09RecommendationSystem

November 10, 2018

1 Recommendation

1. 推薦系統的種類
   - collabrative(協同推薦)
   - content-based(內容推薦)

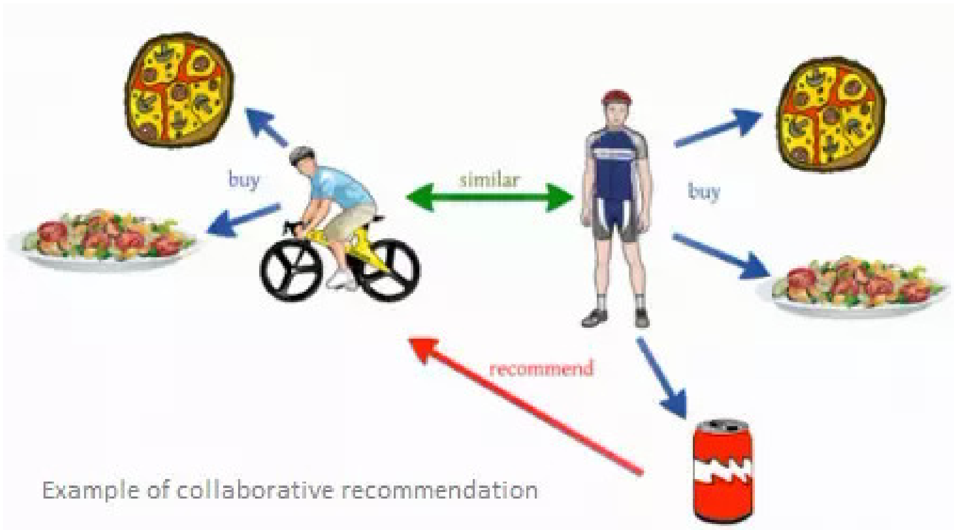
from quora

1.1 IMPORT & DATA

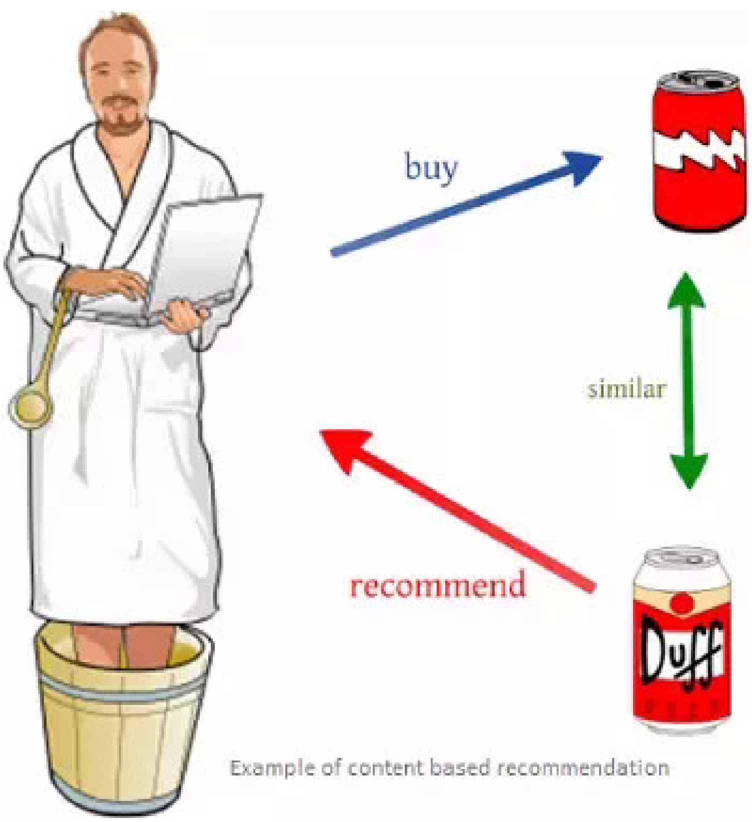
```
In [63]: import numpy as np
         import scipy
         import pandas as pd
         import math
         import random
         import sklearn
         from nltk.corpus import stopwords
         from sklearn.model_selection import train_test_split
         from sklearn.feature_extraction.text import TfidfVectorizer
         from sklearn.metrics.pairwise import cosine_similarity
         from scipy.sparse.linalg import svds
         import matplotlib.pyplot as plt


         import warnings
         warnings.filterwarnings('ignore')
```

```
In [64]: df_articles = pd.read_csv('shared_articles.csv')
         df_articles = df_articles[df_articles['eventType'] == 'CONTENT SHARED']
         df_articles.head(5)
```

```
Out[64]:     timestamp       eventType              contentId         authorPersonId  \
         1  1459193988  CONTENT SHARED -4110354420726924665  4340306774493623681
         2  1459194146  CONTENT SHARED -7292285110016212249  4340306774493623681
         3  1459194474  CONTENT SHARED -6151852268067518688  3891637997717104548
```



collabrative



content-based

```
4  1459194497  CONTENT SHARED  2448026894306402386  4340306774493623681
5  1459194522  CONTENT SHARED -2826566343807132236  4340306774493623681

        authorSessionId authorUserAgent authorRegion authorCountry contentType  \
1  8940341205206233829             NaN          NaN           NaN        HTML
2  8940341205206233829             NaN          NaN           NaN        HTML
3 -1457532940883382585             NaN          NaN           NaN        HTML
4  8940341205206233829             NaN          NaN           NaN        HTML
5  8940341205206233829             NaN          NaN           NaN        HTML

                                                 url  \
1  http://www.nytimes.com/2016/03/28/business/dea...
2  http://cointelegraph.com/news/bitcoin-future-w...
3  https://cloudplatform.googleblog.com/2016/03/G...
4  https://bitcoinmagazine.com/articles/ibm-wants...
5  http://www.coindesk.com/ieee-blockchain-oxford...

                                               title  \
1  Ethereum, a Virtual Currency, Enables Transact...
2  Bitcoin Future: When GBPcoin of Branson Wins O...
3                           Google Data Center 360ř Tour
4  IBM Wants to "Evolve the Internet" With Blockc...
5  IEEE to Talk Blockchain at Cloud Computing Oxf...

                                                text lang
1  All of this work is still very early. The firs...   en
2  The alarm clock wakes me at 8:00 with stream o...   en
3  We're excited to share the Google Data Center ...   en
4  The Aite Group projects the blockchain market ...   en
5  One of the largest and oldest organizations fo...   en
```

In [65]:
```python
df_interactions = pd.read_csv('users_interactions.csv')
df_interactions.head(10)
```

Out[65]:
```
    timestamp eventType            contentId             personId  \
0  1465413032      VIEW -3499919498720038879 -8845298781299428018
1  1465412560      VIEW  8890720798209849691 -1032019229384696495
2  1465416190      VIEW   310515487419366995 -1130272294246983140
3  1465413895    FOLLOW   310515487419366995   344280948527967603
4  1465412290      VIEW -7820640624231356730  -445337111692715325
5  1465413742      VIEW   310515487419366995 -8763398617720485024
6  1465415950      VIEW -8864073373672512525  3609194402293569455
7  1465415066      VIEW -1492913151930215984  4254153380739593270
8  1465413762      VIEW   310515487419366995   344280948527967603
9  1465413771      VIEW  3064370296170038610  3609194402293569455

             sessionId                                          userAgent  \
0  1264196770339959068                                                NaN
```

```
1   3621737643587579081  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2...
2   2631864456530402479                                               NaN
3  -3167637573980064150                                               NaN
4   5611481178424124714                                               NaN
5   1395789369402380392  Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebK...
6   1143207167886864524                                               NaN
7   8743229464706506141  Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/53...
8  -3167637573980064150                                               NaN
9   1143207167886864524                                               NaN

  userRegion userCountry
0        NaN         NaN
1         NY          US
2        NaN         NaN
3        NaN         NaN
4        NaN         NaN
5         MG          BR
6        NaN         NaN
7         SP          BR
8        NaN         NaN
9        NaN         NaN
```

1.2 Preprocessing

In [66]:
```
set(df_interactions['eventType'])
```

Out[66]:
```
{'BOOKMARK', 'COMMENT CREATED', 'FOLLOW', 'LIKE', 'VIEW'}
```

In [67]:
```
event_type_strength = {
   'VIEW': 1.0,
   'LIKE': 2.0,
   'BOOKMARK': 2.5,
   'FOLLOW': 3.0,
   'COMMENT CREATED': 4.0,
}

# 請將 eventType 按照 event_type_strength 進行評分
#=============your works starts===============#
df_interactions['eventStrength'] =
#==============your works ends================#

print("平均互動分數:", np.average(df_interactions['eventStrength']))
# 平均互動分數: 1.2362885828078327
```

```
平均互動分數: 1.2362885828078327
```

在推薦系統中，有一個很常見的問題，稱為 ***cold-start***。因為，很多使用者並沒有真正的根產品產生任何互動，所以並沒有辦法從資料及當中了解到他們偏好。因此，這邊我們將少於五個 interactions 的 user 刪掉。

In [68]:
```python
# 計算出每個使用者有對幾項不同的商品進行互動 (hint: 以 ['personId', 'contentId'] 進行 gr
# 篩選掉互動商品數小於五次的使用者 (hint: return list)
#=============your works starts===============#
df_users_interactions_count =
users_with_enough_interactions =
#==============your works ends================#

print("平均互動次數: ", np.average(df_users_interactions_count), "次")
print("使用者個數: ",  len(df_users_interactions_count))
print("互動大於 5 次使用者個數", len(users_with_enough_interactions))
# 平均互動次數:  21.482849604221634 次
# 使用者個數:  1895
# 互動大於 5 次使用者個數 1140
```

```
平均互動次數:  21.482849604221634 次
使用者個數:  1895
互動大於 5 次使用者個數 1140
```

In [69]:
```python
# 找出 df_interactions 中 personId 在 users_with_enough_interactions 當中的 row
#=============your works starts===============#
df_interactions_from_selected_users =
#==============your works ends================#

print('總互動比數:', len(df_interactions))
print('互動次數大於五用戶總互動比數:', len(df_interactions_from_selected_users))
# 總互動比數: 72312
# 互動次數大於五用戶總互動比數: 69868
```

```
總互動比數: 72312
互動次數大於五用戶總互動比數: 69868
```

In [70]:
```python
def smooth_user_preference(x):
    # 請先 +1 再取 log，以平滑互動分數
    #=============your works starts===============#
    logged =
    #==============your works ends================#
    return logged

print(smooth_user_preference(1))
print(smooth_user_preference(2))
print(smooth_user_preference(3))
# 0.6931471805599453
# 1.0986122886681098
# 1.3862943611198906
```

```
0.6931471805599453
1.0986122886681098
```

```
1.3862943611198906
```

```
In [71]: # 計算使用者對於文章喜愛的加總
         # 並透過上面的 smooth_user_preference 使其平滑
         #=============your works starts===============#
         df_interactions_full_no_smooth =
         df_interactions_full =
         #==============your works ends================#

         print('平均喜好分數 (未平滑)', np.average(df_interactions_full_no_smooth))
         print('平均喜好分數 (平滑)', np.average(df_interactions_full['eventStrength']))
         # 平均喜好分數 (未平滑) 2.214954226972843
         # 平均喜好分數 (平滑) 1.015265936675581
```

```
平均喜好分數 (未平滑) 2.214954226972843
平均喜好分數 (平滑) 1.015265936675581
```

1.3 TRAIN_TEST_SPLIT

```
In [72]: # # 請使用 train_test_split 切分 df_interactions_full
         # # 1. stratify=df_interactions_full['personId'] #stratify 可以按照 y 的比例進行切分
         # # 2. test_size=0.2
         # # 3. random_state=1212
         # #=============your works starts===============#
         # df_interactions_train, df_interactions_test =
         # #==============your works ends================#

         # print('len(df_interactions_train):', len(df_interactions_train))
         # print('len(df_interactions_test):', len(df_interactions_test))
         # # len(df_interactions_train): 31284
         # # len(df_interactions_test): 7822
```

1.4 評價

1.4.1 基礎知識

col	Retrieved	Non Retrieved
Relevant	True Positive(TP)	False Negative(TN)
Irrelevant	False Postive(FP)	True Negative(TN)

1.4.2 解釋

1. Precision at K

$$Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|} = \frac{|TP|}{|TP| + |FP|}$$

2. Recall at K

$$Recall = \frac{|Relevant \cap Retrieved|}{|Relevant|} = \frac{|TP|}{|TP| + |TN|}$$

3. F measure

   1. 算術平均數

$$F = \frac{P+R}{2} \quad where \quad P = Precision, \ R = Recall$$

   2. 幾何平均數

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 B + R} \quad where \quad \beta^2 = \frac{1-\alpha}{\alpha}$$

   3. F_β or F_1

$$F_1 = F_\beta = \frac{1}{0.5\frac{1}{P} + 0.5\frac{1}{R}} = \frac{PR}{0.5P + 0.5R} = \frac{2PR}{P+R}$$

4. R-Precision
5. NDCG
6. MAP

1.4.3 注意事項

1. 注意與 Accuracy 的區別

$$\frac{|TP| + |TF|}{|TP| + |TN| + |FP| + |FN|}$$

2. 參考資訊

1.5 Evaluation

In [73]:
```
# predict result
item_ids = df_articles['contentId'].tolist()
item_ids_mapping = dict([(idx, contentId) for idx, contentId in enumerate(item_ids)])
item_ids_set = set(item_ids)
df_user_preference = df_interactions_full.groupby('personId')['contentId'].apply(set)
```

```
In [74]: # ground truth
         # 找出每一個 personId 曾經互動過「不重複」的文章
         #=============your works starts===============#
         df_answer =
         #==============your works ends================#

         df_answer.head(5)
         # personId
         # -9223121837663643404    {5211673327552264703, -5002383425685129595, -7...
         # -9212075797126931087    {-1995591062742965408, 6852597772196653540, -9...
         # -9207251133131336884    {-4029704725707465084, -1297580205670251233, -...
         # -9199575329909162940    {5293701842202310496, -5002383425685129595, 54...
         # -9196668942822132778    {-7217327053148O3549, -8813724423497152538, -8...
         # Name: contentId, dtype: object
```

```
Out[74]: personId
         -9223121837663643404    {5211673327552264703, -5002383425685129595, -7...
         -9212075797126931087    {-1995591062742965408, 6852597772196653540, -9...
         -9207251133131336884    {-4029704725707465084, -1297580205670251233, -...
         -9199575329909162940    {5293701842202310496, -5002383425685129595, 54...
         -9196668942822132778    {-7217327053148O3549, -8813724423497152538, -8...
         Name: contentId, dtype: object
```

1.6 Popularity model (Base Line)

```
In [75]: # 以 contentId 進行 groupby，按照每篇文章總分數進行排序
         #=============your works starts===============#
         df_item_popularity =
         #==============your works ends================#

         df_item_popularity.head(5).to_dict(orient='record')
         # [{'contentId': -4.029704725707465e+18, 'eventStrength': 213.30481497288199},
         #  {'contentId': -6.783772548752092e+18, 'eventStrength': 162.03158006500846},
         #  {'contentId': -1.3313934239753886e+17, 'eventStrength': 158.05458586966674},
         #  {'contentId': -8.208801367848628e+18, 'eventStrength': 136.62458307425328},
         #  {'contentId': -6.843047699859122e+18, 'eventStrength': 134.34939619163308}]
```

```
Out[75]: [{'contentId': -4.029704725707465e+18, 'eventStrength': 213.30481497288199},
          {'contentId': -6.783772548752092e+18, 'eventStrength': 162.03158006500846},
          {'contentId': -1.3313934239753886e+17, 'eventStrength': 158.05458586966674},
          {'contentId': -8.208801367848628e+18, 'eventStrength': 136.62458307425328},
          {'contentId': -6.843047699859122e+18, 'eventStrength': 134.34939619163308}]
```

```
In [76]: def popularity_recommend(user_id):
             # 直接回傳分數加總最高的十篇文章
             #=============your works starts===============#
             recommend =
             #==============your works ends================#
             return recommend
```

```
# 透過 apply function 使用 popularity_recommend 到 df_interactions_full["personId"] 的
df_user_preference['popularity_recommend'] = df_user_preference["personId"].apply(popu
df_user_preference.head(5)
```

```
Out[76]:                 personId                                          contentId  \
         0 -9223121837663643404  {5211673327552264703, -5002383425685129595, -7...
         1 -9212075797126931087  {-1995591062742965408, 6852597772196653540, -9...
         2 -9207251133131336884  {-9216926795620865886, -4029704725707465084, -...
         3 -9199575329909162940  {5293701842202310496, -5002383425685129595, 54...
         4 -9196668942822132778  {-7217327053148035 49, -8813724423497152538, -8...

                                         popularity_recommend
         0  [-4029704725707465084, -6783772548752091658, -...
         1  [-4029704725707465084, -6783772548752091658, -...
         2  [-4029704725707465084, -6783772548752091658, -...
         3  [-4029704725707465084, -6783772548752091658, -...
         4  [-4029704725707465084, -6783772548752091658, -...
```

```
In [77]: def precision_at_k(row, k=10):
             # 計算每一個 row 的 precision_at_k
             #=============your works starts===============#
             precision =
             #==============your works ends================#
             return precision

         evaluation_result = df_user_preference.apply(precision_at_k, axis=1)
         print("Average Precision At K:", np.average(evaluation_result))
```

```
Average Precision At K: 0.13342105263157894
```

1.7 Content-Based Filtering model

```
In [78]: stopwords_list = stopwords.words('english') + stopwords.words('portuguese')
         vectorizer = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0.003, max_d
```

```
In [79]: # 計算 df_articles['title'] + "" + df_articles['text'] 中每個 row 的 tfidf_matrix
         #=============your works starts===============#
         tfidf_matrix =
         #==============your works ends================#

         np.sum(tfidf_matrix[:5].toarray(), axis=1)
         # array([ 9.50309706,  9.14139363,  7.07473481, 10.43412109,  7.64140829])
```

```
Out[79]: array([ 9.50309706,  9.14139363,  7.07473481, 10.43412109,  7.64140829])
```

```
In [80]: def get_user_vector(like_content_list):
             idxs = [item_ids.index(i) for i in like_content_list if i in item_ids_set]
```

```
    if len(idxs) == 0:
        # 初始劃一條全部為零，與 tfidf_matrix 中每一條向量等長的 np.array()
        #=============your works starts===============#
        average_vector =
        #==============your works ends================#
    else:
        # 使用 idxs 找出 tfidf_matrix 中的對應向量
        # 並 element-wise 的計算每一條向量中每個元素的平均值 (axis=0)
        #=============your works starts===============#
        tfidf_vectors =
        average_vector =
        #==============your works ends================#
    return average_vector

df_user_preference['preference_vector'] = df_user_preference['contentId'].apply(get_us
df_user_preference['preference_vector'].head(5)
# 0    [0.0022163282029446594, 0.0030194389616064613,...
# 1    [0.0, 0.012492323019192434, 0.0, 0.0, 0.003230...
# 2    [0.013699793673353864, 0.002006773900308267, 0...
# 3    [0.0, 0.006253150796304994, 0.0, 0.0, 0.003906...
# 4    [0.0, 0.0, 0.0, 0.0, 0.017006668208970826, 0.0...
# Name: preference_vector, dtype: object
```

Out[80]:
```
0    [0.0022163282029446594, 0.0030194389616064613,...
1    [0.0, 0.012492323019192434, 0.0, 0.0, 0.003230...
2    [0.013699793673353864, 0.002006773900308267, 0...
3    [0.0, 0.006253150796304994, 0.0, 0.0, 0.003906...
4    [0.0, 0.0, 0.0, 0.0, 0.017006668208970826, 0.0...
Name: preference_vector, dtype: object
```

In [81]:
```
user_preference_vector = np.hstack(df_user_preference['preference_vector'].values).res
# 使用 cosine_similarity 去計算每一個 preference_vector 與每一篇待選文章的 cosine simil
#=============your works starts===============#
similarity_metric =
#==============your works ends================#
# 請特別注意：
# 每一個 row 是使用者對每一篇文章的 similarity(preference)
# 所以接下來要篩出，每一個 row 當中 similarity 最高分的 10 篇文章


similarity_metric[:5, :5]
# array([[0.119037  , 0.11127776, 0.23397873, 0.16428804, 0.1500561 ],
#        [0.03351941, 0.03650116, 0.13718512, 0.04648663, 0.04216605],
#        [0.04209047, 0.04084319, 0.04926266, 0.08144102, 0.02540297],
#        [0.08642438, 0.09372568, 0.10180722, 0.1332371 , 0.08941768],
#        [0.04121156, 0.01011304, 0.01477526, 0.04543164, 0.03242129]])
```

Out[81]:
```
array([[0.119037  , 0.11127776, 0.23397873, 0.16428804, 0.1500561 ],
       [0.03351941, 0.03650116, 0.13718512, 0.04648663, 0.04216605],
```

```
        [0.04209047, 0.04084319, 0.04926266, 0.08144102, 0.02540297],
        [0.08642438, 0.09372568, 0.10180722, 0.1332371 , 0.08941768],
        [0.04121156, 0.01011304, 0.01477526, 0.04543164, 0.03242129]])
```

In [82]:
```
# 使用 np.argsort 將每一個 row 的 similarity 進行排序，然後倒過來排續，篩出前 10 個
#=============your works starts===============#
top_10_content_idx =
#==============your works ends================#
# 請注意這邊的產出代表的是每一篇文章的在 tfidf_matrix 的 idx 位置
# 必須與 contentId 區別

top_10_content_idx
# array([[ 650, 1032, 1643, ...,  237, 1034, 3018],
#        [ 977, 1023, 1601, ..., 1548, 1175, 1769],
#        [1671, 1593, 1795, ..., 2477, 1117, 1520],
#        ...,
#        [1622, 1845, 3021, ...,  974, 1035, 1607],
#        [1185, 1636, 1116, ..., 2309, 2357, 2616],
#        [2664, 2781,  659, ..., 2634, 1552, 3018]], dtype=int64)
```

Out[82]:
```
array([[ 650, 1032, 1643, ...,  237, 1034, 3018],
       [ 977, 1023, 1601, ..., 1548, 1175, 1769],
       [1671, 1593, 1795, ..., 2477, 1117, 1520],
       ...,
       [1622, 1845, 3021, ...,  974, 1035, 1607],
       [1185, 1636, 1116, ..., 2309, 2357, 2616],
       [2664, 2781,  659, ..., 2634, 1552, 3018]], dtype=int64)
```

In [83]:
```
# 將 tfidf_matrix 的 idx 轉換成 contentId
#=============your works starts===============#
top_10_contentId =
#==============your works ends================#

top_10_contentId[:5, :3]
# array([[ 8596997246990922861,  2858969450431709251, -4541461982704074404],
#        [-1995591062742965408,  6852597772196653540,  -969155230116728853],
#        [-1297580205670251233, -9216926795620865886, -4434534460030275781],
#        [-1755875383603052680,  5293701842202310496,  5037403311832115000],
#        [ 9175693555063886126,  7013665235990336340, -2069509552243850466]],
#       dtype=int64)
```

Out[83]:
```
array([[ 8596997246990922861,  2858969450431709251, -4541461982704074404],
       [-1995591062742965408,  6852597772196653540,  -969155230116728853],
       [-1297580205670251233, -9216926795620865886, -4434534460030275781],
       [-1755875383603052680,  5293701842202310496,  5037403311832115000],
       [ 9175693555063886126,  7013665235990336340, -2069509552243850466]],
      dtype=int64)
```

In [84]:
```
df_user_preference['content_based_recommended'] = list(top_10_contentId)
```

```
In [85]: def precision_at_k(row, k=10):
             return len(set(row['content_based_recommended']) & set(df_answer[row['personId']])

         evaluation_result = df_user_preference.apply(precision_at_k, axis=1)
         print("Average Precision At K:", np.average(evaluation_result))
         # Average Precision At K: 0.5700877192982456
```

```
Average Precision At K: 0.5700877192982456
```