Recommendations_with_IBM

February 23, 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

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 the necessary libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import project_tests as t
        import pickle
        import seaborn as sns
        sns.set()
        %matplotlib inline

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

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

```
Out[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]: # median
        df.groupby('email')['article_id'].count().median()
```

```
Out[3]: 3.0
```

```
In [4]: # max
        df.groupby('email')['article_id'].count().max()
```

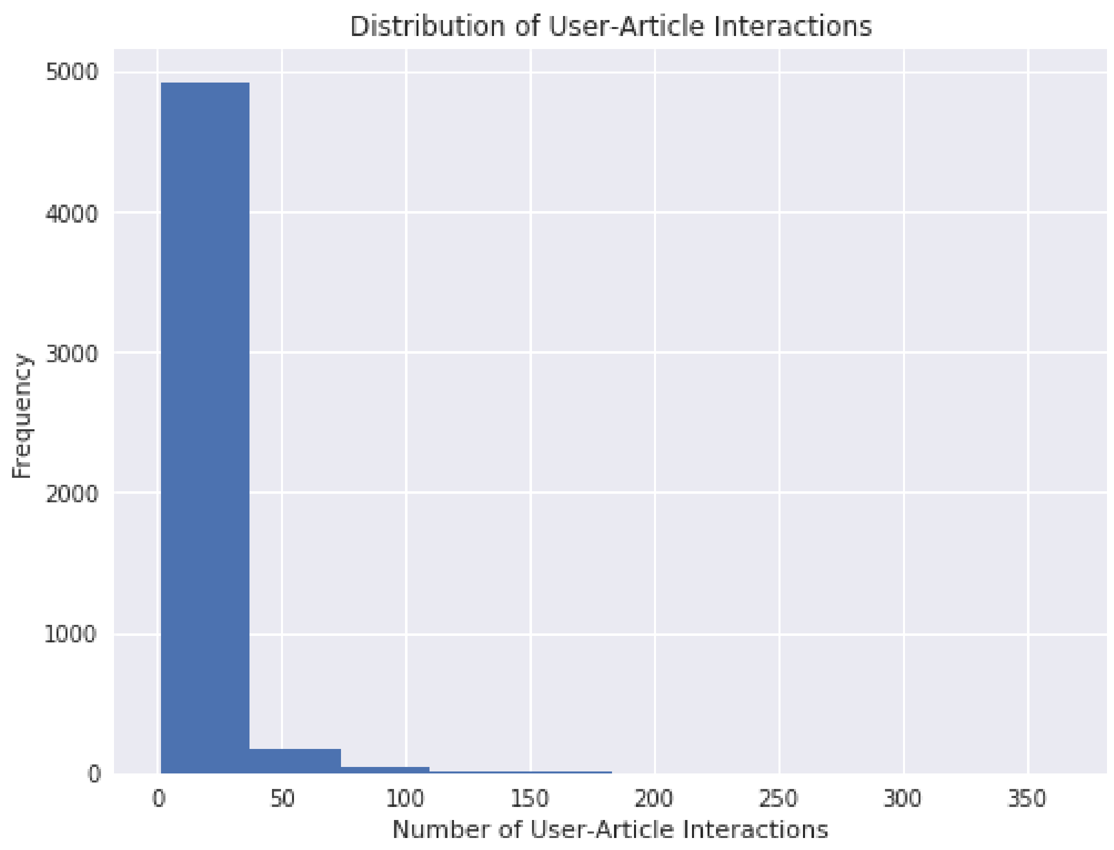
```
Out[4]: 364

In [5]: # median and maximum number of user_article interactions

        median_val = df.groupby('email')['article_id'].count().median() # 50% of individuals int
        max_views_by_user = df.groupby('email')['article_id'].count().max() # The maximum number

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

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]: # summary stats
        user_interacts.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]: # plot graph
        plt.figure(figsize=(8,6))
        user_interacts.plot(kind='hist')
        plt.title('Distribution of User-Article Interactions')
        plt.xlabel('Number of User-Article Interactions');
```



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

```
In [9]: # Find and explore duplicate articles
        df_content.head(3)
```

```
Out[9]:                                             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...

                                             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...

                                               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
```

```
In [10]: # Remove any rows that have the same article_id - only keep the first
         df_content.shape
```

```
Out[10]: (1056, 5)

In [11]: # find duplicate articles
         df_content.article_id.duplicated().sum()

Out[11]: 5

In [12]: ids = df_content['article_id']

In [13]: # explore duplicate articles
         df_content[ids.isin(ids[ids.duplicated()])]

Out[13]:                                               doc_body  \
         50   Follow Sign in / Sign up Home About Insight Da...
         221  * United States\r\n\r\nIBMő * Site map\r\n\r\n...
         232  Homepage Follow Sign in Get started Homepage *...
         365  Follow Sign in / Sign up Home About Insight Da...
         399  Homepage Follow Sign in Get started * Home\r\n...
         578  This video shows you how to construct queries ...
         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  \
         50                       Community Detection at Scale
         221  When used to make sense of huge amounts of con...
         232  If you are like most data scientists, you are ...
         365  During the seven-week Insight Data Engineering...
         399  Todays world of data science leverages data f...
         578  This video shows you how to construct queries ...
         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
         50                       Graph-based machine learning       Live          50
         221  How smart catalogs can turn the big data flood...       Live         221
         232  Self-service data preparation with IBM Data Re...       Live         232
         365                      Graph-based machine learning       Live          50
         399  Using Apache Spark as a parallel processing fr...       Live         398
         578                             Use the Primary Index       Live         577
         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 [14]: # Remove any rows that have the same article_id - only keep the first
         df_content.drop_duplicates(subset=['article_id'], keep='first', inplace=True)
```

```
In [15]: # check this works by pulling out an example
         df_content.iloc[971]

Out[15]: doc_body           Cloudant allows custom Javascript to be run se...
         doc_description    Cloudant allows custom Javascript to be run se...
         doc_full_name                Defensive coding in Map/Index functions
         doc_status                                                      Live
         article_id                                                       971
         Name: 976, dtype: object
```

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 [16]: # The number of unique users
         df.email.nunique()

Out[16]: 5148

In [17]: unique_articles = df.article_id.nunique() # The number of unique articles that have at
         total_articles =  df_content.article_id.nunique() # The number of unique articles on th
         unique_users = df.email.nunique() # The number of unique users
         user_article_interactions = df.shape[0] # 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 [18]: # most viewed article_id
         df.article_id.value_counts().head(2)

Out[18]: 1429.0    937
         1330.0    927
         Name: article_id, dtype: int64

In [19]: most_viewed_article_id = str(df.article_id.value_counts().index[0]) # The most viewed a
         max_views = df.article_id.value_counts().iloc[0] # The most viewed article in the datas

In [20]: ## 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[20]:    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 [21]: ## 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 [22]: 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

             '''
             top_articles = df['title'].value_counts().index.tolist()[:n]
             top_articles = [str(i) for i in top_articles]

             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

             '''
             top_articles = df['article_id'].value_counts().index.tolist()[:n]
             top_articles = [str(i) for i in top_articles]

             return top_articles # Return the top article ids
```

```
In [23]: 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 [24]: # 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 [25]: # 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
             '''
             df_count = df.groupby(['user_id', 'article_id']).count().reset_index() # create a n
             user_item = df_count.pivot_table(values='title', index='user_id', columns='article_
             user_item.replace(np.nan, 0, inplace=True) # replace nulls with 0s
             user_item=user_item.applymap(lambda x: 1 if x > 0 else x) # entries should be a 1 o

             return user_item # return the user_item matrix

         user_item = create_user_item_matrix(df)
```

```
In [26]: ## 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 [27]: 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
             dot_prod_users = user_item.dot(np.transpose(user_item))

             # sort by similarity
             sim_users = dot_prod_users[user_id].sort_values(ascending = False)

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

             # 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 [28]: # 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 [29]: 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)
             '''
             article_names = []

             for idx in article_ids:
                 article_names.append(df[df['article_id']==float(idx)].max()['title'])

             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

             Description:
             Provides a list of the article_ids and article titles that have been seen by a user
             '''
             article_ids = user_item.loc[user_id][user_item.loc[user_id] == 1].index.astype('str

             article_names = []

             for idx in article_ids:
                 article_names.append(df[df['article_id']==float(idx)].max()['title']) # need to

             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

    '''
    recs = np.array([]) # recommendations to be made

    user_articles_seen = get_user_articles(user_id)[0] #seen by our user
    closest_users = find_similar_users(user_id) # users closest to our user

    for others in closest_users:

        others_articles_seen = get_user_articles(others)[0] # articles seen by others l
        new_recs = np.setdiff1d(others_articles_seen, user_articles_seen, assume_unique
        recs = np.unique(np.concatenate([new_recs, recs], axis = 0)) # concate arrays a

        if len(recs) > m-1:
            break

    recs = recs[:m]
    recs.tolist()

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

```
In [30]: # Check Results
         get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

```
Out[30]: ['recommender systems: approaches & algorithms',
          '1448    i ranked every intro to data science course on...\nName: title, dtype: object
          'data tidying in data science experience',
          'a tensorflow regression model to predict house values',
          '520    using notebooks with pixiedust for fast, flexi...\nName: title, dtype: object'
          'airbnb data for analytics: mallorca reviews',
          'airbnb data for analytics: vancouver listings',
          'analyze facebook data using ibm watson and watson studio',
          'analyze accident reports on amazon emr spark',
          'analyze energy consumption in buildings']
```

```
In [31]: # 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 [32]: 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

             '''
             # create neighbors dataframe with empty columns
             neighbors_df = pd.DataFrame(columns=['neighbor_id', 'similarity'])
             # set neighbor_id column equal to user_item index starting from 1
             neighbors_df['neighbor_id'] = user_item.index-1
             # make similarity column equal to most similar using dot product
             dot_prod_users = user_item.dot(np.transpose(user_item))
             neighbors_df['similarity'] = dot_prod_users[user_id]
             # create new df based on number of interactions of users
```

```
        interacts_df = df.user_id.value_counts().rename_axis('neighbor_id').reset_index(nam
        # merge dataframes which creates number of interactions column from interacts_df
        neighbors_df = pd.merge(neighbors_df, interacts_df, on='neighbor_id', how='outer')
        # sortvalues on similarity and then number of interactions
        neighbors_df = neighbors_df.sort_values(by=['similarity', 'num_interactions'], asce
        # reset index
        neighbors_df = neighbors_df.reset_index(drop=True)
        # drop row with the user_id as itself will be most similar
        neighbors_df = neighbors_df[neighbors_df.neighbor_id != user_id]

        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.

        '''
        recs = np.array([]) # recommendations to be made

        user_articles_ids_seen, user_articles_names_seen = get_user_articles(user_id, user_
        closest_neighs = get_top_sorted_users(user_id, df, user_item).neighbor_id.tolist()

        for neighs in closest_neighs:

            neigh_articles_ids_seen, neigh_articles_names_seen = get_user_articles(neighs,
            new_recs = np.setdiff1d(neigh_articles_ids_seen, user_articles_ids_seen, assume
            recs = np.unique(np.concatenate([new_recs, recs], axis = 0)) # concate arrays a

            if len(recs) > m-1:
                break

            recs = recs[:m]
            recs = recs.tolist() # convert to a list

            rec_names = get_article_names(recs, df=df)

            return recs, rec_names
```

```
In [33]: # 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:
['1024.0', '1085.0', '109.0', '1150.0', '1151.0', '1152.0', '1153.0', '1154.0', '1157.0', '1160.

The top 10 recommendations for user 20 are the following article names:
['using deep learning to reconstruct high-resolution audio', 'airbnb data for analytics: chicago
```

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 = get_top_sorted_users(1).iloc[0].neighbor_id # Find the user that is mo
         user131_10th_sim = get_top_sorted_users(131).iloc[9].neighbor_id# Find the 10th most si
```

```
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.

Provide your response here. If given a new user, it would make sense to use Rank Based Recommendations and the get_top_articles function to make recommendations. We would just recommend the most popular articles since we do not have any information about the user or their interactions so cannot tell which other users they are most similar to. Once we have more information about the user we could a blended approach of 3 types of recommendation techniques; Rank, Content, and Collaborative.

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 = get_top_article_ids(10, df)
```

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

            OUTPUT:

            '''
```

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

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

Write an explanation of your content based recommendation system here.

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

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

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


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

1.1.8 Part V: Matrix Factorization

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

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

```
In [38]: # Load the matrix here
         user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

```
In [39]: # quick look at the matrix
         user_item_matrix.head()
```

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

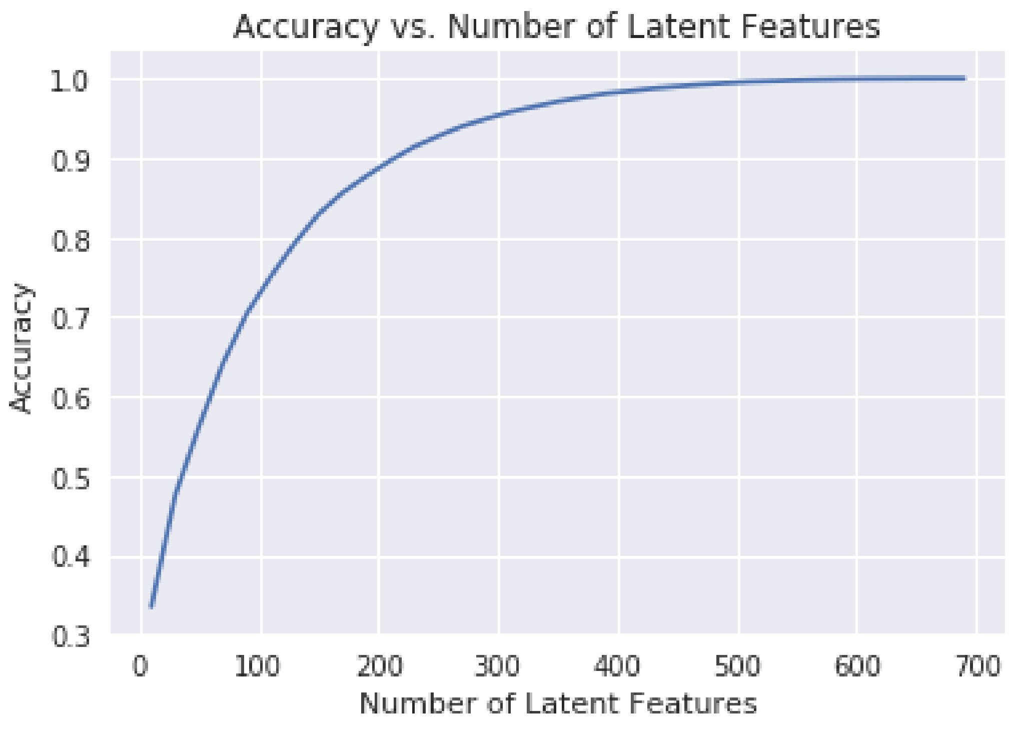
2. In this situation, you can use Singular Value Decomposition from numpy on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

In [40]:
```
# 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. We have no missing values in this matrix therefore we can perform SVD. In the classroom, our matrix had missing values which meant that we had to use FunkSVD

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 [41]:
```
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 [42]: 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)

    test_idx = user_item_test.index
    test_arts = 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 [45]: test_idx # 682 users in test set
```

```
Out[45]: Int64Index([2917, 3024, 3093, 3193, 3527, 3532, 3684, 3740, 3777, 3801,
                     ...
                     5140, 5141, 5142, 5143, 5144, 5145, 5146, 5147, 5148, 5149],
                    dtype='int64', name='user_id', length=682)
```

```
In [46]: train_idx = user_item_train.index # 4487 users in training set
         train_idx
```

```
Out[46]: 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 [47]: test_idx.difference(train_idx) # of 682 users in test set, only 20 of them are in train
```

```
Out[47]: Int64Index([4488, 4489, 4490, 4491, 4492, 4493, 4494, 4495, 4496, 4497,
                     ...
                     5140, 5141, 5142, 5143, 5144, 5145, 5146, 5147, 5148, 5149],
                    dtype='int64', name='user_id', length=662)
```

```
In [48]: test_arts #574 movies in test set
```

```
Out[48]: Float64Index([   0.0,    2.0,    4.0,    8.0,    9.0,   12.0,   14.0,   15.0,
                         16.0,   18.0,
                       ...
                       1432.0, 1433.0, 1434.0, 1435.0, 1436.0, 1437.0, 1439.0, 1440.0,
                       1441.0, 1443.0],
                      dtype='float64', name='article_id', length=574)
```

```
In [49]: train_arts = user_item_train.columns #714 movies in train set
         train_arts
```

```
Out[49]: Float64Index([   0.0,    2.0,    4.0,    8.0,    9.0,   12.0,   14.0,   15.0,
                         16.0,   18.0,
                       ...
                       1434.0, 1435.0, 1436.0, 1437.0, 1439.0, 1440.0, 1441.0, 1442.0,
                       1443.0, 1444.0],
                      dtype='float64', name='article_id', length=714)
```

```
In [50]: test_arts.difference(train_arts) # all articles in test set are in training set too
```

```
Out[50]: Float64Index([], dtype='float64', name='article_id')
```

```
In [51]: # 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 [52]: # 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
```

```
In [53]: # Use these cells to see how well you can use the training
         # decomposition to predict on test data
         s_train.shape, u_train.shape, vt_train.shape
```

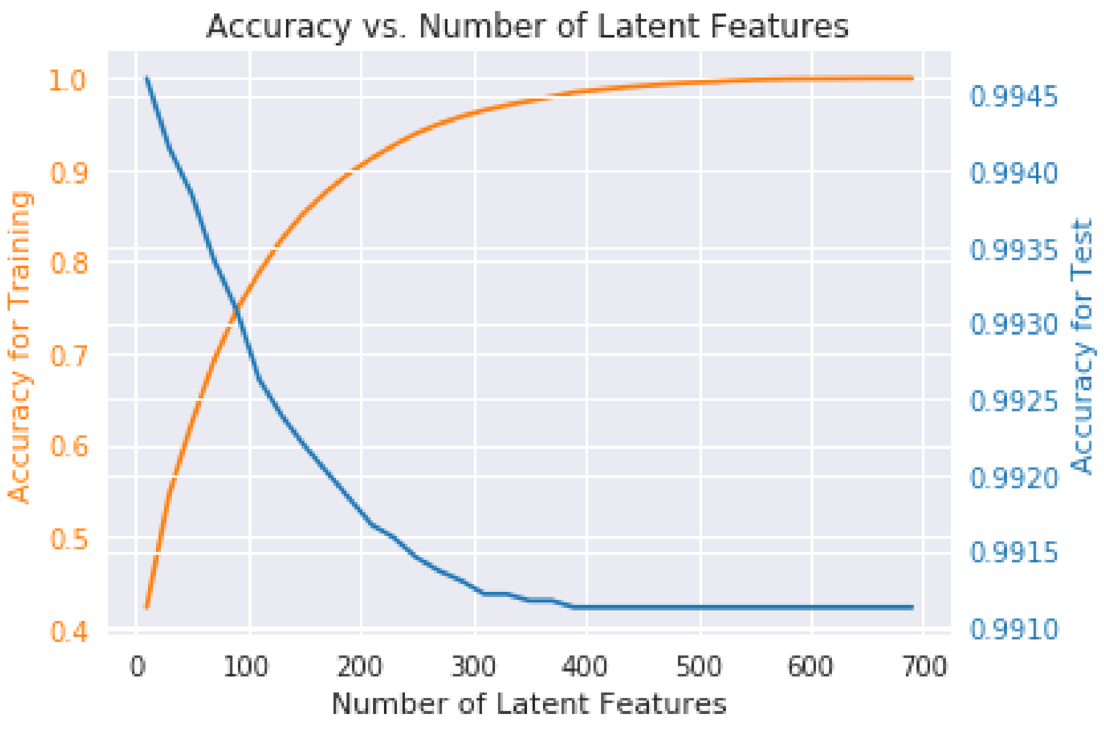
```
Out[53]: ((714,), (4487, 4487), (714, 714))
```

```
In [54]: num_latent_feats = np.arange(10,700+10,20)
         sum_errs_train = []
```

```
        sum_errs_test = []


        #Decomposition
        row_idx = user_item_train.index.isin(test_idx)
        col_idx = user_item_train.columns.isin(test_arts)

        u_test = u_train[row_idx, :]
        vt_test = vt_train[:, col_idx]

        # test users that we can predict for
        users_can_predict = np.intersect1d(list(user_item_train.index),list(user_item_test.inde

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

            # take dot product
            user_item_train_preds = np.around(np.dot(np.dot(u_train_new, s_train_new), vt_train
            user_item_test_preds = np.around(np.dot(np.dot(u_test_new, s_train_new), vt_test_ne

            # compute error for each prediction to actual value
            diffs_train = np.subtract(user_item_train, user_item_train_preds)
            diffs_test = np.subtract(user_item_test.loc[users_can_predict,:], user_item_test_pr

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

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

```
In [55]: # plotting the training and test accuracies
         fig, ax1 = plt.subplots()

         color = 'tab:orange'
         ax1.set_xlabel('Number of Latent Features')
         ax1.set_ylabel('Accuracy for Training', color=color)
         ax1.plot(num_latent_feats, 1 - np.array(sum_errs_train)/df.shape[0], color=color)
         ax1.tick_params(axis='y', labelcolor=color)
         ax1.set_title('Accuracy vs. Number of Latent Features')

         ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

         color = 'tab:blue'
         ax2.set_ylabel('Accuracy for Test', color=color)  # we already handled the x-label with
         ax2.plot(num_latent_feats, 1 - np.array(sum_errs_test)/df.shape[0], color=color)
```

```
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```



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

Your response here. From above, we can see that the accuracy for the training data increases with an increase in the number of latent features, however the opposite is true for the accuracy of the test data. This is most likely due to overfitting of the data with the increase in latent features, therefore the number of latent features should be kept relatively low. It is important to note that using SVD here we can only actually make recommendations for the 20 users in both the training and test dataset, and we have a very sparse matrix which is likely why the test data accuracy is so high at >99%. It would be interesting to look at results if we had more users that appeared in both the test and training data. We can see that at approximately 80 features there is a cross over point when the accuracy for test data begins to drop, therefore this would be a good number of latent features to include, since beyond that our accuracy for training increases but testing decreases. To test how well our recommendation engine works in practice, we could conduct an A/B test for new users to help solve the cold start problem. An example would be to recommend articles to one group using our recommendation engine and then to recommend just the most popular articles to the other group of users. We would then compare the click through rates to effectively measure if our recommendation engine leads to an increase in clicks. If we saw a significant rise in clicks by using our recommendation engine then we could conclude this works well and should be deployed.

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 [ ]:
```