Recommendations_with_IBM

August 5, 2020

1 Recommendations with IBM

In this notebook, you will be putting your recommendation skills to use on real data from the IBM Watson Studio platform.

You may either submit your notebook through the workspace here, or you may work from your local machine and submit through the next page. Either way assure that your code passes the project RUBRIC. **Please save regularly.**

By following the table of contents, you will build out a number of different methods for making recommendations that can be used for different situations.

1.1 Table of Contents

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

```
                                      email
0  ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1  083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2  b96a4f2e92d8572034b1e9b28f9ac673765cd074
3  06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4  f01220c46fc92c6e6b161b1849de11faacd7ccb2
```

In [2]:
```python
# Show df_content to get an idea of the data
df_content.head()
```

Out[2]:
```
                                            doc_body  \
0  Skip navigation Sign in SearchLoading...\r\n\r...
1  No Free Hunch Navigation * kaggle.com\r\n\r\n ...
2   * Login\r\n * Sign Up\r\n\r\n * Learning Pat...
3  DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...
4  Skip navigation Sign in SearchLoading...\r\n\r...

                                     doc_description  \
0  Detect bad readings in real time using Python ...
1  See the forest, see the trees. Here lies the c...
2  Heres this weeks news in Data Science and Bi...
3  Learn how distributed DBs solve the problem of...
4  This video demonstrates the power of IBM DataS...

                                       doc_full_name doc_status  article_id
0  Detect Malfunctioning IoT Sensors with Streami...       Live           0
1  Communicating data science: A guide to present...       Live           1
2         This Week in Data Science (April 18, 2017)       Live           2
3  DataLayer Conference: Boost the performance of...       Live           3
4      Analyze NY Restaurant data using Spark in DSX       Live           4
```

1.1.1 Part I : Exploratory Data Analysis

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

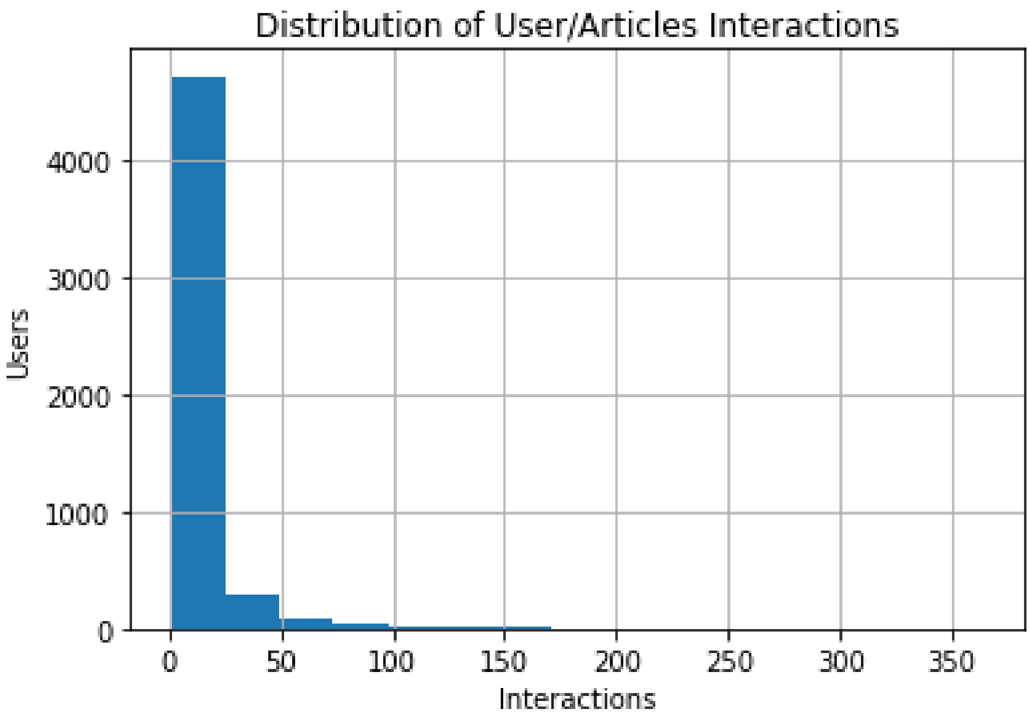
1. What is the distribution of how many articles a user interacts with in the dataset? Provide a visual and descriptive statistics to assist with giving a look at the number of times each user interacts with an article.

In [3]:
```python
# Information of Descriptive statistics of user's interactions with articles.
df['email'].value_counts().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: email, dtype: float64
```

In [4]:
```
# A visual displaying of the user's interactions with articles.
df['email'].value_counts().hist(bins = 15)
plt.title('Distribution of User/Articles Interactions')
plt.xlabel('Interactions')
plt.ylabel('Users')
plt.show()
```



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 fewer.
max_views_by_user = 364# The maximum number of user-article interactions by any 1 user i
```

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

In [6]:
```
# Finding and exploring duplicate articles
df_content[df_content.duplicated(subset=['article_id'])]
```

Out[6]:
```
                                              doc_body  \
365  Follow Sign in / Sign up Home About Insight Da...
692  Homepage Follow Sign in / Sign up Homepage * H...
```

```
761  Homepage Follow Sign in Get started Homepage *...
970  This video shows you how to construct queries ...
971  Homepage Follow Sign in Get started * Home\r\n...

                                       doc_description  \
365  During the seven-week Insight Data Engineering...
692  One of the earliest documented catalogs was co...
761  Todays world of data science leverages data f...
970  This video shows you how to construct queries ...
971  If you are like most data scientists, you are ...

                                         doc_full_name doc_status  article_id
365                         Graph-based machine learning       Live          50
692  How smart catalogs can turn the big data flood...       Live         221
761  Using Apache Spark as a parallel processing fr...       Live         398
970                                Use the Primary Index       Live         577
971  Self-service data preparation with IBM Data Re...       Live         232
```

In [7]:
```
# Remove any rows that have the same article_id - only keep the first

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

Out[8]: 714

In [9]:
```
# b. The number of unique articles in the dataset (whether they have any interactions or
df_content['article_id'].nunique()
```

Out[9]: 1051

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

Out[10]: 5148

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

Out[11]: 45993

In [13]:
```
unique_articles = 714 # The number of unique articles that have at least one interactio
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 [14]:
```
df['article_id'].value_counts().head()
```

Out[14]:
```
1429.0    937
1330.0    927
1431.0    671
1427.0    643
1364.0    627
Name: article_id, dtype: int64
```

In [15]:
```
most_viewed_article_id = '1429.0' # The most viewed article in the dataset as a string
max_views = 937 # The most viewed article in the dataset was viewed how many times?
```

In [16]:
```
## No need to change the code here - this will be helpful for later parts of the notebo
# 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[16]:

	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 [17]:
```
## If you stored all your results in the variable names above,
## you shouldn't need to change anything in this cell
```

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

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

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

1.1.2 Part II: Rank-Based Recommendations

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

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

```
In [20]: 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
             article_idx = list(df['article_id'].value_counts().index[:n])
             top_articles = []
             for i in range(n):
                 article_title = df[df.article_id == article_idx[i]]['title'].iloc[0]
                 top_articles.append(article_title)

             return top_articles

         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
        top_articles_ids = df['article_id'].value_counts()
        top_articles_ids = [str(a_id) for a_id in top_articles_ids.index[:n]]

        return top_articles_ids # Return the top article ids
```

In [21]:
```
print(get_top_articles(10))
print(get_top_article_ids(10))
```

```
['use deep learning for image classification', 'insights from new york car accident reports', 'v
['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0', '1170.0', '1162.0', '1304
```

In [22]:
```
# Test your function by returning the top 5, 10, and 20 articles
top_5 = get_top_articles(5)
top_10 = get_top_articles(10)
top_20 = get_top_articles(20)

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

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

1.1.3 Part III: User-User Based Collaborative Filtering

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

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

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

```
In [23]: # create the user-article matrix with 1's and 0's

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

             OUTPUT:
             user_item - user item matrix

             Description:
             Return a matrix with user ids as rows and article ids on the columns with 1 values
             an article and a 0 otherwise
             '''
             # Fill in the function here
             user_item = df.groupby(['user_id', 'article_id'])['title'].count().unstack()
             user_item = user_item.notnull().astype(int)

             return user_item # return the user_item matrix

         user_item = create_user_item_matrix(df)
```

```
In [24]: ## Tests: You should just need to run this cell.  Don't change the code.
         assert user_item.shape[0] == 5149, "Oops!  The number of users in the user-article matr
         assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-article ma
         assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by user 1 do
         print("You have passed our quick tests!  Please proceed!")
```

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

2. Complete the function below which should take a user_id and provide an ordered list of the most similar users to that user (from most similar to least similar). The returned result should not contain the provided user_id, as we know that each user is similar to him/herself. Because the results for each user here are binary, it (perhaps) makes sense to compute similarity as the dot product of two users.

Use the tests to test your function.

```
In [25]: def find_similar_users(user_id, user_item=user_item):
             '''
             INPUT:
             user_id - (int) a user_id
             user_item - (pandas dataframe) matrix of users by articles:
                         1's when a user has interacted with an article, 0 otherwise

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

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

    '''
    # compute similarity of each user to the provided user
    similarity = user_item.dot(user_item.loc[user_id])

    # sort by similarity
    similarity = similarity.sort_values(ascending=False)

    # create list of just the ids
    similarity = similarity.index

    # remove the own user's id
    most_similar_users = similarity.drop(user_id)
    return most_similar_users # return a list of the users in order from most to least
```

In [26]:
```
# Do a spot check of your function
print("The 10 most similar users to user 1 are: {}".format(find_similar_users(1)[:10]))
print("The 5 most similar users to user 3933 are: {}".format(find_similar_users(3933)[:
print("The 3 most similar users to user 46 are: {}".format(find_similar_users(46)[:3]))
```

```
The 10 most similar users to user 1 are: Int64Index([3933, 23, 3782, 203, 4459, 131, 3870, 46, 4
The 5 most similar users to user 3933 are: Int64Index([1, 23, 3782, 4459, 203], dtype='int64', n
The 3 most similar users to user 46 are: Int64Index([4201, 23, 3782], dtype='int64', name='user_
```

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 [27]:
```
def get_article_names(article_ids, df=df):
    '''
    INPUT:
    article_ids - (list) a list of article ids
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    article_names - (list) a list of article names associated with the list of article
                    (this is identified by the title column)
    '''
    # Your code here
    article_names = []
    for article in article_ids:
        name = df[df['article_id'] == float(article)]['title'].iloc[0]

        article_names.append(name)
```

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


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

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

    Description:
    Provides a list of the article_ids and article titles that have been seen by a user
    '''
    # Your code here
    article_ids = user_item.loc[user_id]
    article_ids = [str(a_id) for a_id in article_ids[article_ids == 1].index]
    article_names = get_article_names(article_ids)
    return article_ids, article_names # return the ids and names


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

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

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

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

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

    '''
    # Your code here
    # Find viewed articles by user
```

```
    seen_ids, seen_names = get_user_articles(user_id)

    # Find similar users
    similar_users = find_similar_users(user_id)

    recs = []
    for user in similar_users:
        article_ids, article_names = get_user_articles(user)

        not_seen = list(set(article_ids) - (set(seen_ids) & set(article_ids)))

        recs.extend(not_seen)

        if len(recs) > m:
            break

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

In [28]:
```
# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

Out[28]:
```
['from spark ml model to online scoring with scala',
 'machine learning and the science of choosing',
 'web picks (week of 4 september 2017)',
 'automating web analytics through python',
 'this week in data science (february 14, 2017)',
 'the nurse assignment problem',
 'learn basics about notebooks and apache spark',
 'working with db2 warehouse on cloud in data science experience',
 '5 practical use cases of social network analytics: going beyond facebook and twitter'
 'data tidying in data science experience',
 'deploy your python model as a restful api',
 'ml algorithm != learning machine',
 'the 3 kinds of context: machine learning and the art of the frame',
 'generalization in deep learning',
 'airbnb data for analytics: vancouver listings',
 'why even a moths brain is smarter than an ai',
 'the unit commitment problem',
 'analyze precipitation data',
 'from scikit-learn model to cloud with wml client',
 'python machine learning: scikit-learn tutorial',
 'awesome deep learning papers',
 'deep learning achievements over the past year ',
 'deep learning from scratch i: computational graphs',
 'programmatic evaluation using watson conversation',
 'this week in data science (april 25, 2017)',
 'gosales transactions for logistic regression model',
 'intents & examples for ibm watson conversation',
```

```
'shaping data with ibm data refinery',
'perform sentiment analysis with lstms, using tensorflow',
'analyze open data sets with pandas dataframes',
'brunel in jupyter',
'timeseries data analysis of iot events by using jupyter notebook',
'simple graphing with ipython and\xa0pandas',
'a tensorflow regression model to predict house values',
'analyzing data by using the sparkling.data library features',
'visualising data the node.js way',
'a dynamic duo  inside machine learning  medium',
'this week in data science (may 2, 2017)',
'using deep learning with keras to predict customer churn',
'data science for real-time streaming analytics',
'predicting churn with the spss random tree algorithm',
'accelerate your workflow with dsx',
'the power of machine learning in spark',
'improving real-time object detection with yolo',
'graph-based machine learning',
'get started with streams designer by following this roadmap',
'better together: spss and data science experience',
'times world university ranking analysis',
'discover hidden facebook usage insights',
'brunel interactive visualizations in jupyter notebooks',
'pixieapp for outlier detection',
'using github for project control in dsx',
'healthcare python streaming application demo',
'pixiedust 1.0 is here!  ibm watson data lab',
'analyze accident reports on amazon emr spark',
'using rstudio in ibm data science experience',
'spark 2.1 and job monitoring available in dsx',
'recent trends in recommender systems',
'1448    i ranked every intro to data science course on...\nName: title, dtype: object
'optimizing a marketing campaign: moving from predictions to actions',
'airbnb data for analytics: mallorca reviews',
'easy json loading and social sharing in dsx notebooks',
'learn tensorflow and deep learning together and now!',
'overlapping co-cluster recommendation algorithm (ocular)',
'using bigdl in dsx for deep learning on spark',
'54174    detect potentially malfunctioning sensors in r...\nName: title, dtype: objec
'recommender systems: approaches & algorithms',
'declarative machine learning',
'spark-based machine learning tools for capturing word meanings',
'using machine learning to predict parking difficulty',
'car performance data',
'dsx: hybrid mode',
'56594    lifelong (machine) learning: how automation ca...\nName: title, dtype: objec
'i am not a data scientist  ibm watson data lab',
'how smart catalogs can turn the big data flood into an ocean of opportunity',
```

```
 'model bike sharing data with spss',
 'small steps to tensorflow',
 'use sql with data in hadoop python',
 'deep learning with data science experience',
 'get social with your notebooks in dsx',
 'insights from new york car accident reports',
 'this week in data science (may 30, 2017)',
 'experience iot with coursera',
 'got zip code data? prep it for analytics.  ibm watson data lab  medium',
 '520    using notebooks with pixiedust for fast, flexi...\nName: title, dtype: object'
 'deep forest: towards an alternative to deep neural networks',
 'modeling energy usage in new york city',
 'machine learning for the enterprise',
 'apache spark lab, part 3: machine learning',
 'this week in data science (april 18, 2017)',
 'flightpredict ii: the sequel   ibm watson data lab',
 'what is smote in an imbalanced class setting (e.g. fraud detection)?',
 'pixiedust gets its first community-driven feature in 1.0.4',
 'variational auto-encoder for "frey faces" using keras',
 'movie recommender system with spark machine learning',
 'maximize oil company profits',
 'using brunel in ipython/jupyter notebooks',
 'ml optimization using cognitive assistant',
 'data science platforms are on the rise and ibm is leading the way',
 'challenges in deep learning',
 'use decision optimization to schedule league games',
 'twelve\xa0ways to color a map of africa using brunel',
 'markdown for jupyter notebooks cheatsheet',
 'apache spark lab, part 2: querying data',
 'brunel 2.0 preview',
 'analyze facebook data using ibm watson and watson studio',
 'higher-order logistic regression for large datasets',
 'process events from the watson iot platform in a streams python application',
 'machine learning exercises in python, part 1',
 'analyze open data sets with spark & pixiedust',
 'leverage python, scikit, and text classification for behavioral profiling',
 'fertility rate by country in total births per woman',
 'analyze energy consumption in buildings',
 'build a python app on the streaming analytics service',
 'aspiring data scientists! start to learn statistics with these 6 books!',
 'visualize data with the matplotlib library',
 '502    forgetting the past to learn the future: long ...\nName: title, dtype: object'
 'data visualization playbook: telling the data story']
```

In [29]:
```python
# Test your functions here - No need to change this code - just run this cell
assert set(get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0', '1427.0
assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing (2015): un
assert set(get_user_articles(20)[0]) == set(['1320.0', '232.0', '844.0'])
```

```
assert set(get_user_articles(20)[1]) == set(['housing (2015): united states demographic
assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1314.0', '14
assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct high-re
print("If this is all you see, you passed all of our tests!  Nice job!")
```

```
If this is all you see, you passed all of our tests!  Nice job!
```

4. Now we are going to improve the consistency of the **user_user_recs** function from above.

- Instead of arbitrarily choosing when we obtain users who are all the same closeness to a given user - choose the users that have the most total article interactions before choosing those with fewer article interactions.
- Instead of arbitrarily choosing articles from the user where the number of recommended articles starts below m and ends exceeding m, choose articles with the articles with the most total interactions before choosing those with fewer total interactions. This ranking should be what would be obtained from the **top_articles** function you wrote earlier.

In [30]:
```
def get_top_sorted_users(user_id, df=df, user_item=user_item):
    '''
    INPUT:
    user_id - (int)
    df - (pandas dataframe) df as defined at the top of the notebook
    user_item - (pandas dataframe) matrix of users by articles:
            1's when a user has interacted with an article, 0 otherwise


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

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

    '''
    # Your code here
    # User - item interactions
    user_int = df.groupby(['user_id'])['article_id'].count()

    # Number of users
    n_users = user_item.shape[0]

    # Neighbor_id column
    neighbor_id = [u_id for u_id in range(1, n_users) if u_id != user_id]

    # Similarity and num_interactions columns
```

```python
    num_interactions = []
    similarity = []
    for u_id in neighbor_id:
        num_interactions.append(user_int.loc[u_id])
        similarity.append(np.dot(user_item.loc[user_id], user_item.loc[u_id]))

    # Create neighbors_df dataframe
    neighbors_df = pd.DataFrame({'neighbor_id': neighbor_id,
                                 'similarity': similarity,
                                 'num_interactions': num_interactions})

    # Sort by similarity
    neighbors_df.sort_values('similarity', ascending=False, inplace=True)

    return neighbors_df # Return the dataframe specified in the doc_string


def user_user_recs_part2(user_id, m=10):
    '''
    INPUT:
    user_id - (int) a user id
    m - (int) the number of recommendations you want for the user

    OUTPUT:
    recs - (list) a list of recommendations for the user by article id
    rec_names - (list) a list of recommendations for the user by article title

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

    Notes:
    * Choose the users that have the most total article interactions
    before choosing those with fewer article interactions.

    * Choose articles with the articles with the most total interactions
    before choosing those with fewer total interactions.

    '''
    # Your code here
    # Get neighbors
    neighbors_df = get_top_sorted_users(user_id)

    # Top neighbors
    top_neighbors = list(neighbors_df[:m]['neighbor_id'])

    # Articles viewed by top neighbors
```

```
        recs = []
        for u_id in top_neighbors:
            article_ids = user_item.loc[u_id]
            recs.extend([str(a_id) for a_id in article_ids[article_ids == 1].index])

        # Unique values
        recs = list(set(recs[:m]))

        # Find unique article names
        rec_names = list(set(df[df['article_id'].isin(recs)]['title']))
        return recs, rec_names
```

In [31]:
```
# Quick spot check - don't change this code - just use it to test your functions
rec_ids, rec_names = user_user_recs_part2(20, 10)
print("The top 10 recommendations for user 20 are the following article ids:")
print(rec_ids)
print()
print("The top 10 recommendations for user 20 are the following article names:")
print(rec_names)
```

```
The top 10 recommendations for user 20 are the following article ids:
['981.0', '727.0', '793.0', '232.0', '1186.0', '53.0', '1053.0', '89.0', '1271.0', '495.0']

The top 10 recommendations for user 20 are the following article names:
['super fast string matching in python', 'from python nested lists to multidimensional numpy arr
```

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 [32]:
```
### 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 [33]:
```
## Dictionary Test Here
sol_5_dict = {
    'The user that is most similar to user 1.': user1_most_sim,
    'The user that is the 10th most similar to user 131': user131_10th_sim,
}

t.sol_5_test(sol_5_dict)
```

```
This all looks good!  Nice job!
```

6. If we were given a new user, which of the above functions would you be able to use to make recommendations? Explain. Can you think of a better way we might make recommendations? Use the cell below to explain a better method for new users.

Provide your response here.

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 [38]: new_user = '0.0'

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

In [39]: assert set(new_user_recs) == set(['1314.0','1429.0','1293.0','1427.0','1162.0','1364.0'

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

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

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

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

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

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

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

            OUTPUT:

            '''
```

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

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

Write an explanation of your content based recommendation system here.

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

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

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


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

1.1.8 Part V: Matrix Factorization

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

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

```
In [40]: # Load the matrix here
         user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

```
In [41]: # quick look at the matrix
         user_item_matrix.head()
```

```
Out[41]: article_id  0.0  100.0  1000.0  1004.0  1006.0  1008.0  101.0  1014.0  1015.0  \
         user_id
         1           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         2           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         3           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         4           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         5           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0

         article_id  1016.0  ...   977.0  98.0  981.0  984.0  985.0  986.0  990.0  \
         user_id             ...
         1              0.0  ...     0.0   0.0    1.0    0.0    0.0    0.0    0.0
         2              0.0  ...     0.0   0.0    0.0    0.0    0.0    0.0    0.0
         3              0.0  ...     1.0   0.0    0.0    0.0    0.0    0.0    0.0
         4              0.0  ...     0.0   0.0    0.0    0.0    0.0    0.0    0.0
         5              0.0  ...     0.0   0.0    0.0    0.0    0.0    0.0    0.0

         article_id  993.0  996.0  997.0
         user_id
         1             0.0    0.0    0.0
```

```
2              0.0    0.0    0.0
3              0.0    0.0    0.0
4              0.0    0.0    0.0
5              0.0    0.0    0.0

[5 rows x 714 columns]
```

2. In this situation, you can use Singular Value Decomposition from numpy on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

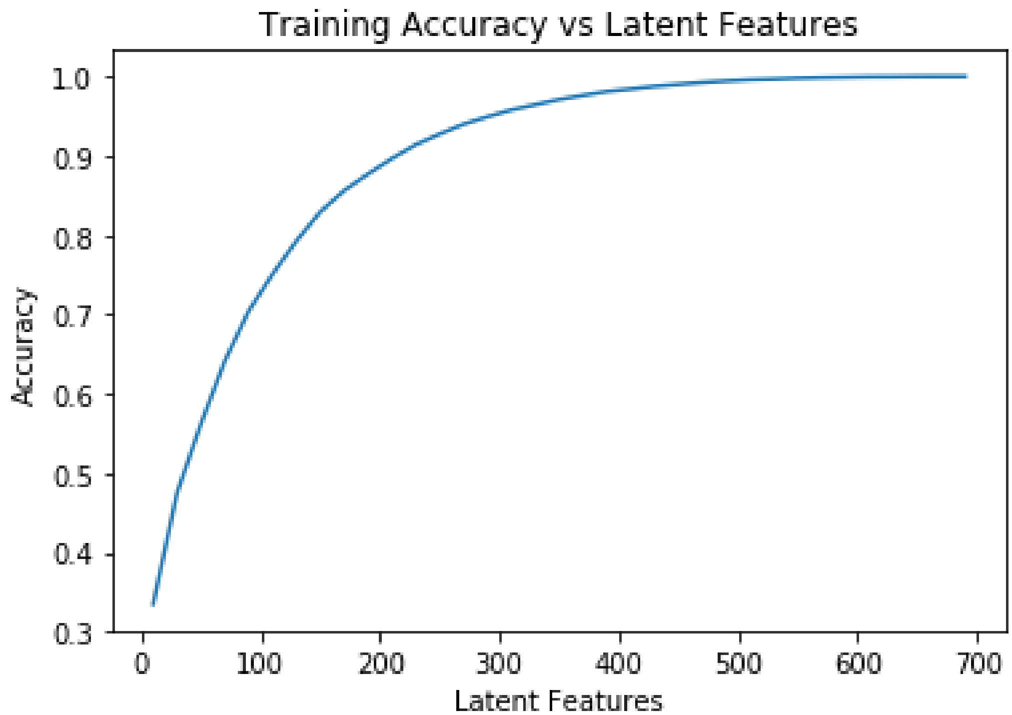
In [42]:
```
# Perform SVD on the User-Item Matrix Here

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

# Checking matrices dimensions
u.shape, s.shape, vt.shape
```

Out[42]:
```
((5149, 5149), (714,), (714, 714))
```

Provide your response 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 [43]:
```
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('Latent Features');
plt.ylabel('Accuracy');
plt.title('Training Accuracy vs 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 [44]: df_train = df.head(40000)
         df_test = df.tail(5993)

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

             OUTPUT:
             user_item_train - a user-item matrix of the training dataframe
```

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

    '''
    # Your code here
    # User-item matrix of the training dataframe
    user_item_train = create_user_item_matrix(df_train)

    # User-item matrix of the testing dataframe
    user_item_test = create_user_item_matrix(df_test)

    # Test user ids
    test_idx = user_item_test.index
    test_idx = list(set(test_idx))

    # Test article ids
    test_arts = user_item_test.columns
    test_arts = list(set(test_arts))

    return user_item_train, user_item_test, test_idx, test_arts

user_item_train, user_item_test, test_idx, test_arts = create_test_and_train_user_item(
```

In [45]:
```
# How many users can we make predictions for in the test set?
len(np.intersect1d(df_train['user_id'].unique(),df_test['user_id'].unique()))
```

Out[45]: 20

In [46]:
```
# How many users in the test set are we not able to make predictions for because of the
len(df_test['user_id'].unique()) - len(np.intersect1d(df_train['user_id'].unique(),df_t
```

Out[46]: 662

In [47]:
```
# How many articles can we make predictions for in the test set?
len(np.intersect1d(df_train['article_id'].unique(),df_test['article_id'].unique()))
```

Out[47]: 574

In [48]:
```
# How many articles in the test set are we not able to make predictions for because of
len(df_test['article_id'].unique()) - len(np.intersect1d(df_train['article_id'].unique(
```

Out[48]: 0

In [57]:
```
a = 662
b = 574
c = 20
```

```
d = 0


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

t.sol_4_test(sol_4_dict)
```

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

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

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

```
In [52]: # fit SVD on the user_item_train matrix
         u_train, s_train, vt_train = u_train, s_train, vt_train = np.linalg.svd(user_item_train
```

```
In [53]: # Use these cells to see how well you can use the training
         # decomposition to predict on test data
         # Users and articles in user_item_train matrix
         train_idx = np.array(user_item_train.index)
         train_arts = np.array(user_item_train.columns)

         # Users and articles of test set in training set
         test_idx_set = np.intersect1d(test_idx, train_idx)
         test_arts_set = np.intersect1d(test_arts, train_arts)

         # Users and articles positions of test subset in training matrix
         train_indexes = np.where(np.in1d(train_idx, test_idx_set))[0]
         train_articles = np.where(np.in1d(train_arts, test_arts_set))[0]

         # Users positions of test subset in test matrix
         test_indexes = np.where(np.in1d(test_idx, test_idx_set))[0]
```

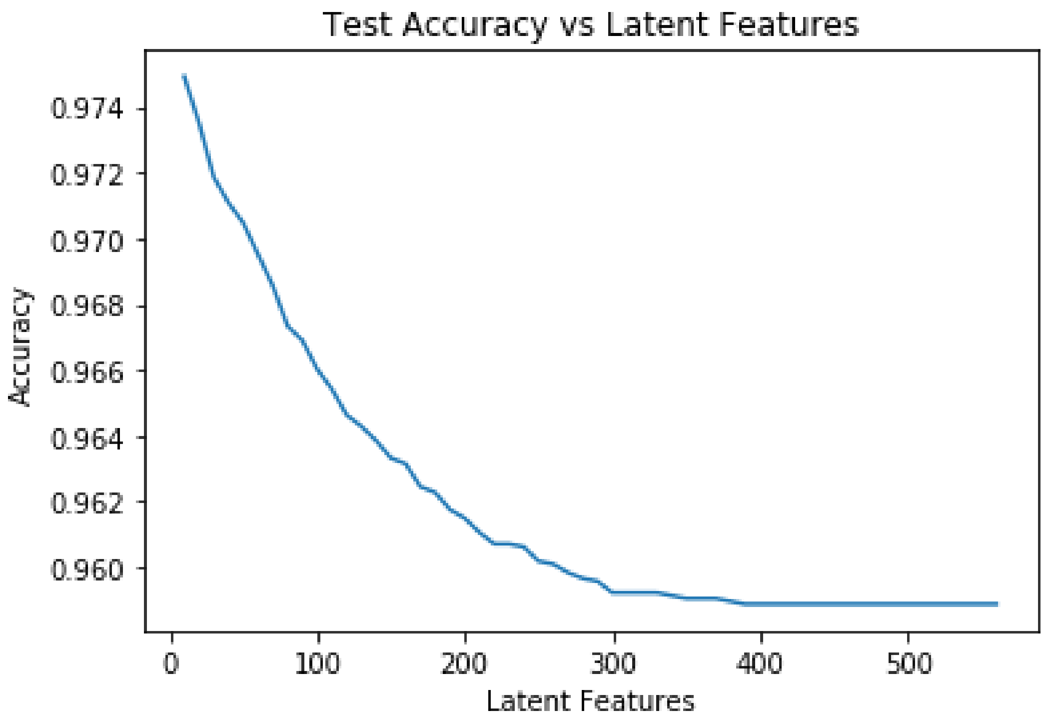
```
In [54]: # FIND SUBSET OF USER_ITEM MATRIX CONTAINING ONLY USER AND ARTICLES THAT ARE SHARED BY
         u_item_test_set = user_item_test.iloc[test_indexes,:]
         u_item_train_set = user_item_train.iloc[train_indexes, train_articles]
```

```
In [55]: latent_feats = np.arange(10,570,10)
         sum_errors = []
```

```
for k in latent_feats:

    # Restructure train matrices using k features
    s_train_k, u_train_k, vt_train_k = np.diag(s_train[:k]), u_train[:, :k], vt_train[:

    # Restructure test matrices using k features
    s_test_k, u_test_k, vt_test_k = s_train_k, u_train_k[train_indexes,:], vt_train_k[:

    # Calculate dot product
    u_item_test_set_pred = np.around(np.dot(np.dot(u_test_k, s_test_k), vt_test_k))

    # Error (prediction - actual values)
    error = np.subtract(u_item_test_set, u_item_test_set_pred)

    # Total errors
    total_error = np.sum(np.sum(np.abs(error)))
    sum_errors.append(total_error)
```

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?

```
In [56]: # Plot test accuracy and latent features
         n_interactions_set = u_item_test_set.shape[0] * u_item_test_set.shape[1]
         plt.plot(latent_feats, 1 - np.array(sum_errors) / n_interactions_set);
         plt.xlabel('Latent Features');
         plt.ylabel('Accuracy');
         plt.title('Test Accuracy vs Latent Features');
```



Your response here.

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

1.2 Conclusion

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

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

1.3 Directions to Submit

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

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

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

```
In [58]: from subprocess import call
         call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])

Out[58]: 0
```