# Recommendations_with_IBM

March 17, 2022

## 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') # Import the user item interactions
        df_content = pd.read_csv('data/articles_community.csv') # Import the articles database d
        del df['Unnamed: 0']
        del df_content['Unnamed: 0']

        # Show df to get an idea of the data
        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]:
```
# 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 [3]:
```
df_email = df.set_index('email') # Set index to the email
df_email_count = df_email.groupby('email')['article_id'].count()
# Group articles by email and extract article_id's count: this gives
# the number of articles each user interacted with
df_email_countunique = df_email.groupby('email')['article_id'].unique()
# Here, we define a dataframe as above BUT we include an article in
# the count only once even if a user has interacted multiple times with it

df_email_count.describe()
```

```
Out[3]: 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 [4]: df_email_countunique_len = df_email_countunique.apply(lambda x: len(x))
        # We extract the number of articles each user interacted with and
        # show the statistics below
        df_email_countunique_len.describe()

Out[4]: count    5148.000000
        mean        6.540210
        std         9.990676
        min         1.000000
        25%         1.000000
        50%         3.000000
        75%         7.000000
        max       135.000000
        Name: article_id, dtype: float64

In [5]: # Fill in the median and maximum number of user_article interactios below

        median_val = 3 # 50% of individuals interact with ____ 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 [6]: # Find and explore duplicate articles

        df_content[df_content['article_id'].duplicated() == True]
        # The above shows the duplicate entries

Out[6]:                                                 doc_body  \
        365  Follow Sign in / Sign up Home About Insight Da...
        692  Homepage Follow Sign in / Sign up Homepage * H...
        761  Homepage Follow Sign in Get started Homepage *...
        970  This video shows you how to construct queries ...
        971  Homepage Follow Sign in Get started * Home\r\n...

                                            doc_description  \
        365  During the seven-week Insight Data Engineering...
        692  One of the earliest documented catalogs was co...
        761  Todays world of data science leverages data f...
        970  This video shows you how to construct queries ...
```

```
971  If you are like most data scientists, you are ...
```

| | doc_full_name | doc_status | article_id |
|---|---|---|---|
| 365 | Graph-based machine learning | Live | 50 |
| 692 | How smart catalogs can turn the big data flood... | Live | 221 |
| 761 | Using Apache Spark as a parallel processing fr... | Live | 398 |
| 970 | Use the Primary Index | Live | 577 |
| 971 | Self-service data preparation with IBM Data Re... | Live | 232 |

In [7]:
```
# Remove any rows that have the same article_id - only keep the first


df_content1 = df_content.drop_duplicates(subset =['article_id'])
df_content1.head()
```

Out[7]:

| | doc_body | doc_description | doc_full_name | doc_status | article_id |
|---|---|---|---|---|---|
| 0 | Skip navigation Sign in SearchLoading...\r\n\r... | Detect bad readings in real time using Python ... | Detect Malfunctioning IoT Sensors with Streami... | Live | 0 |
| 1 | No Free Hunch Navigation * kaggle.com\r\n\r\n ... | See the forest, see the trees. Here lies the c... | Communicating data science: A guide to present... | Live | 1 |
| 2 | * Login\r\n * Sign Up\r\n\r\n * Learning Pat... | Heres this weeks news in Data Science and Bi... | This Week in Data Science (April 18, 2017) | Live | 2 |
| 3 | DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA... | Learn how distributed DBs solve the problem of... | DataLayer Conference: Boost the performance of... | Live | 3 |
| 4 | Skip navigation Sign in SearchLoading...\r\n\r... | This video demonstrates the power of IBM DataS... | Analyze NY Restaurant data using Spark in DSX | Live | 4 |

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 [8]:
```
df4 = df.set_index('article_id')
df5 = df4.groupby('article_id')['title']
df4.describe()
```

Out[8]:

| | title |
|---|---|
| count | 45993 |
| unique | 714 |

```
top     use deep learning for image classification
freq                                           937

                                              email
count                                         45976
unique                                         5148
top     2b6c0f514c2f2b04ad3c4583407dccd0810469ee
freq                                            364
```

```
In [9]: 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
```

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 [10]: df_id_art = df4.groupby(['article_id']) # Create a dataframe grouped
         # by article_id and having the index of article_id

         value_counts = df['article_id'].value_counts(dropna=True, sort=True)
         # Create a value_count series with number of times the article is
         # interacted with. Then sort it with highest count values on the top.
         value_counts.head()
```

```
Out[10]: 1429.0    937
         1330.0    927
         1431.0    671
         1427.0    643
         1364.0    627
         Name: article_id, dtype: int64
```

```
In [11]: 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 [12]: ## 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[12]:

| | 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 [13]: ## 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 [14]: 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

            '''
            value_counts = df['article_id'].value_counts(dropna=True, sort=True)
            top_articles_id = list(value_counts.index[0:n])
            # Return articles with highest value counts i.e. interacted with the most
            top_articles = [df[df['article_id'] == art_id].title.iloc[0] for art_id in top_arti

            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 ids

            '''
            # Return article ids with highest value counts i.e. interacted with the most
            value_counts = df['article_id'].value_counts(dropna=True, sort=True)
            top_articles = list(value_counts.index[0:n])
            return top_articles # Return the top article ids
```

In [15]:
```
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, 1304.0]
```

In [16]:
```
# 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 [96]:
```
# 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
    '''
    # Extract only the user_id and article_id columns

    df6 = df[['user_id', 'article_id']]
    # Extract dummies of the article_id variable and concatenate with user_id variable
    df7 = pd.concat([df6.user_id, pd.get_dummies(df6.article_id)], axis=1)

    # If an article is interacted with more than or equal to once by a user, set it to
    user_item = (df7.groupby('user_id').sum() > 0).astype(int)


    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
```

In [18]:
```
## 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!
```

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 [19]:
```
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
    sim = user_item.dot(user_item.iloc[user_id-1,:])
    # sort by similarity
    sim2 = sim.sort_values(ascending = False)
    # create list of just the ids
    most_similar_users = list(sim2.index)
    # remove the own user's id
    most_similar_users.remove(user_id)

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

In [20]:
```
# 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 [21]: 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)
             '''
             # Extract titles given the article_ids
             article_names = [df.title[df.article_id == float(a)].iloc[0] for a in article_ids]
             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
             '''
             # Extract the articles seen by a user
             article_ids = list(str(x) for x in set(df[df.user_id==user_id].article_id))
             article_names = list(str(x) for x in set(df[df.user_id==user_id].title))
             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

            '''
            # Find similar users (by dot product)
            similar_users = find_similar_users(user_id)
            #Find articles already seen by user
            art_ids1, art_nms1 = get_user_articles(user_id)

            # Find other articles based on similar users that our user has not
            # already seen
            rec_list = []
            for user in similar_users:
                art_ids, art_nms = get_user_articles(user)
                rec_list.append(list(set(art_ids) - set(art_ids1)))
            recs2 = [item for sublist in rec_list for item in sublist]
            recs = recs2[:m]
            return recs # return your recommendations for this user_id
```

In [22]:
```
# Check Results
#get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
list(str(x) for x in set(df[df.user_id==20].article_id))
```

Out[22]:
```
['1320.0', '232.0', '844.0']
```

In [23]:
```
# 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 [97]: 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

             '''

             # To compute number of interactions:
             # Extract dummies of the article_id variable and concatenate with user_id variable
             df_new = pd.concat([df.user_id, pd.get_dummies(df.article_id)], axis=1)

             # Sum the number of interactions of a user
             user_item2 = (df_new.groupby('user_id').sum()).sum(axis=1)


             # Find the users with most similarity and create a new data frame
             neighbors_df = pd.DataFrame(find_similar_users(user_id), columns = ['neighbor_id'])

             # Add columns with the similarities and their number of interactions
             neighbors_df['similarity'] = list(user_item.loc[neighbors_df.neighbor_id].dot(user_




             neighbors_df['num_interactions']=list(user_item2.loc[neighbors_df.neighbor_id])

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

             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.

    '''
    # Get the neighbours
    neighbors_df = get_top_sorted_users(user_id)
    # And the articles that our user has already seen
    art_ids1, art_nms1 = get_user_articles(user_id)
    rec_list = []

    # Get recommendations from neighbours that our user hasn't already seen
    for user in neighbors_df.neighbor_id:
        art_ids, art_nms = get_user_articles(user)
        rec_list.append(list(set(art_ids) - set(art_ids1)))
        recs2 = [item for sublist in rec_list for item in sublist]
        recs = recs2[:m]
    rec_names = get_article_names(recs)
    return recs, rec_names
```

In [99]:
```
# 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:
['1400.0', '1276.0', '1035.0', '1172.0', '903.0', '1162.0', '1357.0', '1314.0', '939.0', '1366.0
```

```
The top 10 recommendations for user 20 are the following article names:
['uci ml repository: chronic kidney disease data set', 'deploy your python model as a restful ap
```

```
In [109]: neighbors_df = get_top_sorted_users(131)
          neighbors_df.iloc[0:12]
```

Out[109]:

| | neighbor_id | similarity | num_interactions |
|---|---|---|---|
| 0 | 3870 | 74 | 144 |
| 1 | 3782 | 39 | 363 |
| 2 | 23 | 38 | 364 |
| 3 | 4459 | 33 | 158 |
| 4 | 203 | 33 | 160 |
| 5 | 98 | 29 | 170 |
| 6 | 3764 | 29 | 169 |
| 7 | 3697 | 29 | 145 |
| 8 | 49 | 29 | 147 |
| 9 | 242 | 25 | 148 |
| 10 | 3910 | 25 | 147 |
| 11 | 4932 | 24 | 76 |

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 [34]: ### Tests with a dictionary of results

         user1_most_sim = neighbors_df.iloc[0].neighbor_id # Find the user that is most similar
         user131_10th_sim = neighbors_df.iloc[9].neighbor_id# Find the 10th most similar user to
```

```
In [35]: ## 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.

The above method only works by finding other similar users, so user-bsed collaborative methods will not work. We could use rank based recommendations i.e. the get_top_articles function. For better ways to make recommendations, we could potentially add filters that the user could use to select articles.

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 [36]: 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 = list(str(x) for x in get_top_article_ids(10)) # Your recommendations he

         new_user_recs

Out[36]: ['1429.0',
          '1330.0',
          '1431.0',
          '1427.0',
          '1364.0',
          '1314.0',
          '1293.0',
          '1170.0',
          '1162.0',
          '1304.0']

In [37]: 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 [38]: def make_content_recs(article_id, m):
             '''
```

```
    INPUT:
    article_id - (str with a number) one article id that the user has interacted with
    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:
    For the given article_id, find the users who have interacted with this article. Fin
    most of these users have interacted with.

    '''
    article_id_float = float(article_id)

    # Find all users who have interacted with this article
    users_df = user_item[user_item.columns[user_item.columns == article_id_float]]
    users_to_use = list(users_df.index[users_df[article_id_float] ==1])

    # Find the other articles that they have most interacted with as a group
    articles_rec = user_item.iloc[users_to_use,:].sum().sort_values(ascending= False)
    articles_rec.drop(labels=article_id_float)
    recs = list(str(x) for x in articles_rec[0:m].index)
    rec_names = get_article_names(recs)

    return recs, rec_names
```

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 [39]: # make recommendations for a brand new user
         new_user_recs = list(str(x) for x in get_top_article_ids(10)) # Your recommendations he

         new_user_recs
```

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

user_rec_ids, user_rec_titles = make_content_recs('1427.0', 5)

user_rec_ids
```

```
Out[39]: ['1330.0', '1429.0', '1427.0', '1364.0', '1314.0']
```

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

In [41]: # quick look at the matrix
         user_item_matrix.head()
```

```
Out[41]: 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]
```

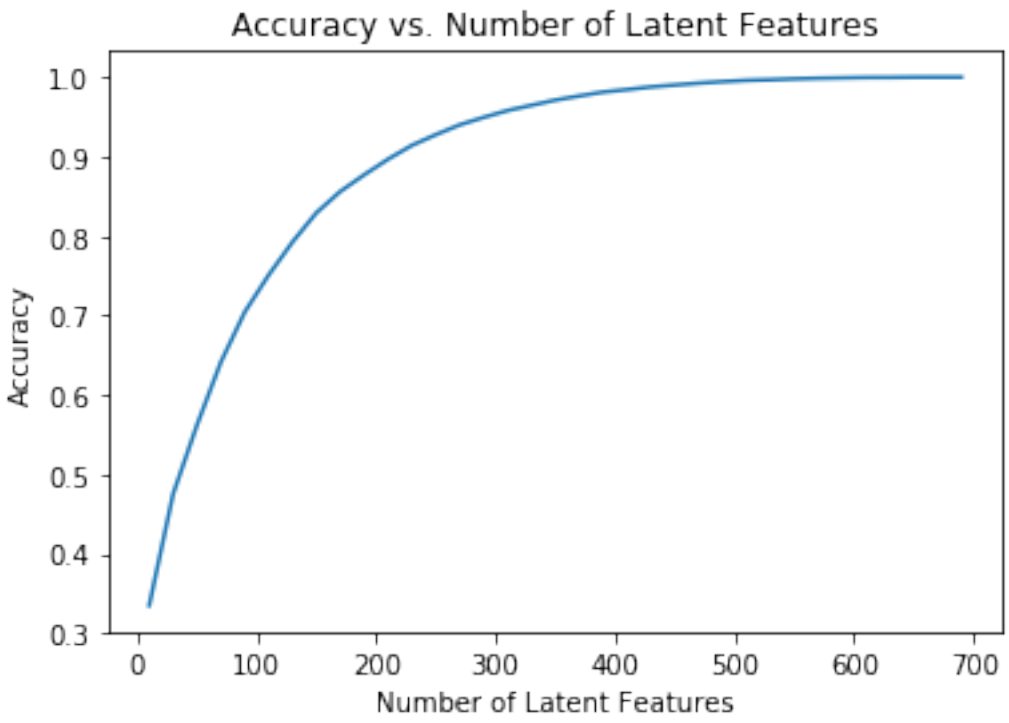
2. In this situation, you can use Singular Value Decomposition from numpy on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

```
In [42]: # 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
```

This matrix has only binary values, so it is different in that sense from the rating matrix used in the lesson. This matrix has nonempty values for every cell, therefore we need not use FunkSVD on it but can do with SVD.

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 [43]: 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');
```




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 [45]: df_train = df.head(40000)
         df_test = df.tail(5993)


         # Create matrices for training and testing separately
         user_item_train = create_user_item_matrix(df_train)
         user_item_test = create_user_item_matrix(df_test)

         # Find users in test that are not in train and articles in test that are not in train
         len(set(user_item_test.index) - set(user_item_train.index))
         len(set(user_item_test.columns) - set(user_item_train.columns))
```

```
#Visualize the test matrix
user_item_test.head()
```

Out[45]:

| user_id | 0.0 | 2.0 | 4.0 | 8.0 | 9.0 | 12.0 | 14.0 | 15.0 | \ |
|---|---|---|---|---|---|---|---|---|---|
| 2917 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3024 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 3093 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3193 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3527 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| user_id | 16.0 | 18.0 | ... | 1432.0 | 1433.0 | 1434.0 | 1435.0 | 1436.0 | \ |
|---|---|---|---|---|---|---|---|---|---|
| | | | ... | | | | | | |
| 2917 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 3024 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 3093 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 1 | |
| 3193 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 3527 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |

| user_id | 1437.0 | 1439.0 | 1440.0 | 1441.0 | 1443.0 |
|---|---|---|---|---|---|
| 2917 | 0 | 0 | 0 | 0 | 0 |
| 3024 | 0 | 0 | 0 | 0 | 0 |
| 3093 | 0 | 0 | 0 | 0 | 0 |
| 3193 | 0 | 0 | 0 | 0 | 0 |
| 3527 | 0 | 0 | 0 | 0 | 0 |

[5 rows x 574 columns]

In [46]:
```
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

    '''
```

```
    # Create train and test dataframes
    user_item_train = create_user_item_matrix(df_train)
    user_item_test = create_user_item_matrix(df_test)

    test_idx = list(user_item_test.index)
    test_arts = list(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 [48]:
```
# 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 articles can we make predictions for in the test set?': b,
    'How many articles in the test set are we not able to make predictions for because
}

t.sol_4_test(sol_4_dict)

# There seems to be some issue with the solution dictionary.
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)

<ipython-input-48-1cf7764adb4b> in <module>()
 13 }
 14
---> 15 t.sol_4_test(sol_4_dict)
 16
 17 # There seems to be some issue with the solution dictionary.


/home/workspace/project_tests.py in sol_4_test(sol_4_dict)
 76     else:
 77         for k, v in sol_4_dict_1.items():
---> 78             if sol_4_dict_1[k] != sol_4_dict[k]:
 79                 print("Sorry it looks like that isn't the right value associated wit
 80
```

```
KeyError: 'How many movies can we make predictions for in the test set?'
```

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.

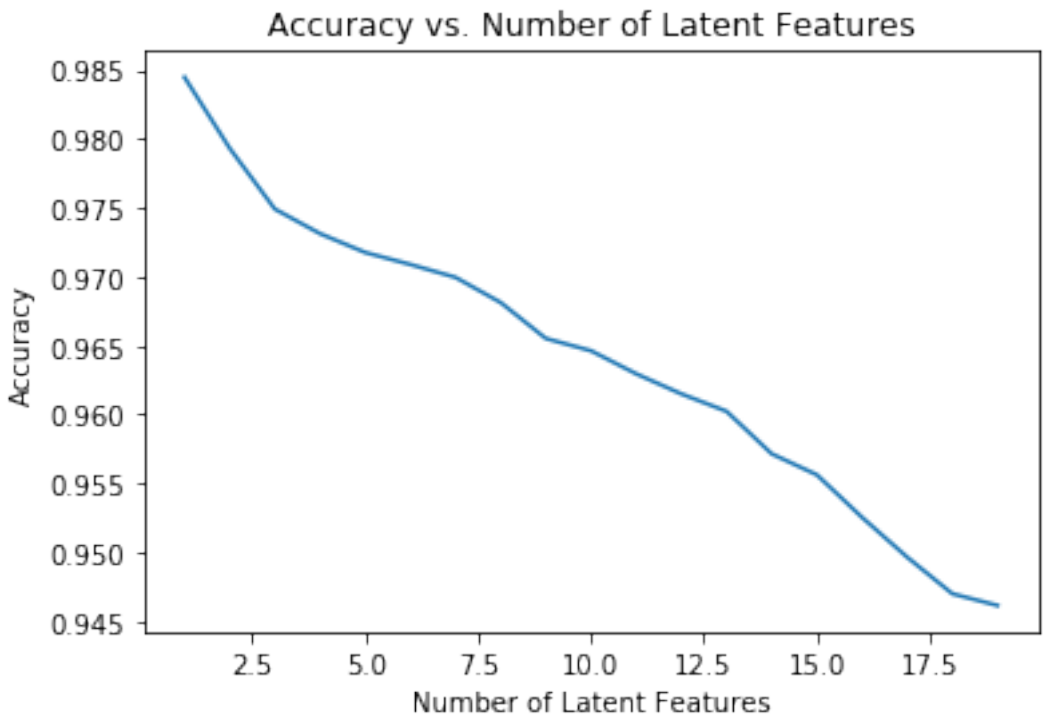
```
In [49]: # fit SVD on the user_item_train matrix
         u_train, s_train, vt_train = np.linalg.svd(user_item_train)# fit svd similar to above t

         # Print all the shapes for understanding what the matrices represent
         print(np.shape(u_train))
         print(np.shape(s_train))
         print(np.shape(vt_train))
         print(np.shape(user_item_train))
```

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

```
In [54]: # Find users that are common in the train and test dataframe
         rows_to_remove = list(set(user_item_test.index) - set(user_item_train.index))
         rows_to_keep = list(set(user_item_test.index) - set(rows_to_remove))
         rows_to_keep

         # Find article_ids that are common in the train and test dataframe
         columns_to_remove = list(set(user_item_test.columns) - set(user_item_train.columns))
         columns_to_keep = list(set(user_item_test.columns) - set(columns_to_remove))

         # Find row indices correponding to common users
         # and column indices correponsing to common article_ids
         users_to_keep = [row - 1 for row in rows_to_keep]
         article_indices = list(user_item_train.columns)
         articles_to_keep = [article_indices.index(i) for i in columns_to_keep]


         # Find the u_train, v_train and s_train corresponding to only common users
         # and articles

         u_train2 = u_train[users_to_keep,:]
         u_train3 = u_train2[:, users_to_keep]
```

```
vt_train2 = vt_train[articles_to_keep,:]
vt_train3 = vt_train2[:, articles_to_keep]
s_train2 = s_train[articles_to_keep]
np.shape(np.around(np.dot(np.dot(u_new, s_new), vt_new)))



# Keep only the common users in the train and test dataframes;
# we keep all the articles as they are all present in the train dataframe
user_item_test2 = user_item_test.loc[user_item_test.index.intersection(rows_to_keep)]
```

In [55]:
```
# Use the reduced u,v,s to make predictions about the test dataframe
# with different number of latent features and compare the results

num_latent_feats = np.arange(1,20,1)
sum_errs = []

for k in num_latent_feats:
    # restructure with k latent features
    s_new, u_new, vt_new = np.diag(s_train2[:k]), u_train3[:, :k], vt_train3[: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_test2, 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');
```




`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?

It seems that the accuracy on the test set is high but that it reduced with the number of latent features. This could be because as the number of latent features increases, the model overfits on the training data to reproduce the training data matrix.

To determine if any of the above recommendation systems are an improvement, we could perhaps perform cross validation on the dataset but splitting it into train-test groups multiple times and then averaging over the prediction accuracies.

To evaluate the performance of the recommendation system, we could do an A/B testing type experiment where we recommend articles to the users based on our predictions and see if they're morely like to follow up on these articles compared to articles that were not predicted.

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

```
Out[56]: 0
```

```
In [ ]:
```