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 (https://review.udacity.com/#!/rubrics/2322/view). **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.

Table of Contents

At the end of the notebook, you will find directions for how to submit your work. Let's get started by importing the necessary libraries and reading in the data.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import project_tests as t
import pickle

%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	email
0	1430.0	using pixiedust for fast, flexible, and easier...	ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1	1314.0	healthcare python streaming application demo	083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2	1429.0	use deep learning for image classification	b96a4f2e92d8572034b1e9b28f9ac673765cd074
3	1338.0	ml optimization using cognitive assistant	06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4	1276.0	deploy your python model as a restful api	f01220c46fc92c6e6b161b1849de11faacd7ccb2

In [2]:

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

Out[2]:

	doc_body	doc_description	doc_full_name	doc_status	article_id
0	Skip navigation Sign in SearchLoading...\r\n\r...	Detect bad readings in real time using Python ...	Detect Malfunctioning IoT Sensors with Streami...	Live	0
1	No Free Hunch Navigation * kaggle.com\r\n\r\n ...	See the forest, see the trees. Here lies the c...	Communicating data science: A guide to present...	Live	1
2	☰ * Login\r\n * Sign Up\r\n\r\n * Learning Pat...	Here’s this week’s news in Data Science and Bi...	This Week in Data Science (April 18, 2017)	Live	2
3	DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...	Learn how distributed DBs solve the problem of...	DataLayer Conference: Boost the performance of...	Live	3
4	Skip navigation Sign in SearchLoading...\r\n\r...	This video demonstrates the power of IBM DataS...	Analyze NY Restaurant data using Spark in DSX	Live	4

In [3]:

```
df.shape
```

Out[3]:

```
(45993, 3)
```

In [4]:

```
df_content.shape
```

Out[4]:

```
(1056, 5)
```

Part I : Exploratory Data Analysis

Use the dictionary and cells below to provide some insight into the descriptive statistics of the data.

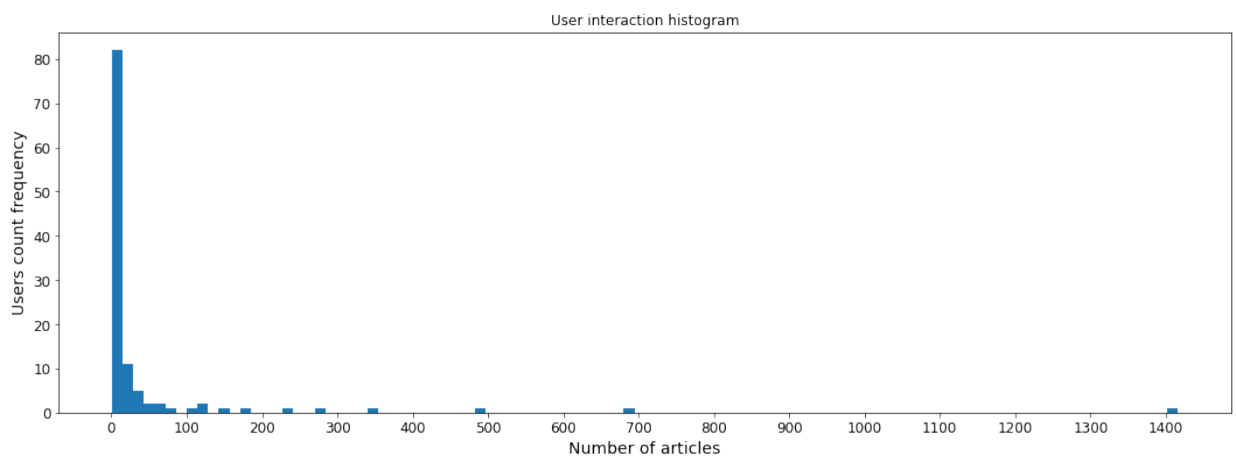
`1.` What is the distribution of how many articles a user interacts with in the dataset? Provide a visual and descriptive statistics to assist with giving a look at the number of times each user interacts with an article.

In [5]:

```
#Let's group the user interaction data by the email id, and then count how many
articles each email has and do a histogram plot
article_interaction = df.groupby('email')['article_id'].count().value_counts()
```

In [6]:

```
ax = article_interaction.plot(kind='hist', title = "User interaction histogram",
figsize=(18,6), fontsize=12, label='what', bins=100)
start, end = ax.get_xlim()
ax.xaxis.set_ticks(np.arange(0, end, 100))
ax.set_xlabel('Number of articles', fontsize=14)
ax.set_ylabel("Users count frequency", fontsize=14)
print(" ")
```



From the bar graph above we can see that the majority of the users reads just a few articles, and a few heavy reader reads hundreds or even over a 1000 articles.

In [7]:

```
user_article_count = df.groupby('email')['article_id'].count()
```

In [8]:

```
user_article_count.describe()
```

Out[8]:

```
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
```

In [9]:

```
# Fill in the median and maximum number of user_article interactios below
median_val = 3 # 50% of individuals interact with ____ number of articles or few
er.
max_views_by_user = 364 # The maximum number of user-article interactions by any
1 user is ______.
```

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

In [10]:

```
# Find and explore duplicate articles
# Let's first just see which articles are duplicated
df_articleCount = pd.DataFrame(df_content.groupby('article_id')['doc_status'].co
unt()).reset_index()
df_articleCount.columns = ['article_id', 'doc_count']
```

In [11]:

```
dup_article = df_articleCount[df_articleCount["doc_count"] > 1]
```

In [12]:

```
dup_article.head()
```

Out[12]:

	article_id	doc_count
50	50	2
221	221	2
232	232	2
398	398	2
577	577	2

In [13]:

```
# We can see from above, the articles that has the same ID may still have slight
ly descriptions but full name looks same.
df_content[df_content.article_id.isin(dup_article.article_id)].sort_values("arti
cle_id").head(20)
```

Out[13]:

	doc_body	doc_description	doc_full_name	doc_status	article_id
50	Follow Sign in / Sign up Home About Insight Da...	Community Detection at Scale	Graph-based machine learning	Live	50
365	Follow Sign in / Sign up Home About Insight Da...	During the seven-week Insight Data Engineering...	Graph-based machine learning	Live	50
221	* United States\r\n\r\nIBM® * Site map\r\n\r\n...	When used to make sense of huge amounts of con...	How smart catalogs can turn the big data flood...	Live	221
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
232	Homepage Follow Sign in Get started Homepage *...	If you are like most data scientists, you are ...	Self-service data preparation with IBM Data Re...	Live	232
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
399	Homepage Follow Sign in Get started * Home\r\n...	Today's world of data science leverages data f...	Using Apache Spark as a parallel processing fr...	Live	398
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
578	This video shows you how to construct queries ...	This video shows you how to construct queries ...	Use the Primary Index	Live	577
970	This video shows you how to construct queries ...	This video shows you how to construct queries ...	Use the Primary Index	Live	577

In [14]:

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

In [15]:

```
print("There are ", df_content.shape[0], "rows in the article dataset")
print("There are ", len(df_content.article_id.unique()), " unique article_id")
```

```
There are  1051 rows in the article dataset
There are  1051  unique article_id
```

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 [16]:

```
# Let's work on item a, the number of unique articles that have an interaction w
ith a user.
np.sum(df.isnull())
```

Out[16]:

```
article_id     0
title          0
email         17
dtype: int64
```

In [17]:

```
# The dataframe df already has the list of articles that each user interacted wi
th, we just need a unique count there.
# We'll also skip the articles that has no email
unique_count = df[~df["email"].isnull()]["article_id"].unique().shape[0]
unique_count
```

Out[17]:

```
714
```

In [18]:

```
# Now let's work on b, unique articles in the data set.
len(df_content["article_id"].unique())
```

Out[18]:

```
1051
```

In [19]:

```
# For c, we'll get the unique email count that's not null
len(df[~df["email"].isnull()]["email"].unique())
```

Out[19]:

```
5148
```

In [20]:

```
# For d, we'll get the total number of articles interacted by some users. This t
ime including null emails.
df["article_id"].shape[0]
```

Out[20]:

```
45993
```

In [21]:

```
unique_articles = 714 # The number of unique articles that have at least one int
eraction
total_articles = 1051# The number of unique articles on the IBM platform
unique_users = 5148 # The number of unique users
user_article_interactions = 45993 # 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 [22]:

```
# To get the most read article ID, we'll group the emails by article_id, and the
n get the index of the max count using idxmax
most_read_article = df.groupby("article_id")["email"].count()
most_read_article_id = most_read_article.idxmax()
print("Most read article id: ", most_read_article_id)
print("Most read article count: ", most_read_article.max())
print("Most read article title: ", df[df["article_id"] == most_read_article_id].
iloc[0]["title"])
```

```
Most read article id:  1429.0
Most read article count:  937
Most read article title:  use deep learning for image classification
```

In [23]:

```
most_viewed_article_id = "1429.0" # The most viewed article in the dataset as a
string with one value following the decimal
max_views = 937 # The most viewed article in the dataset was viewed how many tim
es?
```

In [24]:

```
## 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()
```

Out[24]:

	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 [25]:

```
## 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_view
s,
    '`The article_id of the most viewed article is ______.`': most_viewed_articl
e_id,
    '`The number of unique articles that have at least 1 rating ______.`': uniqu
e_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!
```

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 [26]:

```
# Work area for building the function
df.head()
```

Out[26]:

	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 [27]:

```
# Group the articles by user read count first
most_read_article = df.groupby("article_id")["user_id"].count()
most_read_article
```

Out[27]:

```
article_id
0.0       14
2.0       58
4.0       13
8.0       85
9.0       10
          ..
1440.0    10
1441.0     8
1442.0     4
1443.0    22
1444.0     5
Name: user_id, Length: 714, dtype: int64
```

In [28]:

```
#Let's just test with the top 10. In the function we will need to use n
top_articles = pd.DataFrame(most_read_article.sort_values(ascending=False).iloc[
0:10])
top_articles = top_articles.reset_index()
top_articles.columns = ["article_id", "count"]
top_articles.head()
```

Out[28]:

	article_id	count
0	1429.0	937
1	1330.0	927
2	1431.0	671
3	1427.0	643
4	1364.0	627

In [29]:

```
# We need to get the list of article IDs and their titles, and then join them ba
ck
articles_titles = df[["article_id", "title"]].drop_duplicates(subset=["article_i
d"])
```

In [30]:

```
articles_titles.head()
```

Out[30]:

	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

In [31]:

```
top_articles_title = top_articles.set_index('article_id').join(articles_titles.s
et_index("article_id"))
```

In [32]:

```
top_articles_title.head()
```

Out[32]:

	count	title
article_id		
1429.0	937	use deep learning for image classification
1330.0	927	insights from new york car accident reports
1431.0	671	visualize car data with brunel
1427.0	643	use xgboost, scikit-learn & ibm watson machine...
1364.0	627	predicting churn with the spss random tree alg...

In [33]:

```
list(top_articles_title.iloc[0:5]['title'])
```

Out[33]:

```
['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']
```

In [34]:

```
top_articles_title_string = list(top_articles_title.index.astype(str))
```

In [35]:

```
top_articles_title_string
```

Out[35]:

```
['1429.0',
 '1330.0',
 '1431.0',
 '1427.0',
 '1364.0',
 '1314.0',
 '1293.0',
 '1170.0',
 '1162.0',
 '1304.0']
```

In [36]:

```
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_ids, top_titles = get_top_articles_info(n, df)

    return top_titles # 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

    '''
    top_ids, top_titles = get_top_articles_info(n, df)

    return top_ids # Return the top article ids

# Since we need to get both ids and titles and the code is basically the same to
get both, creating this utility function to get both.
def get_top_articles_info(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

    '''
    #Let's first create a data frame, with the list of articles grouped by the a
rticle id and sorted by the count
    most_read_article = df.groupby("article_id")["user_id"].count()
    top_articles = pd.DataFrame(most_read_article.sort_values(ascending=False).i
loc[0:n])
    top_articles = top_articles.reset_index()
    top_articles.columns = ["article_id", "count"]

    #Now we get the titles, and join them back with the top articles
    articles_titles = df[["article_id", "title"]].drop_duplicates(subset=["artic
le_id"])
    top_articles_title = top_articles.set_index('article_id').join(articles_titl
es.set_index("article_id"))
    top_titles = list(top_articles_title['title'])
    top_ids = list(top_articles_title.index.astype(str))
    return top_ids, top_titles
```

In [37]:

```
print(get_top_articles(10))
print(get_top_article_ids(10))
```

```
['use deep learning for image classification', 'insights from new yo
rk car accident reports', 'visualize car data with brunel', 'use xgb
oost, scikit-learn & ibm watson machine learning apis', 'predicting
churn with the spss random tree algorithm', 'healthcare python strea
ming application demo', 'finding optimal locations of new store usin
g decision optimization', 'apache spark lab, part 1: basic concepts'
, 'analyze energy consumption in buildings', 'gosales transactions f
or logistic regression model']
['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0
', '1170.0', '1162.0', '1304.0']
```

In [38]:

```
# 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.
```

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 [39]:

```
# Work area to build the function
df.head()
```

Out[39]:

	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 [40]:

```
df.shape
```

Out[40]:

```
(45993, 3)
```

In [41]:

```
# We need to create one column per article. Let's do a quick test
df_test = df.iloc[0:5]
```

In [42]:

```
articleList = list(df_test["article_id"])
```

In [43]:

```
userTable = pd.DataFrame(df_test["user_id"])
userTable.columns = ["user_id"]
userTable.head()
```

Out[43]:

	user_id
0	1
1	2
2	3
3	4
4	5

In [44]:

```
# With the blank table with users as index, we need to add one column / article
at a time
#   For each article, we will first get all the unique users who saw the article
, then set the rows accordingly
for aid in articleList:
    articleUserList = set(df_test[df_test["article_id"] == aid]["user_id"].uniqu
e())
    userTable[aid] = userTable["user_id"].apply(lambda x: 1 if x in articleUserL
ist else 0)
```

In [45]:

```
userTable = userTable.set_index("user_id")
```

In [46]:

```
userTable.head()
```

Out[46]:

	1430.0	1314.0	1429.0	1338.0	1276.0
user_id					
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	0	0	1

In [47]:

```
# create the user-article matrix with 1's and 0's
import progressbar

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
    '''
    # We will start with a dataframe with just the list of user ids
    user_item = pd.DataFrame(df["user_id"].unique())
    user_item.columns = ["user_id"]

    # With the blank table with users as index, we need to add one column / arti
cle at a time
    #   For each article, we will first get all the unique users who saw the art
icle, then set the rows accordingly
    #   As this will take a while we will use the progress bar too
    articleList = list(df["article_id"].unique())

    bar = progressbar.ProgressBar(maxval=len(articleList), widgets=[progressbar.
Bar('=', '[', ']'), ' ', progressbar.Percentage()])
    bar.start()
    counter = 0

    for aid in articleList:
        bar.update(counter)
        articleUserList = set(df[df["article_id"] == aid]["user_id"].unique())
        user_item[aid] = user_item["user_id"].apply(lambda x: 1 if x in articleU
serList else 0)
        counter += 1

    bar.finish()

    user_item = user_item.set_index("user_id")

    return user_item # return the user_item matrix
```

In [48]:

```
user_item = create_user_item_matrix(df)
```

```
[====================================================================
=====] 100%
```

In [49]:

```
user_item.shape
```

Out[49]:

```
(5149, 714)
```

In [50]:

```
## 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-artic
le matrix doesn't look right."
assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-art
icle matrix doesn't look right."
assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by us
er 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.

In [51]:

```
# Work area to build the function
user_item.head()
```

Out[51]:

	1430.0	1314.0	1429.0	1338.0	1276.0	1432.0	593.0	1185.0	993.0	14.0	...	1135.0
user_id												
1	1	0	1	0	0	0	0	1	0	0	...	0
2	0	1	0	0	0	0	0	0	0	0	...	0
3	0	1	1	0	0	1	0	0	0	0	...	0
4	0	1	0	1	1	1	0	0	0	0	...	0
5	0	0	0	0	1	0	0	0	0	0	...	0

5 rows × 714 columns

In [52]:

```
# Work area to build the function
user_similarity = {}
currentUserVector = user_item.loc[1]
currentUserVector
```

Out[52]:

```
1430.0    1
1314.0    0
1429.0    1
1338.0    0
1276.0    0
         ..
1156.0    0
555.0     0
708.0     0
575.0     0
972.0     0
Name: 1, Length: 714, dtype: int64
```

In [53]:

```
#Let's calculate similarity for a few users
for i in range(2,6):
    otherUserVector = user_item.loc[i]
    similarity = np.dot(currentUserVector, otherUserVector)
    user_similarity[i] = similarity
```

In [54]:

```
user_similarity
```

Out[54]:

```
{2: 2, 3: 6, 4: 3, 5: 0}
```

In [55]:

```
list(user_similarity.keys())
```

Out[55]:

```
[2, 3, 4, 5]
```

In [56]:

```
similar_user_df = pd.DataFrame({'user_id': list(user_similarity.keys()), 'simila
rity':list(user_similarity.values())})
```

In [57]:

```
similar_user_df
```

Out[57]:

	user_id	similarity
0	2	2
1	3	6
2	4	3
3	5	0

In [58]:

```
similar_user_df = similar_user_df.sort_values("similarity", ascending=False)
```

In [59]:

```
similar_user_df.head()
```

Out[59]:

	user_id	similarity
1	3	6
2	4	3
0	2	2
3	5	0

In [60]:

```
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
    user_similarity = {}
    currentUserVector = user_item.loc[user_id]

    #Let's calculate similarity for a few users
    for i in list(user_item.index):
        #Skip over the user himself
        if i == user_id:
            continue

        otherUserVector = user_item.loc[i]
        similarity = np.dot(currentUserVector, otherUserVector)
        user_similarity[i] = similarity

    similar_user_df = pd.DataFrame({'user_id': list(user_similarity.keys()), 'si
milarity':list(user_similarity.values())})

    # sort by similarity
    similar_user_df = similar_user_df.sort_values("similarity", ascending=False)

    # create list of just the ids
    most_similar_users = list(similar_user_df["user_id"])

    return most_similar_users # return a list of the users in order from most to
least similar
```

In [61]:

```
# 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,
3870, 131, 46, 4201, 395]
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 [62]:

```
#Work areas
top_articles_title_string
```

Out[62]:

```
['1429.0',
 '1330.0',
 '1431.0',
 '1427.0',
 '1364.0',
 '1314.0',
 '1293.0',
 '1170.0',
 '1162.0',
 '1304.0']
```

In [63]:

```
article_names = []
for arid in [1429.0, 1330.0, 1431.0]:
    article_names.append(df[df["article_id"] == arid].iloc[0]["title"])
```

In [64]:

```
article_names
```

Out[64]:

```
['use deep learning for image classification',
 'insights from new york car accident reports',
 'visualize car data with brunel']
```

In [65]:

```
user_item.head()
```

Out[65]:

	1430.0	1314.0	1429.0	1338.0	1276.0	1432.0	593.0	1185.0	993.0	14.0	...	1135.0
user_id												
1	1	0	1	0	0	0	0	1	0	0	...	0
2	0	1	0	0	0	0	0	0	0	0	...	0
3	0	1	1	0	0	1	0	0	0	0	...	0
4	0	1	0	1	1	1	0	0	0	0	...	0
5	0	0	0	0	1	0	0	0	0	0	...	0

5 rows × 714 columns

In [66]:

```
userItems = pd.DataFrame(user_item.loc[1])
userItems.columns = ["read"]
```

In [67]:

```
list(userItems[userItems["read"] == 1].index)
```

Out[67]:

```
[1430.0,
 1429.0,
 1185.0,
 1170.0,
 1052.0,
 1431.0,
 1427.0,
 1368.0,
 1305.0,
 1436.0,
 1400.0,
 310.0,
 1293.0,
 151.0,
 43.0,
 732.0,
 109.0,
 626.0,
 1406.0,
 1391.0,
 981.0,
 268.0,
 668.0,
 525.0,
 968.0,
 1232.0,
 329.0,
 585.0,
 768.0,
 346.0,
 910.0,
 1183.0,
 1439.0,
 494.0,
 390.0,
 1363.0]
```

In [68]:

```
matching_articles = df[df["article_id"] == 1024.0]
```

In [69]:

```
len(matching_articles)
```

Out[69]:

```
74
```

In [70]:

```
matching_articles.iloc[0]["title"]
```

Out[70]:

```
'using deep learning to reconstruct high-resolution audio'
```

In [71]:

```
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 a
rticle ids
                    (this is identified by the title column)
    '''
    article_names = []
    for arid in article_ids:
        matching_articles = df[df["article_id"] == float(arid)]
        if len(matching_articles) > 0:
            article_names.append(matching_articles.iloc[0]["title"])
        else:
            print("Error getting title for article with id ", arid)

    return article_names # Return the article names associated with list of arti
cle 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 a
rticle ids
                    (this is identified by the doc_full_name column in df_conten
t)

    Description:
    Provides a list of the article_ids and article titles that have been seen by
a user
    '''
    if user_id in df.index:
        userItems = pd.DataFrame(user_item.loc[user_id])
        userItems.columns = ["read"]
        article_ids = list(userItems[userItems["read"] == 1].index.astype(str))
        article_names = get_article_names(article_ids, df)
        return article_ids, article_names # return the ids and names
    else:
        return [],[]




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

    '''
    recs = []

    #Get the list of most similar users
    sim_users = find_similar_users(user_id, user_item)

    my_articles_id, my_articles_names = get_user_articles(user_id, user_item)
```

```
    for user in sim_users:
        other_articles_id, other_articles_names = get_user_articles(user, user_i
tem)
        for aid in other_articles_id:
            if not aid in my_articles_id:
                recs.append(aid)
                if len(recs) >= m:
                    return recs

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

In [72]:

```
# Check Results
get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0', '1427.0'])
```

Out[72]:

```
['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']
```

In [73]:

```
# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

Out[73]:

```
['healthcare python streaming application demo',
 'ml optimization using cognitive assistant',
 'deploy your python model as a restful api',
 'visualize data with the matplotlib library',
 'got zip code data? prep it for analytics. – ibm watson data lab –
medium',
 'the unit commitment problem',
 'timeseries data analysis of iot events by using jupyter notebook',
 'the nurse assignment problem',
 'dsx: hybrid mode',
 'predicting churn with the spss random tree algorithm']
```

In [74]:

```
# Check Results
get_user_articles(20)[0]
```

Out[74]:

```
['1320.0', '844.0', '232.0']
```

In [75]:

```
# 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 f
or naive bayes model', 'healthcare python streaming application demo', 'use r da
taframes & ibm watson natural language understanding', 'use xgboost, scikit-lear
n & ibm watson machine learning apis']), "Oops! Your the get_article_names funct
ion doesn't work quite how we expect."
assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing (20
15): united states demographic measures','self-service data preparation with ibm
data refinery','use the cloudant-spark connector in python notebook']), "Oops! Y
our 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 demo
graphic measures', 'self-service data preparation with ibm data refinery','use t
he 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 appli
cation demo', 'use r dataframes & ibm watson natural language understanding', 'u
se 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.

In [76]:

```
# Work areas.
user_id = 1
user_similarity = {}
currentUserVector = user_item.loc[user_id]

for i in list(user_item.index):
    #Skip over the user himself
    if i == user_id:
        continue

    otherUserVector = user_item.loc[i]
    similarity = np.dot(currentUserVector, otherUserVector)
    user_similarity[i] = similarity

similar_user_df = pd.DataFrame({'user_id': list(user_similarity.keys()), 'simila
rity':list(user_similarity.values())})
```

In [77]:

```
similar_user_df.head()
```

Out[77]:

	user_id	similarity
0	2	2
1	3	6
2	4	3
3	5	0
4	6	4

In [78]:

```
def get_user_interaction_count(user_id, user_item=user_item):
    '''
    A utility function to get the interaction count for a particular user
    '''
    arid, arname = get_user_articles(2, user_item=user_item)
    return len(arid)
```

In [79]:

```
similar_user_df["num_interactions"] = similar_user_df["user_id"].apply(get_user_
interaction_count)
```

In [80]:

```
similar_user_df.head()
```

Out[80]:

	user_id	similarity	num_interactions
0	2	2	6
1	3	6	6
2	4	3	6
3	5	0	6
4	6	4	6

In [81]:

```
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 p
rovided 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 o
f interactions where
                    highest of each is higher in the dataframe

    '''
    # First we find the list of users who are similar first, similar to the earl
ier step
    user_similarity = {}
    currentUserVector = user_item.loc[user_id]

    for i in list(user_item.index):
        #Skip over the user himself
        if i == user_id:
            continue

        otherUserVector = user_item.loc[i]
        similarity = np.dot(currentUserVector, otherUserVector)
        user_similarity[i] = similarity

    neighbors_df = pd.DataFrame({'user_id': list(user_similarity.keys()), 'simil
arity':list(user_similarity.values())})

    # Now that we have a list of similar users, we need to find the number of in
teraction for each of them
    neighbors_df["num_interactions"] = neighbors_df["user_id"].apply(get_user_in
teraction_count)

    # sort by similarity
    neighbors_df = neighbors_df.sort_values(["similarity", "num_interactions"],
ascending=False)

    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 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.

    '''
    # Your code here
    recs = []
    rec_names = []

    #Get the list of most similar users
    sim_users = get_top_sorted_users(user_id, df, user_item)

    my_articles_id, my_articles_names = get_user_articles(user_id, user_item)

    for user in sim_users["user_id"].values:
        other_articles_id, other_articles_names = get_user_articles(user, user_i
tem)
        for i in range(0,len(other_articles_id)):
            if not other_articles_id[i] in my_articles_id:
                recs.append(other_articles_id[i])
                rec_names.append(other_articles_names[i])
                if len(recs) >= m:
                    return recs, rec_names

    return recs, rec_names
```

In [82]:

```
my_articles_id, my_articles_names = get_user_articles(20, user_item)
```

In [83]:

```
type(my_articles_id)
```

Out[83]:

```
list
```

In [84]:

```
# 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
:
['1430.0', '1314.0', '1429.0', '1338.0', '1276.0', '1185.0', '1364.0
', '1162.0', '1431.0', '1427.0']

The top 10 recommendations for user 20 are the following article nam
es:
['using pixiedust for fast, flexible, and easier data analysis and e
xperimentation', 'healthcare python streaming application demo', 'us
e deep learning for image classification', 'ml optimization using co
gnitive assistant', 'deploy your python model as a restful api', 'cl
assify tumors with machine learning', 'predicting churn with the sps
s random tree algorithm', 'analyze energy consumption in buildings',
'visualize car data with brunel', 'use xgboost, scikit-learn & ibm w
atson machine learning apis']
```

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 [85]:

```
#Work area
# Find the user that is most similar to user 1
# Find the 10th most similar user to user 131
one_neighbour = get_top_sorted_users(1, df, user_item)
```

In [86]:

```
one_neighbour.head()
```

Out[86]:

	user_id	similarity	num_interactions
3931	3933	35	6
21	23	17	6
3780	3782	17	6
201	203	15	6
4457	4459	15	6

In [87]:

```
ten_neighbour = get_top_sorted_users(131, df, user_item)
```

In [88]:

```
ten_neighbour.head(10)
```

Out[88]:

	user_id	similarity	num_interactions
3868	3870	74	6
3780	3782	39	6
22	23	38	6
201	203	33	6
4457	4459	33	6
48	49	29	6
97	98	29	6
3695	3697	29	6
3762	3764	29	6
240	242	25	6

In [89]:

```
### Tests with a dictionary of results

user1_most_sim = 3933 # Find the user that is most similar to user 1
user131_10th_sim = 242 # Find the 10th most similar user to user 131
```

In [90]:

```
## 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.

Early on in the exercise we created a function to return the most read / interacted with article, which could be a start. An improvement would be classifying the articles by topics too. When a user comes, if we have clues (or if we just ask) regarding what topics the user may like more we can provide different top suggestions based on topic also.

`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 [91]:

```
new_user = '0.0'

# What would your recommendations be for this new user '0.0'?  As a new user, th
ey have no observed articles.
# Provide a list of the top 10 article ids you would give to

new_user_recs = get_top_article_ids(10, df)
```

In [92]:

```
new_user_recs
```

Out[92]:

```
['1429.0',
 '1330.0',
 '1431.0',
 '1427.0',
 '1364.0',
 '1314.0',
 '1293.0',
 '1170.0',
 '1162.0',
 '1304.0']
```

In [93]:

```
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!
```

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.

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 [94]:

```
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?

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.

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 [95]:

```
# make recommendations for a brand new user


# make a recommendations for a user who only has interacted with article id '142
7.0'
```

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 [96]:

```
# Load the matrix here
user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

In [97]:

```
# quick look at the matrix
user_item_matrix.head()
```

Out[97]:

article_id	0.0	100.0	1000.0	1004.0	1006.0	1008.0	101.0	1014.0	1015.0	1016.0	...	977.0
user_id												
1	0.0	0.0	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	0.0	...	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.0
4	0.0	0.0	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	0.0	...	0.0

5 rows × 714 columns

`2.` In this situation, you can use Singular Value Decomposition from [numpy (https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.linalg.svd.html)](https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.linalg.svd.html) on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

In [98]:

```
# Perform SVD on the User-Item Matrix Here

u, s, vt = np.linalg.svd(user_item_matrix, full_matrices=True) # use the built i
n to get the three matrices
```

In [99]:

```
u.shape
```

Out[99]:

```
(5149, 5149)
```

In [100]:

```
s.shape
```

Out[100]:

```
(714,)
```

In [101]:

```
vt.shape
```

Out[101]:

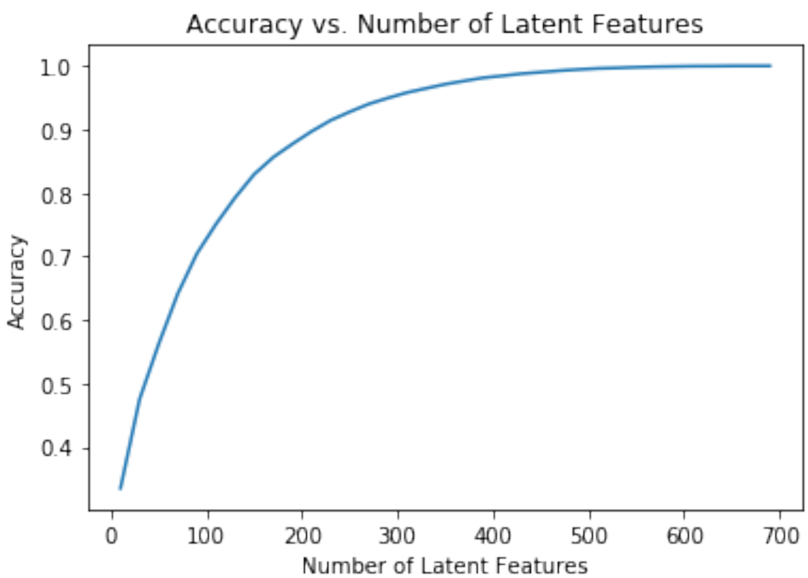
```
(714, 714)
```

As we learned in the movie recommendation lesson, SVD only works if there are no missing values. So for the movie ratings, if there are movies that the users hasn't rated before SVD doesn't work and a common technique is to fill those values with zeros. In our case, the values in the matrix represents whether the user has or has not interacted with the article, and therefore the values are ones and zeros already so we can 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.

In [102]:

```
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 [103]:

```
#Work areas
user_item_matrix.shape
```

Out[103]:

```
(5149, 714)
```

In [104]:

```
df.head()
```

Out[104]:

	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 [105]:

```
user_item.head()
```

Out[105]:

	1430.0	1314.0	1429.0	1338.0	1276.0	1432.0	593.0	1185.0	993.0	14.0	...	1135.0
user_id												
1	1	0	1	0	0	0	0	1	0	0	...	0
2	0	1	0	0	0	0	0	0	0	0	...	0
3	0	1	1	0	0	1	0	0	0	0	...	0
4	0	1	0	1	1	1	0	0	0	0	...	0
5	0	0	0	0	1	0	0	0	0	0	...	0

5 rows × 714 columns

In [106]:

```
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 co
lumn)
    user_item_test - a user-item matrix of the testing dataframe
                    (unique users for each row and unique articles for each colu
mn)
    test_idx - all of the test user ids
    test_arts - all of the test article ids

    '''

    #For the train user item, we will get the list of users in df_train

    user_item_train = user_item_matrix.loc[set(df_train["user_id"].values)]
    user_item_test = user_item_matrix.loc[set(df_test["user_id"].values)]
    test_idx = list(user_item_test.index)
    test_arts = list(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_use
r_item(df_train, df_test)
```

In [107]:

```
#Work area
user_item_train.shape
```

Out[107]:

```
(4487, 714)
```

In [108]:

```
user_item_test.shape
```

Out[108]:

```
(682, 714)
```

In [109]:

```
#Work area for 'How many users can we make predictions for in the test set?
# For us to make predictions, we need to have data on what articles you read.
print("Users that are in both train and test data set:", len(np.intersect1d(user
_item_train.index, user_item_test.index)))
```

```
Users that are in both train and test data set: 20
```

In [110]:

```
#Work area for 'How many users in the test set are we not able to make predictio
ns for because of the cold start problem?'
# If we have not see the user in the training set before, we won't be able to pr
edict
print("Total users in database = ", len(user_item_matrix), " number of users not
trained: ", len(user_item_matrix) - len(user_item_train))
```

```
Total users in database =  5149  number of users not trained:  662
```

In [111]:

```
# For 'How many movies can we make predictions for in the test set?'
# We need to know the number of articles that appeared in the test set
articles_in_test = len(set(df_test["article_id"].values))
articles_in_test
```

Out[111]:

```
574
```

In [112]:

```
# For 'How many movies in the test set are we not able to make predictions for b
ecause of the cold start problem?
# We need to see the number of articles that did not show up in the train set
all_articles_in_train = (set(df_train["article_id"].values))
total_articles_in_db = (set(df["article_id"].values))
print("Total articles len: ", len(total_articles_in_db), ", trained = ", len(all
_articles_in_train))
```

```
Total articles len:  714 , trained =  714
```

In [113]:

```
#Looks like all articles was read by someone and trained
articles_trained = np.intersect1d(list(all_articles_in_train), list(total_articl
es_in_db))
len(articles_trained)
```

Out[113]:

```
714
```

In [114]:

```
# Replace the values in the dictionary below
a = 662
b = 574
c = 20
d = 0

#Looks like there were some issues with the answer key with copy and paste. It s
hould be "articles" and not "movies".
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 beca
use 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 bec
ause 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 train
ing data, but there are only 20 test users that were also in the tra
ining 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 u
sing 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.

In [115]:

```
index_predictable_users = np.intersect1d(user_item_train.index, user_item_test.i
ndex)
index_predictable_users
```

Out[115]:

```
array([2917, 3024, 3093, 3193, 3527, 3532, 3684, 3740, 3777, 3801, 3
968,
       3989, 3990, 3998, 4002, 4204, 4231, 4274, 4293, 4487])
```

In [116]:

```
arrayIndex = index_predictable_users - 1
arrayIndex
```

Out[116]:

```
array([2916, 3023, 3092, 3192, 3526, 3531, 3683, 3739, 3776, 3800, 3
967,
       3988, 3989, 3997, 4001, 4203, 4230, 4273, 4292, 4486])
```

In [117]:

```
# fit SVD on the user_item_train matrix
u_train, s_train, vt_train = np.linalg.svd(user_item_train, full_matrices=True)
# use the built in to get the three matrices# fit svd similar to above then use
the cells below
```

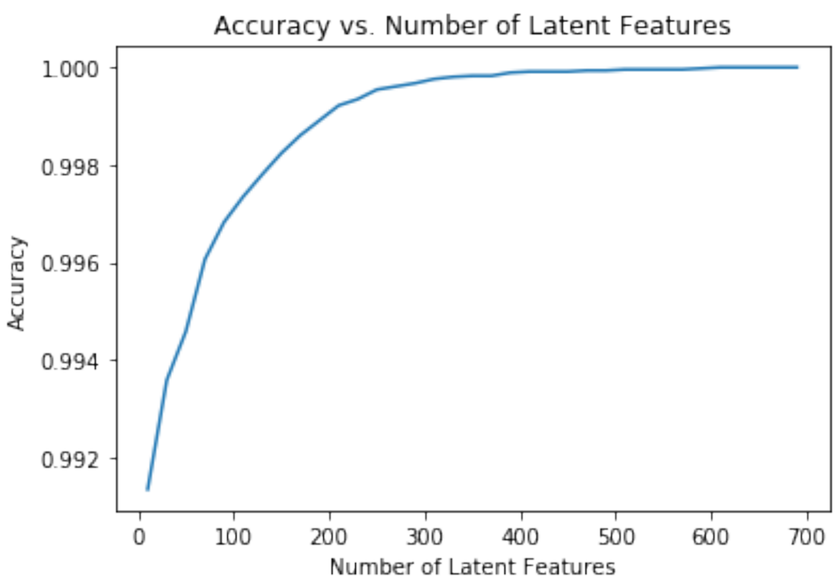
In [118]:

```
print("u_train shape", u_train.shape)
print("s_train shape", s_train.shape)
print("vt_train shape", vt_train.shape)
```

```
u_train shape (4487, 4487)
s_train shape (714,)
vt_train shape (714, 714)
```

In [119]:

```
#Repeat the earlier exercise with the new training and test data

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

for k in num_latent_feats:
    # restructure with k latent features and only the subset of users that we ca
n predict on
    s_new, u_new, vt_new = np.diag(s[:k]), u[arrayIndex, :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.loc[index_predictable_users], 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');
```



`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?

Looking at the results, the accuracy of the recommendation is quite high and the more latent features we keep the better the result is. And as we learned in the lessons, SVD works great only if the data is non-sparse. In the movie rating example, where movie watchers likely only rate a very small set of the movies and usually only the extremes, once they loved and once they hated, SVD did not do well. In the current case for recommending articles, the data is indeed non-sparse. We have a full record of when the user interacted or not interacted with each article and therefore the result using SVD looks great.

Even though the numbers look good, it's probably too early to celebrate anything as these recommendations can be considered great only if we get some additional feedback.

For the earlier exercise for recommending movies, you can validate whether the suggested recommendations are well received if the user actually watches the movie, and then rates the movie. The comparison between the user's rating and what we predict allows usto confirm whether it was a good suggestion. For the articles here it's actually impossible to determine accurately, without some additional metrics. Even if the user actually reads the article, he may thinks it's completely irrelevant or he doesn't like the article and we would not be able to tell. A rating system or at least a like / not like button needs to be added to provide the additional signal for better prediction.

Another alternative without the adding ratings is to separate our users into two groups to do A/B test. One can be the control group without any recommendations from us, and the other group gets suggestions from our recommender. Observations of how the users differ interact with the recommended articles can shed lights on how good the recommended articles are.

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!

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 (https://review.udacity.com/#!/rubrics/2322/view). You should also probably remove all of the "Tips" like this one so that the presentation is as polished as possible.

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!

In [120]:

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

Out[120]:

0