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)](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

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 & 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.

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

Part I : Exploratory Data Analysis

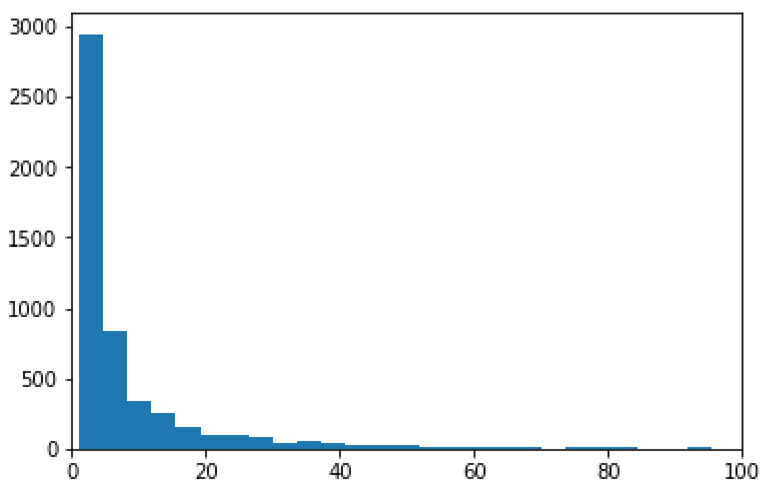
Use the dictionary and cells below to provide some insight into the descriptive statistics of the data.

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

In [3]:
```
# how many articles a user interacts
article_per_user=df.groupby('email')['article_id'].count()
article_per_user.describe()
```

Out[3]:
```
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 [4]:
```
# make a histogram
plt.hist(article_per_user, bins=100);
plt.xlim(0,100);
```



In [5]:
```
# Fill in the median and maximum number of user_article interactios below

median_val = 3 # 50% of individuals interact with ____ number of articles or f
ewer.
max_views_by_user = 364 # The maximum number of user-article interactions by a
ny 1 user is ______.
```

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

In [6]:
```python
# Find and explore duplicate articles
articleID_duplicate=df_content['article_id'][df_content['article_id'].duplicat
ed()]
articleID_duplicate
```

Out[6]:
```
365     50
692    221
761    398
970    577
971    232
Name: article_id, dtype: int64
```

In [7]:
```python
# Remove any rows that have the same article_id - only keep the first
df[df['article_id'].isin(list(articleID_duplicate))].drop_duplicates('article_
id',keep="first")
```

Out[7]:

	article_id	title	email
125	50.0	graph-based machine learning	383273e6185969bd9b93ace8d20cfab0a75e6979
269	221.0	how smart catalogs can turn the big data flood...	3427a5a4065625363e28ac8e85a57a9436010e9c
1008	232.0	self-service data preparation with ibm data re...	fff9fc3ec67bd18ed57a34ed1e67410942c4cd81
16995	398.0	51822 using apache spark as a parallel proc...	731dfdc882e246d08e5fdb4185ff1f11818612e3

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 [8]:
```python
# The number of unique articles that have an interaction with a user.
df['article_id'].nunique()
```

Out[8]: 714

In [9]:
```python
# The number of unique articles in the dataset (whether they have any interact
ions or not).
df_content['article_id'].nunique()
```

Out[9]: 1051

In [10]:
```python
# The number of unique users in the dataset. (excluding null values)
df['email'].nunique()
```

Out[10]: 5148

In [11]:
```
# The number of user-article interactions in the dataset.
df.shape[0]
```

Out[11]:
```
45993
```

In [12]:
```
unique_articles = 714 # The number of unique articles that have at least one i
nteraction
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 [13]:
```
# get the most vieweed article_id and the number of views
df['article_id'].value_counts().head(1)
```

Out[13]:
```
1429.0    937
Name: article_id, dtype: int64
```

In [14]:
```
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 t
imes?
```

In [15]:

```
## No need to change the code here - this will be helpful for later parts of t
he 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[15]:

	article_id	title	user_id
0	1430.0	using pixiedust for fast, flexible, and easier...	1
1	1314.0	healthcare python streaming application demo	2
2	1429.0	use deep learning for image classification	3
3	1338.0	ml optimization using cognitive assistant	4
4	1276.0	deploy your python model as a restful api	5

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

         sol_1_dict = {
             '`50% of individuals have _____ or fewer interactions.`': median_val,
             '`The total number of user-article interactions in the dataset is ______.
         `': user_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_vi
         ews,
             '`The article_id of the most viewed article is ______.`': most_viewed_arti
         cle_id,
             '`The number of unique articles that have at least 1 rating ______.`': uni
         que_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 [17]:
```
def get_top_articles(n, df=df):
    '''
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

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

    '''
    # Your code here
    top_articles=[]
    # get the list of top n article_id
    id_topn=df['article_id'].value_counts().head(n).index
    for item in id_topn:
        top_articles.append(df[df['article_id']==item].drop_duplicates('title'
,keep='first')['title'].values[0])

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

def get_top_article_ids(n, df=df):
    '''
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

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

    '''
    # Your code here
    idx=df['article_id'].value_counts().head(n)
    top_articles=idx.index.tolist()
    return top_articles # Return the top article ids
```

In [18]:
```
print(get_top_articles(10))
print(get_top_article_ids(10))
```

```
['use deep learning for image classification', 'insights from new york car ac
cident reports', 'visualize car data with brunel', 'use xgboost, scikit-learn
& ibm watson machine learning apis', 'predicting churn with the spss random t
ree algorithm', 'healthcare python streaming application demo', 'finding opti
mal locations of new store using decision optimization', 'apache spark lab, p
art 1: basic concepts', 'analyze energy consumption in buildings', 'gosales t
ransactions for logistic regression model']
[1429.0, 1330.0, 1431.0, 1427.0, 1364.0, 1314.0, 1293.0, 1170.0, 1162.0, 130
4.0]
```

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

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

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

Part III: User-User Based Collaborative Filtering

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

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

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

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

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

    OUTPUT:
    user_item - user item matrix

    Description:
    Return a matrix with user ids as rows and article ids on the columns with
 1 values 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']).agg(lambda x: 1).unstac
k().fillna(0)
    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
```

In [21]:
```
## Tests: You should just need to run this cell.  Don't change the code.
assert user_item.shape[0] == 5149, "Oops!  The number of users in the user-art
icle matrix doesn't look right."
assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-a
rticle 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.

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

    OUTPUT:
    similar_users - (list) an ordered list where the closest users (largest do
t 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_item_new=user_item.copy()
    user_item_new['similarity']=np.dot(user_item,user_item.loc[user_id,:])

    # sort by similarity
    user_item_new.sort_values(by='similarity',ascending=False,inplace=True)

    # create list of just the ids
    most_similar_users=user_item_new.index.tolist()
    # remove the own user's id
    most_similar_users.remove(user_id)

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

In [23]:
```
# Do a spot check of your function
print("The 10 most similar users to user 1 are: {}".format(find_similar_users(
1)[:10]))
print("The 5 most similar users to user 3933 are: {}".format(find_similar_user
s(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, 13
1, 4201, 46, 5041]
The 5 most similar users to user 3933 are: [1, 23, 3782, 203, 4459]
The 3 most similar users to user 46 are: [4201, 3782, 23]
```

3. Now that you have a function that provides the most similar users to each user, you will want to use these users to find articles you can recommend. Complete the functions below to return the articles you would recommend to each user.

In [24]:
```
def get_article_names(article_ids, df=df):
    '''
    INPUT:
    article_ids - (list) a list of article ids
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    article_names - (list) a list of article names associated with the list of
article ids
                    (this is identified by the title column)
    '''
    # Your code here
    article_names=df[df['article_id'].isin(article_ids)]['title'].unique().tol
ist()

    return article_names # Return the article names associated with list of ar
ticle 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_cont
ent)

    Description:
    Provides a list of the article_ids and article titles that have been seen
 by a user
    '''
    # Your code here
    article_ids = user_item.loc[user_id][user_item.loc[user_id]==1].title.inde
x.tolist()
    article_ids = [str(item) for item in article_ids]
    article_names = get_article_names(article_ids)

    return article_ids, article_names # return the ids and names


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

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

    Description:
    Loops through the users based on closeness to the input user_id
```

```
    For each user - finds articles the user hasn't seen before and provides th
em 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

    '''
    # Your code here
    article_ids_self, article_names_self =  get_user_articles(user_id)

    most_similar_users=find_similar_users(user_id)

    recs=[]

    for user_id in most_similar_users:
        article_ids, article_names =  get_user_articles(user_id)

        for article_id in article_ids:
            if article_id not in article_ids_self:
                if article_id not in recs and len(recs) < m:
                    recs.append(article_id)
                    if len(recs)>=m:
                        break
        if len(recs)>=m:
            break

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

In [25]:

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

Out[25]:

```
['got zip code data? prep it for analytics. – ibm watson data lab – medium',
 'timeseries data analysis of iot events by using jupyter notebook',
 'graph-based machine learning',
 'using brunel in ipython/jupyter notebooks',
 'experience iot with coursera',
 'the 3 kinds of context: machine learning and the art of the frame',
 'deep forest: towards an alternative to deep neural networks',
 'this week in data science (april 18, 2017)',
 'higher-order logistic regression for large datasets',
 'using machine learning to predict parking difficulty']
```

In [26]:

```
# Test your functions here - No need to change this code - just run this cell
assert set(get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0'
, '1427.0'])) == set(['using deep learning to reconstruct high-resolution audi
o', 'build a python app on the streaming analytics service', 'gosales transact
ions 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_artic
le_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 wi
th 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 de
mographic measures', 'self-service data preparation with ibm data refinery','u
se the cloudant-spark connector in python notebook'])
assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '131
4.0', '1422.0', '1427.0'])
assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruc
t high-resolution audio', 'build a python app on the streaming analytics servi
ce', 'gosales transactions for naive bayes model', 'healthcare python streamin
g application demo', 'use r dataframes & ibm watson natural language understan
ding', '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.

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


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

    '''
    # Your code here
    # compute similarity of each user to the provided user

    user_item_new=user_item.copy()

    # calculate similarity & number of interactions
    user_item_new['similarity']=np.dot(user_item,user_item.loc[user_id,:])
    user_item_new['num_interactions']=user_item.sum(axis=1)

    # sort by similarity & number of interactions
    user_item_new.sort_values(by=['similarity','num_interactions'],ascending=F
alse,inplace=True)

    neighbors_df=user_item_new[['similarity','num_interactions']].reset_index
()
    neighbors_df.columns=['neighbor_id','similarity','num_interactions']

    neighbors_df=neighbors_df[neighbors_df['neighbor_id']!=user_id]

    return neighbors_df # Return the dataframe specified in the doc_string

def get_article_interaction(user_item=user_item):
    article_df=pd.DataFrame(columns=user_item.columns,index=['num_interaction
s'])
    article_df.loc['num_interactions',:]=user_item.sum(axis=0)

    article_df_T=article_df.T.reset_index()
    article_df_T.drop(['level_0'],axis=1, inplace=True)

    return article_df_T

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 th
em 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

    article_ids_self, article_names_self =  get_user_articles(user_id)
    # get close neighbors
    neighbors_df=get_top_sorted_users(user_id, df=df, user_item=user_item)

    neigbhors_lst=neighbors_df['neighbor_id']


    article_interaction_df=get_article_interaction(user_item)

    recs=[]

    for user_id in neigbhors_lst:
        article_ids, article_names =  get_user_articles(user_id)

        df_temp=article_interaction_df[article_interaction_df['article_id'].is
in(article_ids)]
        df_temp=df_temp.sort_values(by='num_interactions')

        for article_id in df_temp['article_id']:

            if article_id not in article_ids_self:
                if article_id not in recs and len(recs) < m:
                    recs.append(article_id)
                    if len(recs)>=m:
                        break
        if len(recs)>=m:
            break

    rec_names=get_article_names(recs)

    return recs, rec_names
```

In [28]:
```
# Quick spot check - don't change this code - just use it to test your functio
ns
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 name
s:")
print(rec_names)
```

```
The top 10 recommendations for user 20 are the following article ids:
[763.0, 857.0, 876.0, 468.0, 347.0, 273.0, 990.0, 858.0, 302.0, 609.0]

The top 10 recommendations for user 20 are the following article names:
['accelerate your workflow with dsx', 'what is hadoop?', 'simple linear regre
ssion? do it the bayesian way', 'load data into rstudio for analysis in dsx',
'this week in data science (january 10, 2017)', 'statistical bias types expla
ined (with examples)', 'r markdown reference guide', 'statistical bias types
explained', 'analyze starcraft ii replays with jupyter notebooks', 'announcin
g dsx environments in beta!']
```

5. Use your functions from above to correctly fill in the solutions to the dictionary below. Then test your dictionary against the solution. Provide the code you need to answer each following the comments below.

In [29]:
```
# Find the user that is most similar to user 1
get_top_sorted_users(1).neighbor_id.values[0]
```

Out[29]:
```
3933
```

In [30]:
```
# Find the 10th most similar user to user 131
get_top_sorted_users(131).neighbor_id.values[10]
```

Out[30]:
```
242
```

In [31]:
```
### 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 [32]:
```
## 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.

Provide your response here. As for a new user, there is no existing data about the user, so user-user based collaborative recommendation won't work. We have to use rank based recommendations and use the function get_top_articles. If we can get some background information when the user sign up, such as education, social media (facebook, linkedin, twitter, pinterest, instagram, snapchat, etc.), geographic location, age, industry, and so on, we have find current users with similar background and interest and use user-user based collaborative recommendation.

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 [33]:
```
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 = get_top_article_ids(10) # Your recommendations here
new_user_recs=[str(x) for x in new_user_recs]
```

In [34]:
```
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 i
n this case we would want to recommend the most popular articles, because we d
on'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 wav to show off vour skills.

In [35]:
```
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 [36]:
```
# make recommendations for a brand new user


# make a recommendations for a user who only has interacted with article id '1
427.0'
```

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 [37]:
```
# Load the matrix here
user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

In [38]:
```
# quick look at the matrix
user_item_matrix.head()
```

Out[38]:

article_id	0.0	100.0	1000.0	1004.0	1006.0	1008.0	101.0	1014.0	1015.0	1016.0	...	977.0	98
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.

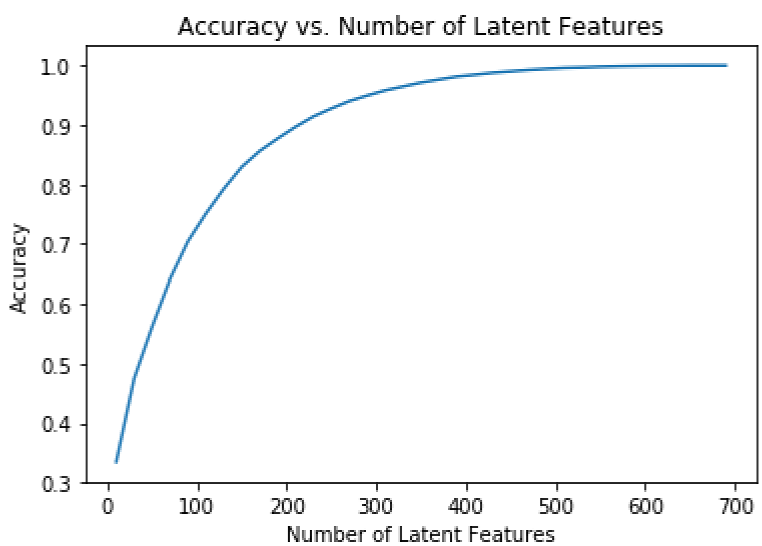
In [39]:
```
# Perform SVD on the User-Item Matrix Here

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

Provide your response here.

We use SVD here, but in the lesson, FunkSVD was used as the data contains missing values. SVD only works with no missing value, which is the case here.

3. Now for the tricky part, how do we choose the number of latent features to use? Running the below cell, you can see that as the number of latent features increases, we obtain a lower error rate on making predictions for the 1 and 0 values in the user-item matrix. Run the cell below to get an idea of how the accuracy improves as we increase the number of latent features.

In [40]:
```
num_latent_feats = np.arange(10,700+10,20)
sum_errs = []

for k in num_latent_feats:
    # restructure with k latent features
    s_new, u_new, vt_new = np.diag(s[:k]), u[:, :k], vt[:k, :]

    # take dot product
    user_item_est = np.around(np.dot(np.dot(u_new, s_new), vt_new))

    # compute error for each prediction to actual value
    diffs = np.subtract(user_item_matrix, user_item_est)

    # total errors and keep track of them
    err = np.sum(np.sum(np.abs(diffs)))
    sum_errs.append(err)


plt.plot(num_latent_feats, 1 - np.array(sum_errs)/df.shape[0]);
plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features');
```



4. From the above, we can't really be sure how many features to use, because simply having a better way to predict the 1's and 0's of the matrix doesn't exactly give us an indication of if we are able to make good recommendations. Instead, we might split our dataset into a training and test set of data, as shown in the cell below.

Use the code from question 3 to understand the impact on accuracy of the training and test sets of data with different numbers of latent features. Using the split below:

- How many users can we make predictions for in the test set?
- How many users are we not able to make predictions for because of the cold start problem?
- How many articles can we make predictions for in the test set?
- How many articles are we not able to make predictions for because of the cold start problem?

In [41]:
```
df_train = df.head(40000)
df_test = df.tail(5993)

def create_test_and_train_user_item(df_train, df_test):
    '''
    INPUT:
    df_train - training dataframe
    df_test - test dataframe

    OUTPUT:
    user_item_train - a user-item matrix of the training dataframe
                      (unique users for each row and unique articles for each
 column)
    user_item_test - a user-item matrix of the testing dataframe
                    (unique users for each row and unique articles for each co
lumn)
    test_idx - all of the test user ids
    test_arts - all of the test article ids

    '''
    # Your code here
    user_item_train=create_user_item_matrix(df_train)

    user_item_test=create_user_item_matrix(df_test)

    test_idx=user_item_test.index

    test_arts=user_item_test.columns

    return user_item_train, user_item_test, test_idx, test_arts

user_item_train, user_item_test, test_idx, test_arts = create_test_and_train_u
ser_item(df_train, df_test)
```

In [42]:
```
# users in both train and test
len(user_item_test.index.intersection(user_item_train.index))
```

Out[42]: 20

In [43]:
```
# users in test but not in train
len(np.setdiff1d(user_item_test.index, user_item_train.index))
```

Out[43]: 662

In [44]:
```
# articles in both train and test
len(user_item_test.columns.intersection(user_item_train.columns))
```

Out[44]: 574

In [45]:
```
# articles in test set but not in training set
len(np.setdiff1d(user_item_test.columns, user_item_train.columns))
```

Out[45]: 0

In [46]:
```
# 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 be
cause 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 b
ecause 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 c
annot make predictions for these users using SVD.
```

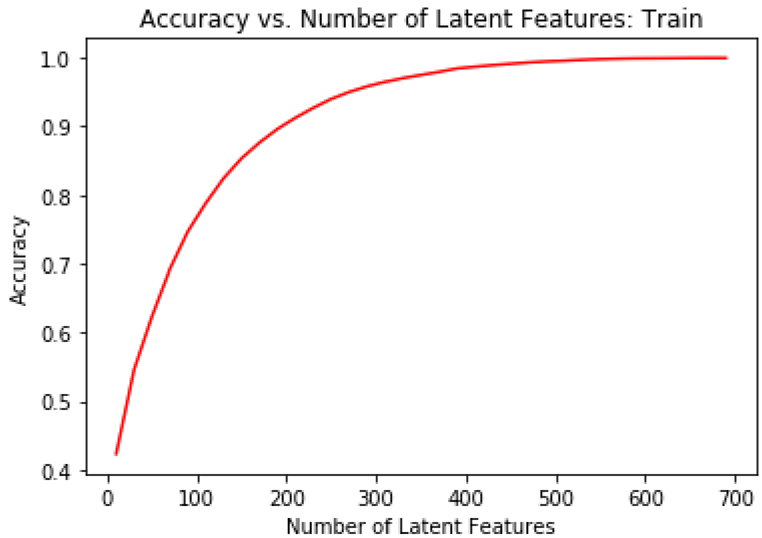
5. Now use the **user_item_train** dataset from above to find U, S, and V transpose using SVD. Then find the subset of rows in the **user_item_test** dataset that you can predict using this matrix decomposition with different numbers of latent features to see how many features makes sense to keep based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

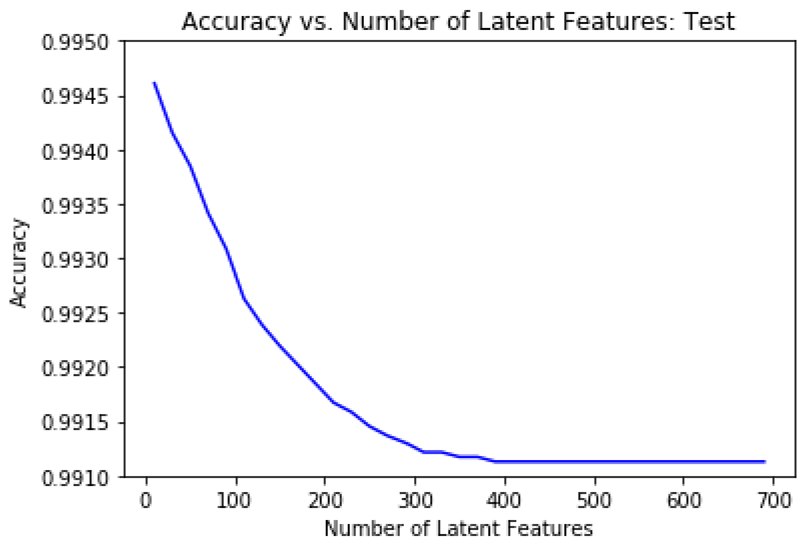
Use the cells below to explore how well SVD works towards making predictions for recommendations on the test data.

In [47]:
```
# fit SVD on the user_item_train matrix
u_train, s_train, vt_train = np.linalg.svd(user_item_train) # fit svd similar
 to above then use the cells below
```

In [48]:
```
# Use these cells to see how well you can use the training
# decomposition to predict on test data
```

In [49]:
```
num_latent_feats = np.arange(10,700+10,20)
sum_errs_train = []
sum_errs_test = []
user_item_test = user_item_test.loc[user_item_test.index.isin(user_item_train.
index), user_item_test.columns.isin(user_item_train.columns)]
u_test = u_train[user_item_train.index.isin(user_item_test.index), :]
vt_test = vt_train[:, user_item_train.columns.isin(test_arts)]


for k in num_latent_feats:
    # restructure with k latent features
    s_new_train, u_new_train, vt_new_train = np.diag(s_train[:k]), u_train[:,
:k], vt_train[:k, :]


    s_new_test, u_new_test, vt_new_test = s_new_train, u_test[:, :k], vt_test
[:k, :]

    # take dot product
    user_item_est_train = np.around(np.dot(np.dot(u_new_train, s_new_train), v
t_new_train))
    user_item_est_test = np.around(np.dot(np.dot(u_new_test, s_new_test), vt_n
ew_test))

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

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

    sum_errs_train.append(err_train)

    sum_errs_test.append(err_test)


plt.plot(num_latent_feats, 1 - np.array(sum_errs_train)/df.shape[0],color='re
d');
plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features: Train');
plt.show()

plt.plot(num_latent_feats, 1 - np.array(sum_errs_test)/df.shape[0], color='blu
e');

plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.ylim(0.991,0.995);
plt.title('Accuracy vs. Number of Latent Features: Test');
plt.show()
```





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

Your response here.

As the number of latent features increase, it is possible to overfit the model thus the prediction accuracy for the test data set decreases. But as we only have 20 users in the test set, it's hard to conlude about the prediction accuracy with so limited number of users. Cross validation will be helpful to randomly split into training and test dataset and repeat many times to see the prediction performance.

We may use A/B testing to determine the performance of the above recommendation systems. The null hypothesis is there is no difference between the way userf currently find articles and with our recommendation systems. We will need to determine the p value and compare the results to reject or accept the null hypothesis. By using cookie or user-based diversion, people will be split into control and experiment groups.

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)](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 [50]:
```
from subprocess import call
call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```

Out[50]: 0