Recommendations_with_IBM

March 4, 2021

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 pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import project_tests as t
import pickle

%matplotlib inline

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

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

Out[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]:
```python
# 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
```

In [3]:
```python
df.count()
```

Out[3]:
```
article_id    45993
title         45993
email         45976
dtype: int64
```

In [4]:
```python
df['email'].nunique()
```

Out[4]:
```
5148
```

1.1.1 Part I : Exploratory Data Analysis

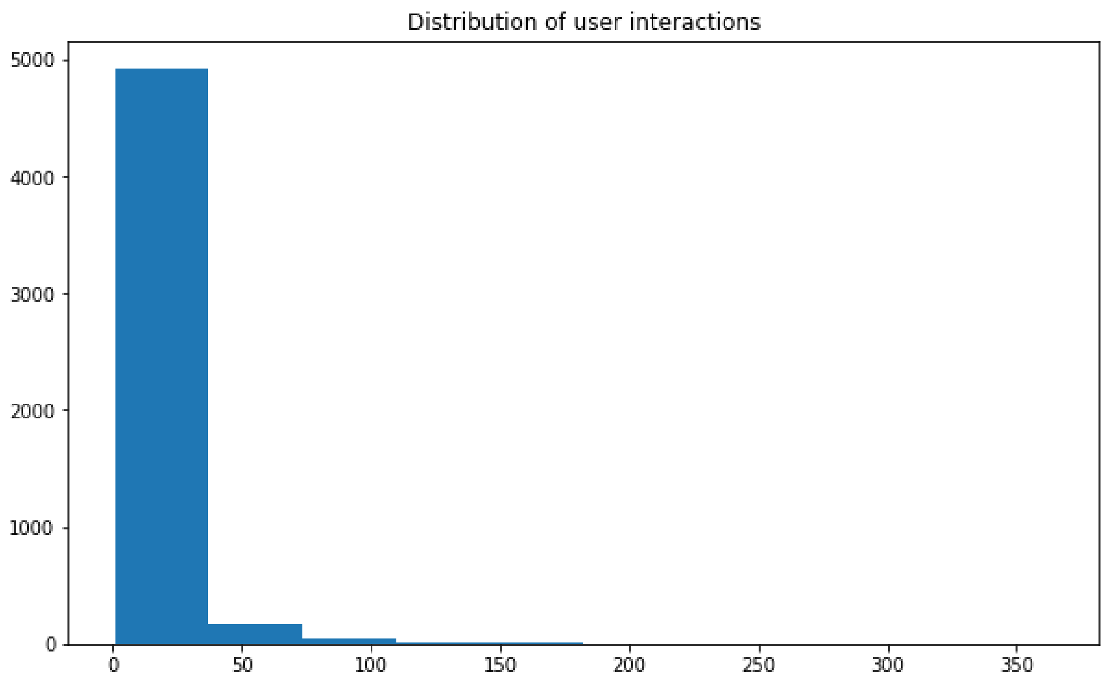
Use the dictionary and cells below to provide some insight into the descriptive statistics of the data.

`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 [5]:
```
users_grouped = df.groupby('email').count()['article_id']
users_grouped.describe()
```

Out[5]:
```
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 [6]:
```
fig, ax = plt.subplots(figsize=(10,6))
ax.hist(users_grouped)
ax.set_title('Distribution of user interactions')
plt.show();
```



In [7]:
```
# 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 [8]:
```
# Find and explore duplicate articles
df_content.duplicated('article_id').sum()
```

```
Out[8]: 5
```

```
In [9]: # Remove any rows that have the same article_id - only keep the first
        df_content.drop_duplicates(subset=['article_id'], keep='first',inplace = True)
        df_content.duplicated('article_id').sum()
```

```
Out[9]: 0
```

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 [10]: df['article_id'].nunique()
```

```
Out[10]: 714
```

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

```
Out[11]: 1051
```

```
In [12]: df['email'].nunique()
```

```
Out[12]: 5148
```

```
In [13]: df.shape[0]
```

```
Out[13]: 45993
```

```
In [14]: unique_articles = 714 # The number of unique articles that have at least one interactio
         total_articles = 1051 # The number of unique articles on the IBM platform
         unique_users = 5148 # The number of unique users
         user_article_interactions = 45993# The number of user-article interactions
```

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

```
In [15]: df.groupby(['article_id'])['email'].count().sort_values(ascending = False).head(5)
```

```
Out[15]: article_id
         1429.0    937
         1330.0    927
         1431.0    671
         1427.0    643
         1364.0    627
         Name: email, dtype: int64
```

In [16]:
```
most_viewed_article_id = '1429.0'# The most viewed article in the dataset as a string w
max_views = 937 # The most viewed article in the dataset was viewed how many times?
```

In [17]:
```
## 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[17]:

	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 [18]:
```
## If you stored all your results in the variable names above,
## you shouldn't need to change anything in this cell

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

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

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

```
In [19]: article_rank = df.groupby(['title'])['user_id'].count().sort_values(ascending = False).

         list(article_rank['title'][:20])
```

```
Out[19]: ['use deep learning for image classification',
          'insights from new york car accident reports',
          'visualize car data with brunel',
          'use xgboost, scikit-learn & ibm watson machine learning apis',
          'predicting churn with the spss random tree algorithm',
          'healthcare python streaming application demo',
          'finding optimal locations of new store using decision optimization',
          'apache spark lab, part 1: basic concepts',
          'analyze energy consumption in buildings',
          'gosales transactions for logistic regression model',
          'welcome to pixiedust',
          'customer demographics and sales',
          'total population by country',
          'deep learning with tensorflow course by big data university',
          'model bike sharing data with spss',
          'the nurse assignment problem',
          'classify tumors with machine learning',
          'analyze accident reports on amazon emr spark',
          'movie recommender system with spark machine learning',
          'putting a human face on machine learning']
```

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

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

             '''
             # Your code here
             # Pull the number of interactions of users for each article
             ranked_articles = df.groupby(['title'])['user_id'].count().sort_values(ascending =
```

```
        top_articles = list(ranked_articles['title'][:n])

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

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

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

        '''
        # Your code here
        ranked_articles = df.groupby(['article_id'])['user_id'].count().sort_values(ascendi

        top_articles = list(ranked_articles['article_id'][:n])
        return top_articles # Return the top article ids
```

In [21]:
```
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 [22]:
```
# 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 [23]: # create the user-article matrix with 1's and 0's

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

             OUTPUT:
             user_item - user item matrix

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

             return user_item # return the user_item matrix

         user_item = create_user_item_matrix(df)
```

```
In [24]: ## 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 [25]: def find_similar_users(user_id, user_item=user_item):
             '''
```

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

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

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

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

            # sort by similarity
            similarity = similarity.sort_values(ascending=False)

            # remove the own user's id
            similarity.drop(user_id, inplace=True)
            # create list of just the ids
            most_similar_users = list(similarity.index)

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

In [26]:
```
# 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 [27]:
```
article_ids = user_item.loc[1]
article_ids
```

Out[27]:
```
article_id
0.0       0
2.0       0
4.0       0
```

```
8.0       0
9.0       0
12.0      0
14.0      0
15.0      0
16.0      0
18.0      0
20.0      0
25.0      0
26.0      0
28.0      0
29.0      0
30.0      0
32.0      0
33.0      0
34.0      0
36.0      0
39.0      0
40.0      0
43.0      1
48.0      0
50.0      0
51.0      0
53.0      0
54.0      0
57.0      0
58.0      0
         ..
1412.0    0
1414.0    0
1415.0    0
1416.0    0
1418.0    0
1419.0    0
1420.0    0
1421.0    0
1422.0    0
1423.0    0
1424.0    0
1425.0    0
1426.0    0
1427.0    1
1428.0    0
1429.0    1
1430.0    1
1431.0    1
1432.0    0
1433.0    0
```

```
1434.0    0
1435.0    0
1436.0    1
1437.0    0
1439.0    1
1440.0    0
1441.0    0
1442.0    0
1443.0    0
1444.0    0
Name: 1, Length: 714, dtype: int64
```

```
In [28]: def get_article_names(article_ids, df=df):
             '''
             INPUT:
             article_ids - (list) a list of article ids
             df - (pandas dataframe) df as defined at the top of the notebook

             OUTPUT:
             article_names - (list) a list of article names associated with the list of article
                             (this is identified by the title column)
             '''
             # Your code here
             article_names = list(set(df[df['article_id'].isin(article_ids)]['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
                             (this is identified by the doc_full_name column in df_content)

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

             article_names = get_article_names(article_ids)
```

```
        return article_ids, article_names # return the ids and names


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

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

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

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

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

        '''
        # Your code here
        recs = []
        # find similar users
        sim_users = find_similar_users(user_id)
        # get articles already seen by user
        seen_articles, _ = get_user_articles(user_id)
        # get articles for similar users
        for user in sim_users:
            if(len(recs)<=m):
                ids, names = get_user_articles(user)
                #find ids that have not been read by user yet
                user_recs = list(set(ids) - (set(seen_articles) & set(ids)))
                # append each iterable to recs list
                recs.extend(user_recs)
            #when we have found enough articles
            else:
                break


        return recs[:m] # return your recommendations for this user_id
```

In [29]:
```
# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

Out[29]:
```
['build a python app on the streaming analytics service',
```

```
'declarative machine learning',
 'markdown for jupyter notebooks cheatsheet',
 'deep learning with data science experience',
 'maximize oil company profits',
 'visualising data the node.js way',
 'shaping data with ibm data refinery',
 'got zip code data? prep it for analytics.  ibm watson data lab  medium',
 'dsx: hybrid mode',
 'brunel 2.0 preview']
```

```
In [30]: # 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 [31]: def get_top_sorted_users(user_id, df=df, user_item=user_item):
             '''
             INPUT:
             user_id - (int)
             df - (pandas dataframe) df as defined at the top of the notebook
             user_item - (pandas dataframe) matrix of users by articles:
                     1's when a user has interacted with an article, 0 otherwise


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

             Other Details - sort the neighbors_df by the similarity and then by number of inter
```

```
                    highest of each is higher in the dataframe

    '''
    # Your code here
    #obtain similar users
    neighbors_df = pd.DataFrame(columns=['neighbor_id', 'similarity', 'num_interactions
    for user in user_item.index:
        if user == user_id:
            continue
        neighbors_df.loc[user] = [user, np.dot(user_item.loc[user_id, :], user_item.loc
                                  df[df['user_id']==user]['article_id'].count()]

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

    return neighbors_df # Return the dataframe specified in the doc_string


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

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

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

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

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

    '''
    recs = []

    neighbors_df = get_top_sorted_users(user_id)

    the_user_articles, the_article_names = get_user_articles(user_id)
    for user in neighbors_df['neighbor_id']:
        article_ids, article_names = get_user_articles(user)
        for id in article_ids:
```

```
                if id not in the_user_articles:
                    recs.append(id)
                if len(recs) >= m:
                    break
            if len(recs) >= m:
                    break

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

        rec_names = get_article_names(recs)

        return recs, rec_names
```

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

```
The top 10 recommendations for user 20 are the following article ids:
['12.0', '109.0', '125.0', '142.0', '164.0', '205.0', '302.0', '336.0', '362.0', '465.0']

The top 10 recommendations for user 20 are the following article names:
['accelerate your workflow with dsx', 'statistics for hackers', "a beginner's guide to variation
```

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 [33]:
```
get_top_sorted_users(1).head(1)
```

Out[33]:
```
      neighbor_id similarity num_interactions
3933         3933         35               45
```

In [34]:
```
get_top_sorted_users(131).iloc[9]
```

Out[34]:
```
neighbor_id          242
similarity            25
num_interactions     148
Name: 242, dtype: object
```

```
In [35]: ### Tests with a dictionary of results

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

In [36]: ## 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. I will use the 'get_top_article_ids' function since we don't have information about articles views of the new user yet.

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 [37]: 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 = [str(id) for id in get_top_article_ids(10)]# Your recommendations here

In [38]: new_user_recs

Out[38]: ['1429.0',
          '1330.0',
          '1431.0',
          '1427.0',
          '1364.0',
          '1314.0',
          '1293.0',
          '1170.0',
          '1162.0',
          '1304.0']

In [39]: 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 [40]: 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 [41]: # 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 [42]: # Load the matrix here
         user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

```
In [43]: # quick look at the matrix
         user_item_matrix.head()
```

```
Out[43]: 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]
```

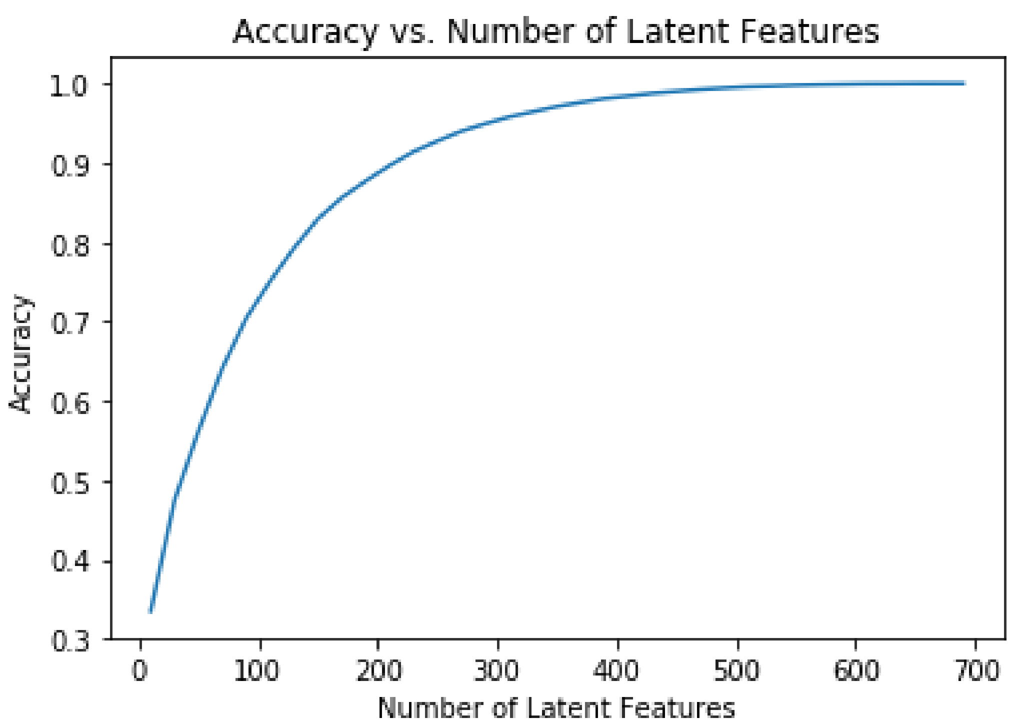
2. In this situation, you can use Singular Value Decomposition from numpy on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

```
In [44]: # 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 are able to use SVD because in this situation our matrix doesn't contain any missing values and SVD only works if this condition is satisfied.

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 [45]:
```
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 [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

             '''
             # Your code here
             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 [47]: user_item_train.shape, user_item_test.shape

Out[47]: ((4487, 714), (682, 574))
```

```
In [73]: train_idx = user_item_train.index
         common_idx = list(set(train_idx) & set(test_idx))
         print(common_idx)
```

```
train_arts = user_item_train.columns
len(set(train_arts) & set(test_arts))
```

```
[3968, 3777, 4002, 3684, 4293, 2917, 4487, 4231, 3527, 4204, 3532, 3024, 4274, 3801, 3093, 3989,
```

```
Out[73]: 574
```

```
In [50]: # 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 [51]: # 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
```

```
In [63]: # Use these cells to see how well you can use the training
         # decomposition to predict on test data
         print('train: ', u_train.shape, s_train.shape, vt_train.shape)
```

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

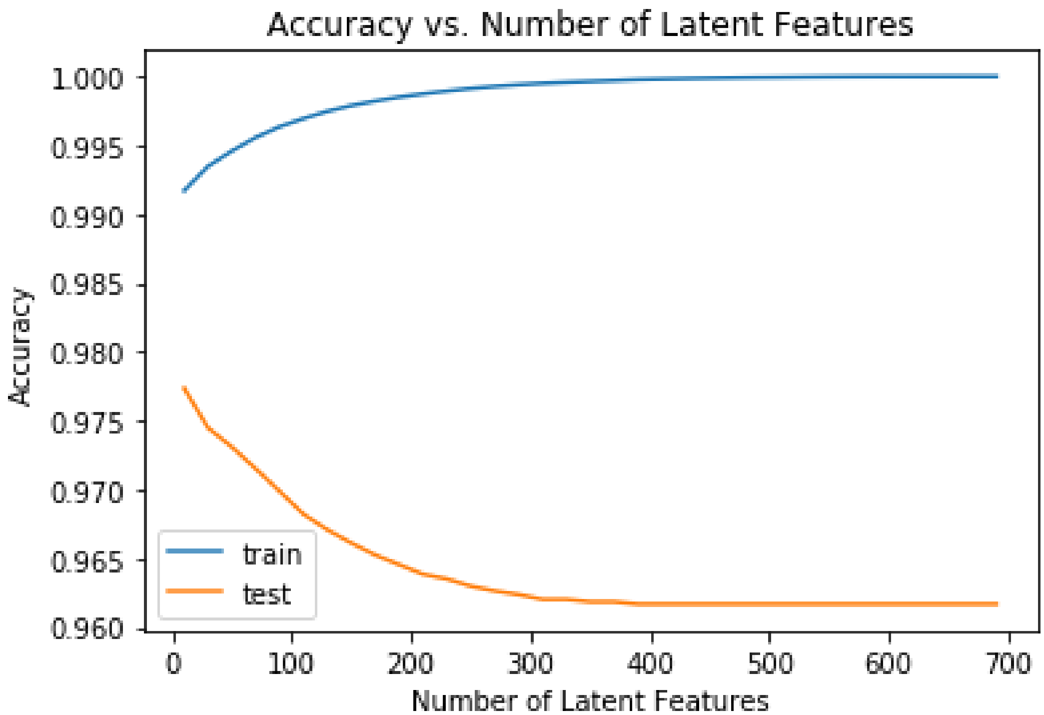
```
In [64]: # common users in both test and train set
         train_common_idx = user_item_train.index.isin(test_idx)
         train_common_col = user_item_train.columns.isin(test_arts)
```

```
In [66]: u_test = u_train[train_common_idx, :]
         vt_test = vt_train[:, train_common_col]
         print('test: ', u_test.shape, vt_test.shape)
```

```
test:  (20, 4487) (714, 574)
```

```
In [74]: user_item_test_subset = user_item_test.loc[common_idx]

         num_latent_feats = np.arange(10,700+10,20)
         train_sum_errs = []
         test_sum_errs = []

         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_est = np.around(np.dot(np.dot(u_train_new, s_train_new), vt_train_n
             user_item_test_est = np.around(np.dot(np.dot(u_test_new, s_train_new), vt_test_new)

             # compute error for each prediction to actual value
             diffs_train = np.subtract(user_item_train, user_item_train_est)
             diffs_test = np.subtract(user_item_test_subset, user_item_test_est)

             # total errors and keep track of them
             train_err = np.sum(np.sum(np.abs(diffs_train)))
             train_sum_errs.append(train_err)

             test_err = np.sum(np.sum(np.abs(diffs_test)))
             test_sum_errs.append(test_err)
```

```
In [75]: plt.plot(num_latent_feats, 1 - (np.array(train_sum_errs)/(user_item_train.shape[0]*user
         plt.plot(num_latent_feats, 1 - (np.array(test_sum_errs)/(user_item_test_subset.shape[0]
         plt.legend(loc='best')
         plt.xlabel('Number of Latent Features');
         plt.ylabel('Accuracy');
         plt.title('Accuracy vs. Number of Latent Features');
         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. It was observed that the accuracy curve is the inverse of the one we saw during training. The explanation for this is that with increasing latent features causes overfitting during training. In view of this we would try to keep fewer latent features.

As there are small datasets to train and test, we can't build our recommendation system solely based on SVD. There were also very few overlaps of users and articles between training and testing, so we can't get recommendation of all users.

It would be best give recommendation to users from various recommendation systems since every method has it's own flaws. So we can generate recommendations from all methods, removing duplicates and give some ranks to it. We can perform A/B testing, so that we can track how well our combinational recommendation works.

As we get more data, we can retrain the SVD model so that more users and articles comes into the model. We can also add a rating feature similar to facebook(like), which will enable us to know if the user actually likes the article, so that we can recommend articles similar to it.

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

Out[53]: 0
```