Amazon Product Reviews

July 16, 2020

1 Amazon Product Reviews Dataset

https://www.kaggle.com/saurav9786/amazon-product-reviews

userId : Every user identified with a unique id

productId : Every product identified with a unique id

Rating : Rating of the corresponding product by the corresponding user

timestamp : Time of the rating (ignore this column for this exercise)

[1]:
```
import pandas as pd
import  numpy as np
import os
import math
import json
import time
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.neighbors import NearestNeighbors



import scipy.sparse
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import svds

import warnings; warnings.simplefilter('ignore')
%matplotlib inline
```

[2]:
```
# Read and give names to data columns
data = pd.read_csv('./../data/ratings_electronics.csv',
                   names = ['userId', 'productId', 'Rating','timestamp'])
```

1.1 Data Exploration and Visualization

[3]:
```
data.head()
```

[3]:
```
            userId   productId  Rating   timestamp
0    AKM1MP6P0OYPR  0132793040     5.0  1365811200
1   A2CX7LUOHB2NDG  0321732944     5.0  1341100800
2   A2NWSAGRHCP8N5  0439886341     1.0  1367193600
3   A2WNBOD3WNDNKT  0439886341     3.0  1374451200
4   A1GI0U4ZRJA8WN  0439886341     1.0  1334707200
```

[4]:
```
print('Data shape: ', data.shape)
```

```
Data shape:  (7824482, 4)
```

Use only 1,000,000 datapoints

[5]:
```
data = data.iloc[:1000000,:]
```

[6]:
```
print('Data shape: ', data.shape)
```

```
Data shape:  (1000000, 4)
```

[7]:
```
print('Data types:\n',data.dtypes)
```

```
Data types:
 userId        object
productId     object
Rating       float64
timestamp      int64
dtype: object
```

[8]:
```
print("Describe: ", data.describe()['Rating'])
```

```
Describe:  count    1000000.000000
mean           3.973620
std            1.399741
min            1.000000
25%            3.000000
50%            5.000000
75%            5.000000
max            5.000000
Name: Rating, dtype: float64
```

[9]:
```
min_rating, max_rating = data.Rating.min(), data.Rating.max()
print('Minimum: {} and maximum: {} rating '.format(min_rating, max_rating ))
```

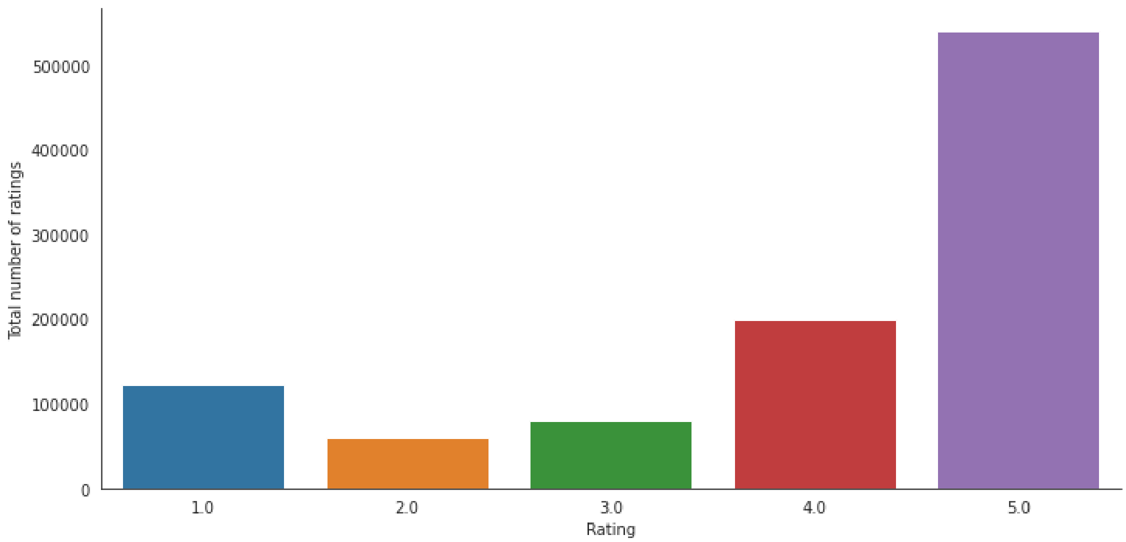
```
Minimum: 1.0 and maximum: 5.0 rating
```

1.2 Data Cleaning

[10]:
```
print("Number of missing values:",
      data.isnull().sum()
     )
```

```
Number of missing values: userId       0
productId    0
Rating       0
timestamp    0
dtype: int64
```

[11]:
```
with sns.axes_style('white'):
    g = sns.factorplot("Rating", data=data, aspect=2.0,kind='count')
    g.set_ylabels("Total number of ratings")
```



[12]:
```
print("Some statistics ")
print("-"*50)
print("\nTotal no of ratings :",data.shape[0])
print("Total No of Users   :", len(np.unique(data.userId)))
print("Total No of products  :", len(np.unique(data.productId)))
```

```
Some statistics
--------------------------------------------------

Total no of ratings : 1000000
Total No of Users   : 754153
Total No of products  : 59634
```

Drop timestamp column

[13]:
```
data.drop(['timestamp'], axis = 1, inplace = True)
```

[14]:
```
data.head()
```

[14]:
```
           userId   productId  Rating
0   AKM1MP6P0OYPR  0132793040     5.0
1  A2CX7LUOHB2NDG  0321732944     5.0
2  A2NWSAGRHCP8N5  0439886341     1.0
3  A2WNBOD3WNDNKT  0439886341     3.0
4  A1GI0U4ZRJA8WN  0439886341     1.0
```

[15]:
```
data.columns
```

[15]:
```
Index(['userId', 'productId', 'Rating'], dtype='object')
```

Only products with 100 or more ratings

[16]:
```
data_count_ratings = data.groupby('productId').filter(lambda d: d['Rating'].
↪count() >= 100 )
```

[17]:
```
selected_items = set(data_count_ratings['productId'])
print("Len of selected items: ", len(selected_items))
print(list(selected_items)[:10])
```

```
Len of selected items:  1712
['B00005BMSN', 'B00009705F', 'B0002ZQHFA', 'B00083Y0YG', 'B00009XVA3',
'B0007OWNCC', 'B00004TWM6', 'B000CBB4N4', 'B00004Z61H', 'B000FED6N0']
```

[18]:
```
data.shape
```

[18]:
```
(1000000, 3)
```

[19]:
```
### Only products with 100 or more reviews
data = data[data.productId.isin(selected_items)]
```

[20]:
```
data.shape
```

[20]:
```
(509370, 3)
```

[21]:
```
data.iloc[15]
```

[21]:
```
userId       A1ZD73MDX4P0AY
productId        0972683275
Rating                    5
Name: 198, dtype: object
```

1.2.1 Training and testing sets

[22]:
```
n,m = data.shape

training_percentage = 0.8
n_training = int(n*0.8)

training, test = data.iloc[:n_training], data.iloc[n_training:]
print( 'Length training {} and test {} '.format(len(training), len(test)))
```

```
Length training 407496 and test 101874
```

2 Recommender System

[23]:
```
from collections import defaultdict

def create_sets(data):
    """
    data:Pandas dataframe
    """
    n,m = data.shape

    items_per_user = defaultdict(set)
    users_per_item = defaultdict(set)

    for i in range(n):
        datapoint  = data.iloc[i]
        user,item = datapoint['userId'], datapoint['productId']
        items_per_user[user].add(item)
        users_per_item[item].add(user)

    return items_per_user ,users_per_item


def create_set_reviews(data):
    reviewsPerUser = defaultdict(list)
    reviewsPerItem = defaultdict(list)


    n,m = data.shape

    for j in range(n):
        d = data.iloc[j]
        user,item = d['userId'], d['productId']
        reviewsPerUser[user].append(d)
        reviewsPerItem[item].append(d)
```

```
    return reviewsPerUser,reviewsPerItem
```

[24]:
```
items_per_user ,users_per_item  = create_sets(training)
```

[25]:
```
reviewsPerUser,reviewsPerItem  = create_set_reviews(training)
```

[28]:
```
n,m = training.shape

overall_mean_rating = training['Rating'].mean()
overall_mean_rating
```

[28]: 4.096099102813279

[29]:
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom

def mostSimilar(iD, n,reviewsPerUser, usersPerItem):
    similarities = []
    users = usersPerItem[iD]
    for i2 in usersPerItem:
        if i2 == iD: continue
        sim = Jaccard(users, usersPerItem[i2])
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:n]
```

[43]:
```
def predictRating(user,item, reviewsPerUser, usersPerItem):
    ratings = []
    similarities = []


    for d in  reviewsPerUser[user]:

        # product
        i2 = d['productId']

        if i2 == item: continue

        ratings.append(d['Rating'])

        similarities.append(Jaccard(usersPerItem[item],usersPerItem[i2]))
```

```
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings,similarities)]
        return sum(weightedRatings) / sum(similarities)

    else:
        # User hasn't rated any similar items
        return overall_mean_rating
```

2.1 Make Recommendation

[44]:
```
item = training.iloc[200]['productId']
item
```

[44]: '0972683275'

[45]:
```
mostSimilar(iD = item, n= 10 ,reviewsPerUser=reviewsPerUser, usersPerItem␣
 ↪=users_per_item )
```

[45]:
```
[(0.0020174848688634837, 'B0001OHH0Q'),
 (0.0017391304347826088, 'B0002855KK'),
 (0.001729106628242075, 'B00005ML7Q'),
 (0.001594896331738437, 'B0002GV876'),
 (0.0015936254980079682, 'B000A2AGYS'),
 (0.0015923566878980893, 'B0007A1IRC'),
 (0.0014803849000740192, 'B0006I09LQ'),
 (0.0013708019191226869, 'B000ARAPQW'),
 (0.0013201320132013201, 'B00005T3N3'),
 (0.0013192612137203166, 'B0007MWE1E')]
```

2.1.1 Evaluate Performace

- Compare against mean rating by user

[46]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

[47]:
```
# Overall mean rating
n,m = training.shape
mean_rating = [overall_mean_rating]*n
n,m = training.shape
labels = [training.iloc[k]['Rating'] for k in range(n)]
```

[50]:
```
mean_rating[:10]
```

[50]:
```
[4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279]
```

[48]:
```
labels[:10]
```

[48]:
```
[4.0, 4.0, 5.0, 4.0, 5.0, 4.0, 5.0, 3.0, 5.0, 5.0]
```

[52]:
```
predictions =[]

for i in range(n):
    d =training.iloc[i]
    user = d['userId']
    item = d['productId']

    stars = predictRating(user,item, reviewsPerUser=reviewsPerUser,␣
 ↪usersPerItem =users_per_item)
    predictions.append(stars)
```

[53]:
```
predictions[50:100]
```

[53]:
```
[4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.0,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
```

```
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 5.0,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.0,
 4.096099102813279,
 5.0,
 4.096099102813279,
 4.096099102813279,
 4.0,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279,
 4.096099102813279]
```

[54]:
```
print(MSE(mean_rating, labels), MSE(predictions, labels))
```

```
1.7587234209338283 1.917202902116438
```

2.2 Conclusion

Collaborative filtering method not always provide the best solution, weighting previous ratings can be or not be a good prediction for future ratings.

2.3 Model-based collaborative filtering system

Item-user matrix M $M[i,j] $ rating given to item i by user j

[ ]:
```

```

```python
def create_item_user_matrix(data):
    M = data.pivot_table(values = 'Rating', index = 'userId', columns =␣
 ↪'productId', fill_value = 0)
    M = M.T

    return M



M = create_item_user_matrix(training)


M.head()


from sklearn.decomposition import TruncatedSVD



def correlation(M, n_componets):
    """
    M: matrix of items-products
    ratings

    return correlation matrix tem-item
    most similar items are more correlated
    """

    # Decompose
    SVD = TruncatedSVD(n_components = 10 )
    decomposed = SVD.fit_transform(M)
    correlation =  np.corrcoef(decomposed)

    return correlation

correlation  = correlation(M, n_componets= 10 )
items_names =  list(M.index)


i = 100
item_name   = M.index[i]
print('Item name: ', item_name)
item_index = items_names.index(item_name)
print('Item index: ', item_index)

### Recommend items based on item B000021YU8
```

```
# Correlations for item B000021YU8
correlations_item = correlation[item_index]
print("Correlations shape {} for item {} ".format(correlations_item.shape,␣
 ↪item_name))


# recommend items with correlation > 0.65
r_items = list(M.index[correlations_item > 0.65] )

# Remove the item itself
r_items.remove(item_name)

print('Recommended first best 24 items: {}, based on item {}'.format(r_items,␣
 ↪item_name))




# Correlations for item B000021YU8
correlations_item = correlation[item_index]
print("Correlations shape {} for item {} ".format(correlations_item.shape,␣
 ↪item_name))


# recommend items with correlation > 0.65
r_items = list(M.index[correlations_item > 0.65] )

# Remove the item itself
r_items.remove(item_name)

print('Recommended first best 24 items: {}, based on item {}'.format(r_items,␣
 ↪item_name))
```

This method uses item-item correlations to recommend more products to an user. It decompose the rating matrix M, representing each item as a vector (embedding).

The method works well with large and sparse matrices. Allow us to recommend many items to users, but with limited personalization.

This approach ignores information user-user about ratings.

[ ]:

```

```