TP5
Système de recommandation de films

Romain Raveaux

Objectif

Sensibiliser les étudiants aux techniques de recommandations automatiques de films utilisées dans les systèmes multimédia et par les diffuseurs de contenu comme Netflix et Youtube.

Mot clés : Régression, Prédicteurs, Reconnaissances des formes

Préambule

Netflix est une entreprise de location de DVD qui a commencé ses activités en 1997. Au début, les DVD loués étaient envoyés par courrier aux clients. Ceux-ci pouvaient garder leur DVD pendant une durée indeterminée. Une fois le film regardé, les clients renvoyaient le DVD par courrier, et indiquaient dans l'enveloppe le prochain film souhaité.
En 2008, Netflix est passé au streaming. En 2013, Netflix a 23 millions de clients. Progressivement, Netflix s'est rendu compte que recommander des films aux utilisateurs augmentait les ventes. Netflix a donc eu l'idée d'intégrer un système de recommandation automatique, sans expert : on recommande un film à un utilisateur suivant les films qu'il a aimés et les films qu'il n'a pas aimés. Netflix conçoit un premier algorithme de recommandation appelé Cinematch.
Afin de mesurer les performances d'un tel algorithme, on utilise la mesure d'erreur RMSE : Root mean square error. L'erreur e définie par cette mesure est :

$$e = \sqrt{\sum_{r_{u,i} connu} \frac{(r_{u,i} - \hat{r}_{u,i})^2}{C}}$$

où C est le nombre de votes ru,i connus dans les données de test.
Dans le but d'améliorer les performances de son système de recommandation, Netflix organise un concours de programmation. À la clé, une récompense de 1000000 $. Les règles du jeu sont les suivantes : les participants se voient confier un ensemble de votes d'utilisateurs de Netflix sur des films (100 millions de votes). Les participants doivent alors soumettre leur propre algorithme de recommandation, et améliorer l'algorithme Cinematch. Si une amélioration d'au moins 10% de l'erreur est constatée, le prix est remporté. Les utilisateurs ont accès à un ensemble de données de test (1,4 millions de votes) sur lequel ils peuvent constater les performances de leur algorithme. Ils peuvent ensuite soumettre leur algorithme à Netflix qui leur renvoie leurs résultats sur l'ensemble Quiz, privé, gardé par Netflix (1,4 millions de votes). L'évaluation finale se fait sur un dernier ensemble de test, gardé secret (1,4 millions de votes). Le problème est donc le suivant : À partir d'un ensemble de notes données par des utilisateurs sur des films,

prédire pour chaque couple (utilisateur,film) qui n'est pas connu la note que va donner cet utilisateur au film. Pour gagner, il faut minimiser l'erreur RMSE. L'algorithme naïf qui consiste à renvoyer la moyenne des votes de tous les utilisateurs sur un film i comme valeur de prédiction $\hat{r}_{u,i}$ obtient une erreur e = 1.0540. L'algorithme Cinematch obtient un score e = 0.9514. En Septembre 2009, Netflix récompense l'équipe «BellKor's pragmatic chaos», qui a réalisé un score de 0.8553, soit une amélioration de 10.09 % sur l'algorithme de Netflix. Dans la suite, nous allons développer les solutions à ce genre de problèmes de collaborative filtering. Modèle de voisinage : on recommande un utilisateur suivant les utilisateurs qui ont un profile voisin du sien Modèle des facteurs latents : on cherche une structure cachée de petite dimension qui modèlise bien les votes que l'on obtient

Les données : http://www.grouplens.org/

Les données se trouvent dans l'archive ml-100k.zip.
http://files.grouplens.org/datasets/movielens/ml-100k.zip

Jetez un coup d'œil au fichier readme.
Le fichier u.data contient les informations dont nous avons besoin :
User id | film id | score |timestamp
196 242 3 881250949
Le caractère de séparation est la tabulation.
Il y a 943 utilisateurs et 1682 films. 100000 votes.

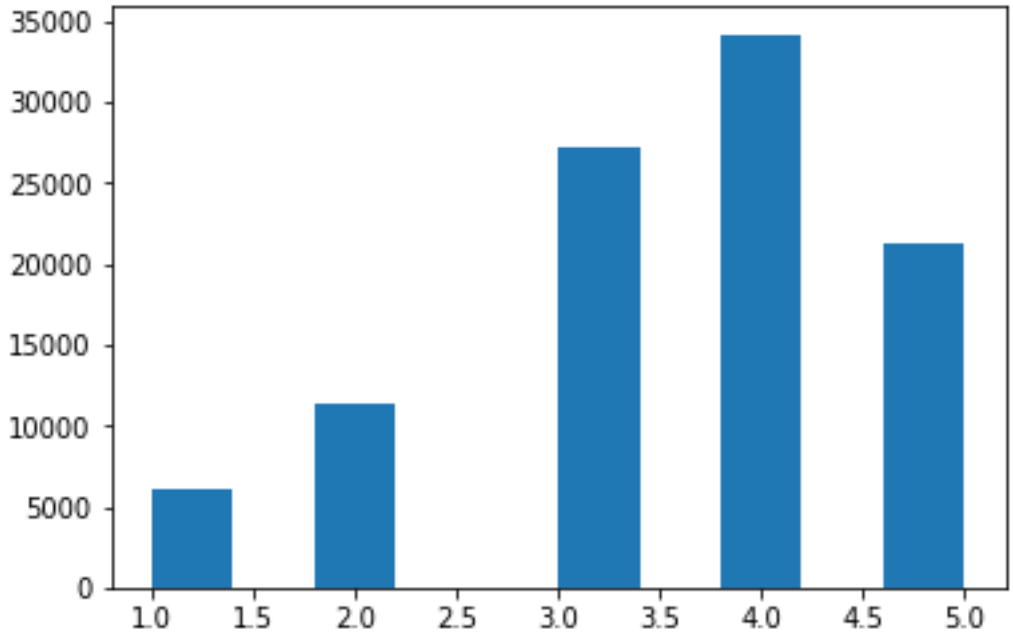

Figure 1Distribution des votes. Moyenne = 3.52

Les identifiants des utilisateurs sont numérotés à partir de 1.
Parser les données et créer une matrice de recommandation où les lignes sont des utilisateurs et les colonnes les films. Si pour un couple (utilisateur, film) il n'y a pas de valeur alors mettez -1 dans la cellule.
Exemple de lecture d'un fichier texte en python :

```
fichier = open(chemin, 'rU')
listeNoeud  = fichier.readlines()
# lecture ligne a ligne
for line in listeNoeud:
        lineSplit = line.split("\t")
```

La fonction de lecture des données peut ressembler à cela :

```
def readdata(file,nbuser,nbfilm):
   res=numpy.ones((nbuser, nbfilm))
   res = res*-1
   fichier = open(file, 'rU')

   lignes  = fichier.readlines()



   # lecture ligne a ligne

   for line in lignes :

      #print(line)
      lineSplit = line.split("\t")
      userid=int(lineSplit[0])-1
      filmid=int(lineSplit[1])-1
      score=int(lineSplit[2])
      #print(str(userid)+" "+str(filmid)+" "+str(score))
      res[userid,filmid]=score
   return res
```

Calculer l'erreur RMSE pour un prédicteur aléatoire.
Le score aléatoire pourra se calculer avec la fonction random :
scorealea=1+ (numpy.random.random()*4)

Prédicteur basique

On note r¯ la moyenne de toutes les notes dont on dispose, N le nombre d'utilisateurs, M le nombre de films, Vu le nombre de votes de l'utilisateur u que l'on a dans notre base de données, Vi le nombre de votes pour le film i que l'on a dans notre base de données.
Dans le modèle basique, on suppose que chaque utilisateur u note tous les films qu'il a regardé avec un écart à la moyenne bu qui lui est propre. Aussi, chaque film a un écart à la moyenne qui lui est propre bi . On obtient alors un prédicateur r^ des notes attribuées par chaque utilisateur à chaque film tel que

$$\hat{r}_{u,i} = \bar{r} + b_u + b_i$$

Remarque : On pourrait prendre :

$$b_u = \frac{\sum_i r_{u,i}}{V_u} - \bar{r}$$

$$b_i = \frac{\sum_u ru, i}{V_i} - \bar{r}$$

Implémenter le prédicteur basique et calculer sa valeur de RMSE.

Méthode du voisinage

Avec cette méthode, on cherche à établir une notion de similarité entre utilisateurs pour pouvoir prédire la note que va mettre l'utilisateur u au film i suivant les notes qu'ont mises les utilisateurs similaires à u.

on note $\tilde{r}_{u,i} = r_{u,i} - (\bar{r} + b_u + b_i)$ et $\tilde{R} = (\tilde{r}_{u,i})_{u,i}$)

Définition 7.3.1 (Score de similarité) *Soient $\tilde{r}_i$ et $\tilde{r}_j$ deux colonnes de $\tilde{R}$ associées aux films i et j. On définit le score de similarité $s_{i,j}$ entre les deux films i et j comme étant la valeur :*

$$s_{i,j} = \frac{\tilde{r}_i^t \tilde{r}_j}{\|\tilde{r}_i\|_2 \|\tilde{r}_j\|_2} = \frac{\sum_u \tilde{r}_{u,i} \tilde{r}_{u,j}}{\sqrt{(\sum_u r_{u,i}^2)(\sum_u r_{u,j}^2)}}$$

en ne sommant bien sûr la valeur $r_{u,i}$ que si on la connaît.

On note $S = (s_{i,j})_{i,j}$. Pour un film i donné, on classe les autres films i par ordre décroissant suivant $|s_{i,j}|$. On prend ensuite les L premiers voisins suivant ce classement comme voisins du film i, dont l'ensemble est noté $\mathcal{L}_i$.

Ainsi, la note prédictive de l'utilisateur u sur le film i sera la note du prédicateur basique plus une somme pondérée par $w_{i,j}$ des films voisins.

Un choix naturel est $w_{i,j} = s_{i,j}$. Le nouveau prédicateur s'écrit donc

$$\tilde{r}_{u,i} = (\bar{r} + b_u + b_i) + \frac{\sum_{j \in \mathcal{L}_i} s_{i,j} \tilde{r}_{u,j}}{\sum_{j \in \mathcal{L}_i} |s_{i,j}|}$$

Pour implémenter ce prédicteur vous allez avoir besoin de faire un tri. Voici le code python pour trier des éléments en gardant en mémoire l'index original de l'objet.
tuple=sorted(enumerate(vec), key=lambda x: x[1])
tuple.reverse()

Chaque tuple contient 2 informations : id et valeur
Pour gagner du temps en phase de développement, il possible de réduire la taille du jeu de données comme suite :

```
print('reducing the number of element by 6)
nbuser=int(943/6)
nbfilm=int(1682/6)
matrice=matrice[0:nbuser,0:nbfilm]
nbscore=computenbvote(matrice,nbuser,nbfilm)

def computenbvote(matrice,nbuser,nbfilm):
   nbvote=0
   for i in range(0,nbuser):
      for j in range(0,nbfilm):
         val=matrice[i,j]
         if(val != -1):
            nbvote=nbvote+1

   return nbvote
```

Implémenter le prédicteur de voisinage et calculer sa valeur de RMSE. On choisira L=10.
Conclure en comparant les différents prédicteurs.

Prédicteur basique par régression linéaire : résolution analytique

Trouver les vecteurs b_u et b_i qui minimiser l'erreur quadratique (donc la RMSE) ?

$$\min_{b_u,b_i} \sum_{u,i} (r_{u,i} - \hat{r}_{u,i})^2$$

Ceci est un problème standard de minimisation des moindres carrés : On cherche à minimiser $\|Ab-c\|_2^2$ suivant b, connaissant la matrice A et le vecteur c avec $b \in \mathbb{R}^{N+M}$, $A \in 0,1^{C\times(N+M)}$, et $c \in \mathbb{R}^C$ où C est le nombre de votes dont on dispose.

On a alors

$$\|Ab - c\| = b^t A^t Ab - 2b^t A^t c + c^t c$$

que l'on dérive suivant b. On cherche ensuite un point d'annulation de la dérivée (pour trouver le minimum), et on trouve :

$$A^t Ab = A^t c$$

Si les colonnes de A sont indépendantes, alors $A^t A$ est définie positive et on peut retrouver $b = (A^t A)^{-1} A^t c$.

Utiliser le prédicteur basique pour construire une matrice R_{pred} (avec toutes les prédictions même pour les valeurs connues).
Pour le $k^{ième}$ vote connu ($r_{u,i}$ connu), construisez le vecteur A_k=[R_{pred}[i, :],R_{pred}[:,u].T]
A la matrice A rajouter une colonne remplie de 1 afin de simuler une entrée à 1. b contiendra donc (N+M+1) paramètres le dernier paramètre étant un biais (« ordonnée à l'origine).

b est facilement calculable avec la bibliothèque Numpy (la fonction « inv »pour inverser une matrice et la fonction « dot » pour faire des produits matriciels)

Prédicteur basique par régression linéaire : résolution par la descente de gradient

Lorsque la matrice A est grande la résolution analytique de la régression linéaire au sens des moindres carrés est impossible. Il est alors possible de résoudre le problème grâce à la méthode de la descente de gradient. Pour cela nous avons besoin d'exprimer la dérivée de la fonction d'erreur par rapport à b. Si maintenant on appelle *L*=||Ab-C|| la fonction d'erreur (la somme des carrés des erreurs) alors la dérivée s'écrit ainsi :

$$\frac{\partial L}{\partial b} = A^T Ab - 2A^T c$$

Data: $\#iter$ is the maximum number of iterations
Data: α learning rate
Result: Learned b. A weight vector
1 $b \leftarrow 0$ and $t \leftarrow 0$
2 **while** $t <$ $\#iter$ **do**
3 $\quad \widehat{c} \leftarrow A.b$
4 $\quad b(t+1) \leftarrow b(t) - \alpha \frac{\partial L}{\partial b}$
5 $\quad$ error$\leftarrow \|c - \widehat{c}\|_2^2$
6 $\quad$ t $\leftarrow$ t $+1$
7 **end**

Algorithm 1: Learning by gradient descent

Remerciement et source :

Ce TP se fonde sur le cours de Marc Lelarge pour la description des modèles.
Source : http://www.di.ens.fr/~lelarge/soc/cours/cours7.pdf