Recommendations_with_IBM

January 15, 2023

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
        import random

        %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]: *# Fill in the median and maximum number of user_article interactios below*

```python
        median_val = (df.groupby('email',as_index = False)['article_id'].count()
                    .rename(columns = {'article_id':'count'})
                    ['count'].median())# 50% of individuals interact with ____ number of articl
        max_views_by_user = (df.groupby('email',as_index = False)['article_id'].count()
                    .rename(columns = {'article_id':'count'})
                    ['count'].max())# The maximum number of user-article interactions by any 1
```

2

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

```
In [4]:  # Remove any rows that have the same article_id - only keep the first
         df_content = df_content.drop_duplicates('article_id', keep = 'first')
```

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

```
In [5]:  unique_articles = df['article_id'].nunique() # The number of unique articles that have a
         total_articles = df_content['article_id'].nunique() # The number of unique articles on t
         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 [6]:  # The most viewed article in the dataset as a string with one value following the decima
         most_viewed_article_id = (df.groupby('article_id',as_index = False)['email'].count()
                                     .rename(columns = {'email':'count'})
                                     .sort_values('count', ascending = False)
                                     .head(1)
                                     .loc[699,'article_id'].astype(str))

         # The most viewed article in the dataset was viewed how many times?
         max_views = (df.groupby('article_id',as_index = False)['email'].count()
                        .rename(columns = {'email':'count'})
                        .sort_values('count', ascending = False)
                        .head(1)
                        .loc[699,'count'])
```

```
In [7]:  ## No need to change the code here - this will be helpful for later parts of the noteboo
         # Run this cell to map the user email to a user_id column and remove the email column

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

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

                 email_encoded.append(coded_dict[val])
```

3

```python
        return email_encoded

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

    # show header
    df.head()
```

```
Out[7]:    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 [8]: sol_1_dict = {
            '`50% of individuals have _____ or fewer interactions.`': median_val,
            '`The total number of user-article interactions in the dataset is _____.`': user_ar
            '`The maximum number of user-article interactions by any 1 user is _____.`': max_vi
            '`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_articl
            '`The number of unique users in the dataset is _____`': unique_users,
            '`The number of unique articles on the IBM platform`': total_articles
        }
```

```
In [9]: ## 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_ar
            '`The maximum number of user-article interactions by any 1 user is _____.`': max_vi
            '`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_articl
            '`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.

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

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


             '''

             top_articles = list(df['title'].value_counts().index)[0: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_article_ids - (list) A list of the top 'n' article ids


             '''
             top_article_ids = list(df['article_id'].astype(str).value_counts().index)[0:n]

             return top_article_ids # Return the top article ids

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

5

```
          # 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.

```python
In [13]: 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
             '''
             user_item = df.groupby(['user_id', 'article_id'])['title'].nunique().unstack()
             user_item = user_item.fillna(0)

             return user_item # return the user_item matrix

         user_item = create_user_item_matrix(df)

In [14]: ## 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.

```python
In [15]: 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
             user_similarity = user_item.dot(user_item.T)
             # sort by similarity and create list of just the ids
             most_similar_users = list(user_similarity.loc[:,user_id].sort_values(ascending = Fa
             # 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
```

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

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

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

```
In [17]: 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)
             '''
             # edge case: to conver the input to list if not provided
             if isinstance(article_ids,list) == False:
                 article_ids = list(article_ids)

             article_names = df.loc[df['article_id'].isin(article_ids), 'title'].unique().tolist

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

In [18]: 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
             '''
             user_matrix = user_item[user_item.index == user_id]
             user_articles = user_matrix.loc[:,user_matrix.eq(1).all()]
             article_ids = user_articles.columns.tolist()
             article_ids = [str(id) for id in article_ids]
             article_ids = np.sort(article_ids)
             article_names = get_article_names(article_ids)

             return article_ids, article_names # return the ids and names

In [19]: def random_ids_sampling(m, df = df):
             '''
```

8

```
        INPUT:
        m - (int) the number of recommendations you want for the user
        df - (pandas dataframe) df as defined at the top of the notebook

        OUTPUT:
        random_ids

        Description:
        From

        '''

        random_ids = random.sample(set(df['article_id']),m)

        return [str(id) for id in random_ids]

In [20]: 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

        '''
        interacted_article_ids, interacted_article_names = get_user_articles(user_id)
        similar_users = find_similar_users(user_id)
        recs = []

        for user in similar_users:
                similar_users_article_ids,similar_users_article_names  = get_user_articles(
                for article in similar_users_article_ids:
                    if article not in recs and article not in interacted_article_ids:
                        if len(recs) < 10:
                            recs.append(article)
                        else:
```

```
                        break

            random_article_ids = random_ids_sampling(m=m)

            for i in range(len(recs),m):
                if random_article_ids[i] not in recs:
                    recs.append(random_article_ids[i])

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

In [21]: # Check Results
         get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1

Out[21]: ['analyze energy consumption in buildings',
          'analyze accident reports on amazon emr spark',
          '520   using notebooks with pixiedust for fast, flexi...\nName: title, dtype: object'
          '1448   i ranked every intro to data science course on...\nName: title, dtype: object
          'data tidying in data science experience',
          'airbnb data for analytics: vancouver listings',
          'recommender systems: approaches & algorithms',
          'airbnb data for analytics: mallorca reviews',
          'analyze facebook data using ibm watson and watson studio',
          'a tensorflow regression model to predict house values']

In [22]: # 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 [23]: 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 provide
                    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

    '''
    neighbor_ids = find_similar_users(user_id)
    user_df = user_item[user_item.index == user_id]

    neighbor_df = user_item.loc[neighbor_ids].dot(user_df.T)

    neighbor_df = (neighbor_df.rename(columns = {neighbor_df.columns[0] : 'similarity'}
                   .reset_index()
                   .rename(columns = {'user_id':'neighbor_id'})
                   .set_index('neighbor_id')
                   .assign(num_interactions = user_item.loc[neighbor_ids].sum(axis=1).v
                   .reset_index()
                   .sort_values(['similarity','num_interactions'],ascending=[False,Fals

    neighbor_df.columns.name = 'index'

    return neighbor_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.

            '''
            interacted_article_ids, interacted_article_names = get_user_articles(user_id)
            neighbor_df = get_top_sorted_users(user_id = user_id)
            similar_users = neighbor_df['neighbor_id']


            recs = []

            for user in similar_users:
                    similar_users_article_ids,similar_users_article_names  = get_user_articles(
                    for article in similar_users_article_ids:
                        if article not in recs and article not in interacted_article_ids:
                            if len(recs) < 10:
                                recs.append(article)
                            else:
                                break

            random_article_ids = random_ids_sampling(m=m)

            for i in range(len(recs),m):
                if random_article_ids[i] not in recs:
                    recs.append(random_article_ids[i])

            rec_names = get_article_names(recs)

            return recs, rec_names

In [24]: # 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:
['1014.0', '1059.0', '109.0', '111.0', '1157.0', '1162.0', '1164.0', '1172.0', '1186.0', '12.0']

The top 10 recommendations for user 20 are the following article names:
```

```
['timeseries data analysis of iot events by using jupyter notebook', 'analyze energy consumption
```

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.

```python
In [25]: ### Tests with a dictionary of results
         # Find the user that is most similar to user 1
         user1_most_sim = get_top_sorted_users(user_id = 1)['neighbor_id'][0]

         # Find the 10th most similar user to user 131
         user131_10th_sim = get_top_sorted_users(user_id = 131)['neighbor_id'][9]

In [26]: ## 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.

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.

When new user comes to the platform, existing system can't be applied, since we don't have a record of interacted articles and neighbors of a new user.

In that case to maximize interaction rate of a new user on the platform, the best solution is to recommend top n interacted articles

```python
In [27]: new_user = '0.0'

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

In [28]: 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 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 [29]: # Load the matrix here
         user_item_matrix = pd.read_pickle('user_item_matrix.p')

In [30]: # quick look at the matrix
         user_item_matrix.head()

Out[30]: 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 [31]: # Perform SVD on the User-Item Matrix Here
         u, s, vt = np.linalg.svd(user_item_matrix)
```

Provide your response here.

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

14

```
In [32]: 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');
```
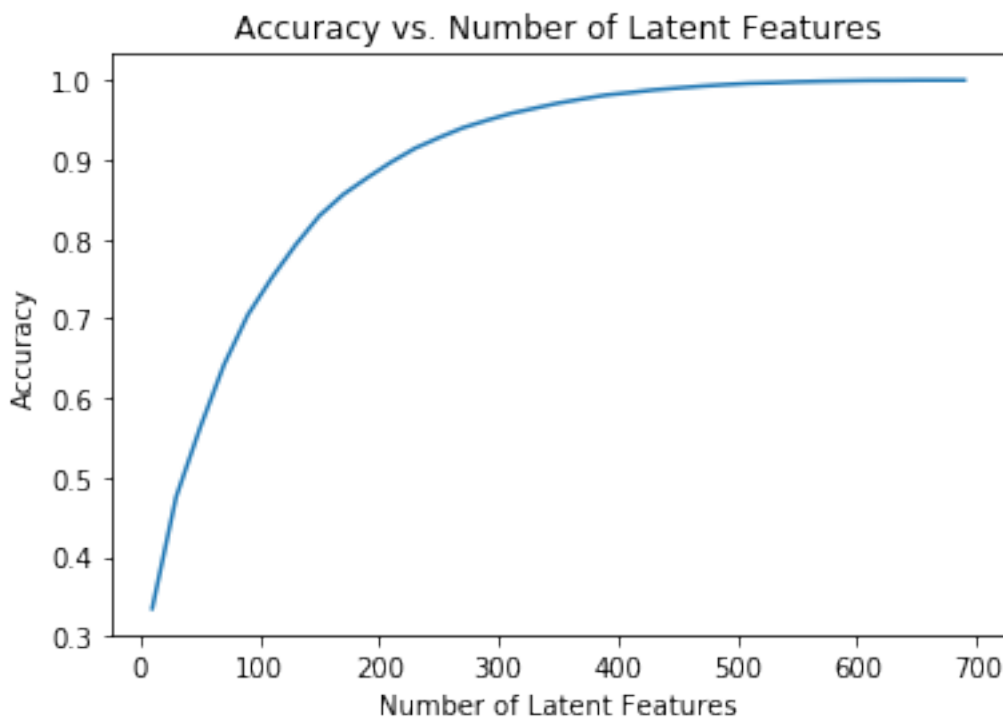

4. From the above, we can't really be sure how many features to use, because simply having a better way to predict the 1's and 0's of the matrix doesn't exactly give us an indication of if we are

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 [33]: 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
             df_train = df_train.drop_duplicates(['article_id','user_id'])
             df_test = df_test.drop_duplicates(['article_id','user_id'])

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

             test_idx = df_test['user_id'].drop_duplicates().tolist()
             test_arts = df_test['article_id'].drop_duplicates().tolist()

             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 [34]: print('1. How many users can we make predictions for in the test set?')
         print(user_item_test.index.isin(user_item_train.index).sum())
         print()
         print('2. How many users in the test set are we not able to make predictions for becaus
```

```
        print(sum(user_item_test.index.isin(user_item_train.index)==False))
        print()
        print('3. How many articles can we make predictions for in the test set?')
        print(len(test_arts))
        print()
        print('4. How many articles in the test set are we not able to make predictions for bec
        print(sum(user_item_test.columns.isin(user_item_train.columns)==False))
```

```
1. How many users can we make predictions for in the test set?
20

2. How many users in the test set are we not able to make predictions for because of the cold st
662

3. How many articles can we make predictions for in the test set?
574

4. How many articles in the test set are we not able to make predictions for because of the cold
0
```

In [35]: # Replace the values in the dictionary below
```
        a = 662
        b = 574
        c = 20
        d = 0



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

        t.sol_4_test(sol_4_dict)
```

Awesome job! That's right! All of the test articles are in the training data, but there are on

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 [36]: # 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 [37]: row_ids = user_item_train.index.isin(test_idx)
         columns_ids = user_item_train.columns.isin(test_arts)
         u_test = u_train[row_ids,:]
         vt_test = vt_train[:,columns_ids]

In [38]: num_latent_feats = np.arange(10,700+10, 20)
         sum_errs_train = []
         sum_errs_test = []


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

             # compute error for each prediction to actual value
             # training
             diffs_train = np.subtract(user_item_train, user_item_est_train)
             # test
             commmon_test_df = user_item_test.loc[(user_item_train.index & user_item_test.index)
             diffs_test = np.subtract(commmon_test_df, user_item_est_test)

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



         plt.plot(num_latent_feats, 1 - np.array(sum_errs_train)/(user_item_est_train.shape[0]*u
         plt.plot(num_latent_feats, 1 - np.array(sum_errs_test)/(user_item_est_test.shape[0]*use
         plt.xlabel('Number of Latent Features');
         plt.ylabel('Accuracy');
         plt.legend()
         plt.title('Accuracy vs. Number of Latent Features');
```
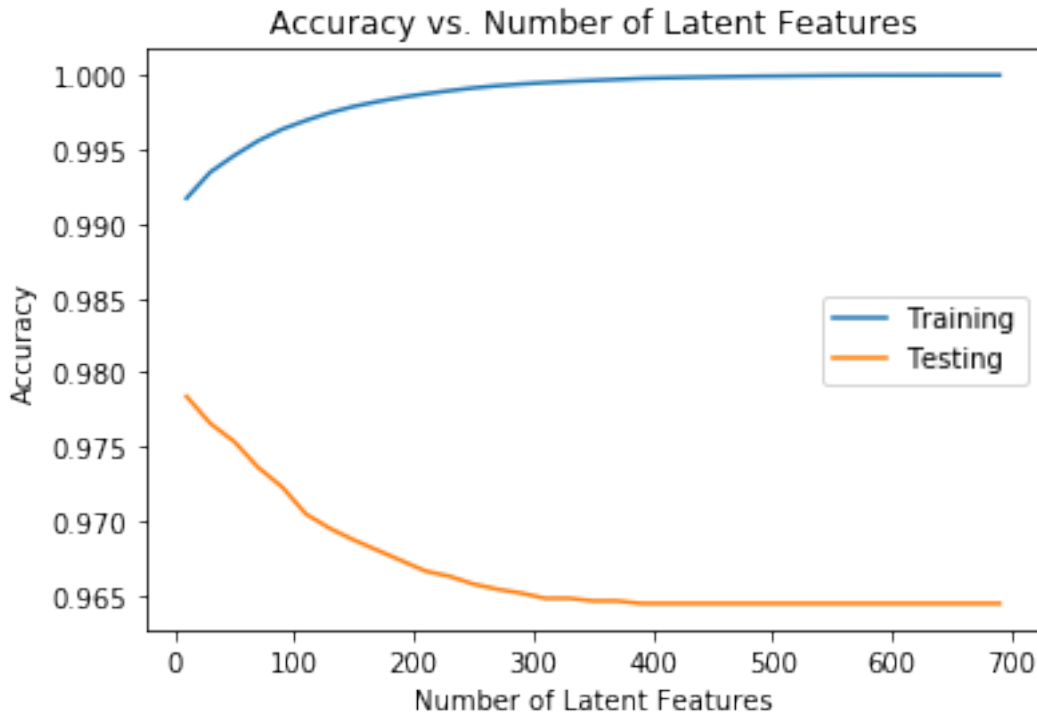
Accuracy vs. Number of Latent Features

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

`In [ ]:`

Your response here.

Observations: 1. Increase the number of Latent factors increases the accuracy on training dataset and at the same decrease the accuracy of the model on testing datase 2. Testing dataset has 574 articles to make inference and only 20 users that intersect between tarining and testing datasets. It means that there's no enough evidence to conclude whether observation 1 is overfit or real trend. 3. If to assume that the observation 1 is a real trend, in that case the most optimal solution is to have as less latent factors as possible, which is not the case in most of the instances of real life examples

Proposed solutions: 1. Resample training and testing datasets with bigger intersection of users between and keep the same amount of articles or more 2. Sample a third dataset from df, to test whether accuracy is robust on chosen number of Latent factors on validation dataset 3. Start collecting data of the satisfaction rate of users by reading the article. E.g there might be a case that article is popular among reader, though it's quality make people never return to the platform. Existing metrics do not track it and proposed recommendation system would keep recommending this article all the time

1.2 Conclusion

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

19

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!

```python
In [39]: from subprocess import call
         call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])

Out[39]: 0
```