Recommendations_with_IBM

January 26, 2020

1 IBM Watson Product Recommendations

1.1 Table of Contents

```
In [98]: # Importing libraries
         import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import project_tests as t
         import pickle
         import warnings
         warnings.filterwarnings("ignore")

         %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[98]:    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 [99]: # Show df_content to get an idea of the data
         df_content.head()

Out[99]:                                              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

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.

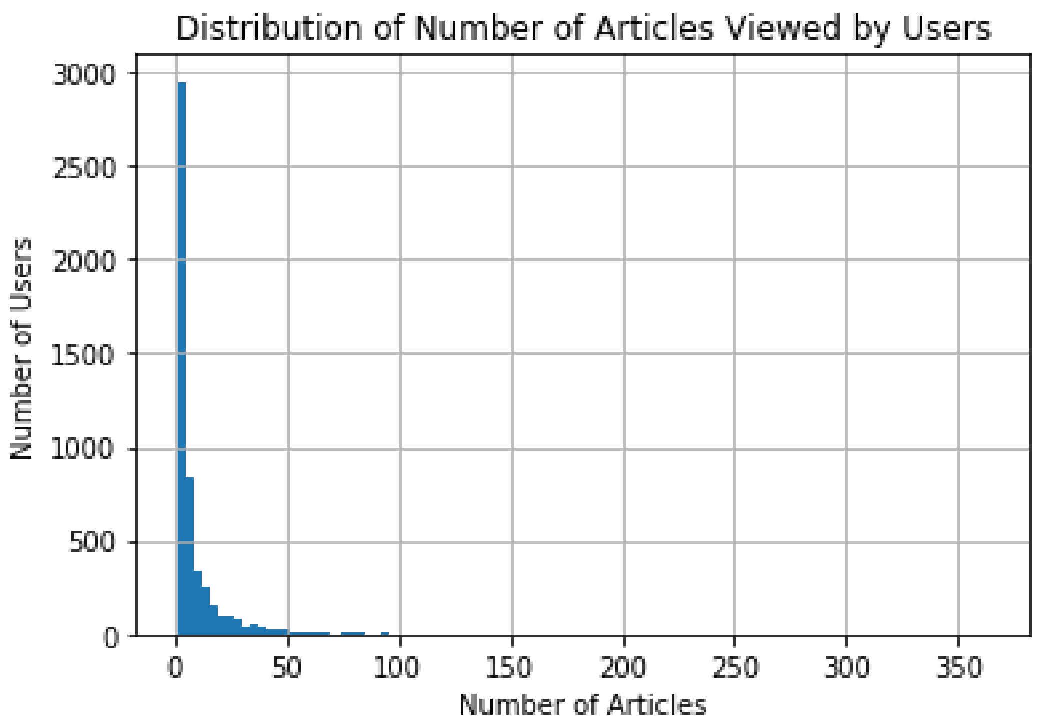
```
In [100]: # Total number of unique articles each user interacts with
          df_user_articles = df.groupby('email')['article_id'].count()
          df_user_articles.head()

Out[100]: email
          0000b6387a0366322d7fbfc6434af145adf7fed1    13
          001055fc0bb67f71e8fa17002342b256a30254cd     4
          00148e4911c7e04eeff8def7bbbdaf1c59c2c621     3
          001a852ecbd6cc12ab77a785efa137b2646505fe     6
          001fc95b90da5c3cb12c501d201a915e4f093290     2
          Name: article_id, dtype: int64

In [101]: # Number of times each user interacts with each article
          df_user_articles_ntimes = df.groupby('email')['article_id'].value_counts()
          df_user_articles_ntimes.head()

Out[101]: email                                     article_id
          0000b6387a0366322d7fbfc6434af145adf7fed1  43.0          2
                                                    124.0         1
```

```
                                          173.0          1
                                          288.0          1
                                          349.0          1
          Name: article_id, dtype: int64
```

```
In [102]: # Summary of user_article interaction
          df.email.value_counts().describe()
```

```
Out[102]: 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: email, dtype: float64
```

```
In [103]: # Distribution of user_article interaction
          df_user_articles.hist(bins=100)
          plt.title('Distribution of Number of Articles Viewed by Users')
          plt.xlabel('Number of Articles')
          plt.ylabel('Number of Users');
```



```
In [104]: # Median and maximum number of user_article interactios below
          median_val = df.email.value_counts().median()
          print("50% of individuals interact with {} number of articles or fewer.".format(median

          max_views_by_user = df.email.value_counts().max()
          print("The maximum number of user-article interactions by any 1 user is {}.".format(ma
```

```
50% of individuals interact with 3.0 number of articles or fewer.
The maximum number of user-article interactions by any 1 user is 364.
```

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

```
In [105]: # Checking the shape of dataset -- 1056 rows
          df_content.shape
```

```
Out[105]: (1056, 5)
```

```
In [106]: # Finding unique articles
          df_content['article_id'].nunique()
```

```
Out[106]: 1051
```

```
In [107]: # Finding duplicate items
          df_content['article_id'].duplicated().sum()
```

```
Out[107]: 5
```

```
In [108]: # Exlporing duplicate articles
          df_content[df_content.duplicated('article_id')]
```

```
Out[108]:                                              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 [109]: # Removing any rows that have the same article_id - only keep the first
          df_content = df_content.drop_duplicates(subset='article_id', keep='first')
```

3. 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 [111]: 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] # The number of user-article interactions
```

4. 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 [112]: most_viewed_article_id = str(df.article_id.value_counts().sort_values(ascending=False)
          print("Most viewed article: {} ".format(most_viewed_article_id))

          max_views = df.article_id.value_counts().sort_values(ascending=False).iloc[0] # The mo
          print("Number of times most viewed article was seen: {} ".format(max_views))
```

```
Most viewed article: 1429.0
Number of times most viewed article was seen: 937
```

```
In [113]: # 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[113]:    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 [114]: # Tests

          sol_1_dict = {
              '`50% of individuals have _____ or fewer interactions.`': median_val,
              '`The total number of user-article interactions in the dataset is ______.`': user_
              '`The maximum number of user-article interactions by any 1 user is ______.`': max_
              '`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_arti
              '`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

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. Return the **n** top articles ordered with most interactions as the top.

```
In [115]: 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.groupby('title').count().sort_values(by='user_id', ascendin

              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 = list(df.article_id.value_counts().head(n).index.astype(str))

        return top_articles # Return the top article ids
```

```
In [116]: 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 [117]: # Test the 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 the 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. Writing a 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**.

```
In [118]: df.head()
```

```
Out[118]:    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 [119]: # 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
              '''
              user_item = df.groupby(['user_id', 'article_id'])['title'].count().unstack().apply

              return user_item # return the user_item matrix

          user_item = create_user_item_matrix(df)
```

```
In [120]: ## Tests
          assert user_item.shape[0] == 5149, "Oops!  The number of users in the user-article mat
          assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-article m
          assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by user 1 d
          print("You have passed our quick tests!  Please proceed!")
```

```
You have passed our quick tests!  Please proceed!
```

2. A function that takes 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.

```
In [121]: 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 produc
                        are listed first

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

        '''
        # compute similarity of each user to the provided user
        sim_users = user_item[user_item.index == user_id].dot(user_item.T)
        # sort by similarity
        most_similar = sim_users.sort_values(user_id, axis = 1, ascending=False)
        # create list of just the ids
        most_similar_users = most_similar.columns.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 [122]: # Do a spot check of the 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, 3870, 131, 4201, 46, 5041]
The 5 most similar users to user 3933 are: [1, 23, 3782, 203, 4459]
The 3 most similar users to user 46 are: [4201, 3782, 23]
```

3. Functions below to return the articles we would recommend to each user.

```
In [123]: 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(set(df[df.article_id.isin(article_ids)]['title']))


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


          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 use
    '''
    article_ids = user_item.loc[user_id][user_item.loc[user_id] == 1].index.values.ast
    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 re
    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

    '''

    # users similar to provided user
    most_similar_users = find_similar_users(user_id)

    # articles read by provided user
    user_articles_ids = set(get_user_articles(user_id)[0])

    # articles seen by similar users
```

```
        recs = []
        for sim_user_id in most_similar_users:
            sim_user_article_ids = set(get_user_articles(sim_user_id)[0])
            recs+= list(set(sim_user_article_ids) - set(user_articles_ids))

            if len(recs) > m:
                break;

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

```
In [124]: # Check Results
          get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

```
Out[124]: ['machine learning and the science of choosing',
           'this week in data science (april 25, 2017)',
           'times world university ranking analysis',
           'analyze open data sets with spark & pixiedust',
           'machine learning exercises in python, part 1',
           'recent trends in recommender systems',
           'machine learning for the enterprise',
           'get social with your notebooks in dsx',
           'flightpredict ii: the sequel   ibm watson data lab',
           'analyze open data sets with pandas dataframes']
```

```
In [126]: # Tests
          assert set(get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0', '1427.
          assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing (2015): u
          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 demographi
          assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1314.0', '1
          assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct high-r
          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 [127]: 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

    Other Details - sort the neighbors_df by the similarity and then by number of inte
                    highest of each is higher in the dataframe

    '''
    neighbors_df = pd.DataFrame(columns=['neighbor_id', 'similarity', 'num_interaction

    for user in user_item.index:
        if user!= user_id:
            neighbor_id = user
            similarity = np.dot(user_item.loc[user_id, :], user_item.loc[user, :])
            num_interactions = df[df['user_id'] == user]['article_id'].count()
            neighbors_df.loc[neighbor_id] = [neighbor_id, similarity, num_interactions

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

    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 re
    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)
            user_article_ids, user_article_names = get_user_articles(user_id)

            for neighbor_user in neighbors_df['neighbor_id']:
                neighbor_user_article_ids, neighbor_user_article_names = get_user_articles(nei
                for id in neighbor_user_article_ids:
                    if id not in user_article_ids:
                        recs.append(id)
                    if len(recs) > m:
                        break;
                if len(recs) > m:
                    break;
            rec_names = get_article_names(recs)


            return recs, rec_names
```

```
In [128]: # Quick spot check
          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', '555.0

The top 10 recommendations for user 20 are the following article names:
['timeseries data analysis of iot events by using jupyter notebook', 'introduction to neural net
```

5. Testing the dictionary against the solution.

```
In [129]: ### Tests with a dictionary of results
          user1_most_sim = get_top_sorted_users(1).iloc[0].neighbor_id # Find the user that is m
          user131_10th_sim = get_top_sorted_users(131).iloc[9].neighbor_id # Find the 10th most
```

```
In [130]: ## 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?
   Since the new user hasn't interacted with any article yet, it is better to show/recommend the most popular articles. As we learn more about the user when he/she interacts with the articles, we can use collaborative filtering to recommend the articles.
7. Provide the top 10 recommended articles you would provide for the a new user below.

```
In [132]: new_user = '0.0'

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

In [133]: 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: Matrix Factorization

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

1. We have already created a **user_item** matrix above in **question 1** of **Part III** above.

```
In [134]: # Load the matrix here
          user_item_matrix = pd.read_pickle('user_item_matrix.p')

In [135]: # quick look at the matrix
          user_item_matrix.head()

Out[135]: 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.

```
In [138]: # 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 matrice
```

```
In [139]: u.shape, s.shape, vt.shape
```

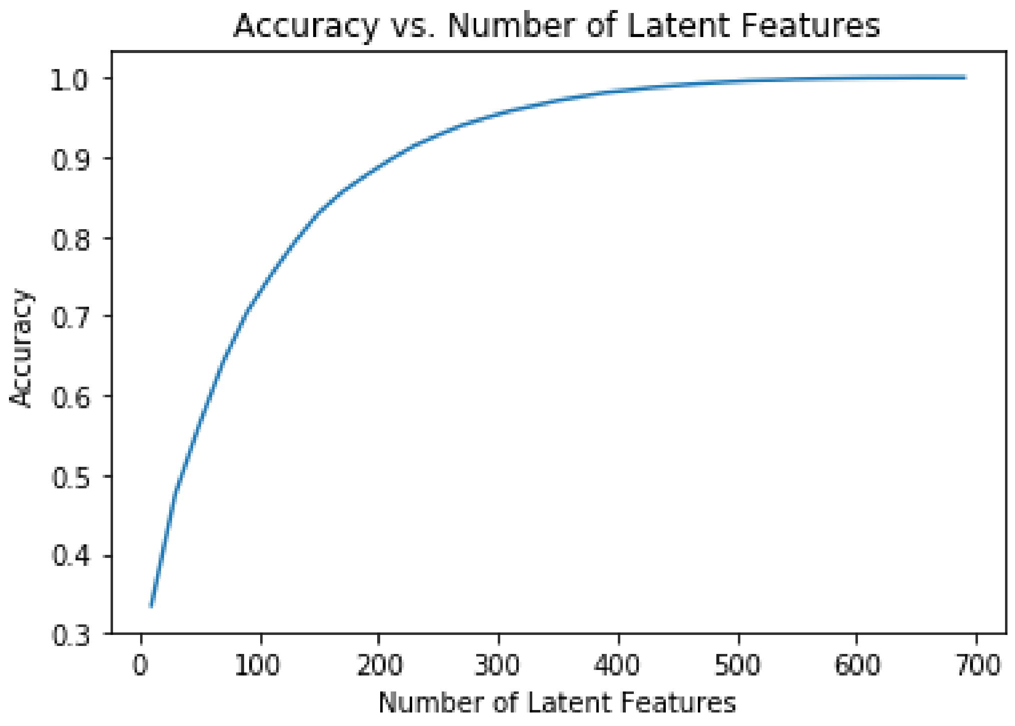
```
Out[139]: ((5149, 5149), (714,), (714, 714))
```

We can use SVD here because the user-item matrix doesn't contain any missing values. Each cell has a value of 0 or 1. If that wouldn't have been the case, we would have used FunkSVD, which can handle missing values.

3. Now for the tricky part, how do we choose the number of latent features to use? We 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.

```
In [140]: 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.

Let's 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 [141]: 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 = set(user_item_test.index)
    test_arts = set(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 [142]: user_item_train.shape
```

```
Out[142]: (4487, 714)
```

```
In [143]: user_item_test.shape
```

```
Out[143]: (682, 574)
```

```
In [144]: # How many users can we make predictions for in the test set?
          len(set(user_item_test.index) & set(user_item_train.index))
```

```
Out[144]: 20
```

```
In [145]: # How many users in the test set are we not able to make predictions for because of th
          len(set(user_item_test.index) - set(user_item_train.index))
```

```
Out[145]: 662
```

```
In [146]: #How many articles can we make predictions for in the test set?
          len(set(user_item_test.columns) & set(user_item_train.columns))
```

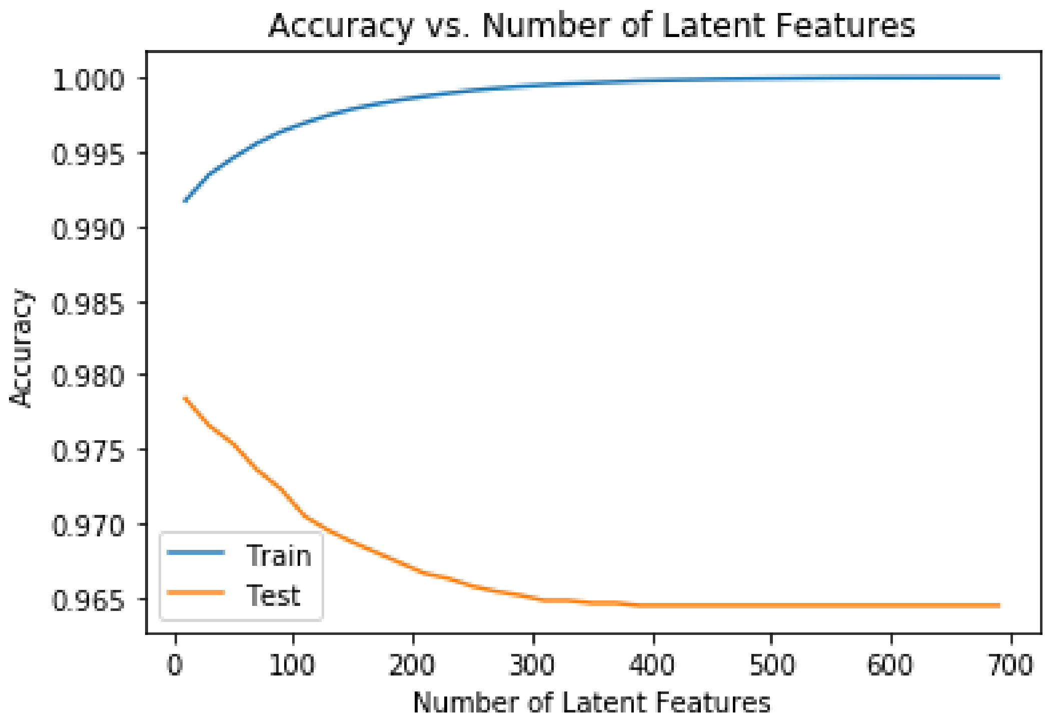
```
Out[146]: 574
```

```
In [147]: #How many articles are we not able to make predictions for because of the cold start p
          len(set(user_item_test.columns) - set(user_item_train.columns))
```

```
Out[147]: 0
```

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.

```
In [148]: # fit SVD on the user_item_train matrix
          u_train, s_train, vt_train = np.linalg.svd(user_item_train) # fit svd similar to above

In [149]: u_train.shape, s_train.shape, vt_train.shape

Out[149]: ((4487, 4487), (714,), (714, 714))

In [151]: common_users = user_item_test.index.isin(user_item_train.index)
          common_arts = user_item_test.columns.isin(user_item_train.columns)
          user_item_test = user_item_test.loc[common_users, common_arts]
          u_test = u_train[user_item_train.index.isin(user_item_test.index), :]
          vt_test = vt_train[:, user_item_train.columns.isin(user_item_test.columns)]

          # decomposition to predict on test data
          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_train, vt_new_train = np.diag(s_train[:k]), u_train[:, :k], vt_train[
              u_new_test, vt_new_test = u_test[:, :k], vt_test[:k, :]

              # take dot product
              user_item_est_train = np.around(np.dot(np.dot(u_new_train, s_new), vt_new_train))
              user_item_est_test = np.around(np.dot(np.dot(u_new_test, s_new), vt_new_test))

              # compute error for each prediction to actual value
              diffs_train = np.subtract(user_item_train, user_item_est_train)
              diffs_test = np.subtract(user_item_test, user_item_est_test)

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


          plt.plot(num_latent_feats, 1 - np.array(sum_errs_train)/user_item_train.size, label='T
          plt.plot(num_latent_feats, 1 - np.array(sum_errs_test)/user_item_test.size, label='Tes
          plt.xlabel('Number of Latent Features');
          plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features');
plt.legend();
```



1.1.5 Part V: Conclusions

Determine if the recommendations we make with any of the above recommendation systems are an improvement to how users currently find articles?

a) The training accuracy increases with the number of latent features, whereas the testing accuracy decreases with the number of latent features. The above graph suggests overfitting as number of latent features increases. Therefore, we can choose a small number of latent features if we want to conitue using this method.

b) Overfitting is a problem here because of a very less number of common users between test and train set. To improve the recommendations, we can use collaborative filtering or content based recommendation.

c) Accuracy is not the right metric to use to represent model performance as our user-tem matrix is imbalanced. In an extreme case, predicting 0 for all user-item pairs would still yield a relatively high accuracy score, but would be a very bad model to trust. It is better to use F1 score to evaluate the performance of the model.

```
In [ ]: # Generating an html file
        from subprocess import call
        call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```

2 Future Work

a) Using FunkSVD: We assumed that the value 0 means that the user didn't like the article but that assumption is not right. It may be the fact that users were unaware of the article. So instead of 0's, NaN's should be used and FunkSVD can be used for the resultant sparse matrix.

b) A/B testing: After hyper-tuning the model in an offline setting, it should be tested online through A/B testing. Users will be divided into control and treatment group. Users in control group won't see new recommendation system and users in treatement group will see this new recommendation system. We can choose click-through rate (CTR) as a metric of success.