Recommendations_with_IBM

December 30, 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. Exploratory Data Analysis II. Rank Based Recommendations III. User-User Based Collaborative Filtering IV. Content Based Recommendations (EXTRA - NOT REQUIRED) V. Matrix Factorization VI. Extras And Concluding

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.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import project_tests as t
import pickle

%matplotlib inline

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

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

[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
```

[2]:
```
print("user-item-interactions shape: ", df.shape)
print("user-item-interactions unique emails", df.email.nunique())
print("user-item-interactions unique articles", df.article_id.nunique())
```

```
user-item-interactions shape:  (45993, 3)
user-item-interactions unique emails 5148
user-item-interactions unique articles 714
```

[3]:
```
# show users with null email
df[df.email.isnull()]
```

[3]:

	article_id	title	email
25131	1016.0	why you should master r (even if it might even…	NaN
29758	1393.0	the nurse assignment problem	NaN
29759	20.0	working interactively with rstudio and noteboo…	NaN
29760	1174.0	breast cancer wisconsin (diagnostic) data set	NaN
29761	62.0	data visualization: the importance of excludin…	NaN
35264	224.0	using apply, sapply, lapply in r	NaN
35276	961.0	beyond parallelize and collect	NaN
35277	268.0	sector correlations shiny app	NaN
35278	268.0	sector correlations shiny app	NaN
35279	268.0	sector correlations shiny app	NaN
35280	268.0	sector correlations shiny app	NaN
35281	415.0	using machine learning to predict value of hom…	NaN
35282	846.0	pearson correlation aggregation on sparksql	NaN
35283	268.0	sector correlations shiny app	NaN
35284	162.0	an introduction to stock market data analysis …	NaN
42749	647.0	getting started with apache mahout	NaN
42750	965.0	data visualization playbook: revisiting the ba…	NaN

[4]:
```
# Show df_content to get an idea of the data
df_content.head()
```

[4]:
```
                                        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  Here's this week's 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
```

[5]:
```
print("\narticles_community shape: ", df_content.shape)
print("\narticles_community columns containing nulls:\n", df_content.isnull().
 ↪mean())
print("\narticles_community doc status' possible values:\n",␣
 ↪df_content['doc_status'].value_counts())
```

```
articles_community shape:  (1056, 5)

articles_community columns containing nulls:
 doc_body           0.013258
doc_description    0.002841
doc_full_name      0.000000
doc_status         0.000000
article_id         0.000000
dtype: float64

articles_community doc status' possible values:
 Live    1056
Name: doc_status, dtype: int64
```

[6]:
```
# show articles where the body or description is null
df_content[df_content.doc_body.isnull() | df_content.doc_description.isnull()]
```

[6]:
```
                                              doc_body  \
206                                                NaN
276                                                NaN
354   The search index lets you create flexible quer…
484                                                NaN
```

```
508                                                    NaN
540                                                    NaN
638                                                    NaN
667                                                    NaN
706                                                    NaN
768   Compose The Compose logo Articles Sign in Free…
842                                                    NaN
876                                                    NaN
889                                                    NaN
919   Cloudant Query is a powerful declarative JSON …
947                                                    NaN
1037                                                   NaN
1054                                                   NaN

                                        doc_description  \
206   Watch how to convert XML data to CSV format to…
276   Love to work in Microsoft Excel? Watch how to …
354                                                    NaN
484   See how to evaluate and convert your DDL and S…
508   Watch how to generate SQL-based reports for Cl…
540   Need to move some data to the cloud for wareho…
638   See how to create a new dashDB instance and po…
667   See how to connect dashDB, as a source and tar…
706   Aginity Workbench is a free application known …
768                                                    NaN
842   Learn how to configure a dashDB connection in …
876   See how to populate data into a table in your …
889   Watch how to apply association rules using R t…
919                                                    NaN
947   Watch how to extract and export dashDB data to…
1037  See how quick and easy it is to set up a dashD…
1054  Learn how to use IBM dashDB as data store for …

                                      doc_full_name doc_status  article_id
206                          Load XML data into dashDB       Live         206
276                        Integrate dashDB with Excel       Live         276
354                  Build the search index in Cloudant       Live         354
484       Convert IBM Puredata for Analytics to dashDB       Live         483
508   Use dashDB with IBM Embeddable Reporting Service       Live         507
540                  Convert data from Oracle to dashDB       Live         539
638       Load JSON from Cloudant database into dashDB       Live         637
667             Integrate dashDB and Informatica Cloud       Live         666
706               Use Aginity Workbench for IBM dashDB       Live         704
768         Announcing the Data Browser for JanusGraph       Live         765
842    Leverage dashDB in Cognos Business Intelligence       Live         839
876             Load data from the desktop into dashDB       Live         873
889   Perform market basket analysis using dashDB and R       Live         886
```

```
919                          Use the new Cloudant query       Live          916
947        Extract and export dashDB data to a CSV file       Live          944
1037                Get started with dashDB on Bluemix       Live         1032
1054                             Use dashDB with Spark       Live         1049
```

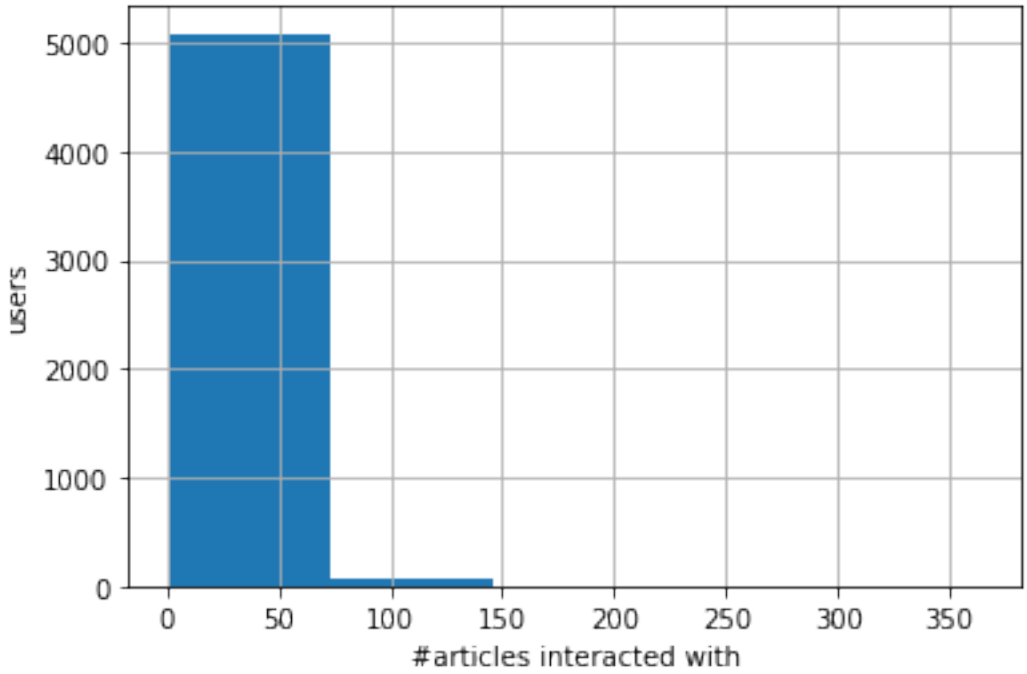
1.1.1 Part I : Exploratory Data Analysis

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.

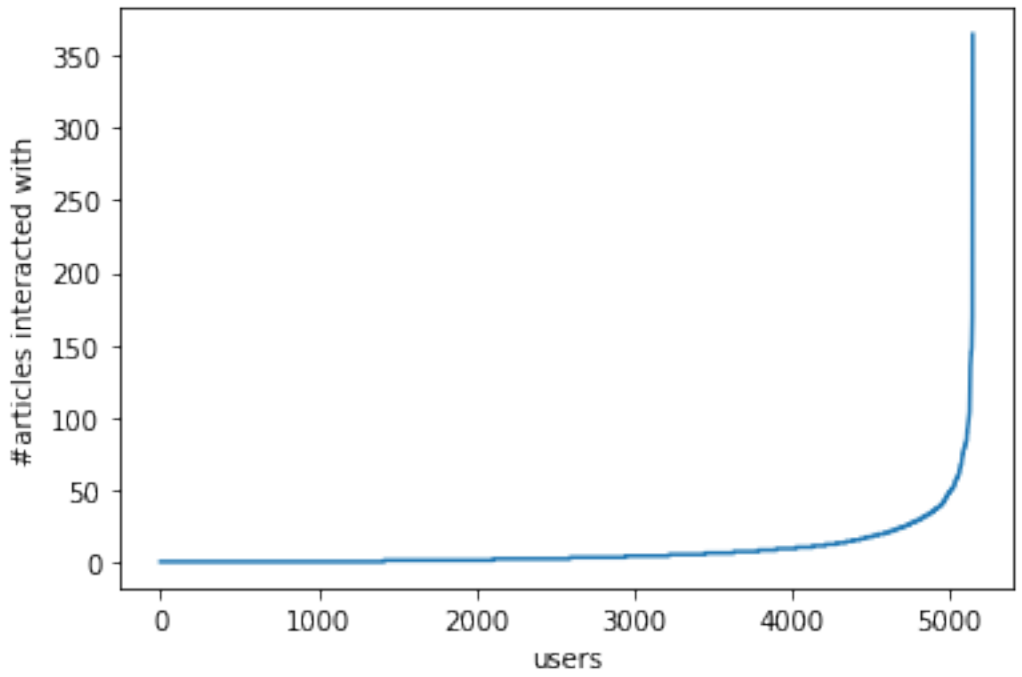
[7]:
```
df.email.value_counts().hist(bins=5)
plt.xlabel("#articles interacted with")
plt.ylabel("users")
```

[7]:
```
Text(0, 0.5, 'users')
```



[8]:
```
article_by_email = df.groupby(by=['email'], dropna=True).article_id.count().
 ↪sort_values()
article_by_email.index = range(len(article_by_email))
article_by_email.plot()
plt.xlabel("users")
plt.ylabel("#articles interacted with")
```

```
plt.show()
```



[9]:
```
print(article_by_email.describe())
print(article_by_email.median())
```

```
count    5148.000000
mean        8.930847
std        16.802267
min         1.000000
25%         1.000000
50%         3.000000
75%         9.000000
max       364.000000
Name: article_id, dtype: float64
3.0
```

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

median_val = article_by_email.median() # 50% of individuals interact with _3_␣
 ↪number of articles or fewer.
max_views_by_user =  article_by_email.max() # The maximum number of␣
 ↪user-article interactions by any 1 user is _364_.
```

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

[11]:
```
# Find and explore duplicate articles
print("number ofduplicated articles:", df_content.duplicated(['article_id']).
 ↪sum())
df_content[df_content.duplicated(['article_id'])]
```

```
number ofduplicated articles: 5
```

[11]:

	doc_body	doc_description	doc_full_name	doc_status	article_id
365	Follow Sign in / Sign up Home About Insight Da…	During the seven-week Insight Data Engineering…	Graph-based machine learning	Live	50
692	Homepage Follow Sign in / Sign up Homepage * H…	One of the earliest documented catalogs was co…	How smart catalogs can turn the big data flood…	Live	221
761	Homepage Follow Sign in Get started Homepage *…	Today's world of data science leverages data f…	Using Apache Spark as a parallel processing fr…	Live	398
970	This video shows you how to construct queries …	This video shows you how to construct queries …	Use the Primary Index	Live	577
971	Homepage Follow Sign in Get started * Home\r\n…	If you are like most data scientists, you are …	Self-service data preparation with IBM Data Re…	Live	232

[12]:
```
# Remove any rows that have the same article_id - only keep the first
df_content.drop_duplicates(subset='article_id', keep='first', inplace=True)
print("number ofduplicated articles after dropping duplicates:", df_content.
 ↪duplicated(['article_id']).sum())
```

```
number ofduplicated articles after dropping duplicates: 0
```

3. Use the cells below to find:

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

[13]:
```
print()
df[df.email.notnull()].email.nunique()
```

[13]: 5148

[14]:
```
print("a. The number of unique articles that have an interaction with a user:␣
 ↪",df.article_id.nunique())
print("b. The number of unique articles in the dataset (whether they have any␣
 ↪interactions or not):", df_content.article_id.nunique())
print("c. The number of unique users in the dataset. (excluding null values):",␣
 ↪df.email.nunique(dropna=True) )
print("d. The number of user-article interactions in the dataset:", df.shape[0])
```

```
a. The number of unique articles that have an interaction with a user:  714
b. The number of unique articles in the dataset (whether they have any
interactions or not): 1051
c. The number of unique users in the dataset. (excluding null values): 5148
d. The number of user-article interactions in the dataset: 45993
```

[15]:
```
unique_articles = df.article_id.nunique() # The number of unique articles that␣
 ↪have at least one interaction
total_articles = df_content.article_id.nunique() # The number of unique␣
 ↪articles on the IBM platform
unique_users = df.email.nunique(dropna=True) # The number of unique users
user_article_interactions = df.shape[0] # The number of user-article␣
 ↪interactions
```

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

[16]:
```
most_viewed_article_id = str(df.article_id.value_counts(sort=True).index[0]) #␣
 ↪The most viewed article in the dataset as a string with one value following␣
 ↪the decimal
max_views = df.article_id.value_counts(sort=True).iloc[0] # The most viewed␣
 ↪article in the dataset was viewed how many times?
print("most viewed article id", most_viewed_article_id)
print("most views article was viewed : ", max_views)
```

```
most viewed article id 1429.0
most views article was viewed :  937
```

[17]:
```
## No need to change the code here - this will be helpful for later parts of␣
 ↪the notebook
# 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()
```

[17]:

	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	user_id
0	ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7	1
1	083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b	2
2	b96a4f2e92d8572034b1e9b28f9ac673765cd074	3
3	06485706b34a5c9bf2a0ecdac41daf7e7654ceb7	4
4	f01220c46fc92c6e6b161b1849de11faacd7ccb2	5

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

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

[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().head(n).index
    return list(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 ids
    '''
    return list(df.article_id.value_counts(sort=True).head(n).index) # Return␣
 ↪the top article ids
```

[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', 'visualize car data with brunel', 'use xgboost, scikit-learn
& ibm watson machine learning apis', 'predicting churn with the spss random tree
algorithm', 'healthcare python streaming application demo', 'finding optimal
```

```
locations of new store using decision optimization', 'apache spark lab, part 1:
basic concepts', 'analyze energy consumption in buildings', 'gosales
transactions for logistic regression model']
[1429.0, 1330.0, 1431.0, 1427.0, 1364.0, 1314.0, 1293.0, 1170.0, 1162.0, 1304.0]
```

[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. 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.

[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 where a user interacted with
    an article and a 0 otherwise
    '''
```

```
    # Fill in the function here
    user_item = df.groupby(by=['user_id','article_id']).title.count().unstack()
    user_item = user_item.applymap(lambda v: 0 if v!=v else 1)
    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
user_item.head()
```

[22]:
```
article_id  0.0     2.0     4.0     8.0     9.0     12.0    14.0    15.0    \
user_id
1                0       0       0       0       0       0       0       0
2                0       0       0       0       0       0       0       0
3                0       0       0       0       0       1       0       0
4                0       0       0       0       0       0       0       0
5                0       0       0       0       0       0       0       0

article_id  16.0    18.0    …  1434.0  1435.0  1436.0  1437.0  1439.0  \
user_id                     …
1                0       0  …       0       0       1       0       1
2                0       0  …       0       0       0       0       0
3                0       0  …       0       0       1       0       0
4                0       0  …       0       0       0       0       0
5                0       0  …       0       0       0       0       0

article_id  1440.0  1441.0  1442.0  1443.0  1444.0
user_id
1                0       0       0       0       0
2                0       0       0       0       0
3                0       0       0       0       0
4                0       0       0       0       0
5                0       0       0       0       0

[5 rows x 714 columns]
```

[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 matrix doesn't look right."
assert user_item.shape[1] == 714, "Oops!  The number of articles in the␣
 ↪user-article matrix doesn't look right."
assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by␣
 ↪user 1 doesn't look right."
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.

[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 users)
                    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
    number_of_same_viewed_articles = user_item.dot(user_item.loc[user_id])
    # sort by similarity
    number_of_same_viewed_articles.sort_values(ascending=False, inplace=True)
    # create list of just the ids
    sorted_ids = list(number_of_same_viewed_articles.index)
    # remove the own user's id
    sorted_ids.remove(user_id)
    return sorted_ids # return a list of the users in order from most to least␣
 ↪similar
```

[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)[:5]))
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.

[26]:
```
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 ids
                    (this is identified by the title column)
    '''
    article_names = [df[df.article_id.isin([article_id])].title.iloc[0] for␣
 ↪article_id in article_ids]
    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 ids
                    (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
    '''

    marked_article_ids = user_item.loc[user_id]
    article_ids = marked_article_ids[marked_article_ids == 1].index.astype(str).
 ↪to_list()
    return article_ids, get_article_names(article_ids) # 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 recs
    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

    '''
    user_article_ids = set(get_user_articles(user_id)[0])
    recs = []
    for similar_user_id in find_similar_users(user_id):
        similar_article_ids = set(get_user_articles(similar_user_id)[0])
        recs.extend(similar_article_ids-user_article_ids)
        if len(recs) > m:
            break;
    return recs[:m] # return your recommendations for this user_id
```

[27]:
```
# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

[27]:
```
['aspiring data scientists! start to learn statistics with these 6 books!',
 'pixiedust 1.0 is here! - ibm watson data lab',
 'programmatic evaluation using watson conversation',
 '502    forgetting the past to learn the future: long …\nName: title, dtype:
object',
 'use decision optimization to schedule league games',
 'the unit commitment problem',
 'deep forest: towards an alternative to deep neural networks',
 'awesome deep learning papers',
 'got zip code data? prep it for analytics. - ibm watson data lab - medium',
 'insights from new york car accident reports']
```

[28]:
```
# 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'])) == set(['using deep learning to reconstruct high-resolution␣
↪audio', 'build a python app on the streaming analytics service', 'gosales␣
↪transactions for naive bayes model', 'healthcare python streaming␣
↪application demo', 'use r dataframes & ibm watson natural language␣
↪understanding', 'use xgboost, scikit-learn & ibm watson machine learning␣
↪apis']), "Oops! Your the get_article_names function doesn't work quite how␣
↪we expect."
assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing␣
↪(2015): united states demographic measures','self-service data preparation␣
↪with ibm data refinery','use the cloudant-spark connector in python␣
↪notebook']), "Oops! Your the get_article_names function doesn't work quite␣
↪how we expect."
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 measures', 'self-service data preparation with ibm data␣
↪refinery','use the cloudant-spark connector in python notebook'])
assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1314.
↪0', '1422.0', '1427.0'])
assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct␣
↪high-resolution audio', 'build a python app on the streaming analytics␣
↪service', 'gosales transactions for naive bayes model', 'healthcare python␣
↪streaming application demo', 'use r dataframes & ibm watson natural language␣
↪understanding', 'use xgboost, scikit-learn & ibm watson machine learning␣
↪apis'])
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.

[29]:
```
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 user_id
                    num_interactions - the number of articles viewed by the␣
↪user - if a u

    Other Details - sort the neighbors_df by the similarity and then by number␣
↪of interactions where
                    highest of each is higher in the dataframe

    '''
    number_of_same_viewed_articles = user_item.dot(user_item.loc[user_id])
    similarity = number_of_same_viewed_articles.sort_values(ascending=False).
↪drop(user_id).to_frame(name='similarity')
    num_interactions = df.groupby(by=['user_id']).article_id.count().
↪drop(user_id).to_frame(name='num_interactions')
    neighbors_df = similarity.merge(num_interactions, left_on='user_id',␣
↪right_on='user_id')
    #neighbors_df.reset_index(level=0, inplace=True)
    neighbors_df.sort_values(by=['similarity','num_interactions'],␣
↪ascending=False, inplace=True)
    neighbors_df['neighbor_id'] = neighbors_df.index
    return neighbors_df[['neighbor_id', 'similarity', 'num_interactions']] #␣
↪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 recs
    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.

    '''
    user_article_ids = set(get_user_articles(user_id)[0])
    neighbors_df = get_top_sorted_users(user_id)
    recs = []
    for similar_user_id in neighbors_df['neighbor_id'].values:
        similar_article_ids = set(get_user_articles(similar_user_id)[0])
        recs.extend(similar_article_ids-user_article_ids)
        if len(recs) > m:
            break;
    rec_names = get_article_names(recs)
    return recs, rec_names
```

[30]:
```
neighbors_df = get_top_sorted_users(1)
neighbors_df.head()
```

[30]:

user_id	neighbor_id	similarity	num_interactions
3933	3933	35	45
23	23	17	364
3782	3782	17	363
203	203	15	160
4459	4459	15	158

[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:
['939.0', '1351.0', '1367.0', '1296.0', '164.0', '981.0', '1154.0', '1163.0',
'1331.0', '1409.0', '1354.0', '1153.0', '1330.0', '1024.0', '1411.0', '1368.0',
'1335.0', '880.0', '1364.0', '1338.0', '1324.0', '1391.0', '761.0', '1396.0',
'1336.0', '302.0', '1276.0', '336.0', '1152.0', '1433.0', '1420.0', '1329.0',
'142.0', '555.0', '686.0', '1085.0', '1150.0', '1360.0', '730.0', '1278.0',
'465.0', '1410.0', '681.0', '1151.0', '1346.0', '1427.0', '1304.0', '911.0',
'1424.0', '12.0', '793.0', '1160.0', '1170.0', '1157.0', '1172.0', '1166.0',
'1407.0', '109.0', '1386.0', '1426.0', '1357.0', '1176.0', '362.0', '1162.0',
'651.0', '125.0', '205.0', '1444.0', '1356.0']
```

```
The top 10 recommendations for user 20 are the following article names:
['deep learning from scratch i: computational graphs', 'model bike sharing data
with spss', 'programmatic evaluation using watson conversation', 'fortune 100
companies', 'learn tensorflow and deep learning together and now!', 'super fast
string matching in python', 'airbnb data for analytics: vienna listings',
'analyze open data sets with spark & pixiedust', 'intentional homicide, number
and rate per 100,000 population, by country', 'uci: red wine quality', 'movie
recommender system with spark machine learning', 'airbnb data for analytics:
vienna calendar', 'insights from new york car accident reports', 'using deep
learning to reconstruct high-resolution audio', 'uci: white wine quality',
'putting a human face on machine learning', 'labor', 'probabilistic graphical
models tutorial\u200a-\u200apart 1 - stats and bots', 'predicting churn with the
spss random tree algorithm', 'ml optimization using cognitive assistant', 'ibm
watson facebook posts for 2015', 'sudoku', 'variational auto-encoder for "frey
faces" using keras', 'times world university ranking analysis', 'learn basics
about notebooks and apache spark', 'accelerate your workflow with dsx', 'deploy
your python model as a restful api', 'challenges in deep learning', 'airbnb data
for analytics: venice reviews', 'visualize the 1854 london cholera outbreak',
'use apache systemml and spark for machine learning', 'ingest data from message
hub in a streams flow', 'neural networks for beginners: popular types and
applications', 'build a naive-bayes model with wml & dsx', 'score a predictive
model built with ibm spss modeler, wml & dsx', 'airbnb data for analytics:
chicago listings', 'airbnb data for analytics: venice calendar', 'pixieapp for
outlier detection', 'developing for the ibm streaming analytics service',
'develop a scala spark model on chicago building violations', 'introduction to
neural networks, advantages and applications', 'uci: sms spam collection',
'real-time sentiment analysis of twitter hashtags with spark (+ pixiedust)',
'airbnb data for analytics: venice listings', 'military expenditure as % of gdp
by country', 'use xgboost, scikit-learn & ibm watson machine learning apis',
'gosales transactions for logistic regression model', 'using machine learning to
predict baseball injuries', 'use spark for python to load data and run sql
queries', 'timeseries data analysis of iot events by using jupyter notebook',
'10 powerful features on watson data platform, no coding necessary', 'analyze
accident reports on amazon emr spark', 'apache spark lab, part 1: basic
concepts', 'airbnb data for analytics: washington d.c. listings', 'apache spark
lab, part 3: machine learning', 'analyzing data by using the sparkling.data
library features', 'uci: poker hand - testing data set', 'tensorflow quick
tips', 'small steps to tensorflow', 'use spark for scala to load data and run
sql queries', 'overlapping co-cluster recommendation algorithm (ocular)', 'build
a python app on the streaming analytics service', 'dsx: hybrid mode', 'analyze
energy consumption in buildings', 'analyzing streaming data from kafka topics',
'statistics for hackers', "a beginner's guide to variational methods",
'worldwide fuel oil consumption by household (in 1000 metric tons)', 'occupation
(2015): united states demographic measures']
```

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.

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

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

[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?

Answer: A new user introduces the cold start problem where we have no historic data. In this case we can't use the collaborative filtering approach and find similar users, but we can "fall back" to the rank-based approach and just present the user the articles users have interacted with most. The `get_top_articles` function from above is a good candidate.

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.

[34]:
```
new_user = '0.0'

# What would your recommendations be for this new user '0.0'?  As a new user,␣
↪they have no observed articles.
# Provide a list of the top 10 article ids you would give to
new_user_recs = list(map(str,get_top_article_ids(10)))
```

[35]:
```
assert set(new_user_recs) == set(['1314.0','1429.0','1293.0','1427.0','1162.
↪0','1364.0','1304.0','1170.0','1431.0','1330.0']), "Oops!  It makes sense␣
↪that in this case we would want to recommend the most popular articles,␣
↪because we don't know anything about these users."

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.

[36]:
```
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.

[37]:
```
# 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.

[38]:
```
# Load the matrix here
user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

[39]:
```
# quick look at the matrix
user_item_matrix.head()
```

[39]:
```
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.

[40]:
```
# Perform SVD on the User-Item Matrix Here

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

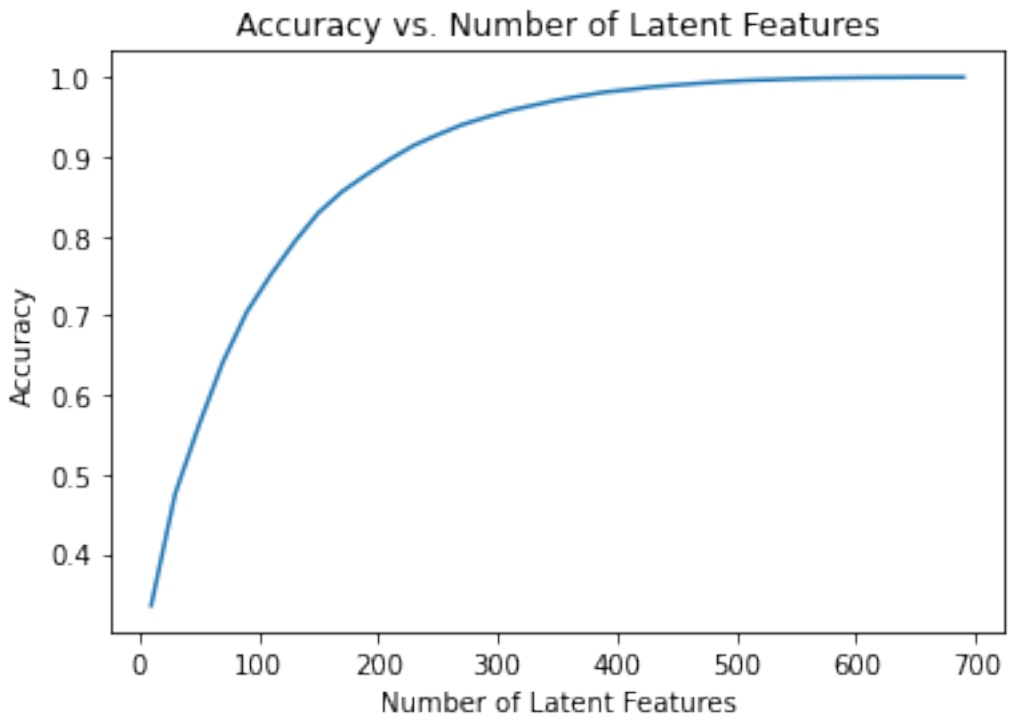
[40]: ((5149, 5149), (714,), (714, 714))

Answer: In the lessons the matrix values had a different coding scheme. They were actual rating numbers. A zero might have been a valid rating and missing ratings were encoded with NaN-values. In this case we could not even use SVD, because of the missing values and converting NaN to zero might lead to false conclusions.

Here, the values are either one or zero (True/False) and tell us if a user as interacted with an item or not. So we do not have to impute missing values, because there are no missing values and can directly use SVD.

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.

[41]:
```
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?

[42]:
```
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 = set(user_item_test.index.to_list())
    test_arts = set(user_item_test.columns.to_list())

    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(df_train, df_test)
```

[43]:
```
# How many users can we make predictions for in the test set?
# => predictions can only be made for the number of unique testing users who␣
 ↪are also part of the training set
train_idx = set(user_item_train.index.to_list())
predictable_user_idx = test_idx.intersection(train_idx)
num_predictable_users = len(predictable_user_idx)
print("num_predictable_users", num_predictable_users)

# How many users in the test set are we not able to make predictions for␣
 ↪because of the cold start problem?
# => total number of test users minus the users for which we can make␣
 ↪predictions
num_cold_start_users = len(test_idx) - num_predictable_users
print("num_cold_start_users", num_cold_start_users)

# How many (_movies_??) articles can we make predictions for in the test set?
# analogous to the number of user questions...
train_arts = set(user_item_test.columns.to_list())
predictable_articles_idx = test_arts.intersection(train_arts)
num_predictable_articles = len(predictable_articles_idx)
print("num_predictable_articles", num_predictable_articles)
num_cold_start_articles = len(test_arts) - num_predictable_articles
print("num_cold_start_articles", num_cold_start_articles)
```

```
num_predictable_users 20
num_cold_start_users 662
```

```
num_predictable_articles 574
num_cold_start_articles 0
```

[44]:
```
# 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 the cold start problem?': a,
    '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 the cold start problem?': d
}

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 20 test users that were also in the training set.  All of the
other users that are in the test set we have no data on.  Therefore, we cannot
make predictions for these users using SVD.
```

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.

[45]:
```
# fit SVD on the user_item_train matrix
u_train, s_train, vt_train = np.linalg.svd(user_item_train)
```

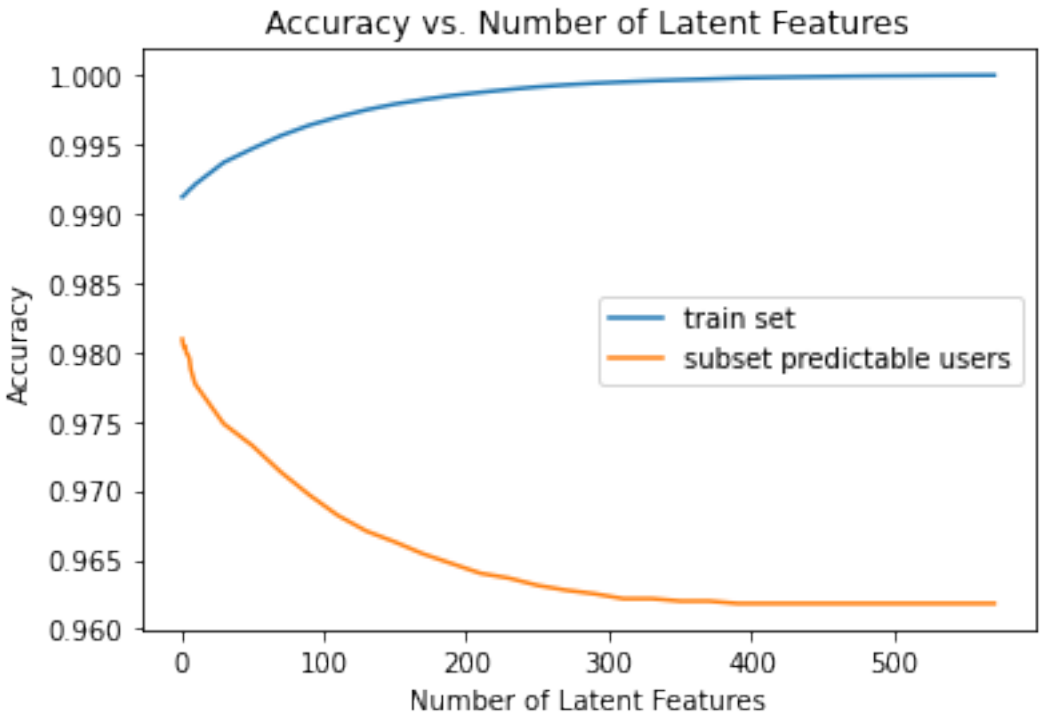
[46]:
```
# Use these cells to see how well you can use the training
# decomposition to predict on test data


# Earlier we've calculated the indices of users and articles that are included␣
 ↪in the training and test dataset
# and stored them in the set variables predictable_user_idx and␣
 ↪predictable_articles_idx
# Once we have performed the single value decomposition we have the same number␣
 ↪of rows in u_train as we have
```

```
# users in user_item_train and we have the same number columns in v_train as we␣
 ↪have articles in user_item_train
# The predictable_*_idx sets contain the real IDs not the index position within␣
 ↪the datasets.
# But np.linals.svd only returns a matrix, not a pandas dataframe. So we need␣
 ↪to convert the IDs to index positions:
## index.isin returns an arrray of True,False where the datasets contains the␣
 ↪ID and np.where returns
## the index position. np.where return values for 2 input arrays, so we select␣
 ↪only the first with [0]
train_predictable_users_index_position = np.where(user_item_train.index.
 ↪isin(predictable_user_idx))[0]
train_predictable_articles_index_position = np.where(user_item_train.columns.
 ↪isin(predictable_articles_idx))[0]

# to measure our prediction from the train set we need just the subset of␣
 ↪predictable users in the test set:
user_item_test_predictable_users = user_item_test.loc[predictable_user_idx,:]
```

[ ]:

[47]:
```
num_latent_feats = list(range(1,9))
num_latent_feats.extend(np.arange(10,num_predictable_articles,20))
sum_errs_train = []
sum_errs_predictable_users = []

for k in num_latent_feats:
    # reconstructure of decomposed train matrices with only k latent features
    s_k, u_k, vt_k = np.diag(s_train[:k]), u_train[:, :k], vt_train[:k, :]

    # from the decomposed matrices we also only take the rows and columns which␣
 ↪corresponds
    # to the index position of predictable users and articles calculated earlier
    u_k_p, vt_k_p = u_k[train_predictable_users_index_position,:], vt_k[:
 ↪,train_predictable_articles_index_position]

    # take dot product to estimate/predict the train matrix with less latent␣
 ↪features
    user_item_train_est = np.around(np.dot(np.dot(u_k, s_k), vt_k))

    # take dot product to estimate/predict the train matrix with less latent␣
 ↪features and only the predictable
    # users and articles between train and test data
    user_item_train_est_predictable_users_and_articles = np.around(np.dot(np.
 ↪dot(u_k_p, s_k), vt_k_p))
```

```
    # compute error between original train set and svd composition with just␣
↪k-latent features
    diffs_train = np.subtract(user_item_train, user_item_train_est)
    # compute error between the subset of predicatable users of the svd␣
↪composition and user_item_test
    diffs_predictable_users = np.subtract(user_item_test_predictable_users,␣
↪user_item_train_est_predictable_users_and_articles)
    # total errors
    sum_errs_train.append(np.sum(np.sum(diffs_train)))
    sum_errs_predictable_users.append(np.sum(np.sum(np.
↪abs(diffs_predictable_users))))


plt.plot(num_latent_feats,
         1 - np.array(sum_errs_train)/(user_item_train.shape[0]*user_item_train.
↪shape[1]),
         label="train set")
plt.plot(num_latent_feats,
         1 - np.array(sum_errs_predictable_users)/
↪(user_item_test_predictable_users.shape[0]*user_item_test_predictable_users.
↪shape[1]),
         label='subset predictable users');
plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.legend()
plt.title('Accuracy vs. Number of Latent Features');
```



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

Answer: The more latent features you include the closer you reconstruct the original matrix. So it is clear that the accuracy increases for the reconstruction of the decomposed training matrices with more features. But even for a very few latent features it is already quite accurate. This is due to the fact that the matrix contains far more zeros than ones, so the error will always be very low.

The current subset of users for which we can test our prediction after splitting the data into train and test data is just 0.39%. If we compute the error between the predicted articles of those users, the accuracy even decreases with more latent features. The fact that the sample size we can predict is very low and the unbalanced distribution of values in our data set (zeroes and ones) lead to underfitting. The assumption that less latent features decrease the accuracy is misleading.

With the current data set we cannot mathematically determine if e.g. matrix factorization leads to a more accurate prediction model. So in practice we would use multiple methods and present them to different user groups. Based on other metrics that somehow measure the satisfaction of the user we could conclude which prediction method is more successful.

Extras Using your workbook, you could now save your recommendations for each user, develop a class to make new predictions and update your results, and make a flask app to deploy your results. These tasks are beyond what is required for this project. However, from what you learned in the lessons, you certainly capable of taking these tasks on to improve upon your work here!

1.2 Conclusion

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

Tip: Once you are satisfied with your work here, check over your report to make sure that it is satisfies all the areas of the rubric. You should also probably remove all of the "Tips" like this one so that the presentation is as polished as possible.

1.3 Directions to Submit

Before you submit your project, you need to create a .html or .pdf version of this notebook in the workspace here. To do that, run the code cell below. If it worked correctly, you should get a return code of 0, and you should see the generated .html file in the workspace directory (click on the orange Jupyter icon in the upper left).

Alternatively, you can download this report as .html via the **File** > **Download as** submenu, and then manually upload it into the workspace directory by clicking on the orange Jupyter icon in the upper left, then using the Upload button.

Once you've done this, you can submit your project by clicking on the "Submit Project" button in the lower right here. This will create and submit a zip file with this .ipynb doc and the .html or .pdf version you created. Congratulations!

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