Recommendations_with_IBM

February 11, 2023

1 Recommendations with IBM

In this notebook, I will be putting my recommendation skills to use on real data from the IBM Watson Studio platform.

1.1 Table of Contents


```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        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]: # 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

I will be providing some insight into the descriptive statistics of the data.

1. What is the distribution of how many articles a user interacts with in the dataset? Provide a visual and descriptive statistics to assist with giving a look at the number of times each user interacts with an article.

```
In [3]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45993 entries, 0 to 45992
Data columns (total 3 columns):
article_id    45993 non-null float64
title         45993 non-null object
email         45976 non-null object
dtypes: float64(1), object(2)
memory usage: 1.1+ MB
```

```
In [4]: df.shape
```

```
Out[4]: (45993, 3)
```

```
In [5]: df.groupby(["email","article_id"]).max().unstack().head(3) #email-article interaction (m
```

```
Out[5]:                                          title                              \
        article_id                               0.0    2.0    4.0    8.0    9.0
        email
        0000b6387a0366322d7fbfc6434af145adf7fed1    NaN    NaN    NaN    NaN    NaN
        001055fc0bb67f71e8fa17002342b256a30254cd    NaN    NaN    NaN    NaN    NaN
        00148e4911c7e04eeff8def7bbbdaf1c59c2c621    NaN    NaN    NaN    NaN    NaN

                                                                                    \
        article_id                               12.0   14.0   15.0   16.0   18.0
        email
        0000b6387a0366322d7fbfc6434af145adf7fed1    NaN    NaN    NaN    NaN    NaN
        001055fc0bb67f71e8fa17002342b256a30254cd    NaN    NaN    NaN    NaN    NaN
        00148e4911c7e04eeff8def7bbbdaf1c59c2c621    NaN    NaN    NaN    NaN    NaN

                                                  ...                               \
        article_id                                ...   1434.0 1435.0 1436.0 1437.0
        email                                     ...
        0000b6387a0366322d7fbfc6434af145adf7fed1  ...      NaN    NaN    NaN    NaN
        001055fc0bb67f71e8fa17002342b256a30254cd  ...      NaN    NaN    NaN    NaN
        00148e4911c7e04eeff8def7bbbdaf1c59c2c621  ...      NaN    NaN    NaN    NaN

                                                                                    \
        article_id                               1439.0 1440.0 1441.0 1442.0 1443.0
        email
        0000b6387a0366322d7fbfc6434af145adf7fed1    NaN    NaN    NaN    NaN    NaN
        001055fc0bb67f71e8fa17002342b256a30254cd    NaN    NaN    NaN    NaN    NaN
        00148e4911c7e04eeff8def7bbbdaf1c59c2c621    NaN    NaN    NaN    NaN    NaN


        article_id                               1444.0
        email
        0000b6387a0366322d7fbfc6434af145adf7fed1    NaN
        001055fc0bb67f71e8fa17002342b256a30254cd    NaN
        00148e4911c7e04eeff8def7bbbdaf1c59c2c621    NaN

        [3 rows x 714 columns]
```

Explore the Data

```
In [6]: user_article_i = df.groupby("email").count()
        user_article_i.head(6)
```

```
Out[6]:                                           article_id  title
        email
        0000b6387a0366322d7fbfc6434af145adf7fed1          13     13
        001055fc0bb67f71e8fa17002342b256a30254cd           4      4
        00148e4911c7e04eeff8def7bbbdaf1c59c2c621           3      3
```

```
001a852ecbd6cc12ab77a785efa137b2646505fe           6      6
001fc95b90da5c3cb12c501d201a915e4f093290           2      2
0042719415c4fca7d30bd2d4e9d17c5fc570de13           2      2
```

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

Out[7]:

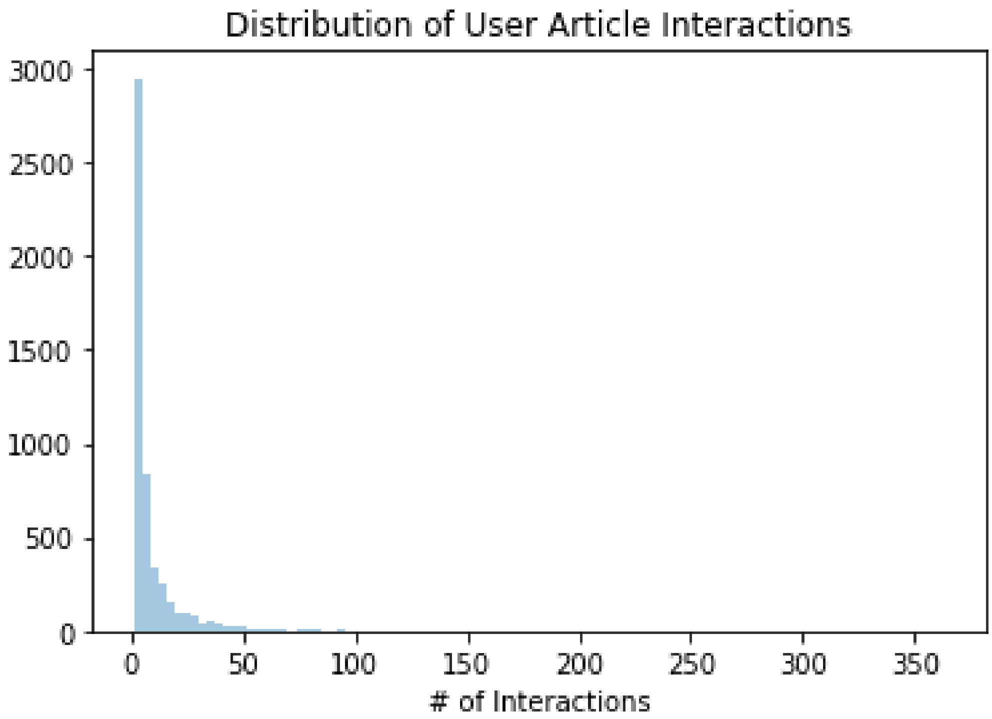
	article_id	title
count	5148.000000	5148.000000
mean	8.930847	8.930847
std	16.802267	16.802267
min	1.000000	1.000000
25%	1.000000	1.000000
50%	3.000000	3.000000
75%	9.000000	9.000000
max	364.000000	364.000000

In [8]:
```
user_article_i[user_article_i["article_id"]==364]
```

Out[8]:

	article_id	title
email		
2b6c0f514c2f2b04ad3c4583407dccd0810469ee	364	364

In [9]:
```
sns.distplot(user_article_i["article_id"],bins = 100, kde = False) # Distribution of the
plt.title('Distribution of User Article Interactions')
plt.xlabel('# of Interactions');
```



```
In [10]: # Fill in the median and maximum number of user_article interactios below

         median_val = df.groupby("email").count()["article_id"].median()
         max_views_by_user = df.groupby("email").count()["article_id"].max()

         print(median_val)
         print("\n")
         print(max_views_by_user)
```

```
3.0


364
```

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

```
In [11]: df_content.shape # shape of the dataset
```

```
Out[11]: (1056, 5)
```

```
In [12]: df_content[df_content.duplicated("article_id")]
```

```
Out[12]:                                               doc_body  \
         365  Follow Sign in / Sign up Home About Insight Da...
         692  Homepage Follow Sign in / Sign up Homepage * H...
         761  Homepage Follow Sign in Get started Homepage *...
         970  This video shows you how to construct queries ...
         971  Homepage Follow Sign in Get started * Home\r\n...

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

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

```
In [13]: # Remove any rows that have the same article_id - only keep the first
         df_content.drop_duplicates(subset="article_id",inplace=True,keep="first")
```

```
In [14]: # After droping dublicates
         df_content.shape
```

Out[14]: (1051, 5)

3. Use the cells 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 [15]: unique_articles = df["article_id"].nunique() # The number of unique articles that have
         total_articles = df_content["article_id"].nunique() # The number of unique articles on
         unique_users = df["email"].nunique() # The number of unique users
         user_article_interactions = df.shape[0]

         print(unique_articles)
         print(total_articles)
         print(unique_users)
         print(user_article_interactions)
```

```
714
1051
5148
45993
```

4. Using the cells below to find the most viewed **article_id**, as well as how often it was viewed.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 [16]: most_viewed_article_id = str(df["article_id"].value_counts().index[0])  # Most viewed a
         max_views = df.groupby("article_id").count().max()["email"] # Max Views
```

```
In [17]: # email_mapper function, used to map the email address to user ID
         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]: # Finding solutions to below questions

         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 [19]: 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()[:n].index.tolist()
             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 titles

    '''
    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 [20]: 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 [21]: # 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. Using 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**.

Using the tests to make sure the basic structure of your matrix matches what is expected by the solution.

```
In [22]: # 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"]).count().unstack()
             user_item.fillna(0, inplace = True)
             user_item = user_item.applymap(lambda x: 1 if x>0 else x )

             return user_item["title"] # return the user_item matrix

         user_item = create_user_item_matrix(df)
```

```
In [23]: ## 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 [24]: 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
            simi_matrix = user_item.dot(np.transpose(user_item))

            # sort by similarity
            simi_matrix = simi_matrix.loc[user_id].sort_values(ascending = False)

            # create list of just the ids
            most_similar_users = simi_matrix.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 [25]:
```
# Do a spot check of your function
print("The 10 most similar users to user 1 are: {}".format(find_similar_users(1)[:10]))
print("The 5 most similar users to user 3933 are: {}".format(find_similar_users(3933)[:
print("The 3 most similar users to user 46 are: {}".format(find_similar_users(46)[:3]))
```

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

In [26]:
```
df_content.head() # for reference
```

Out[26]:
```
                                            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
```

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]:
```
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 = list()
    df = df.set_index("article_id")

    for idd in article_ids:
        article_names.append(df.loc[float(idd)].max()["title"])

    return article_names

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].values == 1].index.asty
```

```
    article_names = []

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

    return article_ids, article_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

    '''

    neighbor_users = find_similar_users(user_id) # closest neighbor to our user_id
    user_articles = get_user_articles(user_id)[0] #seen by our user

    recs = np.array([])


    for user in neighbor_users:

        neighbor_articles_seen = get_user_articles(user)[0] # movies seen by others lik
        recs1 = np.setdiff1d(neighbor_articles_seen, user_articles, assume_unique=True)
        recs = np.concatenate([recs1, recs], axis = 0) #concanate arrays
        recs = np.unique(recs) # find unique items in array

        if len(recs) > m-1:
            break

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

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

```
In [28]: # Check Results
         get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

```
Out[28]: ['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 [29]: # 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 [30]: 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

    '''
    # dot product of user_item with user_item transpose which gives the similarities bt
    user_dot_pro = user_item.dot(np.transpose(user_item))

    # dataframe with neighbor_id and similarity columns
    neighbors_df = user_dot_pro.loc[user_id].rename_axis("neighbor_id").reset_index(nam

    # dataframe with the user-article interactions
    interaction_df = df["user_id"].value_counts().rename_axis("neighbor_id").reset_inde

    # Merged dataframe of neighbors_df and interaction_df
    neighbors_df = pd.merge(neighbors_df,interaction_df,on="neighbor_id",how = "outer")

    # sort the dataframe with similarity first and then number of interactions
    neighbors_df = neighbors_df.sort_values(by=['similarity', 'num_interactions'], asce

    # Remove the row with the input user
    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([])

            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

            return recs, rec_names
```

```
In [31]: # 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 [32]: ### Tests with a dictionary of results
```

```
user1_most_sim = get_top_sorted_users(1)["neighbor_id"].iloc[0]
user131_10th_sim = get_top_sorted_users(131)["neighbor_id"].iloc[9]
```

```
In [33]: ## Dictionary Test Here
         sol_5_dict = {
             'The user that is most similar to user 1.': user1_most_sim,
             'The user that is the 10th most similar to user 131': user131_10th_sim,
         }

         t.sol_5_test(sol_5_dict)
```

```
This all looks good!  Nice job!
```

6. If we were given a new user, which of the above functions would you be able to use to make recommendations? Explain. Can you think of a better way we might make recommendations? Use the cell below to explain a better method for new users.

The function that is the best amongst the functions created is the knowledge based function (get_top_articles) as it gives the top articles interacted with. It is the best one for a new user as we do not have enough information regarding the new user, to which the new user has interacted with. There can be a better way to recommend if we have enough information regarding the user and we can use the combination of knowledge, collaboration type recommendatio

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.

```
In [34]: 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=df)
```

```
In [35]: 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 [36]: # Load the matrix here
         user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

```
In [37]: # quick look at the matrix
         user_item_matrix.head()
```

```
Out[37]: 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 [38]: # Perform SVD on the User-Item Matrix Here

         u, s, vt = np.linalg.svd(user_item_matrix)
```

```
In [39]: print(u.shape)
         print(s.shape)
         print(vt.shape)
```

```
(5149, 5149)
(714,)
(714, 714)
```

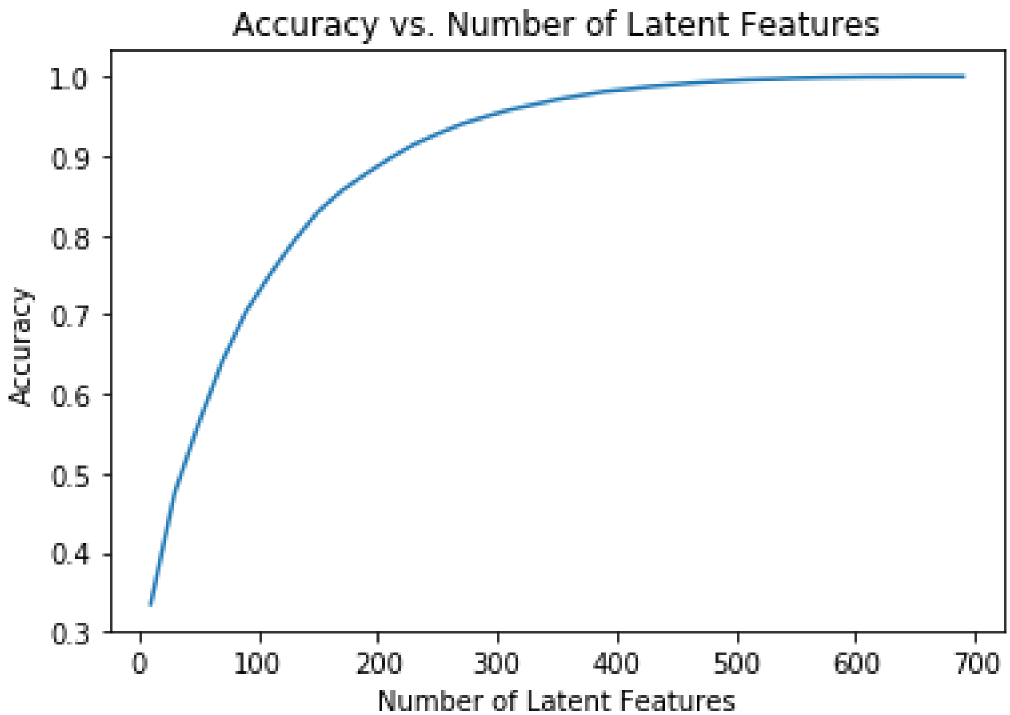
Provide your response here.

1.1.5 Response

In this scenario, instead of using FunkSVD, we opt to employ Numpy's SVD because our user-item matrix doesn't have any missing values. This is achieved by filling in the empty spaces with zeros. However, if our goal is to predict ratings for items that have not been rated by users, we wouldn't fill in the missing values. In this particular case, it's acceptable to fill in the missing values with zeros, as it indicates that the user hasn't viewed the movie yet.

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 [40]:
```
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 [41]: 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(
```

1.1.6 Exploring and Answering Questions Below

```
In [42]: print(test_idx.shape)
         print(len(test_arts))

(682,)
574


In [48]: train_idx = user_item_train.index # 4487 users are in training set
         train_idx

Out[48]: 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 [49]: test_idx # 682 users are in test set

Out[49]: 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 [50]: test_idx.difference(train_idx) #out of 682 users in test set, only 20 of them are in tr

Out[50]: 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 [51]: test_arts #574 movies are in test set
```

```
Out[51]: 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 [54]: train_a = user_item_train.columns #714 movies are in train set
         train_a
```

```
Out[54]: 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 [55]: test_arts.difference(train_a) # 0
```

```
Out[55]: Float64Index([], dtype='float64', name='article_id')
```

```
In [56]: # The above are the solutions to the questions asked below
```

```
In [43]: # 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 [44]:
```
# fit SVD on the user_item_train matrix
u_train, s_train, vt_train = np.linalg.svd(user_item_train)
```

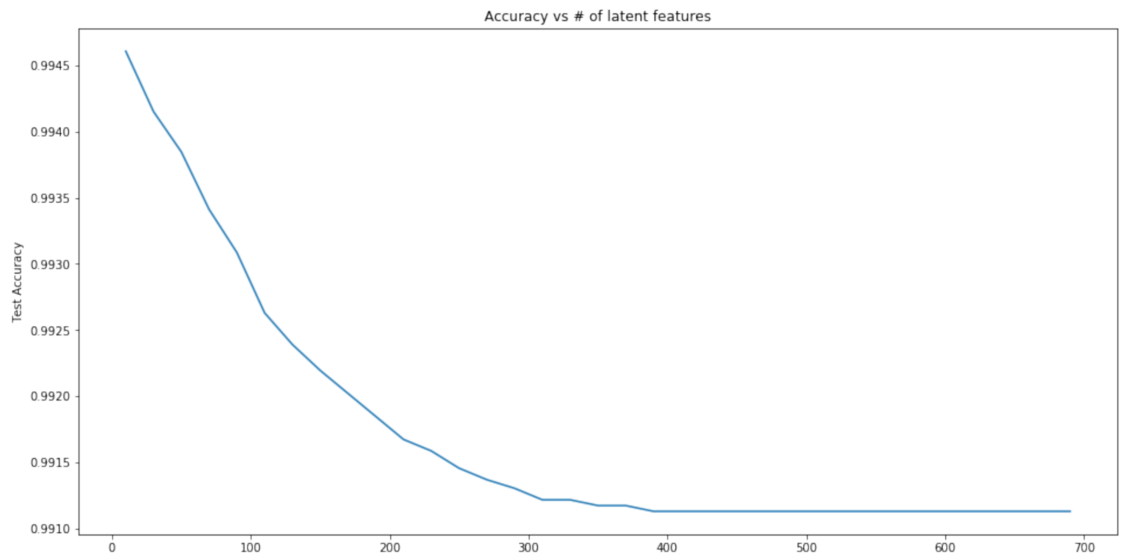
In [46]:
```
# decomposition to predict on test data
print(u.shape)
print(s.shape)
print(vt.shape)


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

#Decomposition
row_i = user_item_train.index.isin(test_idx)
col_i = user_item_train.columns.isin(test_arts)

u_test = u_train[row_i, :]
vt_test = vt_train[:, col_i]

users_predict = np.intersect1d(list(user_item_train.index),list(user_item_test.index))
for k in num_latent_feats:
    # restructure with k latent features
    s_train_n, u_train_n, vt_train_n = np.diag(s_train[:k]), u_train[:, :k], vt_train[:
    u_test_n, vt_test_n = u_test[:, :k], vt_test[:k, :]

    # take dot product
    user_item_train_preds = np.around(np.dot(np.dot(u_train_n, s_train_n), vt_train_n))
    user_item_test_preds = np.around(np.dot(np.dot(u_test_n, s_train_n), vt_test_n))

    # 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_predict,:], user_item_test_preds)

    # total errors
    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)
```

```
(5149, 5149)
(714,)
(714, 714)
```

In [47]:
```
f, ax = plt.subplots(figsize=(16, 8))
ax.set_ylabel('Test Accuracy')
```

```
ax.plot(num_latent_feats, 1 - np.array(sum_errs_test)/df.shape[0])
ax.tick_params(axis='y')
ax.set_title("Accuracy vs # of latent features")
```

Out[47]: Text(0.5,1,'Accuracy vs # 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?

Your response here.

We see that the accuracy for the test data decreases with an increase in latent features. This is mostly due to over fitting of the data. It is best to keep the latent features relatively low.

Conduct A/B test to solve the cold start problem and evaluate recommendation engine performance. Recommend articles to one group using the engine and to the other with most popular articles. Compare click-through rates (maybe using cookies) to measure increase in clicks. If a statistically significant(p-value test) rise in clicks is observed, the recommendation engine is deemed successful.

1.1.7 In Conclusion

In conclusion, this project aimed to develop a recommendation engine for the IBM Watson Studio platform to suggest articles to users based on their past interactions and preferences. The project consisted of four distinct tasks: Exploratory Data Analysis, Rank Based Recommendations, User-User Based Collaborative Filtering, and Matrix Factorization. Through these tasks, the effectiveness of different recommendation approaches was evaluated, and it was found that the Matrix Factorization method produced the best results. It is therefore recommended that this approach be deployed for the IBM Watson Studio platform to enhance the user experience and increase engagement with the articles.

What we can to do make the project more interesting! 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!

```
In [57]: from subprocess import call
         call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])

Out[57]: 0

In [ ]:
```