Recommendations_with_IBM

June 21, 2020

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

I. Section **??** II. Section **??** III. Section **??** IV. Section **??** V. Section **??** VI. Section **??**

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
        from sklearn.metrics import accuracy_score

        %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]:
```
# Find missing values
print('Number of rows with NaNs in email column:', df.email.isnull().sum())
df[df.email.isnull()]

# So, replace unknown email with "" in email column
df['email'] = df.email.astype(str)
```

```
Number of rows with NaNs in email column: 17
```

In [4]:
```
# Create a dataset - user interactions with articles

user_article = df.groupby('email').agg({'article_id': 'count'})
user_article.columns = ['num_of_articles']
print('user_article shape:', user_article.shape)
user_article.head()
```

```
user_article shape: (5149, 1)
```

Out[4]:

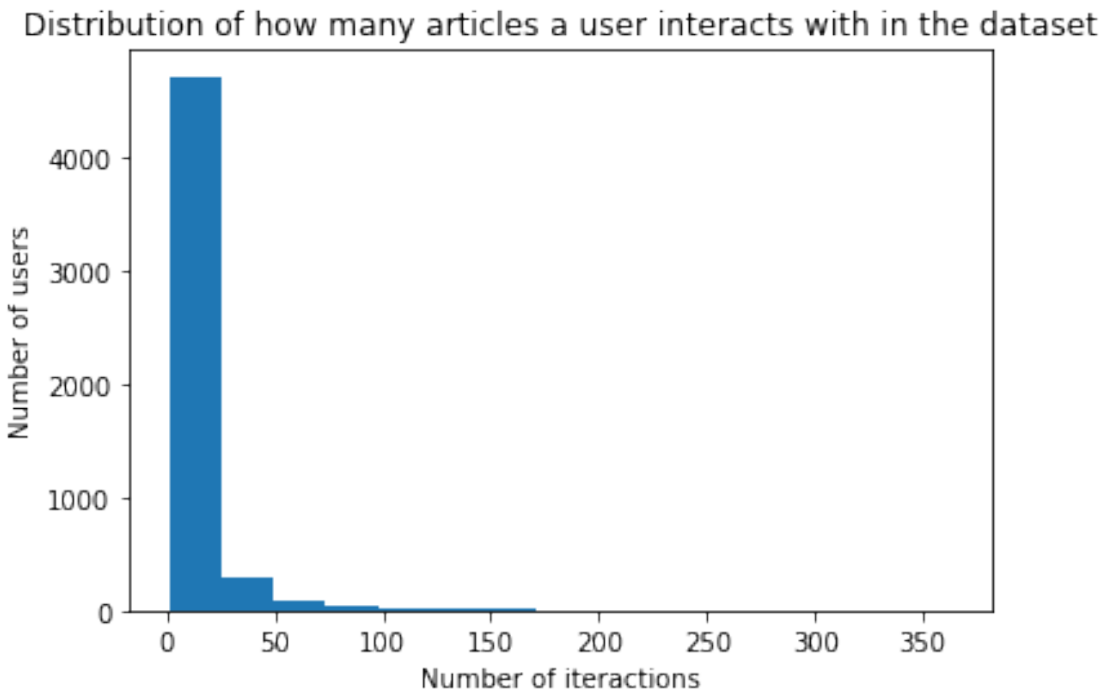
email	num_of_articles
0000b6387a0366322d7fbfc6434af145adf7fed1	13
001055fc0bb67f71e8fa17002342b256a30254cd	4
00148e4911c7e04eeff8def7bbbdaf1c59c2c621	3
001a852ecbd6cc12ab77a785efa137b2646505fe	6
001fc95b90da5c3cb12c501d201a915e4f093290	2

In [5]:
```
# Descriptive statistics to assist with giving a look at the number of times each user i

user_article.describe()
# user_article.describe().loc[['50%','max']] # to answer the following questions (median
```

Out[5]:

	num_of_articles
count	5149.000000
mean	8.932414
std	16.801011
min	1.000000
25%	1.000000
50%	3.000000
75%	9.000000
max	364.000000

In [6]:
```
# Create a histogram

# plt.hist?

plt.hist(user_article.num_of_articles, bins=15)
plt.title('Distribution of how many articles a user interacts with in the dataset')
plt.xlabel('Number of iteractions')
plt.ylabel('Number of users')
plt.show()
```




```
In [7]: # Fill in the median and maximum number of user_article interactios below

        median_val = user_article.describe().loc['50%', 'num_of_articles'] # 50% of individuals
        max_views_by_user = user_article.describe().loc['max', 'num_of_articles'] # The maximum
```

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

```
In [8]: # Find and explore duplicate articles

        df_content[df_content.duplicated(subset='article_id')]

Out[8]:                                                 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 [9]: # Remove any rows that have the same article_id - only keep the first

        df_content.drop_duplicates(subset=['article_id'], keep='first', inplace=True)
        df_content[df_content.duplicated(subset='article_id')]
```

```
Out[9]: Empty DataFrame
        Columns: [doc_body, doc_description, doc_full_name, doc_status, article_id]
        Index: []
```

3. Use the cells below to find:

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

```
In [10]: # Print the answers

         print('a:', df.article_id.unique().shape[0])
         print('b:', df_content.article_id.unique().shape[0])
         print('c:', user_article.shape[0])
         print('d:', df.shape[0])
```

```
a: 714
b: 1051
c: 5149
d: 45993
```

```
In [11]: unique_articles = df.article_id.unique().shape[0] # The number of unique articles that
         total_articles = df_content.article_id.unique().shape[0] # The number of unique article
         unique_users = user_article.shape[0] - 1 # The number of unique users (exclude missing
         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 [12]: # Find the most viewed article_id and how often it was viewed (create a dataset)

         article_user = df.groupby('article_id').agg({'email': 'count'})
         article_user.sort_values('email', ascending=False, inplace=True)
         article_user.head()
```

Out[12]:

article_id	email
1429.0	937
1330.0	927
1431.0	671
1427.0	643
1364.0	627

In [13]:
```
# Find the most viewed article_id and how often it was viewed (answers)

article_idx = article_user.head(1).index[0]
frequency = article_user.loc[article_idx, 'email']
```

In [14]:
```
most_viewed_article_id = article_idx.astype(str) # The most viewed article in the datas
max_views = frequency # The most viewed article in the dataset was viewed how many time
```

In [15]:
```
## No need to change the code here - this will be helpful for later parts of the noteb
# Run this cell to map the user email to a user_id column and remove the email column

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

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

        email_encoded.append(coded_dict[val])
    return email_encoded

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

# show header
df.head()
```

Out[15]:

	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 [16]:
```
## If you stored all your results in the variable names above,
## you shouldn't need to change anything in this cell
```

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

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

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

1.1.2 Part II: Rank-Based Recommendations

Unlike in the earlier lessons, we don't actually have ratings for whether a user liked an article or not. We only know that a user has interacted with an article. In these cases, the popularity of an article can really only be based on how often an article was interacted with.

1. Fill in the function below to return the **n** top articles ordered with most interactions as the top. Test your function using the tests below.

```
In [17]: 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

             '''
             #article_user = df.groupby('article_id').count()
             #article_user.sort_values('user_id', ascending=False, inplace=True)
             #top_articles = df.loc[df.article_id.isin(article_user.index)].title.tolist()[:n]

             article_user = df.groupby('article_id').count().sort_values('user_id', ascending=Fa
             titles = dict(zip(df.article_id, df.title))
             top_articles = [titles[i] for i in article_user][:n]

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

         def get_top_article_ids(n, df=df):
             '''
             INPUT:
```

```
            n - (int) the number of top articles to return
            df - (pandas dataframe) df as defined at the top of the notebook

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

            '''
            top_articles = (df.groupby('article_id').count().sort_values('user_id', ascending=F
                                    .index.astype(str).tolist()[:n])

            return top_articles # Return the top article ids
```

```
In [18]: 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 [19]: # Test your function by returning the top 5, 10, and 20 articles
         top_5 = get_top_articles(5)
         top_10 = get_top_articles(10)
         top_20 = get_top_articles(20)

         # Test each of your three lists from above
         t.sol_2_test(get_top_articles)
```

```
Your top_5 looks like the solution list! Nice job.
Your top_10 looks like the solution list! Nice job.
Your top_20 looks like the solution list! Nice job.
```

1.1.3 Part III: User-User Based Collaborative Filtering

1. Use the function below to reformat the **df** dataframe to be shaped with users as the rows and articles as the columns.

- Each **user** should only appear in each **row** once.
- Each **article** should only show up in one **column**.
- **If a user has interacted with an article, then place a 1 where the user-row meets for that article-column**. It does not matter how many times a user has interacted with the article, all entries where a user has interacted with an article should be a 1.
- **If a user has not interacted with an item, then place a zero where the user-row meets for that article-column**.

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

```
In [20]: # create the user-article matrix with 1's and 0's

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

             OUTPUT:
             user_item - user item matrix

             Description:
             Return a matrix with user ids as rows and article ids on the columns with 1 values
             an article and a 0 otherwise
             '''
             # Fill in the function here
             user_item = df.groupby(['user_id', 'article_id'])['title'].agg(lambda x: 1).unstack
             user_item.fillna(0, inplace=True)
             return user_item # return the user_item matrix

         user_item = create_user_item_matrix(df)

In [21]: ## 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 [22]: def find_similar_users(user_id, user_item=user_item):
             '''
             INPUT:
             user_id - (int) a user_id
             user_item - (pandas dataframe) matrix of users by articles:
                         1's when a user has interacted with an article, 0 otherwise

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

             Description:
```

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

            '''
            # compute similarity of each user to the provided user
            similarity = user_item.dot(user_item.loc[user_id])
            # sort by similarity
            similarity.sort_values(ascending=False, inplace=True)
            # create list of just the ids
            most_similar_users = similarity.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 [23]: # 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 [24]: 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)
             '''
             df = df.drop_duplicates(subset=['article_id'])
             df.set_index('article_id', inplace=True)
             article_ids = [float(i) for i in article_ids]
             article_names = df.loc[article_ids].title
             article_names.tolist()

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


         def get_user_articles(user_id, user_item=user_item):
```

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

    OUTPUT:
    article_ids - (list) a list of the article ids seen by the user
    article_names - (list) a list of article names associated with the list of article
                    (this is identified by the doc_full_name column in df_content)

    Description:
    Provides a list of the article_ids and article titles that have been seen by a user
    '''
    user_articles = user_item.loc[user_id, :]
    article_ids = user_articles[user_articles > 0].index.tolist()
    article_names = get_article_names(article_ids)
    article_ids = [str(i) for i in article_ids]

    return article_ids, article_names # return the ids and names


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

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

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

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

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

    '''
    similar_users = find_similar_users(user_id)
    user_articles, _ = get_user_articles(user_id)

    recs = []
```

```
            for user in similar_users:
                user_ids, _ = get_user_articles(user)
                user_not_seen_articles = list(set(user_ids) - (set(user_articles) & set(user_id
                recs.extend(user_not_seen_articles)

                if len(recs) >= m:
                    break

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

In [25]:
```
# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

Out[25]:
```
article_id
1171.0              apache spark lab, part 2: querying data
1393.0                           the nurse assignment problem
210.0         this week in data science (february 14, 2017)
967.0                       ml algorithm != learning machine
1163.0        analyze open data sets with spark & pixiedust
367.0                       visualising data the node.js way
409.0               using github for project control in dsx
825.0     what is smote in an imbalanced class setting (...
744.0     spark-based machine learning tools for capturi...
996.0     5 practical use cases of social network analyt...
1304.0    gosales transactions for logistic regression m...
720.0     data visualization playbook: telling the data ...
1148.0        airbnb data for analytics: vancouver listings
237.0           deep learning with data science experience
821.0          using rstudio in ibm data science experience
1395.0                           the unit commitment problem
12.0      timeseries data analysis of iot events by usin...
1330.0         insights from new york car accident reports
1160.0        analyze accident reports on amazon emr spark
1162.0             analyze energy consumption in buildings
444.0                         declarative machine learning
1314.0        healthcare python streaming application demo
1432.0          visualize data with the matplotlib library
952.0         why even a moths brain is smarter than an ai
1101.0          airbnb data for analytics: mallorca reviews
1354.0    movie recommender system with spark machine le...
1366.0    process events from the watson iot platform in...
1172.0          apache spark lab, part 3: machine learning
1276.0           deploy your python model as a restful api
299.0                                    brunel in jupyter
                                ...
969.0     flightpredict ii: the sequel   ibm watson dat...
617.0     pixiedust gets its first community-driven feat...
124.0        python machine learning: scikit-learn tutorial
```

```
812.0          machine learning exercises in python, part 1
29.0                          experience iot with coursera
1298.0     from scikit-learn model to cloud with wml client
843.0     aspiring data scientists! start to learn stati...
50.0                          graph-based machine learning
131.0               simple graphing with ipython andăpandas
120.0      a dynamic duo  inside machine learning  medium
28.0      deep forest: towards an alternative to deep ne...
2.0              this week in data science (april 18, 2017)
721.0              the power of machine learning in spark
283.0     twelveăways to color a map of africa using brunel
1165.0                          analyze precipitation data
221.0     how smart catalogs can turn the big data flood...
108.0     520    using notebooks with pixiedust for fast...
1299.0     from spark ml model to online scoring with scala
295.0                         awesome deep learning papers
33.0               using brunel in ipython/jupyter notebooks
1367.0    programmatic evaluation using watson conversation
510.0              this week in data science (may 30, 2017)
16.0      higher-order logistic regression for large dat...
1164.0        analyze open data sets with pandas dataframes
1357.0    overlapping co-cluster recommendation algorith...
1025.0             data tidying in data science experience
693.0     better together: spss and data science experience
302.0                     accelerate your workflow with dsx
641.0     perform sentiment analysis with lstms, using t...
517.0                   shaping data with ibm data refinery
Name: title, Length: 118, dtype: object
```

```
In [26]: # 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 [27]: def get_top_sorted_users(user_id, df=df, user_item=user_item):
             '''
             INPUT:
             user_id - (int)
             df - (pandas dataframe) df as defined at the top of the notebook
             user_item - (pandas dataframe) matrix of users by articles:
                     1's when a user has interacted with an article, 0 otherwise


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

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

             '''
             # create similarity dataframe with user_ids and their similarities, except given us
             similarity = user_item.dot(user_item.loc[user_id])
             similarity.sort_values(ascending=False, inplace=True)
             similarity.drop(user_id, axis=0, inplace=True)
             similarity = similarity.reset_index()

             # create interactions dataframe with user_ids and their number of interactions, exc
             interactions = df.groupby('user_id').article_id.count().drop(user_id, axis=0).reset

             # merge similarity and interactions datasets
             neighbors_df = pd.merge(similarity, interactions, on='user_id')
             neighbors_df.columns = ['neighbor_id', 'similarity', 'num_interactions']
             neighbors_df.sort_values(['similarity', 'num_interactions'], ascending=False, inpla

             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.

        '''
        neighbours = get_top_sorted_users(user_id).neighbor_id.tolist()
        user_articles, _ = get_user_articles(user_id)

        recs = []

        for user in neighbours:
            user_ids, _ = get_user_articles(user)
            user_not_seen_articles = list(set(user_ids) - (set(user_articles) & set(user_id
            recs.extend(user_not_seen_articles)

            if len(recs) >= m:
                break

        recs = recs[:m]
        rec_names = get_article_names(recs)

        return recs, rec_names
```

In [28]:
```
# 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:
['1368.0', '651.0', '1411.0', '1163.0', '911.0', '1386.0', '981.0', '1304.0', '12.0', '555.0']

The top 10 recommendations for user 20 are the following article names:
article_id
1368.0             putting a human face on machine learning
651.0            analyzing streaming data from kafka topics
1411.0                              uci: white wine quality
```

```
1163.0        analyze open data sets with spark & pixiedust
911.0      using machine learning to predict baseball inj...
1386.0                              small steps to tensorflow
981.0                   super fast string matching in python
1304.0     gosales transactions for logistic regression m...
12.0       timeseries data analysis of iot events by usin...
555.0               build a naive-bayes model with wml & dsx
Name: title, dtype: object
```

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 [29]: ### Tests with a dictionary of results

         # Find the user that is most similar to user 1
         user1_most_sim = get_top_sorted_users(1).head(1).neighbor_id.values[0]
         # Find the 10th most similar user to user 131
         user131_10th_sim = get_top_sorted_users(131).neighbor_id.values[9]
```

```
In [30]: ## 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.

Probably, using items (articles) or users similarity algorithms wouldn't work for new users well. And only after some time, for example knowledge-based recommendation systems will provide a miningfull results. So, on the very begining I would suggest using the most interacted articles (or highly ranked movies, etc.) among currect users.

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 [31]: new_user = '0.0'

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

```
In [32]: assert set(new_user_recs) == set(['1314.0','1429.0','1293.0','1427.0','1162.0','1364.0'

         print("That's right!  Nice job!")
```

```
That's right!  Nice job!
```

1.1.4 Part IV: Content Based Recommendations (EXTRA - NOT REQUIRED)

Another method we might use to make recommendations is to perform a ranking of the highest ranked articles associated with some term. You might consider content to be the **doc_body**, **doc_description**, or **doc_full_name**. There isn't one way to create a content based recommendation, especially considering that each of these columns hold content related information.

1. Use the function body below to create a content based recommender. Since there isn't one right answer for this recommendation tactic, no test functions are provided. Feel free to change the function inputs if you decide you want to try a method that requires more input values. The input values are currently set with one idea in mind that you may use to make content based recommendations. One additional idea is that you might want to choose the most popular recommendations that meet your 'content criteria', but again, there is a lot of flexibility in how you might make these recommendations.

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

```
In [33]: def make_content_recs():
             '''
             INPUT:

             OUTPUT:

             '''
```

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

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

Write an explanation of your content based recommendation system here.

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

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

```
In [34]: # make recommendations for a brand new user
```

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

1.1.8 Part V: Matrix Factorization

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

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

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

In [36]:
```
# quick look at the matrix
user_item_matrix.head()
```

Out[36]:

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.

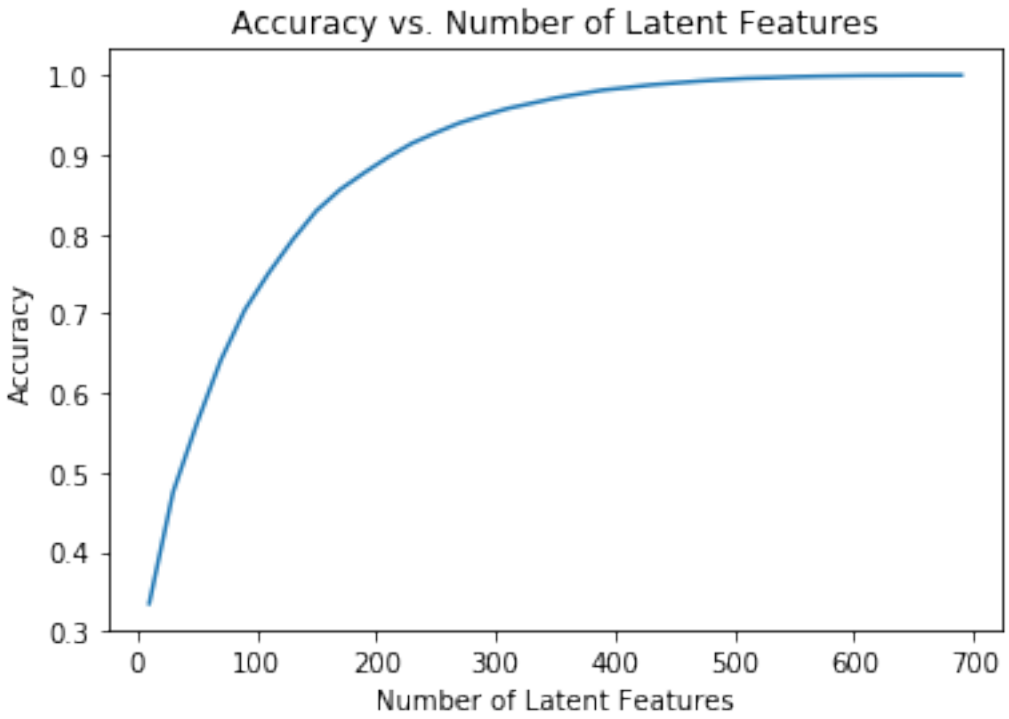
In [37]:
```
# Perform SVD on the User-Item Matrix Here

u, s, vt = np.linalg.svd(user_item_matrix) # use the built in to get the three matrices
print(s.shape, u.shape, vt.shape)
```

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

In this situation we have only two values in user-item matrix - 1 represents whether user has interacted with an article and 0 represents whether user hasn't interacted with an article. While in the lesson we had ratings from 0 to 10 which represent interaction, and missing values represent non-interection. SVD can't be used with DataFrame with missing values, so in the lesson we used FunkSVD or just droped missing values. In this project we can use SVD to make recommendations.

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 [38]: 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 [39]: 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 = np.array(user_item_test.index)
        test_arts = np.array(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 [40]:
```
    #1
    print(len(user_item_test.index & user_item_train.index))
    #2
    print(len(set(user_item_test.index) - set(user_item_train.index)))
    #3
    print(test_arts.shape[0])
    #4
    print(len(set(user_item_test.columns) - set(user_item_train.columns)))
```

```
20
662
574
0
```

In [41]:
```
    # Replace the values in the dictionary below
    a = 662
    b = 574
    c = 20
    d = 0


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

    t.sol_4_test(sol_4_dict)
```

```
Awesome job!  That's right!  All of the test movies are in the training data, but there are only
```

5. Now use the **user_item_train** dataset from above to find U, S, and V transpose using SVD. Then find the subset of rows in the **user_item_test** dataset that you can predict using this matrix decomposition with different numbers of latent features to see how many features makes sense to keep based on the accuracy on the test data. This will require combining what was done in questions 2 - 4.

Use the cells below to explore how well SVD works towards making predictions for recommendations on the test data.

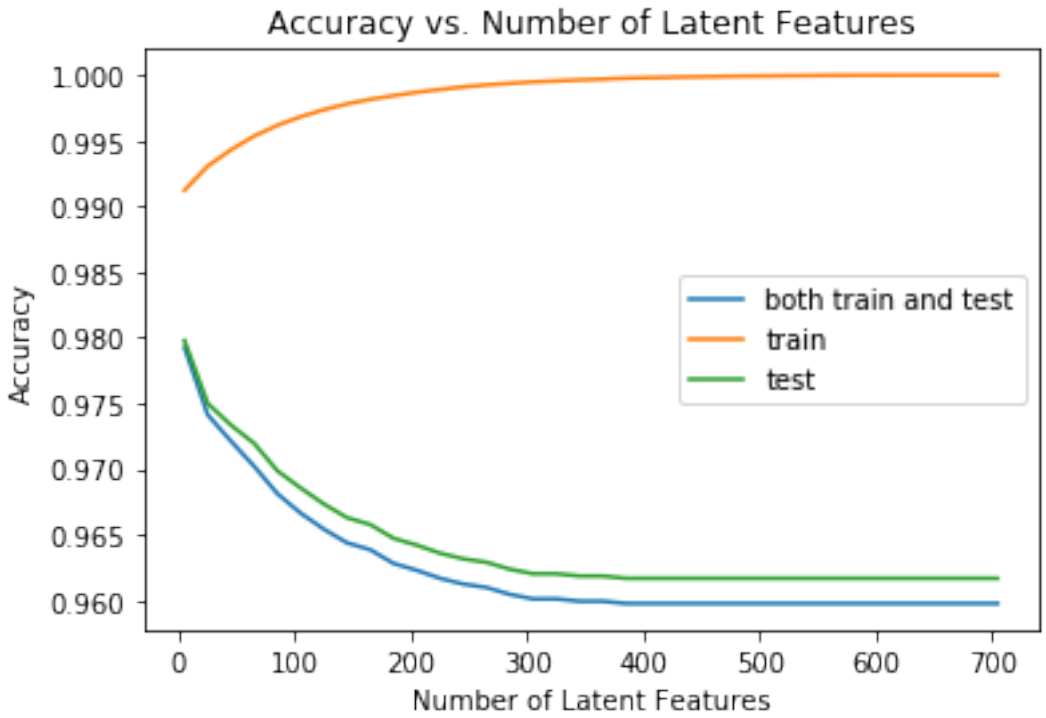
In [42]:
```
# fit SVD on the user_item_train matrix

u_train, s_train, vt_train = np.linalg.svd(user_item_train) # fit svd similar to above
print(u_train.shape, s_train.shape, vt_train.shape)
```

```
(4487, 4487) (714,) (714, 714)
```

In [43]:
```
# Use only data about 20 users for whom we can make predictions in the test set

# train user_ids
train_idx = user_item_train.index

# 20 common user_ids and article_ids
common_idx = list(set(train_idx) & set(test_idx))

# 20 common train user_ids and article_ids
train_common_idx = user_item_train.index.isin(test_idx)
train_common_col = user_item_train.columns.isin(test_arts)

# create test u and vt arrays
u_test = u_train[train_common_idx, :]
vt_test= vt_train[:, train_common_col]

# subset user_item_test dataset
user_item_test = user_item_test.loc[common_idx]
```

In [44]:
```
# Use these cells to see how well you can use the training decomposition to predict on

num_latent_feats = np.arange(5,700+10,20)
sum_errs, train_errs, test_errs = [] ,[], []

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

    # take dot product
    user_item_est_train = np.around(np.dot(np.dot(u_train_new, s_train_new), vt_train_n
    user_item_est_test = np.around(np.dot(np.dot(u_test_new, s_train_new), vt_test_new)
```

```
        # compute error for each prediction to actual value
        diffs_train = np.subtract(user_item_train, user_item_est_train)
        diffs_test = np.subtract(user_item_test.loc[common_idx, :], 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)))
        err_all = 1 - ((np.sum(user_item_est_test) + np.sum(np.sum(user_item_test))) /
                       (user_item_test.shape[0] * user_item_test.shape[1]))

        train_errs.append(err_train)
        test_errs.append(err_test)
        sum_errs.append(err_all)

    # Plot test, train, and both, train and test, accuracy scores
    plt.plot(num_latent_feats, sum_errs, label='both train and test');
    plt.plot(num_latent_feats, 1-np.array(train_errs)/(user_item_train.shape[0]*user_item_t
    plt.plot(num_latent_feats, 1-np.array(test_errs)/(user_item_test.shape[0]*user_item_tes
    plt.xlabel('Number of Latent Features');
    plt.ylabel('Accuracy');
    plt.title('Accuracy vs. Number of Latent Features')
    plt.legend();
```



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?

First of all, from the figure above we can observe an overfitting of our model. Since our dataset (common users from test and train data) is too small, it's quite expected results.

Second, nevertheles, we don't have big enough data to build our recommendation model, the results on test data are high (> 96%). This means that we could try to increase a dataset to look at performance. The other way, when we still won't have a big dataset, cross validation could be used to avoid oberfitting in our data. After 350 latent features the accuracy of our model remain the same, so in future research it will also need to be paid attention to.

```
In [45]: from subprocess import call
         call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```

```
Out[45]: 0
```