Recommendations_with_IBM

May 5, 2019

1 Recommendations with IBM

In this notebook, you will be putting your recommendation skills to use on real data from the IBM Watson Studio platform.

You may either submit your notebook through the workspace here, or you may work from your local machine and submit through the next page. Either way assure that your code passes the project RUBRIC. **Please save regularly.**

By following the table of contents, you will build out a number of different methods for making recommendations that can be used for different situations.

1.1 Table of Contents

At the end of the notebook, you will find directions for how to submit your work. Let's get started by importing the necessary libraries and reading in the data.

```
In [ ]:

In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import project_tests as t
        import pickle

        %matplotlib inline

        df = pd.read_csv('data/user-item-interactions.csv')
        df_content = pd.read_csv('data/articles_community.csv')
        del df['Unnamed: 0']
        del df_content['Unnamed: 0']

        # Show df to get an idea of the data
        df.head()
```

```
Out[2]:    article_id                                              title  \
        0      1430.0  using pixiedust for fast, flexible, and easier...
        1      1314.0       healthcare python streaming application demo
        2      1429.0         use deep learning for image classification
```

```
3      1338.0          ml optimization using cognitive assistant
4      1276.0          deploy your python model as a restful api

                                      email
0  ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1  083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2  b96a4f2e92d8572034b1e9b28f9ac673765cd074
3  06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4  f01220c46fc92c6e6b161b1849de11faacd7ccb2
```

```
In [2]: # Show df_content to get an idea of the data
        df_content.head()
```

```
Out[2]:                                            doc_body  \
        0  Skip navigation Sign in SearchLoading...\r\n\r...
        1  No Free Hunch Navigation * kaggle.com\r\n\r\n ...
        2   * Login\r\n * Sign Up\r\n\r\n * Learning Pat...
        3  DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...
        4  Skip navigation Sign in SearchLoading...\r\n\r...

                                             doc_description  \
        0  Detect bad readings in real time using Python ...
        1  See the forest, see the trees. Here lies the c...
        2  Heres this weeks news in Data Science and Bi...
        3  Learn how distributed DBs solve the problem of...
        4  This video demonstrates the power of IBM DataS...

                                               doc_full_name doc_status  article_id
        0  Detect Malfunctioning IoT Sensors with Streami...       Live           0
        1  Communicating data science: A guide to present...       Live           1
        2           This Week in Data Science (April 18, 2017)       Live           2
        3  DataLayer Conference: Boost the performance of...       Live           3
        4      Analyze NY Restaurant data using Spark in DSX       Live           4
```

1.1.1 Part I : Exploratory Data Analysis

Use the dictionary and cells below to provide some insight into the descriptive statistics of the data.

1. What is the distribution of how many articles a user interacts with in the dataset? Provide a visual and descriptive statistics to assist with giving a look at the number of times each user interacts with an article.

```
In [18]: df['email'].shape
```

```
Out[18]: (45993,)
```

```
In [40]: df['article_id'].nunique()
```

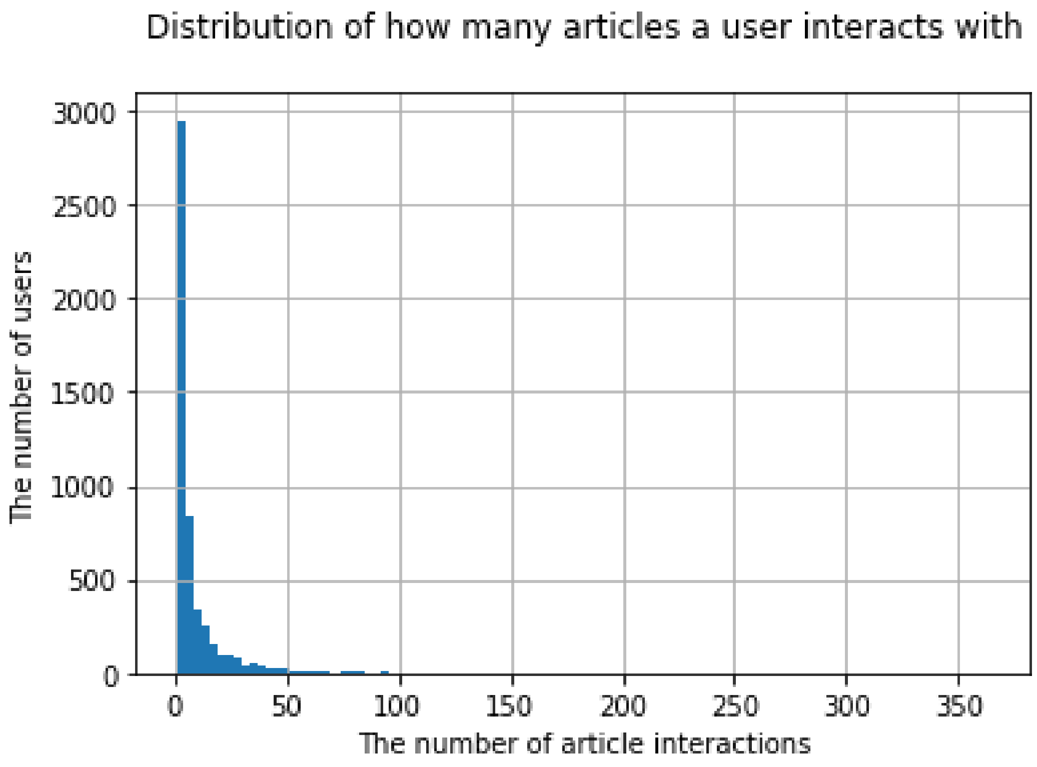
```
Out[40]: 714
```

```
In [16]: df['email'].value_counts().describe()

Out[16]: count    5148.000000
         mean        8.930847
         std        16.802267
         min         1.000000
         25%         1.000000
         50%         3.000000
         75%         9.000000
         max       364.000000
         Name: email, dtype: float64
```

```
In [3]: df['email'].value_counts()

Out[3]: 2b6c0f514c2f2b04ad3c4583407dccd0810469ee    364
        77959baaa9895a7e2bdc9297f8b27c1b6f2cb52a    363
        2f5c7feae533ce046f2cb16fb3a29fe00528ed66    170
        a37adec71b667b297ed2440a9ff7dad427c7ac85    169
        8510a5010a5d4c89f5b07baac6de80cd12cfaf93    160
        f8c978bcf2ae2fb8885814a9b85ffef2f54c3c76    158
        284d0c17905de71e209b376e3309c0b08134f7e2    148
        18e7255ee311d4bd78f5993a9f09538e459e3fcc    147
        d9032ff68d0fd45dfd18c0c5f7324619bb55362c    147
        c60bb0a50c324dad0bffd8809d121246baef372b    145
        276d9d8ca0bf52c780b5a3fc554fa69e74f934a3    145
        56832a697cb6dbce14700fca18cffcced367057f    144
        b2d2c70ed5de62cf8a1d4ded7dd141cfbbdd0388    142
        ceef2a24a2a82031246814b73e029edba51e8ea9    140
        8dc8d7ec2356b1b106eb3d723f3c234e03ab3f1e    137
        e38f123afecb40272ba4c47cb25c96a9533006fa    136
        53db7ac77dbb80d6f5c32ed5d19c1a8720078814    132
        6c14453c049b1ef4737b08d56c480419794f91c2    131
        fd824fc62b4753107e3db7704cd9e8a4a1c961f1    116
        c45f9495a76bf95d2633444817f1be8205ad542d    114
        12bb8a9740400ced27ae5a7d4c990ac3b7e3c77d    104
        3427a5a4065625363e28ac8e85a57a9436010e9c    103
        497935037e41a94d2ae02488d098c7abda9a30bc    102
        0d644205ecefdef33e3346bb3551f5e68dc57c58    102
        015aaf617598e413a35d6d2249e26b7f3c40adb7    101
        e90de4b883d9de64a47774ad7ad49ca6fd69d4fe    101
        db1c400ffb74f14390deba2140bd31d2e1dc5c4e     98
        7dc02db8b76fffbdfe29542da672d4d5fd5ed4ae     97
        2e205a44014ca7bdbf07fc32f3c9d17699671d03     96
        b2926913d95598ec0c007746d693fe3e466ff2d4     95
                                                    ...
        c09552a9ca740feb1a607a70955744bf980586e0      1
        93232606cbe402601e97ba75dbfd5bb8442c0ac2      1
        20fe91be96e8f9210ef659357c01fb87a04629a0      1
```

```
df9db2665c30b3cceccc1b9d21e0a2c6931df1cf      1
83ce0d8e31448e9cbe32c934db725ba06c8931ea      1
f63174ae8078034633394a2f20bd7d10fd490fef      1
bc6443bb456344dbdfc572eb968e0a5837804695      1
ebb5e3e81d4ee0d03500ffb964531554f9ac6c1d      1
c5d4a048f92e24020b4f1afcf08c104a9efe256f      1
4504d555ed2945f100bfcca72200d60d02d0a476      1
88d64c9cad700e5644c10371b469fc96018fcfe1      1
d48906d5eb86921deb4bd075a68dbb32690b1fa6      1
745e68ec6174a3c2b23f49e99eda9dfa6cb09ee7      1
0ccb993486dfa100eace7f35793b31cc309048dd      1
0d1d08864305b14728ad906fd0ca4892487a1522      1
36f02a0721c579eed5eb7019b5d143fe651a8e67      1
1f906928825d3fc2f2ec7a1ba2355faa9d2e5655      1
e9e79868f43be3879a8f92aa1c8cea6b7657a890      1
a01a568d797b737daf0a2834cde7c2f542c46cc2      1
20edf32bc2bbe549b80e2f587bb9c93b6f2ed349      1
fb3514baf9695867f8a75abb7b9bfc0d74714701      1
f1ad630657cd50db734c10fa793ba26b80e78bce      1
691ffcffea3504411ef642da7b2a34e3567e3323      1
c5930efce53d8c865ca42188533e5d8ee890a763      1
99cf31d041ca7d43ce37e3e789ff85c90f01145a      1
d11916c7939e22f00b23139a21cbaa34dd3c55de      1
b19e1fa09b30dd486ad551ed0409899460b72c10      1
fbc7bbad65de237bb5434c62007c56e575946a4a      1
39a67319c2d225db7f06ae03884a9ce8c766b682      1
ec2bb5a01af2c2f2aee40bba390da3f68b09fb1e      1
Name: email, Length: 5148, dtype: int64
```

In [6]:
```python
# Distribution of how many articles a user interacts with.
df['email'].value_counts().hist(bins=100)
plt.title('Distribution of how many articles a user interacts with\n')
plt.xlabel('The number of article interactions')
plt.ylabel('The number of users')
plt.show()
```



```
In [82]: # Fill in the median and maximum number of user_article interactios below

         median_val = 3 # 50% of individuals interact with __3__ number of articles or fewer.
         max_views_by_user = 364 # The maximum number of user-article interactions by any 1 user
```

2. Explore and remove duplicate articles from the **df_content** dataframe.

```
In [45]: # Find and explore duplicate articles
         unique_articles = []
         duplicate_articles = []
         for article in df_content['doc_full_name']:
             if article not in unique_articles:
                 unique_articles.append(article)
             else:
                 duplicate_articles.append(article)
         len(unique_articles), len(duplicate_articles)

         df_content_deduplicated = df_content.drop_duplicates(subset = 'doc_full_name')
         len(df_content_deduplicated)

Out[45]: 1051

In [7]: # Remove any rows that have the same article_id - only keep the first
        df_content = df_content.drop_duplicates(subset = 'doc_full_name')
        len(df_content)
```

```
Out[7]: 1051
```

3. Use the cells below to find:
a. The number of unique articles that have an interaction with a user.
b. The number of unique articles in the dataset (whether they have any interactions or not). **c.** The number of unique users in the dataset. (excluding null values) **d.** The number of user-article interactions in the dataset.

```
In [11]: df['article_id'].nunique()

Out[11]: 714

In [34]: len(df['email'])

Out[34]: 45993

In [32]: df['email'].nunique()

Out[32]: 5148

In [89]: unique_articles = 714 # The number of unique articles that have at least one interactio
         total_articles = 1051 # The number of unique articles on the IBM platform
         unique_users = 5148 # The number of unique users
         user_article_interactions = 45993 # The number of user-article interactions
```

4. Use the cells below to find the most viewed **article_id**, as well as how often it was viewed. After talking to the company leaders, the `email_mapper` function was deemed a reasonable way to map users to ids. There were a small number of null values, and it was found that all of these null values likely belonged to a single user (which is how they are stored using the function below).

```
In [87]: most_viewed_article_id = '1429.0' # The most viewed article in the dataset as a string
         max_views = 937 # The most viewed article in the dataset was viewed how many times?

In [8]: ## No need to change the code here - this will be helpful for later parts of the noteboo
        # Run this cell to map the user email to a user_id column and remove the email column

        def email_mapper():
            coded_dict = dict()
            cter = 1
            email_encoded = []

            for val in df['email']:
                if val not in coded_dict:
                    coded_dict[val] = cter
                    cter+=1

                email_encoded.append(coded_dict[val])
            #print(email_encoded)

            return email_encoded
        email_encoded = email_mapper()
        del df['email']
        df['user_id'] = email_encoded

        # show header
        df.head()
```

```
Out[8]:    article_id                                              title  user_id
        0      1430.0  using pixiedust for fast, flexible, and easier...        1
        1      1314.0       healthcare python streaming application demo        2
        2      1429.0         use deep learning for image classification        3
        3      1338.0          ml optimization using cognitive assistant        4
        4      1276.0          deploy your python model as a restful api        5
```

```
In [90]: ## If you stored all your results in the variable names above,
         ## you shouldn't need to change anything in this cell

         sol_1_dict = {
             '`50% of individuals have _____ or fewer interactions.`': median_val,
             '`The total number of user-article interactions in the dataset is ______.`': user_a
             '`The maximum number of user-article interactions by any 1 user is ______.`': max_v
             '`The most viewed article in the dataset was viewed _____ times.`': max_views,
             '`The article_id of the most viewed article is ______.`': most_viewed_article_id,
             '`The number of unique articles that have at least 1 rating ______.`': unique_artic
             '`The number of unique users in the dataset is ______`': unique_users,
             '`The number of unique articles on the IBM platform`': total_articles
         }

         # Test your dictionary against the solution
         t.sol_1_test(sol_1_dict)
```

```
It looks like you have everything right here! Nice job!
```

1.1.2 Part II: Rank-Based Recommendations

Unlike in the earlier lessons, we don't actually have ratings for whether a user liked an article or not. We only know that a user has interacted with an article. In these cases, the popularity of an article can really only be based on how often an article was interacted with.

`1.` Fill in the function below to return the **n** top articles ordered with most interactions as the top. Test your function using the tests below.

```
In [68]: article_ids = [0,2,4,6,8]
         df[df['article_id'].isin(article_ids)]['title'].drop_duplicates().values.tolist()
```

```
Out[68]: ['detect malfunctioning iot sensors with streaming analytics',
          'this week in data science (april 18, 2017)',
          'data science bowl 2017',
          'analyze ny restaurant data using spark in dsx']
```

```
In [165]: df['title'].value_counts().reset_index().head(10)['index']

Out[165]: 0           use deep learning for image classification
          1          insights from new york car accident reports
          2                        visualize car data with brunel
          3    use xgboost, scikit-learn & ibm watson machine...
          4    predicting churn with the spss random tree alg...
          5         healthcare python streaming application demo
          6    finding optimal locations of new store using d...
          7               apache spark lab, part 1: basic concepts
          8                analyze energy consumption in buildings
          9    gosales transactions for logistic regression m...
          Name: index, dtype: object

In [10]: def get_top_articles(n, df=df):
             '''
             INPUT:
             n - (int) the number of top articles to return
             df - (pandas dataframe) df as defined at the top of the notebook

             OUTPUT:
             top_articles - (list) A list of the top 'n' article titles

             '''
             # Your code here
             top_articles = list(df.title.value_counts().reset_index().head(n)['index'])

             return top_articles # Return the top article titles from df (not df_content)

         def get_top_article_ids(n, df=df):
             '''
             INPUT:
             n - (int) the number of top articles to return
             df - (pandas dataframe) df as defined at the top of the notebook

             OUTPUT:
             top_articles - (list) A list of the top 'n' article titles

             '''
             # Your code here
             top_articles = df['article_id'].value_counts().reset_index().head(n)['index'].tolis
             top_articles = [str(aid) for aid in top_articles]
             return top_articles # Return the top article ids

In [5]: print(get_top_articles(10))
        print(get_top_article_ids(10))

['use deep learning for image classification', 'insights from new york car accident reports', 'v
['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0', '1170.0', '1162.0', '1304
```

```
In [39]: # Test your function by returning the top 5, 10, and 20 articles
         top_5 = get_top_articles(5)
         top_10 = get_top_articles(10)
         top_20 = get_top_articles(20)

         # Test each of your three lists from above
         t.sol_2_test(get_top_articles)
```

```
Your top_5 looks like the solution list! Nice job.
Your top_10 looks like the solution list! Nice job.
Your top_20 looks like the solution list! Nice job.
```

1.1.3 Part III: User-User Based Collaborative Filtering

`1.` Use the function below to reformat the **df** dataframe to be shaped with users as the rows and articles as the columns.

- Each **user** should only appear in each **row** once.
- Each **article** should only show up in one **column**.
- **If a user has interacted with an article, then place a 1 where the user-row meets for that article-column**. It does not matter how many times a user has interacted with the article, all entries where a user has interacted with an article should be a 1.
- **If a user has not interacted with an item, then place a zero where the user-row meets for that article-column**.

Use the tests to make sure the basic structure of your matrix matches what is expected by the solution.

```
In [111]: df.groupby(['user_id', 'article_id'])['title'].max().unstack().notnull().astype(np.int
```

```
Out[111]: article_id  0.0     2.0     4.0     8.0     9.0     12.0    14.0    15.0    \
          user_id
          1                0       0       0       0       0       0       0       0
          2                0       0       0       0       0       0       0       0
          3                0       0       0       0       0       1       0       0
          4                0       0       0       0       0       0       0       0
          5                0       0       0       0       0       0       0       0

          article_id  16.0    18.0     ...    1434.0  1435.0  1436.0  1437.0  1439.0  \
          user_id                      ...
          1                0       0   ...         0       0       1       0       1
          2                0       0   ...         0       0       0       0       0
          3                0       0   ...         0       0       1       0       0
          4                0       0   ...         0       0       0       0       0
          5                0       0   ...         0       0       0       0       0
```

```
article_id  1440.0  1441.0  1442.0  1443.0  1444.0
user_id
1                0       0       0       0       0
2                0       0       0       0       0
3                0       0       0       0       0
4                0       0       0       0       0
5                0       0       0       0       0

[5 rows x 714 columns]
```

```
In [12]: # create the user-article matrix with 1's and 0's

         def create_user_item_matrix(df):
             '''
             INPUT:
             df - pandas dataframe with article_id, title, user_id columns

             OUTPUT:
             user_item - user item matrix

             Description:
             Return a matrix with user ids as rows and article ids on the columns with 1 values
             an article and a 0 otherwise
             '''
             # Fill in the function here
             #df.article_id = df.article_id.astype(str)
             user_item = df.groupby(['user_id', 'article_id'])['title'].count().unstack().notnul

             return user_item # return the user_item matrix

         user_item = create_user_item_matrix(df)
```

```
In [7]: ## Tests: You should just need to run this cell.  Don't change the code.
        assert user_item.shape[0] == 5149, "Oops!  The number of users in the user-article matri
        assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-article mat
        assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by user 1 doe
        print("You have passed our quick tests!  Please proceed!")
```

```
You have passed our quick tests!  Please proceed!
```

2. Complete the function below which should take a user_id and provide an ordered list of the most similar users to that user (from most similar to least similar). The returned result should not contain the provided user_id, as we know that each user is similar to him/herself. Because the results for each user here are binary, it (perhaps) makes sense to compute similarity as the dot product of two users.

Use the tests to test your function.

```
In [35]: similarity = user_item.dot(user_item.loc[1])
         similarity.head()
```

```
Out[35]: user_id
         1    36
         2     2
         3     6
         4     3
         5     0
         dtype: int64
```

```
In [13]: def find_similar_users(user_id, user_item=user_item):
             '''
             INPUT:
             user_id - (int) a user_id
             user_item - (pandas dataframe) matrix of users by articles:
                         1's when a user has interacted with an article, 0 otherwise

             OUTPUT:
             similar_users - (list) an ordered list where the closest users (largest dot product
                             are listed first

             Description:
             Computes the similarity of every pair of users based on the dot product
             Returns an ordered

             '''
             # compute similarity of each user to the provided user
             similarity = user_item.dot(user_item.loc[user_id])
             #print(similarity)
             # sort by similarity
             similarity = similarity.sort_values(ascending=False)

             # remove the own user's id
             similarity.drop(user_id, inplace=True)

             # create list of just the ids
             most_similar_users = similarity.index.tolist()

             return most_similar_users # return a list of the users in order from most to least
```

```
In [31]:
```

```
In [16]: # Do a spot check of your function
         print("The 10 most similar users to user 1 are: {}".format(find_similar_users(1)[:10]))
         print("The 5 most similar users to user 3933 are: {}".format(find_similar_users(3933)[:
         print("The 3 most similar users to user 46 are: {}".format(find_similar_users(46)[:3]))

The 10 most similar users to user 1 are: [3933, 23, 3782, 203, 4459, 131, 3870, 46, 4201, 5041]
The 5 most similar users to user 3933 are: [1, 23, 3782, 4459, 203]
The 3 most similar users to user 46 are: [4201, 23, 3782]
```

3. Now that you have a function that provides the most similar users to each user, you will want to use these users to find articles you can recommend. Complete the functions below to return the articles you would recommend to each user.

```
In [79]: np.where(user_item.loc[23][user_item.loc[23]

Out[79]: (array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
                  13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
                  26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
                  39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
                  52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
                  65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
                  78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
                  91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
                 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
                 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
                 130, 131, 132, 133, 134]),)
```

```
In [78]: list(user_item.loc[23][user_item.loc[23] == 1].index)

Out[78]: [2.0,
          12.0,
          14.0,
          16.0,
          26.0,
          28.0,
          29.0,
          33.0,
          43.0,
          50.0,
          74.0,
          76.0,
          108.0,
          109.0,
          120.0,
          124.0,
          131.0,
          164.0,
          193.0,
          194.0,
          210.0,
          213.0,
          221.0,
          223.0,
          236.0,
          237.0,
241.0,
252.0,
253.0,
283.0,
295.0,
299.0,
302.0,
316.0,
336.0,
337.0,
339.0,
348.0,
359.0,
362.0,
367.0,
409.0,
422.0,
444.0,
477.0,
482.0,
510.0,
517.0,
524.0,
617.0,
634.0,
641.0,
656.0,
658.0,
665.0,
682.0,
693.0,
720.0,
721.0,
729.0,
744.0,
761.0,
800.0,
812.0,
821.0,
825.0,
833.0,
843.0,
887.0,
910.0,
939.0,
943.0,
952.0,
957.0,
967.0,
969.0,
973.0,
981.0,
996.0,
1000.0,
1014.0,
1025.0,
1051.0,
1052.0,
1101.0,
1148.0,
1159.0,
1160.0,
1162.0,
1163.0,
1164.0,
1165.0,
1166.0,
1170.0,
1171.0,
1172.0,
1176.0,
1181.0,
1185.0,
1276.0,
1277.0,
1291.0,
1293.0,
1298.0,
1299.0,
1304.0,
1305.0,
1314.0,
1330.0,
1332.0,
1336.0,
1338.0,
1343.0,
1351.0,
1354.0,
1357.0,
1360.0,
1364.0,
1366.0,
1367.0,
1368.0,
1386.0,
         1391.0,
         1393.0,
         1395.0,
         1396.0,
         1423.0,
         1427.0,
         1428.0,
         1429.0,
         1430.0,
         1431.0,
         1432.0,
         1436.0,
         1439.0]
```

```
In [14]: def get_article_names(article_ids, df=df):
             '''
             INPUT:
             article_ids - (list) a list of article ids
             df - (pandas dataframe) df as defined at the top of the notebook

             OUTPUT:
             article_names - (list) a list of article names associated with the list of article
                             (this is identified by the title column)
             '''
             # Your code here

             article_names = df[df['article_id'].isin(article_ids)]['title'].drop_duplicates().v

             return article_names # Return the article names associated with list of article ids


         def get_user_articles(user_id, user_item=user_item):
             '''
             INPUT:
             user_id - (int) a user id
             user_item - (pandas dataframe) matrix of users by articles:
                         1's when a user has interacted with an article, 0 otherwise

             OUTPUT:
             article_ids - (list) a list of the article ids seen by the user
             article_names - (list) a list of article names associated with the list of article
                             (this is identified by the doc_full_name column in df_content)

             Description:
             Provides a list of the article_ids and article titles that have been seen by a user
             '''
             # Your code here
             article_ids = list(user_item.loc[user_id][user_item.loc[user_id] == 1].index)
```

```
    article_ids = [str(aid) for aid in article_ids]
    article_names = get_article_names(article_ids)

    return article_ids, article_names # return the ids and names


def user_user_recs(user_id, m=10):
    '''
    INPUT:
    user_id - (int) a user id
    m - (int) the number of recommendations you want for the user

    OUTPUT:
    recs - (list) a list of recommendations for the user

    Description:
    Loops through the users based on closeness to the input user_id
    For each user - finds articles the user hasn't seen before and provides them as rec
    Does this until m recommendations are found

    Notes:
    Users who are the same closeness are chosen arbitrarily as the 'next' user

    For the user where the number of recommended articles starts below m
    and ends exceeding m, the last items are chosen arbitrarily

    '''
    # Your code here

    recs = []
    read_ids, read_names = get_user_articles(user_id)
    similar_users_id = find_similar_users(user_id)
    print(read_ids)
    for user in similar_users_id:
        ids, names = get_user_articles(user)
        not_read = list(set(ids) - set(read_ids))
        recs.extend(not_read)
        if len(recs) >= m:
            break
        #print(recs)
    return recs[:m] # return your recommendations for this user_id
```

```
In [34]: # Check Results
         get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

```
['43.0', '109.0', '151.0', '268.0', '310.0', '329.0', '346.0', '390.0', '494.0', '525.0', '585.0
```

```
Out[34]: ['use sql with data in hadoop python',
 'better together: spss and data science experience',
 'brunel in jupyter',
 'challenges in deep learning',
 'this week in data science (april 25, 2017)',
 'recent trends in recommender systems',
 'get social with your notebooks in dsx',
 'using bigdl in dsx for deep learning on spark',
 'deep learning achievements over the past year ',
 'from scikit-learn model to cloud with wml client']
```

```
In [26]: # Test your functions here - No need to change this code - just run this cell
         assert set(get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0', '1427.0
         assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing (2015): un
         assert set(get_user_articles(20)[0]) == set(['1320.0', '232.0', '844.0'])
         assert set(get_user_articles(20)[1]) == set(['housing (2015): united states demographic
         assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1314.0', '14
         assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct high-re
         print("If this is all you see, you passed all of our tests!  Nice job!")
```

```
If this is all you see, you passed all of our tests!  Nice job!
```

4. Now we are going to improve the consistency of the **user_user_recs** function from above.

- Instead of arbitrarily choosing when we obtain users who are all the same closeness to a given user - choose the users that have the most total article interactions before choosing those with fewer article interactions.
- Instead of arbitrarily choosing articles from the user where the number of recommended articles starts below m and ends exceeding m, choose articles with the articles with the most total interactions before choosing those with fewer total interactions. This ranking should be what would be obtained from the **top_articles** function you wrote earlier.

```
In [135]: similarity=similarity.sort_values(ascending=False).drop(1).to_frame(name='similarity')
          similarity.head()
```

```
Out[135]:    user_id  similarity
          0     3933          35
          1       23          17
          2     3782          17
          3      203          15
          4     4459          15
```

```
In [132]: num_interactions = df.user_id.value_counts().to_frame('num_interactions')
          num_interactions.head()
```

```
Out[132]:       num_interactions
          23                 364
          3782               363
          98                 170
          3764               169
          203                160
```

```
In [142]: neighbors_df = similarity.merge(num_interactions, left_on='user_id',
                                    right_index=True).rename(columns={'user_id':'neighbor_id'})
          neighbors_df.head()

Out[142]:    neighbor_id  similarity  num_interactions
          0         3933          35                45
          1           23          17               364
          2         3782          17               363
          3          203          15               160
          4         4459          15               158

In [144]: neighbors_df.sort_values(by = ['similarity', 'num_interactions'], ascending=False).hea

Out[144]:    neighbor_id  similarity  num_interactions
          0         3933          35                45
          1           23          17               364
          2         3782          17               363
          3          203          15               160
          4         4459          15               158

In [31]: def get_top_sorted_users(user_id, df=df, user_item=user_item):
             '''
             INPUT:
             user_id - (int)
             df - (pandas dataframe) df as defined at the top of the notebook
             user_item - (pandas dataframe) matrix of users by articles:
                     1's when a user has interacted with an article, 0 otherwise


             OUTPUT:
             neighbors_df - (pandas dataframe) a dataframe with:
                             neighbor_id - is a neighbor user_id
                             similarity - measure of the similarity of each user to the provided
                             num_interactions - the number of articles viewed by the user - if a

             Other Details - sort the neighbors_df by the similarity and then by number of inter
                             highest of each is higher in the dataframe

             '''
             # Your code here

             #find user similarity w/ dot product
             similarity = user_item.dot(user_item.loc[user_id])

             # sort by similarity
             similarity = similarity.sort_values(ascending=False).drop(user_id).to_frame(name='s

             # get number of interactions for each user
             num_interactions = df.user_id.value_counts().to_frame('num_interactions')
```

```
    # combine the value counts with similarity
    neighbors_df = similarity.merge(num_interactions, left_on='user_id',
                        right_index=True).rename(columns={'user_id':'neighbor_id'})

    neighbors_df.sort_values(by=['similarity', 'num_interactions'], ascending=False, in

    return neighbors_df # Return the dataframe specified in the doc_string


def user_user_recs_part2(user_id, m=10):
    '''
    INPUT:
    user_id - (int) a user id
    m - (int) the number of recommendations you want for the user

    OUTPUT:
    recs - (list) a list of recommendations for the user by article id
    rec_names - (list) a list of recommendations for the user by article title

    Description:
    Loops through the users based on closeness to the input user_id
    For each user - finds articles the user hasn't seen before and provides them as rec
    Does this until m recommendations are found

    Notes:
    * Choose the users that have the most total article interactions
    before choosing those with fewer article interactions.

    * Choose articles with the articles with the most total interactions
    before choosing those with fewer total interactions.

    '''
    # Your code here

    neighbors_df = get_top_sorted_users(user_id)

    # set top-m neighbor_id
    top_neighbors_id = list(neighbors_df[:m]['neighbor_id'])

    # set article_ids seen by top-m neighbors
    recs = []
    for user in top_neighbors_id:
        recs.extend([str(aid) for aid in user_item.loc[user][user_item.loc[user] == 1].
        # recs = [str(aid) for aid in user_item.loc[user][user_item.loc[user] == 1].ind

    # remove duplicate
    recs = list(set(recs[:m]))
```

```
            # set article names and remove duplicate
            rec_names = list(set(df[df['article_id'].isin(recs)]['title']))

            return recs, rec_names
```

```
In [32]: # Quick spot check - don't change this code - just use it to test your functions
         rec_ids, rec_names = user_user_recs_part2(20, 10)
         print("The top 10 recommendations for user 20 are the following article ids:")
         print(rec_ids)
         print()
         print("The top 10 recommendations for user 20 are the following article names:")
         print(rec_names)
```

```
The top 10 recommendations for user 20 are the following article ids:
['362.0', '109.0', '232.0', '142.0', '125.0', '336.0', '164.0', '205.0', '302.0', '12.0']

The top 10 recommendations for user 20 are the following article names:
['dsx: hybrid mode', 'learn tensorflow and deep learning together and now!', 'timeseries data an
```

5. Use your functions from above to correctly fill in the solutions to the dictionary below. Then test your dictionary against the solution. Provide the code you need to answer each following the comments below.

```
In [ ]:

In [44]: get_top_sorted_users(1).iloc[0]

Out[44]: neighbor_id         3933
         similarity            35
         num_interactions      45
         Name: 0, dtype: int64

In [43]: get_top_sorted_users(131).iloc[9]

Out[43]: neighbor_id         242
         similarity           25
         num_interactions    148
         Name: 9, dtype: int64

In [45]: ### Tests with a dictionary of results

         user1_most_sim = 3933 # Find the user that is most similar to user 1
         user131_10th_sim = 242 # Find the 10th most similar user to user 131

In [46]: ## Dictionary Test Here
         sol_5_dict = {
             'The user that is most similar to user 1.': user1_most_sim,
            'The user that is the 10th most similar to user 131': user131_10th_sim,
        }

        t.sol_5_test(sol_5_dict)
```

```
This all looks good!  Nice job!
```

6. If we were given a new user, which of the above functions would you be able to use to make recommendations? Explain. Can you think of a better way we might make recommendations? Use the cell below to explain a better method for new users.

Provide your response here. Since the new given users have no interaction with any of the articles(has same similarity), collaborative filtering can not be used for making recommendations in this case. Instead, we use a rank based recommendation approach

7. Using your existing functions, provide the top 10 recommended articles you would provide for the a new user below. You can test your function against our thoughts to make sure we are all on the same page with how we might make a recommendation.

```
In [32]: new_user = '0.0'

         # What would your recommendations be for this new user '0.0'?  As a new user, they have
         # Provide a list of the top 10 article ids you would give to
         new_user_recs = get_top_article_ids(10) # Your recommendations here
         new_user_recs
```

```
Out[32]: ['1429.0',
          '1330.0',
          '1431.0',
          '1427.0',
          '1364.0',
          '1314.0',
          '1293.0',
          '1170.0',
          '1162.0',
          '1304.0']
```

```
In [33]: assert set(new_user_recs) == set(['1314.0','1429.0','1293.0','1427.0','1162.0','1364.0'

         print("That's right!  Nice job!")
```

```
That's right!  Nice job!
```

1.1.4 Part IV: Content Based Recommendations (EXTRA - NOT REQUIRED)

Another method we might use to make recommendations is to perform a ranking of the highest ranked articles associated with some term. You might consider content to be the **doc_body**, **doc_description**, or **doc_full_name**. There isn't one way to create a content based recommendation, especially considering that each of these columns hold content related information.

1. Use the function body below to create a content based recommender. Since there isn't one right answer for this recommendation tactic, no test functions are provided. Feel free to change the function inputs if you decide you want to try a method that requires more input values. The input values are currently set with one idea in mind that you may use to make content based recommendations. One additional idea is that you might want to choose the most popular recommendations that meet your 'content criteria', but again, there is a lot of flexibility in how you might make these recommendations.

1.1.5 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

```
In [ ]: def make_content_recs():
            '''
            INPUT:

            OUTPUT:

            '''
```

2. Now that you have put together your content-based recommendation system, use the cell below to write a summary explaining how your content based recommender works. Do you see any possible improvements that could be made to your function? Is there anything novel about your content based recommender?

1.1.6 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

Write an explanation of your content based recommendation system here.

3. Use your content-recommendation system to make recommendations for the below scenarios based on the comments. Again no tests are provided here, because there isn't one right answer that could be used to find these content based recommendations.

1.1.7 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

```
In [ ]: # make recommendations for a brand new user


        # make a recommendations for a user who only has interacted with article id '1427.0'
```

1.1.8 Part V: Matrix Factorization

In this part of the notebook, you will build use matrix factorization to make article recommendations to the users on the IBM Watson Studio platform.

1. You should have already created a **user_item** matrix above in **question 1** of **Part III** above. This first question here will just require that you run the cells to get things set up for the rest of **Part V** of the notebook.

```
In [16]: # Load the matrix here
         user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

```
In [37]: # quick look at the matrix
         user_item_matrix.head()
```

```
Out[37]: article_id  0.0  100.0  1000.0  1004.0  1006.0  1008.0  101.0  1014.0  1015.0  \
         user_id
         1           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         2           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         3           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         4           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         5           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0

         article_id  1016.0  ...    977.0  98.0  981.0  984.0  985.0  986.0  990.0  \
         user_id             ...
         1              0.0  ...      0.0   0.0    1.0    0.0    0.0    0.0    0.0
         2              0.0  ...      0.0   0.0    0.0    0.0    0.0    0.0    0.0
         3              0.0  ...      1.0   0.0    0.0    0.0    0.0    0.0    0.0
         4              0.0  ...      0.0   0.0    0.0    0.0    0.0    0.0    0.0
         5              0.0  ...      0.0   0.0    0.0    0.0    0.0    0.0    0.0

         article_id  993.0  996.0  997.0
         user_id
         1             0.0    0.0    0.0
         2             0.0    0.0    0.0
         3             0.0    0.0    0.0
         4             0.0    0.0    0.0
         5             0.0    0.0    0.0

         [5 rows x 714 columns]
```

2. In this situation, you can use Singular Value Decomposition from numpy on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

```
In [17]: # Perform SVD on the User-Item Matrix Here

         u, s, vt = np.linalg.svd(user_item_matrix, full_matrices=False) # use the built in to g
         u.shape, s.shape,vt.shape
```

```
Out[17]: ((5149, 714), (714,), (714, 714))
```

```
In [45]: s
```

```
Out[45]: array([5.53457037e+01, 2.32486418e+01, 2.17600228e+01, 2.06583341e+01,
                1.99708867e+01, 1.95569462e+01, 1.91632342e+01, 1.87197508e+01,
                1.83438615e+01, 1.80639261e+01, 1.76377686e+01, 1.74335474e+01,
                1.74301733e+01, 1.70930876e+01, 1.67638529e+01, 1.64669871e+01,
                1.64559512e+01, 1.63068949e+01, 1.61666030e+01, 1.57444047e+01,
1.57038805e+01, 1.55878977e+01, 1.55013389e+01, 1.53746023e+01,
1.52212789e+01, 1.50401111e+01, 1.49482365e+01, 1.47737529e+01,
1.46704332e+01, 1.46097896e+01, 1.45312074e+01, 1.44025711e+01,
1.43073643e+01, 1.40654598e+01, 1.38995696e+01, 1.38576259e+01,
1.37810084e+01, 1.36388289e+01, 1.36055830e+01, 1.35749215e+01,
1.35196309e+01, 1.33503810e+01, 1.33107854e+01, 1.31843779e+01,
1.30764996e+01, 1.30470484e+01, 1.29137238e+01, 1.28999043e+01,
1.27923679e+01, 1.26848013e+01, 1.26361984e+01, 1.26069082e+01,
1.24039390e+01, 1.23562882e+01, 1.22810586e+01, 1.21600337e+01,
1.21294028e+01, 1.20983395e+01, 1.20100229e+01, 1.19449827e+01,
1.19335388e+01, 1.18192562e+01, 1.17660208e+01, 1.17113450e+01,
1.16297027e+01, 1.15799753e+01, 1.14721374e+01, 1.14021968e+01,
1.13255385e+01, 1.12968801e+01, 1.12121296e+01, 1.11029330e+01,
1.10772469e+01, 1.09611247e+01, 1.09451910e+01, 1.08831594e+01,
1.08319400e+01, 1.07571231e+01, 1.06870041e+01, 1.06441857e+01,
1.05803518e+01, 1.05366876e+01, 1.04634654e+01, 1.04180469e+01,
1.03771561e+01, 1.03662447e+01, 1.02972427e+01, 1.02237610e+01,
1.01930571e+01, 1.01339378e+01, 1.01169304e+01, 1.00605483e+01,
1.00018886e+01, 9.93186806e+00, 9.91523152e+00, 9.88725887e+00,
9.83833681e+00, 9.81794312e+00, 9.79808989e+00, 9.74279916e+00,
9.65791949e+00, 9.61063591e+00, 9.55883047e+00, 9.51331520e+00,
9.48748290e+00, 9.45900984e+00, 9.39838850e+00, 9.36018902e+00,
9.32590972e+00, 9.27660547e+00, 9.22234450e+00, 9.19492811e+00,
9.12079895e+00, 9.08028506e+00, 9.04719849e+00, 9.01448607e+00,
8.94330252e+00, 8.91267694e+00, 8.87882780e+00, 8.85559026e+00,
8.81374437e+00, 8.78139555e+00, 8.70823928e+00, 8.64315751e+00,
8.59659084e+00, 8.59089274e+00, 8.53272777e+00, 8.50286441e+00,
8.48514063e+00, 8.42762658e+00, 8.42009926e+00, 8.40527042e+00,
8.37288303e+00, 8.35604403e+00, 8.32677986e+00, 8.30086653e+00,
8.24938269e+00, 8.20057355e+00, 8.18933771e+00, 8.14272125e+00,
8.11444934e+00, 8.10420578e+00, 8.08620432e+00, 8.03714827e+00,
8.03435923e+00, 7.98780703e+00, 7.94742524e+00, 7.90823847e+00,
7.87805786e+00, 7.85398957e+00, 7.82775774e+00, 7.81104366e+00,
7.76390298e+00, 7.74886149e+00, 7.73839798e+00, 7.69615259e+00,
7.65996085e+00, 7.63113016e+00, 7.60482040e+00, 7.58811574e+00,
7.54589770e+00, 7.52310211e+00, 7.49563792e+00, 7.44964803e+00,
7.43085513e+00, 7.40567789e+00, 7.37869555e+00, 7.35476941e+00,
7.30586187e+00, 7.29266284e+00, 7.26523271e+00, 7.22145211e+00,
7.19880747e+00, 7.15331558e+00, 7.13863328e+00, 7.13034140e+00,
7.08541261e+00, 7.04808650e+00, 7.04357453e+00, 7.03050132e+00,
7.01032763e+00, 6.98657129e+00, 6.95974054e+00, 6.91059037e+00,
6.86950831e+00, 6.83127731e+00, 6.82586163e+00, 6.80808512e+00,
6.78169211e+00, 6.77536696e+00, 6.74540883e+00, 6.72600536e+00,
6.69537884e+00, 6.67322552e+00, 6.65197879e+00, 6.63579835e+00,
6.60456904e+00, 6.58102192e+00, 6.56098147e+00, 6.52976231e+00,
6.51227257e+00, 6.49675351e+00, 6.47147417e+00, 6.44801276e+00,
6.41872750e+00, 6.37532316e+00, 6.35586647e+00, 6.32965091e+00,
6.30709431e+00, 6.27623526e+00, 6.25938849e+00, 6.23982897e+00,
6.23078247e+00, 6.20891126e+00, 6.20088452e+00, 6.17234111e+00,
6.14321030e+00, 6.10348533e+00, 6.08255754e+00, 6.05421974e+00,
6.04398767e+00, 6.02552069e+00, 6.00245107e+00, 5.98865778e+00,
5.96981173e+00, 5.96066667e+00, 5.93584006e+00, 5.88738844e+00,
5.87946429e+00, 5.86845213e+00, 5.85225619e+00, 5.82207158e+00,
5.78835052e+00, 5.77230349e+00, 5.75247863e+00, 5.73248156e+00,
5.70414494e+00, 5.70350083e+00, 5.68447651e+00, 5.66670739e+00,
5.65713461e+00, 5.61679181e+00, 5.60583162e+00, 5.58725591e+00,
5.57313212e+00, 5.53267945e+00, 5.52220473e+00, 5.51223540e+00,
5.50056731e+00, 5.49066387e+00, 5.46215406e+00, 5.42562817e+00,
5.39816863e+00, 5.38684198e+00, 5.37887548e+00, 5.34758543e+00,
5.33933318e+00, 5.31134464e+00, 5.30441619e+00, 5.29467028e+00,
5.27110936e+00, 5.26113856e+00, 5.23614647e+00, 5.23013414e+00,
5.21590406e+00, 5.20553050e+00, 5.16160064e+00, 5.14699168e+00,
5.11957090e+00, 5.11021148e+00, 5.10045361e+00, 5.07484324e+00,
5.05643010e+00, 5.04117683e+00, 5.03175203e+00, 5.00799720e+00,
4.98757351e+00, 4.97415479e+00, 4.96850526e+00, 4.95510476e+00,
4.91914825e+00, 4.90926126e+00, 4.88879760e+00, 4.87780406e+00,
4.86462924e+00, 4.85409363e+00, 4.82792553e+00, 4.81112468e+00,
4.80519464e+00, 4.77407754e+00, 4.77262582e+00, 4.76052276e+00,
4.74233756e+00, 4.73909899e+00, 4.72464868e+00, 4.70695590e+00,
4.69133076e+00, 4.66230409e+00, 4.63556566e+00, 4.62269159e+00,
4.60961205e+00, 4.59337688e+00, 4.58622380e+00, 4.57372843e+00,
4.54722829e+00, 4.52991503e+00, 4.52509516e+00, 4.50947617e+00,
4.49897278e+00, 4.49449383e+00, 4.46943058e+00, 4.45395221e+00,
4.44291146e+00, 4.42280618e+00, 4.41111791e+00, 4.40218052e+00,
4.36206135e+00, 4.35549651e+00, 4.34546507e+00, 4.34297454e+00,
4.32805340e+00, 4.31003682e+00, 4.30429999e+00, 4.29038786e+00,
4.27087575e+00, 4.25139418e+00, 4.24359271e+00, 4.23062900e+00,
4.22467891e+00, 4.20344923e+00, 4.18514808e+00, 4.17221161e+00,
4.16329502e+00, 4.15958969e+00, 4.13735636e+00, 4.12901197e+00,
4.11852107e+00, 4.10613561e+00, 4.08313378e+00, 4.07993926e+00,
4.06191178e+00, 4.05310105e+00, 4.03960883e+00, 4.03092237e+00,
4.01257079e+00, 4.00319049e+00, 3.99357696e+00, 3.98303572e+00,
3.96252519e+00, 3.94562253e+00, 3.93874338e+00, 3.93227174e+00,
3.90554164e+00, 3.90196940e+00, 3.87382484e+00, 3.86845387e+00,
3.85990642e+00, 3.83996220e+00, 3.83638866e+00, 3.83135703e+00,
3.82157806e+00, 3.81859624e+00, 3.80393146e+00, 3.78729611e+00,
3.77941994e+00, 3.76149598e+00, 3.75825720e+00, 3.73133267e+00,
3.72308854e+00, 3.71955067e+00, 3.71006940e+00, 3.69765827e+00,
3.66976594e+00, 3.66123420e+00, 3.64933183e+00, 3.63935763e+00,
3.63006856e+00, 3.61740170e+00, 3.60945254e+00, 3.59155064e+00,
3.57488301e+00, 3.57044582e+00, 3.55840236e+00, 3.53422815e+00,
3.52344797e+00, 3.50278593e+00, 3.50010535e+00, 3.49755759e+00,
3.48298876e+00, 3.47905220e+00, 3.46303567e+00, 3.45210778e+00,
3.44693045e+00, 3.41939036e+00, 3.41019313e+00, 3.38994417e+00,
3.38031958e+00, 3.37348726e+00, 3.36502994e+00, 3.35355995e+00,
3.34155440e+00, 3.32420230e+00, 3.30423719e+00, 3.29859320e+00,
3.27906363e+00, 3.27564600e+00, 3.26454825e+00, 3.25582792e+00,
3.24253348e+00, 3.23060930e+00, 3.22471924e+00, 3.21282005e+00,
3.20352919e+00, 3.19516318e+00, 3.19144412e+00, 3.18436050e+00,
3.16402966e+00, 3.15858376e+00, 3.14605093e+00, 3.13955409e+00,
3.11063288e+00, 3.10651061e+00, 3.09313055e+00, 3.08022823e+00,
3.06778768e+00, 3.05694322e+00, 3.05153813e+00, 3.05086745e+00,
3.04338916e+00, 3.02858899e+00, 3.01198157e+00, 2.99657852e+00,
2.98402312e+00, 2.98073257e+00, 2.96382273e+00, 2.95399643e+00,
2.94357687e+00, 2.92976181e+00, 2.92281231e+00, 2.91588583e+00,
2.91343448e+00, 2.90636535e+00, 2.89063390e+00, 2.87264247e+00,
2.86779118e+00, 2.85952610e+00, 2.83365704e+00, 2.81741778e+00,
2.81432389e+00, 2.80046069e+00, 2.78790133e+00, 2.78622139e+00,
2.76872382e+00, 2.75894203e+00, 2.74971815e+00, 2.73900358e+00,
2.72222412e+00, 2.71878522e+00, 2.70387523e+00, 2.69910111e+00,
2.68412441e+00, 2.65402752e+00, 2.64698577e+00, 2.64433340e+00,
2.63597647e+00, 2.62770090e+00, 2.61441733e+00, 2.59289681e+00,
2.58961932e+00, 2.57741018e+00, 2.57261016e+00, 2.56762542e+00,
2.55865031e+00, 2.52887544e+00, 2.50641338e+00, 2.49378477e+00,
2.48124464e+00, 2.48052049e+00, 2.47013378e+00, 2.45105230e+00,
2.44032568e+00, 2.43328763e+00, 2.42849630e+00, 2.42101957e+00,
2.40895291e+00, 2.39176455e+00, 2.37387972e+00, 2.36548317e+00,
2.33643789e+00, 2.31076457e+00, 2.30152967e+00, 2.29179930e+00,
2.29166856e+00, 2.28120204e+00, 2.27193089e+00, 2.25834323e+00,
2.25310784e+00, 2.23791393e+00, 2.23074938e+00, 2.22475822e+00,
2.21669665e+00, 2.21086883e+00, 2.19127974e+00, 2.18470277e+00,
2.17616391e+00, 2.16337382e+00, 2.14959926e+00, 2.14220931e+00,
2.13315549e+00, 2.11725659e+00, 2.10776972e+00, 2.09552989e+00,
2.09077554e+00, 2.07588643e+00, 2.07438003e+00, 2.06299073e+00,
2.05301240e+00, 2.03619233e+00, 2.02379345e+00, 2.01485936e+00,
2.00858207e+00, 1.99861610e+00, 1.99537789e+00, 1.96417523e+00,
1.95874234e+00, 1.95435504e+00, 1.94813566e+00, 1.93960031e+00,
1.92819813e+00, 1.91818728e+00, 1.91290383e+00, 1.90721084e+00,
1.89201878e+00, 1.87902208e+00, 1.86512209e+00, 1.85949693e+00,
1.85809402e+00, 1.83862972e+00, 1.83709038e+00, 1.82292300e+00,
1.80962973e+00, 1.79939288e+00, 1.79498641e+00, 1.78516123e+00,
1.76936451e+00, 1.76367540e+00, 1.75888313e+00, 1.74697359e+00,
1.74072011e+00, 1.73853051e+00, 1.72961438e+00, 1.71640328e+00,
1.70789308e+00, 1.69747501e+00, 1.69136455e+00, 1.68771024e+00,
1.67316665e+00, 1.66886095e+00, 1.64761890e+00, 1.63630945e+00,
1.63070917e+00, 1.62423427e+00, 1.61462495e+00, 1.60500711e+00,
1.58958746e+00, 1.58567210e+00, 1.57404822e+00, 1.56795054e+00,
1.55163340e+00, 1.54027459e+00, 1.53070874e+00, 1.51765625e+00,
1.51456650e+00, 1.50507953e+00, 1.49657495e+00, 1.48700541e+00,
1.48282660e+00, 1.48034701e+00, 1.45417472e+00, 1.44638689e+00,
1.43708481e+00, 1.43079690e+00, 1.42705387e+00, 1.40253243e+00,
1.38961745e+00, 1.38356917e+00, 1.37248825e+00, 1.36508086e+00,
1.35621822e+00, 1.33974671e+00, 1.33385383e+00, 1.33181740e+00,
1.32289038e+00, 1.31151825e+00, 1.30266774e+00, 1.29575171e+00,
       1.28972572e+00, 1.27395791e+00, 1.26319111e+00, 1.25992490e+00,
       1.24380681e+00, 1.23016975e+00, 1.23005940e+00, 1.21715372e+00,
       1.20767744e+00, 1.19808258e+00, 1.19486895e+00, 1.17830756e+00,
       1.16606520e+00, 1.16331107e+00, 1.15263219e+00, 1.15010409e+00,
       1.13549495e+00, 1.13429860e+00, 1.11607200e+00, 1.10713485e+00,
       1.10663241e+00, 1.09438820e+00, 1.09059946e+00, 1.08067651e+00,
       1.07506271e+00, 1.06658024e+00, 1.06159999e+00, 1.05658342e+00,
       1.04777664e+00, 1.03810761e+00, 1.03299165e+00, 1.02799203e+00,
       1.02132735e+00, 1.01299848e+00, 1.00799704e+00, 1.00000000e+00,
       1.00000000e+00, 9.97151293e-01, 9.95646653e-01, 9.92377954e-01,
       9.91093970e-01, 9.88720232e-01, 9.87577772e-01, 9.86873498e-01,
       9.78684152e-01, 9.78097749e-01, 9.75592917e-01, 9.73164909e-01,
       9.70631406e-01, 9.66533661e-01, 9.60145188e-01, 9.55902675e-01,
       9.52086327e-01, 9.48015987e-01, 9.44924709e-01, 9.28371887e-01,
       9.09968959e-01, 9.08495993e-01, 9.05365913e-01, 8.96691589e-01,
       8.81762234e-01, 8.65122835e-01, 8.53057696e-01, 8.48314426e-01,
       8.39616121e-01, 8.33471361e-01, 8.23228199e-01, 8.12138781e-01,
       7.97612585e-01, 7.84907175e-01, 7.71300265e-01, 7.60149739e-01,
       7.56924159e-01, 7.51075704e-01, 7.43328867e-01, 7.25717885e-01,
       7.15666246e-01, 7.04395015e-01, 6.99101196e-01, 6.91625676e-01,
       6.78114990e-01, 6.73228217e-01, 6.59772497e-01, 6.56037259e-01,
       6.42698362e-01, 6.26688782e-01, 6.15608898e-01, 6.11677828e-01,
       6.09907904e-01, 5.94162404e-01, 5.82797981e-01, 5.81314355e-01,
       5.76307450e-01, 5.17766691e-01, 4.98033321e-01, 4.93127928e-01,
       4.87002388e-01, 4.74889484e-01, 4.54666156e-01, 4.42244880e-01,
       4.01197648e-01, 3.96104245e-01, 3.68386761e-01, 3.52762180e-01,
       3.25557926e-01, 3.00974784e-01, 2.82709464e-01, 2.68497243e-01,
       2.45784298e-14, 4.38358974e-15, 4.38358974e-15, 4.38358974e-15,
       4.38358974e-15, 4.38358974e-15, 4.38358974e-15, 1.35360768e-15,
       7.67704043e-16, 7.63703710e-16])
```

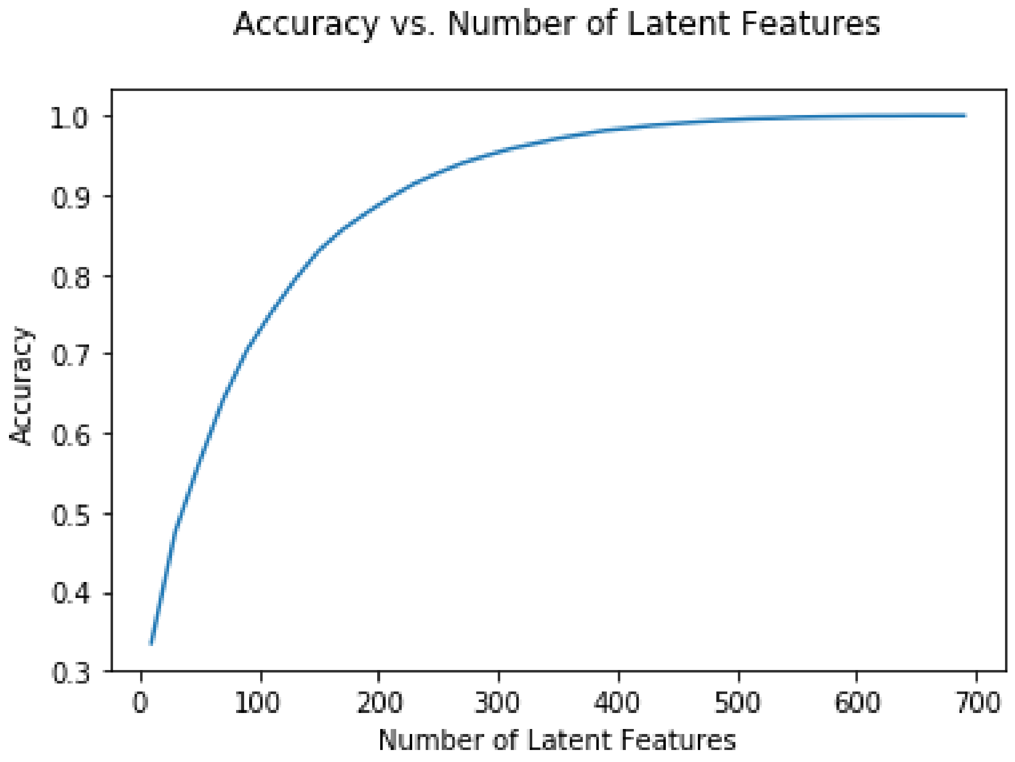
Provide your response here.

This case is different than the observed in the lesson because this matrix have missing values. SVD perfectly works with this satisfied condition. Thus, we do not need to fill values since a 1 denotes interaction and 0 denotes no interaction with the item.

3. Now for the tricky part, how do we choose the number of latent features to use? Running the below cell, you can see that as the number of latent features increases, we obtain a lower error rate on making predictions for the 1 and 0 values in the user-item matrix. Run the cell below to get an idea of how the accuracy improves as we increase the number of latent features.

```
In [46]: num_latent_feats = np.arange(10,700+10,20)
         sum_errs = []

         for k in num_latent_feats:
             # restructure with k latent features
             s_new, u_new, vt_new = np.diag(s[:k]), u[:, :k], vt[:k, :]

             # take dot product
```

```
        user_item_est = np.around(np.dot(np.dot(u_new, s_new), vt_new))

        # compute error for each prediction to actual value
        diffs = np.subtract(user_item_matrix, user_item_est)

        # total errors and keep track of them
        err = np.sum(np.sum(np.abs(diffs)))
        sum_errs.append(err)


    plt.plot(num_latent_feats, 1 - np.array(sum_errs)/df.shape[0]);
    plt.xlabel('Number of Latent Features');
    plt.ylabel('Accuracy');
    plt.title('Accuracy vs. Number of Latent Features\n');
```



4. From the above, we can't really be sure how many features to use, because simply having a better way to predict the 1's and 0's of the matrix doesn't exactly give us an indication of if we are able to make good recommendations. Instead, we might split our dataset into a training and test set of data, as shown in the cell below.

Use the code from question 3 to understand the impact on accuracy of the training and test sets of data with different numbers of latent features. Using the split below:

- How many users can we make predictions for in the test set?

- How many users are we not able to make predictions for because of the cold start problem?
- How many articles can we make predictions for in the test set?

- How many articles are we not able to make predictions for because of the cold start problem?

```
In [19]: df_train = df.head(40000)
         df_test = df.tail(5993)

         def create_test_and_train_user_item(df_train, df_test):
             '''
             INPUT:
             df_train - training dataframe
             df_test - test dataframe

             OUTPUT:
             user_item_train - a user-item matrix of the training dataframe
                               (unique users for each row and unique articles for each column)
             user_item_test - a user-item matrix of the testing dataframe
                             (unique users for each row and unique articles for each column)
             test_idx - all of the test user ids
             test_arts - all of the test article ids

             '''
             # Your code here

             user_item_train = create_user_item_matrix(df_train)
             user_item_test = create_user_item_matrix(df_test)

             test_idx = list(set(user_item_test.index))
             test_arts = list(set(user_item_test.columns))

             return user_item_train, user_item_test, test_idx, test_arts

         user_item_train, user_item_test, test_idx, test_arts = create_test_and_train_user_item(

In [65]: user_item_train.shape, user_item_test.shape

Out[65]: ((4487, 714), (682, 574))

In [29]: common_idx = np.intersect1d(user_item_train.index, test_idx) #users in both train and t
         common_cols = user_item_train.columns.intersection(test_arts)
         print("The common users for both testing and training sets are:\n",common_idx, "with le
         print("\nThe common articles for both testing and training sets are;\n",common_cols, "w

The common users for both testing and training sets are:
 [2917 3024 3093 3193 3527 3532 3684 3740 3777 3801 3968 3989 3990 3998
 4002 4204 4231 4274 4293 4487] with length:  20

The common users for both testing and training sets are;
```

```
Float64Index([   0.0,    2.0,    4.0,    8.0,    9.0,   12.0,   14.0,   15.0,
                16.0,   18.0,
              ...
              1432.0, 1433.0, 1434.0, 1435.0, 1436.0, 1437.0, 1439.0, 1440.0,
              1441.0, 1443.0],
             dtype='float64', length=574) with length:  574
```

```
In [30]: # Number of users in the test set are we not able to make predictions for because of th
         len(set(user_item_test.index) - set(user_item_train.index))

Out[30]: 662

In [31]: # Number of articles in the test set are we unable to make predictions for because of t
         len(set(user_item_test.columns) - set(user_item_train.columns))

Out[31]: 0

In [24]: # Replace the values in the dictionary below
         a = 662
         b = 574
         c = 20
         d = 0


         sol_4_dict = {
             'How many users can we make predictions for in the test set?': c,
             'How many users in the test set are we not able to make predictions for because of
             'How many movies can we make predictions for in the test set?': b,
             'How many movies in the test set are we not able to make predictions for because of
         }

         t.sol_4_test(sol_4_dict)

Awesome job!  That's right!  All of the test movies are in the training data, but there are only
```

5. Now use the **user_item_train** dataset from above to find U, S, and V transpose using SVD. Then find the subset of rows in the **user_item_test** dataset that you can predict using this matrix decomposition with different numbers of latent features to see how many features makes sense to keep based on the accuracy on the test data. This will require combining what was done in questions 2 - 4.

Use the cells below to explore how well SVD works towards making predictions for recommendations on the test data.

```
In [61]: # fit SVD on the user_item_train matrix
         u_train, s_train, vt_train = np.linalg.svd(np.array(user_item_train, dtype='int'), full
         print('train: ', u_train.shape, s_train.shape, vt.shape)
```

```
train:  (4487, 714) (714,) (714, 714)
```

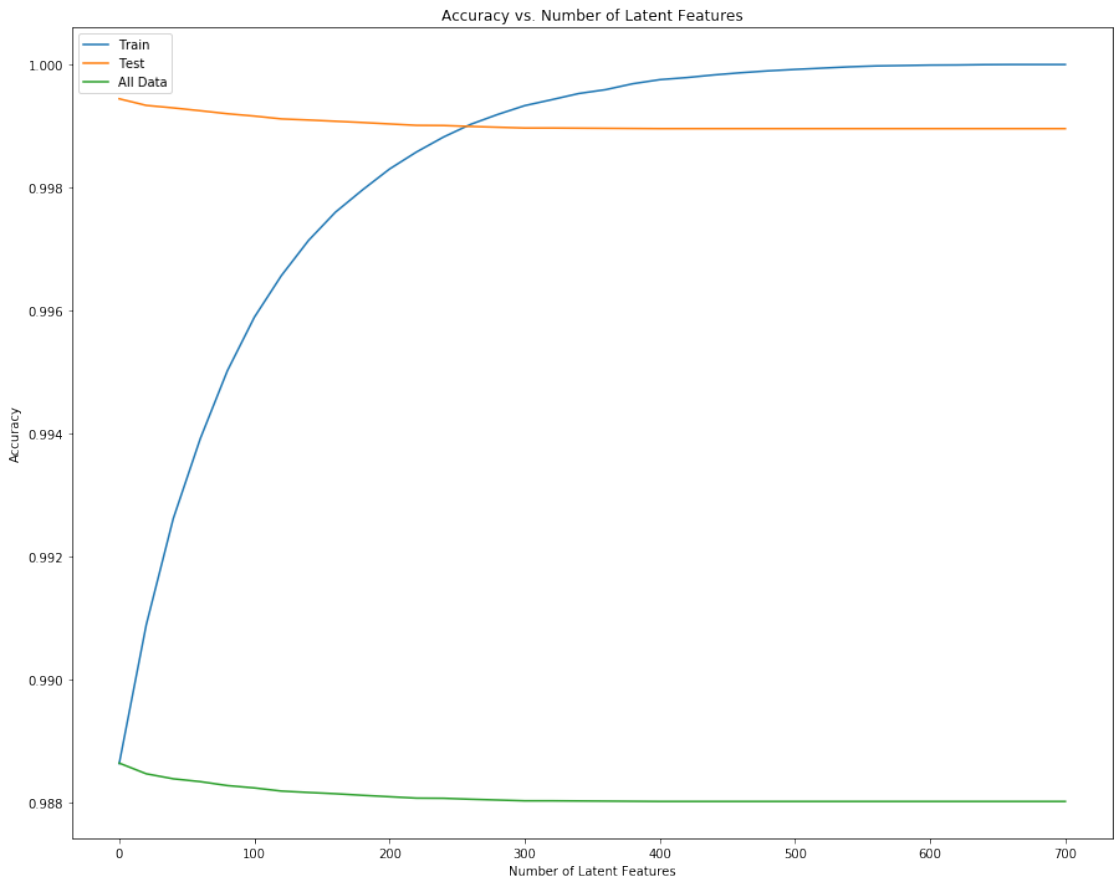
```
In [62]: # get the index of the common users and articles in both train and test idx
         train_common_idx = user_item_train.index.isin(test_idx)
         train_common_col = user_item_train.columns.isin(test_arts)

         # Get the subset of user and article matrices that could be predicted(i.e., calculated
         u_test = u_train[train_common_idx, :]
         vt_test= vt_train[:, train_common_col]
         print('test: ', u_test.shape, vt_test.shape)
```

```
test:  (20, 714) (714, 574)
```

```
In [63]: # finding the subset of users and articles existing in both training annd testing set (
         test_users = np.intersect1d(user_item_train.index, user_item_test.index)
         test_articles = user_item_train.columns.intersection(user_item_test.columns)
         user_item_test_predictable = user_item_test.loc[test_users, test_articles]
```

```
In [69]: # Use these cells to see how well you can use the training
         # decomposition to predict on test data

         no_latent_feats = np.arange(0,700+10,20)
         sum_errs_train = []
         sum_errs_test = []
         all_errs = []

         for k in no_latent_feats:
             # restructure with k latent features
             s_train_new, u_train_new, vt_train_new = np.diag(s_train[:k]), u_train[:, :k], vt_t
             u_test_new, vt_test_new = u_test[:, :k], vt_test[:k, :]

             # take dot product
             user_item_train_preds = np.around(np.dot(np.dot(u_train_new, s_train_new), vt_train
             user_item_test_preds = np.around(np.dot(np.dot(u_test_new, s_train_new), vt_test_ne
             all_errs.append(1 - ((np.sum(user_item_test_preds)+np.sum(np.sum(user_item_test)))/


             # compute error for each prediction to actual value
             diffs_train = np.subtract(user_item_train, user_item_train_preds)
             diffs_test = np.subtract(user_item_test_predictable, user_item_test_preds)

             # total errors and keep track of them
             err_train = np.sum(np.sum(np.abs(diffs_train)))
             err_test = np.sum(np.sum(np.abs(diffs_test)))

             sum_errs_train.append(err_train)
             sum_errs_test.append(err_test)
```

```
plt.figure(figsize=(15,12))
plt.plot(no_latent_feats, 1 - np.array(sum_errs_train)/(user_item_train.shape[0]*user_i
plt.plot(no_latent_feats, 1 - np.array(sum_errs_test)/(user_item_test.shape[0]*user_ite
plt.plot(no_latent_feats, all_errs, label='All Data');
plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features');
plt.legend();
```



In [ ]:

In [ ]:

In [ ]:

6. Use the cell below to comment on the results you found in the previous question. Given the circumstances of your results, discuss what you might do to determine if the recommendations you make with any of the above recommendation systems are an improvement to how users currently find articles?

Your response here. * The training accuracy slightly decreases with the test accuracy indicative of overfitting. Therefore, it is advisable to use a lower number of latent features to minimize this effect. * It was clearly seen that only 20 users present in the test set are contained in the training set. This limits the predictive power of SVD as it suffers from a cold start problem * In this case, it will be more beneficial to use a rank based recommendation system * However, the online testing(A/B) is the most adequate way of actually testing which of the recommendation methods is more relevant to the user. Here, we deploy recommendations and just watch metrics caarefully to determine if the new recommendation system is better tahn the old. * The null hypothesis is that new recommendation system does not improve users' interaction with the articles while the alternative hypothesis stipulates it does * The first half is the control group which receive recommendations based on previous method while the second half is the experimental group which receive recommendations based on new methods

Extras Using your workbook, you could now save your recommendations for each user, develop a class to make new predictions and update your results, and make a flask app to deploy your results. These tasks are beyond what is required for this project. However, from what you learned in the lessons, you certainly capable of taking these tasks on to improve upon your work here!

1.2 Conclusion

Congratulations! You have reached the end of the Recommendations with IBM project!

Tip: Once you are satisfied with your work here, check over your report to make sure that it is satisfies all the areas of the rubric. You should also probably remove all of the "Tips" like this one so that the presentation is as polished as possible.

1.3 Directions to Submit

Before you submit your project, you need to create a .html or .pdf version of this notebook in the workspace here. To do that, run the code cell below. If it worked correctly, you should get a return code of 0, and you should see the generated .html file in the workspace directory (click on the orange Jupyter icon in the upper left).

Alternatively, you can download this report as .html via the **File** > **Download as** submenu, and then manually upload it into the workspace directory by clicking on the orange Jupyter icon in the upper left, then using the Upload button.

Once you've done this, you can submit your project by clicking on the "Submit Project" button in the lower right here. This will create and submit a zip file with this .ipynb doc and the .html or .pdf version you created. Congratulations!

```
In [ ]: from subprocess import call
        call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```