Recommandation-Systeme

December 1, 2019

1 IFT 599/799 – Science des données

1.1 TP4 : Recommandation par filtarge collaboratif

Ce TP porte sur le développement d'une application du filtrage collaboratif pour construire un système de recommandation. Les données à utiliser sont celles fournies par le MovieLens 100K. Le but de ce TP est de se familliariser avec une méthode de recommandation de base et la méthode de crossvalidation pour évaluer la performance d'un système d'apprentissage.

[1]:
```
## Dans un premier temps, il est nécessaire d'importer les différente␣
↪librairies utilisées
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import warnings
warnings.filterwarnings('ignore')
```

1.1.1 Introduction : Compréhension des données

[2]:
```
# Afin de comprendre mieux nos données, on regarde movieLens-Dataset
data_path = 'MovieLens-Dataset/ml-100K/'
columns =␣
↪['movie_id','movie_title','release_date','video_release_date','IMDb_URL',
          ␣
↪'nknown','Action','Adventure','Animation','Children\'s','Comedy','Crime','Documentary',
          ␣
↪'Drama','Fantasy','Film-Noir','Horror','Musical','Mystery','Romance','Sci-Fi',
           'Thriller','War','Western']
u_item = pd.read_csv(os.path.join(data_path, 'u_item.csv'),
                     sep='|', encoding='latin-1', names=columns).drop(
                    labels=['IMDb_URL', 'video_release_date'], axis=1)
u_item.head(3)
```

[2]:

	movie_id	movie_title	release_date	nknown	Action	Adventure	\
0	1	Toy Story (1995)	01-Jan-1995	0	0	0	

```
1          2   GoldenEye (1995)  01-Jan-1995       0       1          1
2          3  Four Rooms (1995)  01-Jan-1995       0       0          0

   Animation  Children's  Comedy  Crime  …  Fantasy  Film-Noir  Horror  \
0          1           1       1      0  …        0          0       0
1          0           0       0      0  …        0          0       0
2          0           0       0      0  …        0          0       0

   Musical  Mystery  Romance  Sci-Fi  Thriller  War  Western
0        0        0        0       0         0    0        0
1        0        0        0       0         1    0        0
2        0        0        0       0         1    0        0

[3 rows x 22 columns]
```

On remaque qu'un item possède 22 caractéristiques, à savoir son titre, sa date de sortie, mais également, sous la forme d'un "one-hot" vector, le type de film dont il s'agit, qui peut être :'unknown','Action','Adventure','Animation','Children's','Comedy','Crime','Documentary', 'Drama','Fantasy','Film-Noir','Horror','Musical','Mystery','Romance','Sci-Fi', 'Thriller','War' ou'Western'. Nous ne nous servirons pas de ce jeu de données à la suite de ce TP, il nous sert uniquement à comprendre ce que contient le dataset de l'exercice.

[3]:
```
# Désormais on regarde les données de notation
u_data = pd.read_csv(os.path.join(data_path, 'u_data.csv'),
                     sep='\t', header=None,
                     names=['user_id','item_id','rating','timestamp'])
# On supprime la caractéristique "timestamp" qui ne nous apporte pas␣
 ↪d'informations
# pour la recommandation
u_data = u_data.drop(labels=['timestamp'], axis=1).sort_values('item_id')
u_data.head(10)
```

[3]:
```
       user_id  item_id  rating
25741       84        1       2
93639      806        1       4
55726      768        1       5
49529       92        1       4
89079      419        1       4
80525      403        1       4
14401      322        1       2
99282      709        1       4
35188      120        1       4
29031      514        1       5
```

On remarque que nos données, après avoir supprimé la caractéristique "timestamp", comprend l'id d'un utilisateur, l'item que cet utilisateur a considéré et la note associée.

1.1.2 Système de recommandation en se basant sur la méthode par voisinage

Nous avons effectué le choix pour ce TP de développer un système de recommandation basé sur la méthode par voisinage, qui consiste à choisir les k voisins d'un utilisateur et de considérer la moyenne de la note donnée par ces voisins sur un item. Une première fonction permet le calcul des moyennes et écart-types des notes données par chaque utilisateur. Cela sera très utile pour la prédiction de la note finale.

[17]:
```
# Afin d'augmenter la vitesse de calcul, on créé un nouveau dataframe contenant:
 ↪
# id_user, note_moyenne_donnée, écart-type des notes données
def get_u_ids(u_data):
    u_ids = u_data.drop_duplicates('user_id', keep='first')
    mean = []
    std = []
    for index, row in u_ids.iterrows():
        mean.append(u_data[u_data['user_id']==row['user_id']].
 ↪mean(axis=0)['rating'])
        std.append(u_data[u_data['user_id']==row['user_id']].
 ↪std(axis=0)['rating'])
    u_ids.insert(1, "std", std)
    u_ids.insert(1, "mean", mean)
    u_ids = u_ids.drop(labels=['item_id','rating'], axis=1)
    return u_ids
u_ids = get_u_ids(u_data)
```

[87]:
```
def pearson_correlation(id_indiv_1, id_indiv_2, u_data, gamma):
    # On cherche quels items ont été noté par les 2 personnes:
    indiv_1 = u_data['user_id'] == id_indiv_1
    indiv_2 = u_data['user_id'] == id_indiv_2
    filter_indiv = ( indiv_1 | indiv_2 )
    df_individus = u_data[filter_indiv]
    items_in_common = df_individus[df_individus.
 ↪duplicated('item_id',keep=False)]
    if items_in_common.empty:
        return 0
    else:
        mean_rate_1 = df_individus[df_individus['user_id']==id_indiv_1].
 ↪mean(axis=0)['rating']
        mean_rate_2 = df_individus[df_individus['user_id']==id_indiv_2].
 ↪mean(axis=0)['rating']
        kept_items_1 = np.squeeze(items_in_common[indiv_1].to_numpy()[:, -1:])
        kept_items_2 = np.squeeze(items_in_common[indiv_2].to_numpy()[:, -1:])
        norm_1 = kept_items_1-mean_rate_1
        norm_2 = kept_items_2-mean_rate_2
        denom = np.sqrt((np.sum((norm_1)**2))*np.sum(((norm_2)**2)))
        if denom == 0:
```

```
            pc = 0
        else:
            pc = np.sum((norm_1)*(norm_2)) / np.sqrt((np.sum((norm_1)**2))*np.
↪sum(((norm_2)**2)))
            pc *= min([kept_items_1.size, gamma]) / gamma
        return pc

def find_neighboors_id(u_data, u_ids, indiv_id, k, w_min, item, gamma=4,␣
↪verbose=False):
    """
    Parameters:
    u_data: (dataframe) contenant l'ensemble des données ('user_id', 'item_id',␣
↪'rating')
    u_ids: (data_frame) contenant des données propres à chaque utilisateur␣
↪('user_id', 'mean_rate', 'std_rate')
    indiv_id: (int) contenant l'id de l'individu dont on veut prédire la note
    k: (int) nomnbre de candidats sur lesquels se baser pour la prédiction
    w_min: (float) similitude minimale pour estimer un candidat comme voisin␣
↪potentiel
    item: (int) contenant l'id de l'item dont on souhaite prédire la note
    gamma: (int) nombre de films minimal en commun sinon une penalité␣
↪s'applique sur la similitude
    Returns:
    float: note prédite
    """
    # Avant de supprimer notre individu on récupère sa note moyenne et son␣
↪écart-type :
    mean_user_rate = u_ids[u_ids['user_id']==indiv_id]['mean'].to_numpy()
    std_user_rate = u_ids[u_ids['user_id']==indiv_id]['std'].to_numpy()
    # On créée le dataframe des users ayant déjà noté le film:
    # Cela permet d'accelerer les calculs (l'algo étant au minima linéaire en␣
↪le nombre de données)
    u_item = u_data[u_data['item_id']==item].drop_duplicates('user_id',␣
↪keep='first')['user_id']
    if verbose :
        print("{} individus ont déjà voté pour ce film".format(len(u_item)))
    # Afin de limiter le nombre de variables on utilise ce même tableau pour␣
↪conserver tous les utilisateurs :
    u_ids.insert(3,"is_in",list(u_ids['user_id'].isin(u_item)))
    # On supprime du dataframe notre utilisateur et les utilisateurs n'ayant␣
↪pas voté pour le film::
    df = u_ids[(u_ids['user_id']!=indiv_id) & (u_ids['is_in']==True)]
    pc = []
    for index, row in df.iterrows():
        pc.append(pearson_correlation(indiv_id, row['user_id'], u_data, gamma))
```

```
    # Conversion de df vers un numpy pour utiliser les bonnes propriétés de␣
↪mask:
    df = df['user_id'].to_numpy()
    pc = np.array(pc)
    # On ne conserve que les voisins ayant la similitude supérieure au seuil:
    mask = pc > 0.3
    res = np.array([df[mask], pc[mask]])
    # On trie les voisins selon leur similitude croissante
    knn = res.T[res.T[:,1].argsort()]
    if verbose:
        print("Après calcul des similitudes, on ne conserve que {} candidats.".
↪format(len(knn)))
        print("Puis les k={} voisins les plus proches sont finalement␣
↪conservées".format(k))
    # On ne conserve que les k plus proches
    if k < knn.shape[0]:
        knn = knn[len(knn)-k:, :]
    if len(knn) == 0:
        u_ids = u_ids.drop(columns='is_in', inplace=False)
        return -1, u_ids
    # On effectue la prédiction sur les k plus proches
    denom = np.sum(np.abs(knn[:, 1]))
    rate_pred = 0
    for elt in knn:
        note_v =␣
↪u_data[(u_data['user_id']==int(elt[0]))&(u_data['item_id']==item)]['rating'].
↪values
        if verbose:
            print("L'utilisateur {} a voté {} pour cet item, sa similitude est␣
↪{}.". \
                  format(int(elt[0]), note_v, elt[1]))
        user_v_mean = u_ids[u_ids['user_id']==int(elt[0])]['mean'].to_numpy()
        std_v_mean = u_ids[u_ids['user_id']==int(elt[0])]['std'].to_numpy()
        rate_pred += elt[1] * (note_v - user_v_mean) / std_v_mean
    rate_pred = rate_pred / denom
    rate_pred = mean_user_rate + std_user_rate*rate_pred
    # Pour une réutilisation du dataframe, on supprime la colonne spécifique à␣
↪l'item:
    u_ids = u_ids.drop(columns='is_in', inplace=False)
    return rate_pred, u_ids

res, u_ids = find_neighboors_id(u_data, u_ids, 78, k=20, w_min=10, item=5,␣
↪verbose=True)
res
```

```
86 individus ont déjà voté pour ce film
Après calcul des similitudes, on ne conserve que 23 candidats.
```

```
Puis les k=20 voisins les plus proches sont finalement conservées
L'utilisateur 276 a voté [3] pour cet item, sa similitude est
0.34158468920776675.
L'utilisateur 43 a voté [4] pour cet item, sa similitude est
0.34257829469755874.
L'utilisateur 378 a voté [3] pour cet item, sa similitude est
0.3597051434745263.
L'utilisateur 280 a voté [4] pour cet item, sa similitude est
0.38944863236080846.
L'utilisateur 864 a voté [4] pour cet item, sa similitude est
0.4009338683510621.
L'utilisateur 393 a voté [3] pour cet item, sa similitude est
0.42148267966859826.
L'utilisateur 255 a voté [2] pour cet item, sa similitude est
0.4389479684796693.
L'utilisateur 880 a voté [3] pour cet item, sa similitude est
0.45497065785967955.
L'utilisateur 311 a voté [3] pour cet item, sa similitude est
0.4586285569750353.
L'utilisateur 109 a voté [3] pour cet item, sa similitude est
0.48764935757562383.
L'utilisateur 709 a voté [4] pour cet item, sa similitude est
0.4927692111216915.
L'utilisateur 303 a voté [2] pour cet item, sa similitude est
0.49864691389691806.
L'utilisateur 643 a voté [3] pour cet item, sa similitude est 0.5.
L'utilisateur 28 a voté [3] pour cet item, sa similitude est 0.5127954669727754.
L'utilisateur 593 a voté [4] pour cet item, sa similitude est
0.5535115773725051.
L'utilisateur 92 a voté [4] pour cet item, sa similitude est 0.5578217116404051.
L'utilisateur 291 a voté [5] pour cet item, sa similitude est
0.5720539055840977.
L'utilisateur 399 a voté [3] pour cet item, sa similitude est
0.6532425407273184.
L'utilisateur 422 a voté [3] pour cet item, sa similitude est
0.7081237603984349.
L'utilisateur 671 a voté [2] pour cet item, sa similitude est
0.9138311535740187.
```

[87]:
```
array([3.05501194])
```

Ci-dessus est un exemple de sortie de l'algorithme implémenté : le nombre d'individus ayant voté pour cet item, le nombre de candidat retenu, la note effectuée par ce candidat pour cet item mais également la similitude entre ce candidat et le candidat pour qui on souhaite faire la recommandation. Finalement, l'algorithme retourne la note qui pourrait être attribuée à cet item par ce candidat, ici 3,05.

1.1.3 Test du système

Dans un second temps, nous souhaitons tester le système de recommandation sur cinq bases de données différentes. Nous avons pris la décision de ne conserver que les 100 premiers objets de chacune des bases de données de test afin de réduire considérablement les temps de calculs.

[88]:
```
# Pour ne pas afficher une cellule remplie d'import, l'ensemble des␣
↪manipulations d'ouverture / traitement a été
# effectué dans un script externe
from import_validation_data import u1_base, u2_base, u3_base, u4_base, u5_base,␣
↪\
                                   u1_test, u2_test, u3_test, u4_test, u5_test
```

[92]:
```
def get_error(base, test):
    """
    Fonction qui permet, grâce aux algorithmes précédents de calculer
    l'erreur effectuée par l'algorithme sur une base de données de test
    """
    u_ids = get_u_ids(base)
    error = 0
    for index, row in test.iterrows():
        res, u_ids = find_neighboors_id(base, u_ids, indiv_id=row['user_id'],␣
↪k=20, w_min=10,
                                        item=row['item_id'],verbose=False)
        if res != -1:
            error = error + np.abs(res - row['rating'])
    error = error / test.shape[0]
    return error
```

[93]:
```
from time import time
top = time()
err1 = get_error(u1_base, u1_test[:100])
print("Erreur sur jeu 1: {}".format(err1))
print("Temps écoulé: {}".format(abs(time()-top)))
err2 = get_error(u2_base, u2_test[:100])
print("erreur sur jeu 2: {}".format(err2))
print("Temps écoulé: {}".format(abs(time()-top)))
err3 = get_error(u3_base, u3_test[:100])
print("erreur sur jeu 3: {}".format(err3))
print("Temps écoulé: {}".format(abs(time()-top)))
err4 = get_error(u4_base, u4_test[:100])
print("erreur sur jeu 4: {}".format(err4))
print("Temps écoulé: {}".format(abs(time()-top)))
err5 = get_error(u5_base, u5_test[:100])
print("erreur sur jeu 5: {}".format(err5))
print("Temps écoulé: {}".format(abs(time()-top)))
mean_err = (err1 + err2 + err3 + err4 + err5)/5
```

```
print("Erreur moyenne: {}".format(mean_err))
print("Temps total {}".format(abs(time()-top)))
```

```
Erreur sur jeu 1: [0.71692126]
Temps écoulé: 85.22398400306702
erreur sur jeu 2: [0.8415115]
Temps écoulé: 184.64445996284485
erreur sur jeu 3: [0.80086194]
Temps écoulé: 280.69833302497864
erreur sur jeu 4: [0.7829049]
Temps écoulé: 359.84035301208496
erreur sur jeu 5: [0.81823037]
Temps écoulé: 449.34778094291687
Erreur moyenne: [0.79208599]
Temps total 449.3481719493866
```

Ainsi, on remarque que l'erreur moyenne de l'algorithme sur l'ensemble des jeux de test est de 0.79. La note finale étant normalement un entier on pourrait donc qualifier notre incertitude de prédiction à plus ou moins 1.
Ces résultats ne sont pas parfaits car il existe des hyperparamètres à optimiser: nombre de candidats à retenir pour la prédiction, nombre de films minimal en commun n'entraînant pas une pénalité de la similitude ou encore seuil de similitude minimal pour le préfiltrage des candidats. Ajoutons aussi que cette méthode pourrait être améliorée en ajoutant d'autre stratégies (comme par exemple un item-based model) et en faisant "voter" ces stratégies pour une note finale (stratégie de bagging).