Recommendations_with_IBM

May 21, 2020

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

I. Section **??** II. Section **??** III. Section **??** IV. Section **??** V. Section **??** VI. Section **??**

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 [1]: 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']


        df.head()
```

```
Out[1]:    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]:
```
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 [3]:
```
df.describe()
```

Out[3]:
```
         article_id
count  45993.000000
mean     908.846477
std      486.647866
min        0.000000
25%      460.000000
50%     1151.000000
75%     1336.000000
max     1444.000000
```

```
In [4]: df_content.describe()

Out[4]:         article_id
        count  1056.000000
        mean    523.913826
        std     303.480641
        min       0.000000
        25%     260.750000
        50%     523.500000
        75%     786.250000
        max    1050.000000

In [5]: df['article_id']=df['article_id'].astype(str)
        df_content['article_id']=df_content['article_id'].astype(str)

In [6]: user_per_article = df.groupby('email')['article_id'].count()
        user_per_article

Out[6]: email
        0000b6387a0366322d7fbfc6434af145adf7fed1    13
        001055fc0bb67f71e8fa17002342b256a30254cd     4
        00148e4911c7e04eeff8def7bbbdaf1c59c2c621     3
        001a852ecbd6cc12ab77a785efa137b2646505fe     6
        001fc95b90da5c3cb12c501d201a915e4f093290     2
        0042719415c4fca7d30bd2d4e9d17c5fc570de13     2
        00772abe2d0b269b2336fc27f0f4d7cb1d2b65d7     3
        008ba1d5b4ebf54babf516a2d5aa43e184865da5    10
        008ca24b82c41d513b3799d09ae276d37f92ce72     1
        008dfc7a327b5186244caec48e0ab61610a0c660    13
        009af4e0537378bf8e8caf0ad0e2994f954d822e     1
        00bda305223d05f6df5d77de41abd2a0c7d895fe     4
        00c2d5190e8c6b821b0e3848bf56f6e47e428994     3
        00ced21f957bbcee5edf7b107b2bd05628b04774     4
        00d9337ecd5f70fba1c4c7a78e21b3532e0112c4     3
        00e524e4f13137a6fac54f9c71d7769c6507ecde    11
        00f8341cbecd6af00ba8c78b3bb6ec49adf83248     3
        00f946b14100f0605fa25089437ee9486378872c     1
        01041260c97ab9221d923b0a2c525437f148d589     2
        0108ce3220657a9a89a85bdec959b0f2976dd51c     4
        011455e91a24c1fb815a4deac6b6eaf5ad16819e     9
        01198c58d684d79c9026abe355cfb532cb524dc5     1
        011ae4de07ffb332b0f51c155a35c23c80294962    35
        011fcfb582be9534e9a275336f7e7c3717100381    11
        0129dfcdb701b6e1d309934be6393004c6683a2d    15
        01327bbc4fd7bfe8ad62e599453d2876b928e725     3
        01455f0ab0a5a22a93d94ad35f6e78431aa90625     7
        014dedab269f1453c647598c92a3fa37b39eed97     2
        014e4fe6e6c5eb3fe5ca0b16c16fb4599df6375c     1
        01560f88312a91894d254e6406c25df19f0ad5e8    11
```

```
                                                ..
fe5396e3762c36767c9c915f7ed1731691d7e4b4     1
fe5480ff15f0ac51eeb2314a192351f168d7aad7     1
fe56a49b62752708ed2f6e30677c57881f7b78d1    15
fe5885b80e91be887510a0b6dd04e011178d6364     3
fe5f9d7528518e00b0a73c7a3994afc335496961     3
fe66aa534c7824eca663b84b99a437a98a9b026e     2
fe69c72c964a8346dbc7763309c4e07d818d360f     4
fe88d1f683f308b32fb3d7554f007cc55cc48df5     1
fe8c1cb974e39d8ea8c005044e927b3f0de8acd0     3
fe90d98b0287090fe8e653bafba6ed3eff19331e     1
fe9327be39fd457df70e83d3fc8cba9b8b3f95b1     1
feaea388105a4ccc48795b191bbf0c26a23b1356     4
fef0c6be3a2ed226e1fb8a811b0ee68a389f6f3c    13
fef28e45f7217026b2684d1783a2e18b061bdffb     3
fef3bc88def1aa787c99957ded7d5b2c0edc040e     4
ff27ffd93e21154b8a9cf2722f2cc0f75dc39eff     1
ff288722b76eba5209cdbf9158c6dfbf229b9129     1
ff452614b91f4c9bd965150b1a82e7bf18f59334     2
ff4d3e1c359cfbb73bcae07fa1eb62c45da2b161     3
ff55d0c0b2a4f56aae87c2a21afb7070ab34383d     1
ff6e82c763fe2443643e48a03e239eb635f406dc    14
ff7a0f59ba022102ad22981141a7182c4d8273c3     7
ff833869969184d86f870f98405e7988eccc2309     9
ff979e07f9d906a32ba35a9b75fd9585f6306dbc    38
ffaefa3a1bc2d074d9a14c9924d4e67a46c35410     1
ffc6cfa435937ca0df967b44e9178439d04e3537     2
ffc96f8fbb35aac4cb0029332b0fc78e7766bb5d     4
ffe3d0543c9046d35c2ee3724ea9d774dff98a32    32
fff9fc3ec67bd18ed57a34ed1e67410942c4cd81    10
fffb93a166547448a0ff0232558118d59395fecd    13
Name: article_id, Length: 5148, dtype: int64
```

In [7]:
```
user_per_article.describe()
```

Out[7]:
```
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: article_id, dtype: float64
```

In [8]:
```
np.median(user_per_article)
```

Out[8]:
```
3.0
```

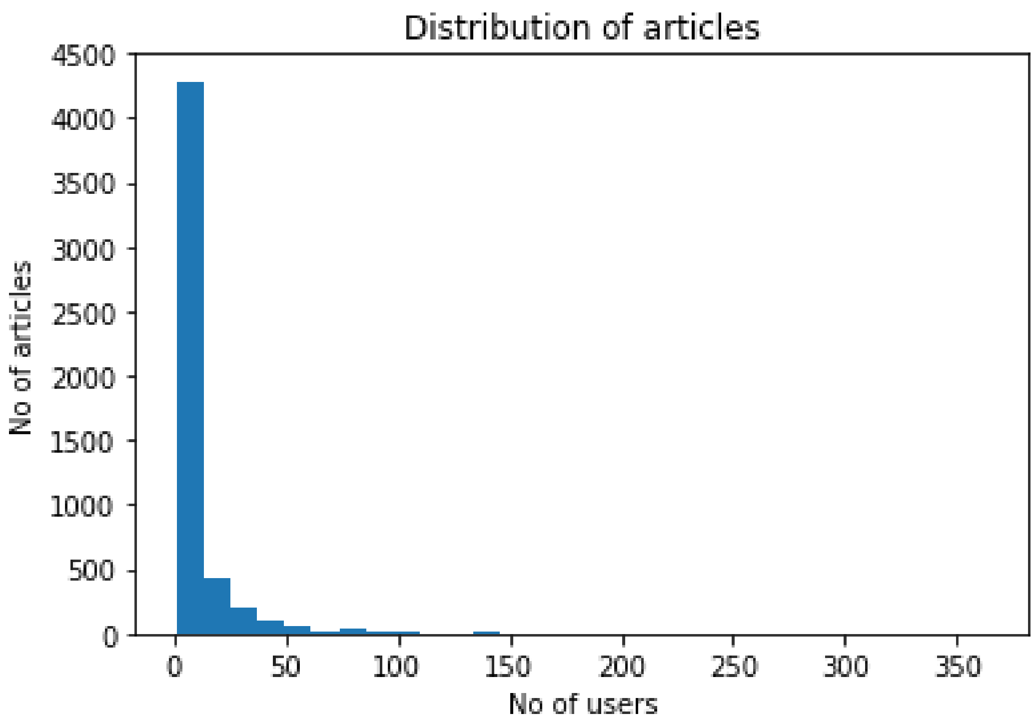
In [9]:
```python
median_val = 3
max_views_by_user = 364
```

In [10]:
```python
max(user_per_article)
```

Out[10]: 364

In [11]:
```python
plt.hist(user_per_article,bins=30)
plt.xlabel('No of users')
plt.ylabel('No of articles')
plt.title('Distribution of articles')
```

Out[11]: Text(0.5,1,'Distribution of articles')



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

In [12]:
```python
df['article_id'].duplicated().sum()# Find and explore duplicate articles
```

Out[12]: 45279

In [13]:
```python
df_content['article_id'].duplicated().sum()
```

Out[13]: 5

In [14]:
```python
df_con=df_content.drop_duplicates(['article_id'],keep='first')
df_=df.drop_duplicates(['article_id'],keep='first')
```

```
In [15]: df_['article_id'].duplicated().sum()
```

```
Out[15]: 0
```

```
In [16]: # Remove any rows that have the same article_id - only keep the first
```

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 [17]: df.nunique()
```

```
Out[17]: article_id     714
         title          714
         email         5148
         dtype: int64
```

```
In [18]: df_con.nunique()
```

```
Out[18]: doc_body           1031
         doc_description    1019
         doc_full_name      1051
         doc_status            1
         article_id         1051
         dtype: int64
```

```
In [19]: df.shape[0]
```

```
Out[19]: 45993
```

```
In [20]: unique_articles = 714# The number of unique articles that have at least one interaction
         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
```

```
In [ ]:
```

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 [21]: df['article_id'].value_counts().head()
```

```
Out[21]: 1429.0    937
         1330.0    927
         1431.0    671
         1427.0    643
         1364.0    627
         Name: article_id, dtype: int64
```

In [22]:
```
most_viewed_article_id = '1429.0'# The most viewed article in the dataset as a string u
max_views = 937 # The most viewed article in the dataset was viewed how many times?
```

In [ ]:

In [23]:
```
## No need to change the code here - this will be helpful for later parts of the notebo
# 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])
    return email_encoded

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

# show header
df.head()
```

Out[23]:

	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 [ ]:

In [24]:
```
## 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!
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

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 [25]: 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

             '''
             article_id_count=df['article_id'].value_counts().index[:n]
             top_articles=[]
             for i in article_id_count:
                 idx=df[df['article_id']==i]['title'].iloc[0]
                 top_articles.append(idx)


             # Your code here

             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

    '''
    top_articles=list(df['article_id'].value_counts().index[:n])


    # Your code here

    return top_articles # Return the top article ids
```

In [26]:
```
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 [27]:
```
# 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 [28]:
```python
# 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
    user_item=df.groupby(['user_id', 'article_id']).agg(lambda x:1).unstack().fillna(0)

    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
```

In [29]:
```python
## 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 matr
assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-article ma
assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by user 1 do
print("You have passed our quick tests!  Please proceed!")
```

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

In [30]:
```python
user_item.loc[1,:].dot(user_item.T).sort_values(ascending=False)
```

Out[30]:
```
user_id
1       36.0
3933    35.0
23      17.0
3782    17.0
203     15.0
4459    15.0
131     14.0
3870    14.0
46      13.0
4201    13.0
5041    12.0
49      12.0
3697    12.0
395     12.0
3910    11.0
322     11.0
```

```
3622    11.0
242     11.0
4642    10.0
290     10.0
2982    10.0
912     10.0
3540    10.0
98      10.0
754     10.0
3764    10.0
256      9.0
52       9.0
268      9.0
40       9.0
        ...
2906     0.0
2909     0.0
2954     0.0
2910     0.0
2952     0.0
2951     0.0
2950     0.0
2947     0.0
2945     0.0
2944     0.0
2943     0.0
2942     0.0
2939     0.0
2938     0.0
2937     0.0
2936     0.0
2933     0.0
2931     0.0
2930     0.0
2929     0.0
2928     0.0
2927     0.0
2923     0.0
2922     0.0
2921     0.0
2920     0.0
2918     0.0
2916     0.0
2911     0.0
2575     0.0
Name: 1, Length: 5149, dtype: float64
```

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 [31]: 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.loc[user_id,:].dot(user_item.T)
             similarity=similarity.sort_values(ascending=False)
             most_similar_users=list(similarity.loc[~(similarity.index==user_id)].index)
             # sort by similarity

             # create list of just the ids

             # remove the own user's id

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

```
In [32]: # 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]
```

```
In [33]: list(user_item.loc[1][user_item.loc[1]==1].index)
```

```
Out[33]: [('title', '1052.0'),
          ('title', '109.0'),
```

```
('title', '1170.0'),
('title', '1183.0'),
('title', '1185.0'),
('title', '1232.0'),
('title', '1293.0'),
('title', '1305.0'),
('title', '1363.0'),
('title', '1368.0'),
('title', '1391.0'),
('title', '1400.0'),
('title', '1406.0'),
('title', '1427.0'),
('title', '1429.0'),
('title', '1430.0'),
('title', '1431.0'),
('title', '1436.0'),
('title', '1439.0'),
('title', '151.0'),
('title', '268.0'),
('title', '310.0'),
('title', '329.0'),
('title', '346.0'),
('title', '390.0'),
('title', '43.0'),
('title', '494.0'),
('title', '525.0'),
('title', '585.0'),
('title', '626.0'),
('title', '668.0'),
('title', '732.0'),
('title', '768.0'),
('title', '910.0'),
('title', '968.0'),
('title', '981.0')]
```

In [ ]:

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 [34]: 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.loc[df['article_id'].isin(article_ids)].title.drop_duplicates()

    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 = [str(id) for id in user_item.loc[user_id][user_item.loc[user_id]==1][

    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=[]
    similar_users=find_similar_users(user_id)
    user_articles_ids,user_articles_names=get_user_articles(user_id)
    for art in similar_users:
        some_ids,some_names=get_user_articles(art)
        for ids in some_ids:
            if ids not in user_articles_ids:
                recs.append(ids)
            if(len(recs)>m-1):
                break
        if(len(recs)>m-1):
            break
    if(len(recs)<m):
        new_art=df['article_id']
        for id in new_art:
            if id not in user_articles_ids:
                recs.append(id)
            if(len(recs)>m-1):
                break



    return recs # return your recommendations for this user_id
```

In [ ]:

In [35]:
```
# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

Out[35]:
```
31                  analyze energy consumption in buildings
158           analyze accident reports on amazon emr spark
219      520    using notebooks with pixiedust for fast...
558      1448    i ranked every intro to data science c...
2704                data tidying in data science experience
3075         airbnb data for analytics: vancouver listings
3391          recommender systems: approaches & algorithms
13127           airbnb data for analytics: mallorca reviews
21668    analyze facebook data using ibm watson and wat...
22820    a tensorflow regression model to predict house...
Name: title, dtype: object
```

```
In [36]: # 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!
```

```
In [37]: user_item.index.values
```

```
Out[37]: array([   1,    2,    3, ..., 5147, 5148, 5149])
```

```
In [38]: user_item.loc[1].dot(user_item.loc[1].T)
```

```
Out[38]: 36.0
```

```
In [39]: len(user_item.loc[1].values)
```

```
Out[39]: 714
```

```
In [40]: df_article_views = df.groupby('user_id').count()
         similarity = []
         for user in range(1, user_item.shape[0]+1):
             sim = np.dot(user_item.loc[20], user_item.loc[user])
             similarity.append((user, sim))

             # sort by similarity
         similarity.sort(key=lambda x: x[1], reverse=True)

             # create dataframe
         df_sims = pd.DataFrame()
         df_sims['user_id'] = [x[0] for x in similarity]
         df_sims
```

Out[40]:

	user_id
0	20
1	40
2	113
3	170
4	196
5	204
6	419
7	2164
8	2354
9	2932

10	3169
11	3927
12	4298
13	4883
14	4932
15	5123
16	5138
17	45
18	47
19	55
20	64
21	86
22	91
23	98
24	102
25	107
26	117
27	120
28	122
29	129
...	...
5119	5116
5120	5117
5121	5118
5122	5119
5123	5120
5124	5121
5125	5122
5126	5124
5127	5125
5128	5126
5129	5127
5130	5128
5131	5130
5132	5131
5133	5132
5134	5134
5135	5135
5136	5136
5137	5137
5138	5139
5139	5140
5140	5141
5141	5142
5142	5143
5143	5144
5144	5145
5145	5146

```
5146     5147
5147     5148
5148     5149

[5149 rows x 1 columns]
```

In [41]:
```
df_sims.columns
```

Out[41]:
```
Index(['user_id'], dtype='object')
```

In [ ]:

In [43]:
```
df_.columns = [['neighbor_id', 'similarity', 'num_articles']]
```

In [44]:
```
df_["neighbor_id"]
```

Out[44]:
```
      neighbor_id
0          1430.0
1          1314.0
2          1429.0
3          1338.0
4          1276.0
5          1432.0
7           593.0
9          1185.0
10          993.0
11           14.0
12         1395.0
14         1170.0
15          542.0
16           12.0
19          173.0
22         1320.0
23         1052.0
25         1393.0
28          362.0
29         1364.0
30          194.0
31         1162.0
32         1324.0
33          460.0
37         1431.0
38          189.0
40         1164.0
42         1427.0
43         1332.0
44         1172.0
...           ...
```

```
21977      1167.0
22244       384.0
22248       319.0
22287       363.0
22314       446.0
22365       870.0
22406      1390.0
22457      1235.0
22458      1303.0
22495       675.0
22618       297.0
22655       662.0
22820      1051.0
22890      1421.0
22893      1247.0
22940      1086.0
22976      1371.0
23005      1372.0
23008       145.0
23129       567.0
23540      1135.0
23581       881.0
23584       183.0
23964       655.0
24235      1233.0
24278      1156.0
24616       555.0
24726       708.0
24737       575.0
24827       972.0

[714 rows x 1 columns]
```

In [ ]:

In [ ]:

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 [45]:
```
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

    '''
    user_interactions = df.groupby(["user_id"])["article_id"].count()

    n_obs = user_item.shape[0]

    # colnames
    some_id = [id_ for id_ in range(1, n_obs) if id_ != user_id]
    similarity = []
    num_interactions = []

    # get similarity and num_interactions
    for id_ in some_id:
        similarity.append(np.dot(user_item.loc[user_id],user_item.loc[id_]))
        num_interactions.append(user_interactions.loc[id_])
    # create dataframe
    neighbors_df = pd.DataFrame({"neighbor_id": some_id,
                                 "similarity": similarity,
                                 "num_interactions": num_interactions})

    return neighbors_df







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.

    '''
    recs = []

    neighbors_df = get_top_sorted_users(user_id)

    the_user_articles, the_article_names = get_user_articles(user_id)
    for user in neighbors_df['neighbor_id']:
        article_ids, article_names = get_user_articles(user)
        for id in article_ids:
            if id not in the_user_articles:
                recs.append(id)
            if len(recs) >= m:
                break
        if len(recs) >= m:
                break

    if len(recs) < m:
        for id in [str(id) for id in get_top_article_ids(100)]:
            if id not in the_user_articles:
                recs.append(id)
            if len(recs) >= m:
                    break

    rec_names = get_article_names(recs)

    return recs, rec_names
```

In [46]:
```
# 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:
['1052.0', '109.0', '1170.0', '1183.0', '1185.0', '1232.0', '1293.0', '1305.0', '1363.0', '1368.

The top 10 recommendations for user 20 are the following article names:
9                    classify tumors with machine learning
14                 apache spark lab, part 1: basic concepts
23       access db2 warehouse on cloud and db2 with python
47                 putting a human face on machine learning
48               gosales transactions for naive bayes model
154      finding optimal locations of new store using d...
512                                  tensorflow quick tips
2047          country statistics: life expectancy at birth
19378                             categorize urban density
21974    predict loan applicant behavior with tensorflo...
Name: title, dtype: object
```

```
In [47]: print(get_top_sorted_users(1).head())
```

```
   neighbor_id  similarity  num_interactions
0            2         2.0                 6
1            3         6.0                82
2            4         3.0                45
3            5         0.0                 5
4            6         4.0                19
```

```
In [48]: print(get_top_sorted_users(131).head(10))
```

```
   neighbor_id  similarity  num_interactions
0            1        14.0                47
1            2         3.0                 6
2            3        13.0                82
3            4         9.0                45
4            5         2.0                 5
5            6        12.0                19
6            7         1.0                 4
7            8        17.0                82
8            9         6.0                32
9           10         7.0                22
```

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 [49]: ### 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 [50]: ## 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!
```

```
In [51]: print(get_top_article_ids(10))
```

```
['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0', '1170.0', '1162.0', '1304
```

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 we don't have any knowledge about the user and don't have enough data so we wouldn't be able to use. We can think of using a rank based recomendation or content based filtering. We can use get_top_article_ids to make recommendation as we don't already have information about the user.

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 [52]: 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 =['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0', '
```

```
In [53]: 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 [54]: 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 [55]: # 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 [56]:
```
# Load the matrix here
user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

In [57]:
```
# quick look at the matrix
user_item_matrix.head()
```

Out[57]:
```
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.

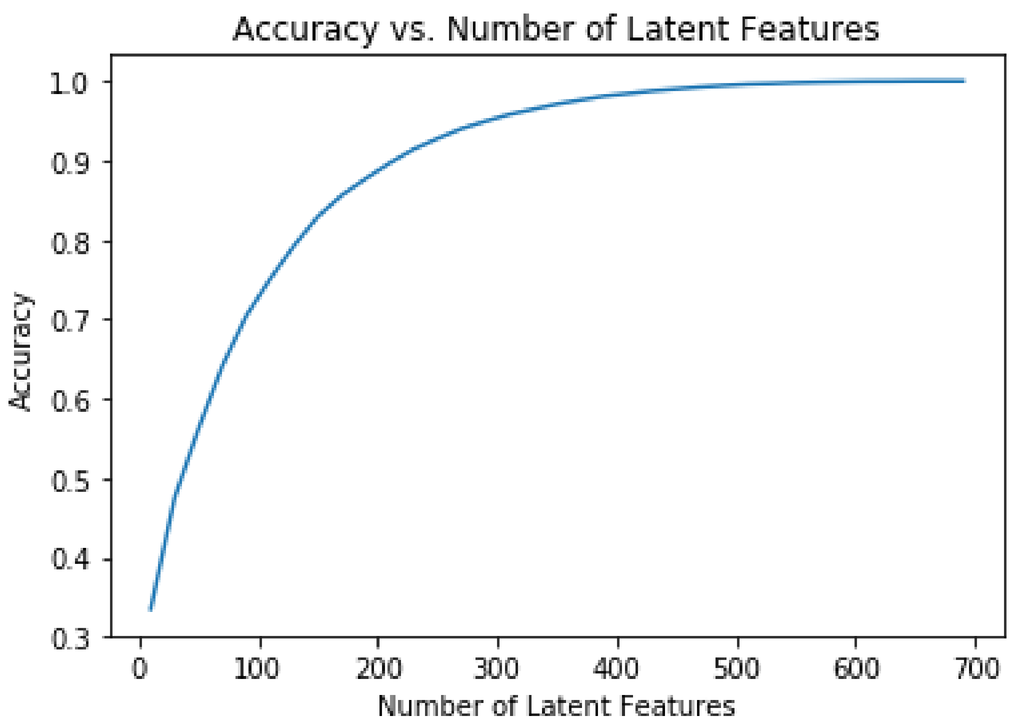
In [58]:
```
# Perform SVD on the User-Item Matrix Here

u, s, vt = np.linalg.svd(user_item_matrix)# use the built in to get the three matrices
```

Provide your response here. In the lesson we had user_item _matrix containing the rating but here the matrix conteinas whether the user has interacted with the article or not.Also we had seen a user_item_matrix containing nan and processing it was not possible with normal SVD so we used FunkSVD there but that is not the case here.

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 [59]:
```
num_latent_feats = np.arange(10,700+10,20)
sum_errs = []
```

```python
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');
```



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 [60]: 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

             '''
             user_item_train=create_user_item_matrix(df_train)
             user_item_test=create_user_item_matrix(df_test)
             # Your code here
             test_idx = list(user_item_test.index.values)
             test_arts=user_item_test['title'].columns.values

             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 [61]: user_item_train.index.isin(test_idx).sum()

Out[61]: 20

In [62]: user_item_train.title.index

Out[62]: Int64Index([   1,    2,    3,    4,    5,    6,    7,    8,    9,   10,
                     ...
                     4478, 4479, 4480, 4481, 4482, 4483, 4484, 4485, 4486, 4487],
                    dtype='int64', name='user_id', length=4487)

In [63]: user_item_train.title.columns.isin(test_arts).sum()
```

```
Out[63]: 574
```

```
In [64]: # 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 [65]: # fit SVD on the user_item_train matrix
         u_train, s_train, vt_train =np.linalg.svd(user_item_train)
         u_train.shape,s_train.shape,vt_train.shape
         # fit svd similar to above then use the cells below
```

```
Out[65]: ((4487, 4487), (714,), (714, 714))
```

```
In [66]: com_idx=user_item_train.index.isin(test_idx)
         com_a=user_item_train.title.columns.isin(test_arts)
         u_test=u_train[com_idx,:]
         vt_test=vt_train[:,com_a]
         s_new, u_new, vt_new = np.diag(s[:k]), u[:, :k], vt[:k, :]
         s_new, u_new, vt_new = np.diag(s_train[:10]), u_train[:, :10], vt_train[:10, :]
         u_test_new, vt_test_new = u_test[:, :10], vt_test[:10, :]

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

```
In [67]: print(u_train.shape,s_train.shape,vt_train.shape)
         user_item_matrix.loc[com_idx, :].shape, user_item_est.shape
```

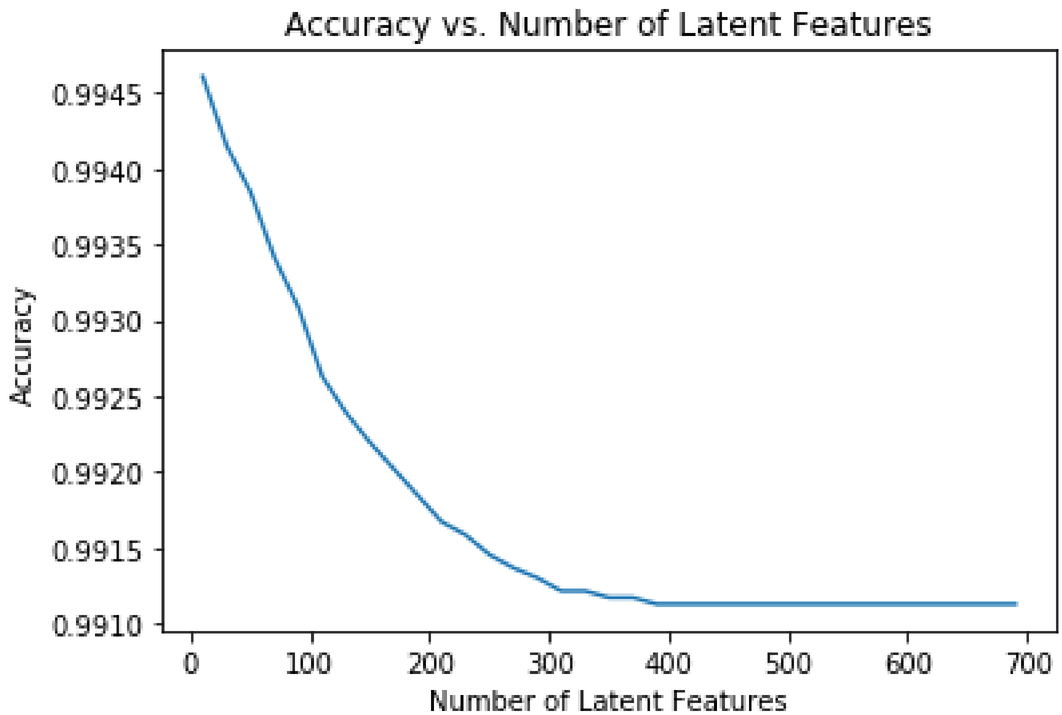
```
(4487, 4487) (714,) (714, 714)
```

```
Out[67]: ((20, 714), (4487, 714))
```

```
In [68]: num_latent_feats = np.arange(10,700+10,20)
         sum_errs = []
         test_sum_errs = []

         for k in num_latent_feats:
             # restructure with k latent features
             s_new, u_new, vt_new = np.diag(s_train[:k]), u_train[:, :k], vt_train[:k, :]
             u_test_new, vt_test_new = u_test[:, :k], vt_test[:k, :]

             # take dot product
             #user_item_est = np.around(np.dot(np.dot(u_new, s_new), vt_new))
             user_test_item_est = np.around(np.dot(np.dot(u_test_new, s_new), vt_test_new))

             # compute error for each prediction to actual value
             #diffs = np.subtract(user_item_matrix, user_item_est)
             #test_diffs = np.subtract(user_item_train.loc[common_idx, common_arts], user_test_i
             test_diffs = np.subtract(user_item_test.loc[user_item_matrix.loc[com_idx, :].index,

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

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?

```
In [69]: plt.plot(num_latent_feats, 1 - np.array(test_sum_errs)/df.shape[0]);
         plt.xlabel('Number of Latent Features');
         plt.ylabel('Accuracy');
         plt.title('Accuracy vs. Number of Latent Features');
```




```
In [133]: num_latent_feats[0 : 29]
```

```
Out[133]: array([ 10,  30,  50,  70,  90, 110, 130, 150, 170, 190, 210, 230, 250,
                 270, 290, 310, 330, 350, 370, 390, 410, 430, 450, 470, 490, 510,
                 530, 550, 570])
```

Your response here. From the above graph we can infer that with an increase in latent features there is a decrease in accuracy which can be attributed to overfitting with class imbalance.Also with the amount of test data less we should not use svd. Since the common users between train and test set are less so other recommendation methods like collaborative filtering or content based recommendation can help improve our recommendation. We could take help from A/B testing to test our recommedation engine and check if its working satisfactorily and increase the engagementof users. We could make two groups of people namely the control and experiment group with one using this recommendation system and other using the other one to check its practical feasibility. THhis would be helpful to realise if this recommendation system is efficient enough to be deployed for all.

```
In [ ]:
```

```
In [ ]:
```

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

Out[70]: 0

In [ ]:
```