta} x_j^T - \tilde{y}_{.,j}\|^2}{\partial x_{j,1}^T} \\ \frac{1}{2} \frac{\partial \|\tilde{\theta} x_j^T - \tilde{y}_{.,j}\|^2}{\partial x_{j,2}^T} \\ \vdots \\ \frac{1}{2} \frac{\partial \|\tilde{\theta} x_j^T - \tilde{y}_{.,j}\|^2}{\partial x_{j,n}^T} \end{pmatrix} = \nabla_{x_j^T} \frac{1}{2} \|\tilde{\theta} x_j^T - \tilde{y}_{.,j}\|^2$$

Et nous pouvons calculer $(\nabla_{x_j^T} \|\tilde{\theta} x_j^T - \tilde{y}_{.,j}\|)$ grâce à une formule qui est prouvé en annexe : $\forall n \in \mathbb{N}^* \forall p \in \mathbb{N}^*$ si A est une matrice de dimension $p * n$, x une matrice colonne de dimension $n * 1$ et b une matrice colonne de dimension $p * 1$ alors on a $\nabla_x \|Ax - b\| = A^T(Ax - b)$.

D'où, $\nabla_{x_j^T} \|\tilde{\theta} x_j^T - \tilde{y}_{.,j}\| = \tilde{\theta}^T(\tilde{\theta} x_j^T - \tilde{y}_{.,j}) = \nabla_{x_j^T} J(\theta, X)$.

Et ceci est rapidement calculable grâce aux fonctions de la bibliothèque Numpy que nous utilisons pour notre programme.

Un raisonnement similaire amène à faire de même avec la partie de l'ancien algorithme modifiant θ ce qui nous donne ce nouvel algorithme beaucoup plus rapide :

Une étape :
    Faire simultanément :
        Pour tout j $\in \{1, 2, ..., nf\}$:
            Affecter à $x_{j,k}$ la valeur $x_{j,k} - \alpha(\tilde{\theta}^T(\tilde{\theta} x_j^T - \tilde{y}_{.,j}))^T$
    Faire simultanément :
        Pour tout i $\in \{1, 2, ..., nu\}$:
            Affecter à $\theta_{i,k}$ la valeur $\theta_{i,k} - \alpha(\tilde{X}^T(\tilde{X}\theta_i^T - \tilde{y}_i))^T$

4 Mise en place de la recommandation pour un utilisateur spécifique

5 conclusion

A Preuve

$$[\nabla_x \frac{1}{2}||Ax-b||_2^2]_j = [\nabla_x \frac{1}{2}(Ax-b).(Ax-b)]_j$$

$$[\nabla_x \frac{1}{2}||Ax-b||_2^2]_j = [\nabla_x \frac{1}{2}(\sum_{i=1}^{n}(\sum_{k=1}^{p} A_{i,k}x_k)^2 - 2b_i(\sum_{k=1}^{p} A_{i,k}x_k) + b_i^2)]_j$$

$$[\nabla_x \frac{1}{2}||Ax-b||_2^2]_j = \frac{\partial \frac{1}{2}(\sum_{i=1}^{n}(\sum_{k=1}^{p} A_{i,k}x_k)^2 - 2b_i(\sum_{k=1}^{p} A_{i,k}x_k) + b_i^2)}{\partial x_j}$$

$$[\nabla_x \frac{1}{2}||Ax-b||_2^2]_j = \frac{1}{2}(\sum_{i=1}^{n} 2(\sum_{k=1}^{p} A_{i,k}x_k)A_{i,j} - 2b_i A_{i,j})$$

$$[\nabla_x \frac{1}{2}||Ax-b||_2^2]_j = (\sum_{i=1}^{n}(\sum_{k=1}^{p} A_{i,k}x_k)A_{i,j} - b_i A_{i,j})$$

$$[\nabla_x \frac{1}{2}||Ax-b||_2^2]_j = (\sum_{i=1}^{n} A_{i,j}(\sum_{k=1}^{p} A_{i,k}x_k) - b_i)$$

$$[\nabla_x \frac{1}{2}||Ax-b||_2^2]_j = \sum_{i=1}^{n} A_{j,i}^T[Ax-b]_i$$

$$[\nabla_x \frac{1}{2}||Ax-b||_2^2]_j = [A^T(Ax-b)]_j$$