Recommendations_with_IBM

January 1, 2021

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 [ ]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import project_tests as t
        import pickle
        import seaborn as sns
        from sklearn.metrics import accuracy_score
        %matplotlib inline

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

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

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

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 [ ]: # group article count by user
        df.groupby('email').count().head()
```

```
In [ ]: #descriptive statistics
        df.groupby('email')['article_id'].count().describe()
```

The maximum article views per user is 364, whereas minimun is 1. Mean of the article_user interactions is 9. 50% of individuals interact with 3.0 articles or fewer.

```
In [ ]: # distribution of article count per user
        # set seaborn styles
        sns.set_style("whitegrid", {"axes.facecolor": "1"})
        plot_size = (12,8)
        fig, ax = plt.subplots(figsize=plot_size)

        sns.distplot(df.groupby('email').count()['article_id'], bins=50,
                     kde=True, color='b', rug=False, ax=ax);
        plt.title("Article user interaction distribution")
        sns.set_context("talk")
        ax.set(xlabel='Number of article read', ylabel='Number of user')

        plt.show()
```

```
In [ ]: # 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 [ ]: # Find and explore duplicate articles
        article_count = df_content.article_id.value_counts(dropna=False)
        dup_articles = article_count[article_count > 1]
        print('number of duplicate articles is: ', len(dup_articles))
```

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

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 [ ]: unique_articles = df.article_id.nunique()
        print('The number of unique articles that have at least one interaction is {}'.format(un
        total_articles = df_content.article_id.count()
        print('The number of unique articles on the IBM platform is {}'.format(total_articles))
        unique_users = df.email.nunique()
        print('The number of unique users is {}'.format(unique_users))
        user_article_interactions = df.shape[0]
        print('The number of user-article interactions is {}'.format(user_article_interactions))
```

```
In [ ]: unique_articles = unique_articles # The number of unique articles that have at least one
        total_articles = df_content.article_id.count() # The number of unique articles on the IB
        unique_users = df.email.nunique()# The number of unique users
        user_article_interactions = df.shape[0] # The number of user-article interactions
```

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

```
In [ ]: df_by_article = df.groupby('article_id').count()
        df_by_article.sort_values('title',ascending=False).head()
```

```
In [ ]: most_viewed_article_id = df_by_article.sort_values('title',ascending=False)['title'].ind
        max_views = df_by_article.sort_values('title',ascending=False)['title'].values[0] # The
```

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

```
In [ ]: ## 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_ar
    '`The maximum number of user-article interactions by any 1 user is ______.`': max_vi
    '`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_articl
    '`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)
```

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 [ ]: 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 = df['title'].value_counts().sort_values(ascending=False).head(n).index

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

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

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

            '''
```

```
    # Your code here
    top_articles = list(df.groupby('article_id').size().sort_values(ascending=False).hea
    return top_articles # Return the top article ids
```

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

```
In [ ]: # 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)
```

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 [ ]: # 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 u
            an article and a 0 otherwise
            '''
            # Fill in the function here
```

```
    user_item = df.groupby('user_id')['article_id'].value_counts().unstack().fillna(0)
    user_item[user_item > 1] = 1

    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
```

In [ ]:
```
## 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 matri
assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-article mat
assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by user 1 doe
print("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 [ ]:
```
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 = pd.DataFrame(np.dot(user_item[user_item.index == user_id],user_item.T),

    # sort by similarity
    similarity = similarity.sort_values(0,axis=1, ascending=False)
    # create list of just the ids and # remove the own user's id
    most_similar_users = similarity.columns.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 s
```

```python
In [ ]: # 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]))
```

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.

```python
In [ ]: 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 i
                            (this is identified by the title column)
            '''
            # Find the names of the articles by modifying the list of articles and names and sub
            article_list = df.drop('user_id',axis=1)
            article_list = article_list.drop_duplicates()
            #article_list = article_list.reset_index()
            article_names = article_list[article_list['article_id'].isin(article_ids)]['title'].

            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 i
                            (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
            '''
            # get the articles ids by selecting the columns that are = 1 for the selected user_i
            article_ids = user_item.columns[user_item.loc[user_id] == 1].tolist()
            article_ids = list(map(str, article_ids))

            # find the names of the articles using the function defined before
```

```python
        article_names = get_article_names(article_ids, df=df)

        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 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

        '''

        # Get user articles
        article_ids, _ = get_user_articles(user_id)
        # Find similar users
        most_similar_users = find_similar_users(user_id)
        # How many users for progress bar
        n_users = len(most_similar_users)
        recs = []
        for user in most_similar_users:
            # Get user articles
            ids, _ = get_user_articles(user)
            article_not_seen = np.setdiff1d(np.array(ids), np.array(article_ids))
            article_not_recs = np.setdiff1d(article_not_seen, np.array(recs))
            recs.extend(list(article_not_recs))
            # If there are more than
            if len(recs) > m:
                break
        recs = recs[:m]
        return recs
```

```
In [ ]: # Check Results
```

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

In [ ]:
```
# 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): uni
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', '142
assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct high-res
print("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 [ ]:
```
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 intera
                    highest of each is higher in the dataframe

    '''
    # Your code here
    # similarity
    user_vector = np.array(user_item.loc[user_id]).reshape(-1, 1)
    Matrix_item = user_item.drop(user_id)
    similarity = np.dot(Matrix_item.values, user_vector)
    # sort by similarity
    df_smly = pd.DataFrame({'neighbor_id': Matrix_item.index,
                            'similarity': similarity.flatten()})
```

```
    # Number of interaction
    count_inter = df.groupby('user_id')['article_id'].count()
    df_inter = pd.DataFrame({'neighbor_id': count_inter.index,
                             'num_interactions': count_inter.values})

    # Merging the two dataframes
    neighbors_df = df_smly.merge(df_inter)

    # sort the neighbors_df
    neighbors_df.sort_values(by=['similarity', 'num_interactions'],
                             inplace=True, 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.

    '''
    recs = []
    top_df = get_top_sorted_users(user_id)
    uid_list = top_df['neighbor_id'].values.tolist()
    recs = []
    name_ids = []

    exp_article_ids = list(set(df[df['user_id'] == user_id]['article_id'].values.tolist(

    for uid in uid_list:
```

```
                recs += df[df['user_id'] == uid]['article_id'].values.tolist()

            recs = list(set(recs))
            recs = [ x for x in recs if x not in exp_article_ids ]

            rec_all = df[df.article_id.isin(recs)][['article_id','title']].drop_duplicates().hea
            recs = rec_all['article_id'].values.tolist()
            rec_names = rec_all['title'].values.tolist()

            return recs, rec_names

            rec_names = get_article_names(recs)
            return recs, rec_names
```

In [ ]:
```
# 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("The top 10 recommendations for user 20 are the following article names:")
print(rec_names)
```

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 [ ]:
```
print(get_top_sorted_users(131).head(10))
```

In [ ]:
```
### Tests with a dictionary of results
user1_most_sim = get_top_sorted_users(1).neighbor_id.values[0]
user131_10th_sim = get_top_sorted_users(131).neighbor_id.values[9]
```

In [ ]:
```
## 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)
```

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.

For a new user, we can use get_top_articles function to suggest top articles

We can try content based recommendations. For eaxample, the most trendy or attactive aricles have more chances to read by new user.

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 [ ]: 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 = list(map(str,get_top_article_ids(10)))# Your recommendations here
```

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

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

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

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

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

```
In [ ]: print(s.shape, u.shape, vt.shape)
```

Provide your response here.

In the lectures, we saw that we have not use all the latent features where as here we are using all.

For this example there is no missing values, so no imputation needed like previous lesson.

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 [ ]: 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 [ ]: 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_idx = list(user_item_train.index)  # test user ids
    test_arts = list(user_item_train.columns)  # test article ids

    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(d
```

In [ ]:
```
user_item_test.describe()
```

In [ ]:
```
user_item_train.shape
```

In [ ]:
```
len(np.setdiff1d(user_item_test.index, user_item_train.index))
```

In [ ]:
```
len(np.setdiff1d(user_item_test.columns, user_item_train.columns))
```

In [ ]:
```
# 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, # letter here,
    'How many users in the test set are we not able to make predictions for because of t
    'How many movies can we make predictions for in the test set?': b,# letter here,
    'How many movies in the test set are we not able to make predictions for because of
}

t.sol_4_test(sol_4_dict)
```

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 [ ]:
```
# fit SVD on the user_item_train matrix
u_train, s_train, vt_train = np.linalg.svd(np.array(user_item_train, dtype='int'), full_
```

In [ ]:
```
u_train.shape, s_train.shape, vt_train.shape
```

In [ ]:
```
# Subset of rows in the user_item_test dataset that you can predict

#  Rows that match the test set
test_idx = user_item_test.index
row_idx = user_item_train.index.isin(test_idx)
u_test = u_train[row_idx, :]

#  Columns that match the test set
test_column = user_item_test.columns
col_idx = user_item_train.columns.isin(test_column)
vt_test = vt_train[:, col_idx]

train_idx = user_item_train.index
row_idx_2 = user_item_test.index.isin(train_idx)
subset_user_item_test = user_item_test.loc[row_idx_2]
```

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

all_errs, train_errs, test_errs = [], [], []


for k in lat_features:
    # restructure with k latent features
    s_train_lat, u_train_lat, vt_train_lat = np.diag(s_train[:k]), u_train[:, :k], vt_tr

    u_test_lat, vt_test_lat = u_test[:, :k], vt_test[:k, :]

    # take dot product
    user_item_train_preds = np.around(np.dot(np.dot(u_train_lat, s_train_lat), vt_train_
    user_item_test_preds = np.around(np.dot(np.dot(u_test_lat, s_train_lat), vt_test_lat

    all_errs.append(1 - ((np.sum(user_item_test_preds)+np.sum(np.sum(subset_user_item_te

    # compute prediction accuracy
    train_errs.append(accuracy_score(user_item_train.values.flatten(), user_item_train_p
    test_errs.append(accuracy_score(subset_user_item_test.values.flatten(), user_item_te


plt.figure()
plt.plot(lat_features, all_errs, label='All Errors')
plt.plot(lat_features, train_errs, label='Train')
plt.plot(lat_features, test_errs, label='Test')
plt.xlabel('Number of Latent Features')
plt.ylabel('Accuracy')
plt.title('Plot of Accuracy vs. Number of Latent Features')
plt.legend()
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.

In trining, the accuracy increases to near 100% as the number of latent features increases. However, in the test data the model works differently. It appears that the accuracy is plummiting as the number of Latent Features increases. This is a case of overfitting, which means the trained model is not generalised.To overcome this issue, we could try followings * incorporate more data, which will increase the varinace of the data and will help to diversify the input space. * we could try n fold cross validation. * We could regularise the model while traininig.

We could also try an A/B test on the plateform by grouping the audicence in different blocks and applying them different types of recommandations algorithms and see which of the algorithms gives the best recommandations to the users.

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