anime_recs

April 24, 2022

1 Anime Recommendations System

Group partners' names:

- Soumeng Chea
- Fay Feghali
- Erina Kitamura
- Kristine Umeh

1.1 Data Pre-Processing - MyAnimeList Data

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import ast
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
from fuzzywuzzy import process
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import *
from statistics import mean
from sklearn.preprocessing import StandardScaler
from pandas.core.common import SettingWithCopyWarning
import warnings
```

1.1.1 Preprocess User Data

Following are the code used to preprocess the user_score_data.csv and user_favorited_data which are originally derived from user_data.csv. This section was commented out and data was exported into a csv since it takes a while to execute.

[2]:
```
# user_df = pd.read_csv('./datasets/user_data.csv')
# user_df.insert(0, 'user_id', range(1, 1 + len(user_df)))
# user_watched = user_df[['user_id', 'watched']]

# import ast
# user_data = []
```

```
# for i in range(len(user_df)):
#     row = user_watched.iloc[i].watched
#     row = row.strip('][').split('}, ')
#     for item in row:
#         row_dict = {}
#         if (item[-1] != "}"):
#             item = item + "}"
#         item_dict = ast.literal_eval(item)
#         row_dict['user_id'] = user_watched.iloc[i].user_id
#         row_dict['mal_id'] = item_dict['mal_id']
#         row_dict['rating'] = item_dict['score']
#         user_data.append(row_dict)

# df_user_data = pd.DataFrame(user_data)
# df_user_data.to_csv('user_score_data')
```

[3]:
```
# user_df = pd.read_csv('./datasets/user_data.csv')
# user_df.insert(0, 'user_id', range(1, 1 + len(user_df)))
# user_favorites = user_df[['user_id', 'favorites']]

# import ast
# import re
# user_data = []

# for i in range(len(user_df)):
#     row = user_favorites.iloc[i].favorites
#     row_dict = ast.literal_eval(row)
#     favorites_lst = row_dict['anime']
#     mal_ids = []
#     for item in favorites_lst:
#         before, key, after = row.partition("mal_id': ")
#         mal_ids = re.findall(r'\b\d+\b', after)
#     for mal_id in mal_ids:
#         row_dict = {}
#         row_dict['user_id'] = user_favorites.iloc[i].user_id
#         row_dict['mal_id'] = mal_id
#         row_dict['favorited'] = 1
#         user_data.append(row_dict)

# df_user_favorite_data = pd.DataFrame(user_data)
# df_user_favorite_data.to_csv('user_favorited_data')
```

[4]:
```
user_rating_data_df = pd.read_csv('./datasets/user_score_data.csv',␣
↪usecols=['user_id', 'mal_id', 'rating'],
                                  dtype={'user_id':'int32', 'mal_id':'int32',␣
↪'rating':'float32'})
```

```
user_favorite_data_df = pd.read_csv('./datasets/user_favorited_data.csv',␣
↪usecols=['user_id', 'mal_id', 'favorited'],
                                   dtype={'user_id':'int32', 'mal_id':'int32',␣
↪'rating':'int32'})
user_data_df = user_rating_data_df
```

[5]:
```
user_rate_fave_df = pd.concat([user_rating_data_df, user_favorite_data_df],␣
↪axis=0)
user_rate_fave_df.favorited = user_rate_fave_df.favorited.fillna(0)
```

[6]:
```
animes_df = pd.read_csv('./datasets/anime_data.csv', usecols=['mal_id',␣
↪'title'],
                        dtype={'mal_id':'int32', 'title':'string'})
```

1.2 Linear Regression

Not all users will rate every anime. Therefore, there are missing data in the ratings of animes. To have a better prediction, linear regression can be used to generate predictions of missing data based on existing values.

[7]:
```
def getTestTrainData(y):
    test_data = y[y['rating'].isna()]
    train_data = y.dropna(subset=['rating'])

    y_train = train_data['rating']
    X_train = train_data.drop('rating', axis=1)
    return test_data, train_data, y_train, X_train
```

[8]:
```
def fillMissingRatingDataLinReg(y):
    test_data, train_data, y_train, X_train = getTestTrainData(y)
    lin_model = LinearRegression().fit(X_train, y_train)

    X_test = test_data.drop('rating', axis=1)
    # case for no data to predict
    if (len(X_test.index) == 0):
        print("no missing data to replace")
        return y

    y_pred = lin_model.predict(X_test)

    with warnings.catch_warnings():
        warnings.filterwarnings('ignore', category=pd.core.common.
↪SettingWithCopyWarning)
        test_data.loc[test_data.rating.isna(), 'rating'] = y_pred

    new = pd.concat([test_data, train_data], axis=0).sort_values(by=['mal_id'],␣
↪ascending=True)
```

```
    new.rename(columns={'rating':'rating'}, inplace=True)

    return new
```

[9]:
```
def getComprehensiveUserRating(user_data_df, user_id):
    '''
        Takes user data and fills missing data based on linear regression
        using collaborative anime rating. Predicts what user of specified
        id will rate each anime.
    '''
    # get all user rating
    y = (user_data_df[user_data_df['user_id'] == user_id])
    y = y.drop(columns=['user_id'])

    comprehensive_df = fillMissingRatingDataLinReg(y)

    return comprehensive_df
```

Tables below show initial ratings for user with user_id of 3 and its predicted ratings using linear regression.

[10]:
```
print("Initial User Data for user_id 10")
old = user_rate_fave_df.loc[user_rate_fave_df.user_id == 10].
 ↪sort_values(by=['mal_id'], ascending=True).head(5)
old[["mal_id", "rating", "favorited"]]
```

Initial User Data for user_id 10

[10]:

	mal_id	rating	favorited
4568	1	9.0	0.0
97	3	NaN	1.0
4569	5	6.0	0.0
5107	6	6.0	0.0
106	11	NaN	1.0

[11]:
```
new = getComprehensiveUserRating(user_rate_fave_df, 10)
print("User Data for user_id 10 With Predicted Ratings")
new.head(5)
```

User Data for user_id 10 With Predicted Ratings

[11]:

	mal_id	rating	favorited
4568	1	9.000000	0.0
97	3	5.992471	1.0
4569	5	6.000000	0.0
5107	6	6.000000	0.0
106	11	5.992329	1.0

1.3 K-Nearest Neighbors

K-nearest neighbors can be used to generate recommendation based on specified anime. Using collaborative filtering, k-nearest neighbors will search for what other animes were enjoyed by other users who also enjoyed watching the specified anime.

[12]:
```
pip install fuzzywuzzy
```

```
Requirement already satisfied: fuzzywuzzy in
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages
(0.18.0)
WARNING: You are using pip version 22.0.3; however, version 22.0.4 is
available.
You should consider upgrading via the
'/Library/Frameworks/Python.framework/Versions/3.8/bin/python3 -m pip install
--upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

[13]:
```
animes_users = user_data_df.pivot(index='mal_id', columns='user_id',␣
↪values='rating').fillna(0)
animes_users_mat = csr_matrix(animes_users.values)
```

[14]:
```
model_knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20)
model_knn.fit(animes_users_mat)
```

[14]:
```
NearestNeighbors(algorithm='brute', metric='cosine', n_neighbors=20)
```

[15]:
```
def getRecommendations(movie_title, data_matrix, animes_df, model_knn,␣
↪n_recommendations):
    model_knn.fit(data_matrix)
    anime_index = process.extractOne(movie_title, animes_df['title'])[2]
    distances, indices = model_knn.kneighbors(data_matrix[anime_index],␣
↪n_neighbors=n_recommendations)
    for i in indices:
        print(animes_df['title'][i].where(i != anime_index))
```

[16]:
```
getRecommendations('Bleach', animes_users_mat, animes_df, model_knn, 5)
```

```
3990                                                  <NA>
6198    Iizuka-senpai x Blazer: Ane Kyun! yori The Ani…
5435                              Kanashimi no Belladonna
3093    New Mobile Report Gundam Wing: Frozen Teardrop…
3295                                       Plastic Little
Name: title, dtype: string
```

1.4 K-Mean Clustering

[17]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans
import pickle
import sys
from sys import exc_info
```

Loading user_score_data dataset Source and Tutorial: https://asdkazmi.medium.com/ai-movies-recommendation-system-with-clustering-based-k-means-algorithm-f04467e02fcd

[18]:
```
user_rating = pd.read_csv('./datasets/user_score_data.csv', usecols=['user_id',␣
 ↪'mal_id', 'rating'],
                                  dtype={'user_id':'int32', 'mal_id':'int32',␣
 ↪'rating':'float32'})
```

Pick only data user that have 4+ rating

[19]:
```
user_rating = user_rating[user_rating['rating'] >= 4.0]
users_list = np.unique(user_rating['user_id'])[:100]
ratings = user_rating.loc[user_rating['user_id'].isin(users_list)]
```

Create new dataframe after the filtering

[20]:
```
fav_movies = ratings.loc[:, ['user_id', 'mal_id']]
```

Prep for Sparse Matrix

[21]:
```
fav_movies = ratings.reset_index(drop = True)
fav_movies.T
```

[21]:
```
               0       1        2        3      4      5        6      7    \
user_id      1.0     1.0      1.0      1.0    1.0    1.0      1.0    1.0
mal_id   29978.0  2467.0  28789.0  34881.0  101.0  713.0  36032.0  656.0
rating       6.0    10.0      6.0      6.0   10.0    8.0      8.0    5.0

              8        9   …   30359    30360    30361    30362    30363  \
user_id     1.0      1.0  …   100.0    100.0    100.0    100.0    100.0
mal_id   1485.0  17901.0  …  4224.0  33352.0  10015.0  15489.0  21595.0
rating     10.0      6.0  …    10.0     10.0      9.0      7.0      7.0

           30364   30365    30366   30367   30368
user_id    100.0   100.0    100.0   100.0   100.0
mal_id   16576.0  1195.0  11319.0  1840.0  3712.0
rating       8.0    10.0      9.0    10.0     9.0

[3 rows x 30369 columns]
```

[22]:
```
fav_movies.to_csv('./datasets/filtered_ratings.csv')
```

[23]:
```
def userMovieList(users, users_data):
    # users = a list of user_ids
    # users_data = a dataframe of users and mal IDs and their rating
    users_list = []
    for user in users:
        users_list.append(str(list(users_data[users_data['user_id'] ==␣
 ↪user]['mal_id'])).split('[')[1].split(']')[0])
    return users_list
```

[24]:
```
user = np.unique(fav_movies['user_id'])
users_list = userMovieList(user, fav_movies)
```

Perform Sparse Matrix on the dataset

[25]:
```
def prepMatrix(listStr):
    # list_of_str = A list, which contain strings of users favourite movies␣
 ↪separate by comma ",".
    countVec = CountVectorizer(token_pattern = r'[^\,\ ]+', lowercase = False)
    sm = countVec.fit_transform(listStr)
    return sm.toarray(), countVec.get_feature_names()
```

[26]:
```
sm, feature_names = prepMatrix(users_list)
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/sklearn/utils/deprecation.py:87: FutureWarning: Function
get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will
be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

[27]:
```
sparseMatrix = pd.DataFrame(sm, index = user, columns = feature_names)
sparseMatrix.head(10)
```

[27]:

	1	100	1000	10012	10015	10017	1002	10020	10029	1003	…	996	\
1	0	0	0	0	0	0	0	0	0	0	…	0	
2	1	0	0	0	0	0	0	1	0	1	…	0	
3	1	0	0	0	0	0	0	1	0	0	…	0	
4	0	0	0	0	0	0	0	0	0	0	…	0	
5	0	0	0	0	0	0	0	0	0	0	…	0	
6	1	0	0	0	0	0	0	0	0	0	…	0	
7	1	0	0	0	0	0	0	0	1	0	…	0	
8	0	0	0	0	0	0	0	0	0	0	…	0	
9	0	0	0	0	0	0	0	0	0	0	…	0	
10	1	0	0	0	0	0	0	0	0	0	…	0	

	9963	9969	997	9981	9982	9988	9989	9996	9999
1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	1	0	0
3	0	0	0	0	0	0	1	0	0

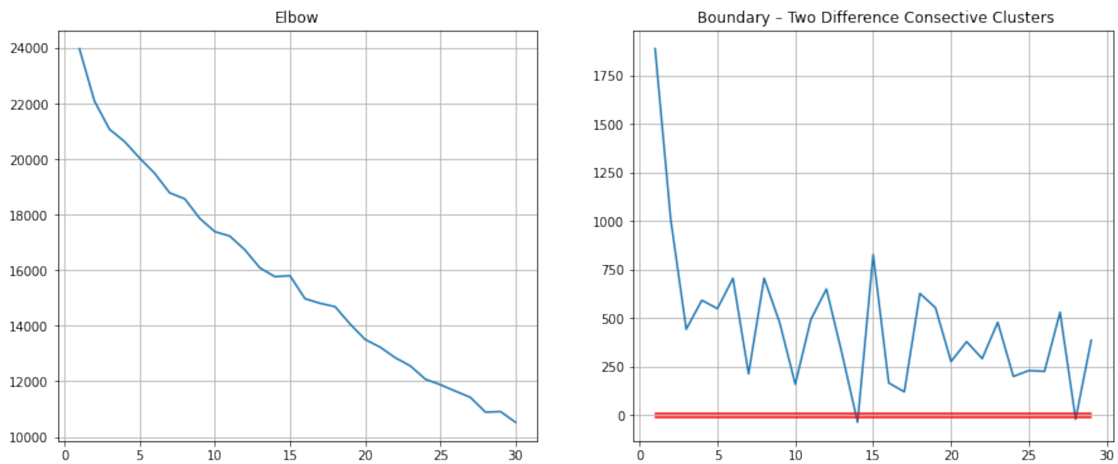
```
4       0      0     0      0      0      0      0      0      0
5       0      0     0      0      0      0      1      0      0
6       0      1     0      0      0      0      1      0      0
7       0      0     0      0      0      0      0      1      0
8       0      0     0      0      0      0      1      0      0
9       0      0     0      0      0      0      0      0      0
10      0      1     0      0      0      0      1      0      0

[10 rows x 4914 columns]
```

Plot for cluster elbor and boudary for optimal n_cluster

[28]:
```python
class method():
    def __init__(self, sm):
        self.sm = sm
        self.wcss = list()
        self.dif = list()
    def run(self, init, upto, max_iterations = 300):
        for _ in range(init, upto + 1):
            kmeans = KMeans(n_clusters = _, init = 'k-means++', max_iter = max_iterations, n_init = 5, random_state = 99)
            kmeans.fit(sm)
            self.wcss.append(kmeans.inertia_)
        self.dif = list()
        for i in range(len(self.wcss) - 1):
            self.dif.append(self.wcss[i] - self.wcss[i + 1])
    def showPlot(self, boundary = 500, up_cluster = None):
        if up_cluster is None:
            WCSS = self.wcss
            DIFF = self.dif
        else:
            WCSS = self.wcss[:up_cluster]
            DIFF = self.dif[:up_cluster - 1]
        plt.figure(figsize=(15, 6))
        plt.subplot(121).set_title('Elbow')
        plt.plot(range(1, len(WCSS) + 1), WCSS)
        plt.grid(b = True)
        plt.subplot(122).set_title(' Boundary - Two Difference Consective Clusters')
        len_dif = len(DIFF)
        X_dif = range(1, len_dif + 1)
        plt.plot(X_dif, DIFF)
        plt.plot(X_dif, np.ones(len_dif) * boundary, 'r')
        plt.plot(X_dif, np.ones(len_dif) * (-boundary), 'r')
        plt.grid()
        plt.show()
```

[29]:
```python
elbow = method(sm)
elbow.run(1, 30)
elbow.showPlot(boundary = 10)
```



K-Mean Clustering Table Fitting

[30]:
```python
kmeans = KMeans(n_clusters = 14, init = 'k-means++', max_iter = 300, n_init =␣
↪10, random_state = 99)
clusters = kmeans.fit_predict(sm)
```

[31]:
```python
users_cluster = pd.DataFrame(np.concatenate((user.reshape(-1, 1), clusters.
↪reshape(-1, 1)), axis = 1), columns = ['user_id', 'Cluster'])
users_cluster.T
```

[31]:
```
          0   1   2   3   4   5   6   7   8   9  …  90  91  92  93  94  95  \
user_id   1   2   3   4   5   6   7   8   9  10  …  91  92  93  94  95  96
Cluster   9  12   8   1   1   1   1   1   3   7  …   3   3   1   1   1   1

         96  97  98   99
user_id  97  98  99  100
Cluster   1   1   1    1

[2 rows x 100 columns]
```

[32]:
```python
for _ in range(14):
    user = users_cluster[users_cluster['Cluster'] == _].shape[0]
    print('User within the cluster ' + str(_) + ' =', user)
```

```
User within the cluster 0 = 1
User within the cluster 1 = 70
User within the cluster 2 = 1
User within the cluster 3 = 18
```

```
User within the cluster 4 = 1
User within the cluster 5 = 1
User within the cluster 6 = 1
User within the cluster 7 = 1
User within the cluster 8 = 1
User within the cluster 9 = 1
User within the cluster 10 = 1
User within the cluster 11 = 1
User within the cluster 12 = 1
User within the cluster 13 = 1
```

A Function to get user movies list

[33]:
```python
def getMovies(user_id, users_data):
    return list(users_data[users_data['user_id'] == user_id]['mal_id'])
```

Save and load training data Source: https://asdkazmi.medium.com/ai-movies-recommendation-system-with-clustering-based-k-means-algorithm-f04467e02fcd

[34]:
```python
class saveLoadFiles:
    def save(self, filename, data):
        try:
            file = open('datasets/' + filename + '.pkl', 'wb')
            pickle.dump(data, file)
        except:
            err = 'Error: {0}, {1}'.format(exc_info()[0], exc_info()[1])
            print(err)
            file.close()
            return [False, err]
        else:
            file.close()
            return [True]
    def load(self, filename):
        try:
            file = open('datasets/' + filename + '.pkl', 'rb')
        except:
            err = 'Error: {0}, {1}'.format(exc_info()[0], exc_info()[1])
            print(err)
            file.close()
            return [False, err]
        else:
            data = pickle.load(file)
            file.close()
            return data
    def loadClusterMoviesDataset(self):
        return self.load('clusters_movies_dataset')
    def loadUsersClusters(self):
        return self.load('users_clusters')
```

Creating a class function for genre recommendatio based on on user history

[35]:
```
class request:
    def __init__(self, user_id, users_data):
        self.users_data = users_data.copy()
        self.user_id = user_id

        # Find User Cluster
        usersCluster = saveLoadFiles().loadUsersClusters()
        self.usersCluster = int(usersCluster[usersCluster['user_id'] == self.
 ↪user_id]['Cluster'])

        # Load User Cluster Movies Dataframe
        self.moviesList = saveLoadFiles().loadClusterMoviesDataset()
        self.cluster_movies = self.moviesList[self.usersCluster] # dataframe
        self.clusterMovies = list(self.cluster_movies['mal_id']) # list

    def recommendGenre(self):
        try:
            user_movies = getMovies(self.user_id, self.users_data)
            clusterMovies= self.clusterMovies.copy()
            for user_movie in user_movies:
                if user_movie in clusterMovies:
                    clusterMovies.remove(user_movie)
            return [True, clusterMovies]
        except KeyError:
            return False
```

Merging two datasets based on 'mal_id'

[36]:
```
animes_df = pd.read_csv('./datasets/anime_data.csv', usecols=['mal_id',␣
 ↪'title', 'genres'])
animes_df.head(3)
```

[36]:
```
   mal_id                                             genres  \
0       1  ['Action', 'Adventure', 'Comedy', 'Drama', 'Sc…
1     100  ['Comedy', 'Drama', 'Fantasy', 'Magic', 'Roman…
2    1000  ['Action', 'Sci-Fi', 'Adventure', 'Space', 'Dr…

                                   title
0                           Cowboy Bebop
1  Shin Shirayuki-hime Densetsu Prétear
2          Uchuu Kaizoku Captain Herlock
```

[37]:
```
filled_data = pd.read_csv('./datasets/complete_user_ratings.csv',␣
 ↪usecols=['mal_id', 'rating', 'favorited'])
filled_data.head(3)
```

[37]:
```
   mal_id  rating  favorited
0   29978     6.0        0.0
1    2467    10.0        0.0
2   28789     6.0        0.0
```

[38]:
```
df = fav_movies.merge(animes_df, on = 'mal_id')
df.head(3)
```

[38]:
```
   user_id  mal_id  rating      genres title
0        1   29978     6.0  ['Comedy']   001
1       36   29978     5.0  ['Comedy']   001
2       70   29978     5.0  ['Comedy']   001
```

Merge with dataset that applied with linear regression

[39]:
```
new_merge = df.merge(filled_data, on = 'mal_id')
new_merge.head(3)
```

[39]:
```
   user_id  mal_id  rating_x      genres title  rating_y  favorited
0        1   29978       6.0  ['Comedy']   001       6.0        0.0
1        1   29978       6.0  ['Comedy']   001       0.0        0.0
2        1   29978       6.0  ['Comedy']   001       1.0        0.0
```

Genre Recommendation system based on user_id

[40]:
```
genresRecommendations = request(21, fav_movies).recommendGenre()
for movie in genresRecommendations[:1]:
    title = list(new_merge.loc[new_merge['user_id'] == 21]['title'])
    if title != []:
        genres = ast.literal_eval(new_merge.loc[new_merge['user_id'] ==␣
↪21]['genres'].values[0].split('[')[1].split(']')[0])
        for genre in genres:
            print(genre)
```

```
Action
Fantasy
Game
```

Test comparing two different user_id

[70]:
```
genresRecommendations = request(85, fav_movies).recommendGenre()
for movie in genresRecommendations[:1]:
    title = list(new_merge.loc[new_merge['user_id'] == 34]['title'])
    if title != []:
        genres = ast.literal_eval(new_merge.loc[new_merge['user_id'] ==␣
↪34]['genres'].values[0].split('[')[1].split(']')[0])
        for genre in genres:
            print(genre)
```

```
Action
Comedy
```

```
School
Shounen
Super Power
```

1.5 Apriori Algorithm

[41]:
```python
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from mlxtend.frequent_patterns import fpgrowth
```

[42]:
```python
user_rating_data_df.head(10)
```

[42]:
```
   user_id  mal_id  rating
0        1   29978     6.0
1        1    2467    10.0
2        1   28789     6.0
3        1   34881     6.0
4        1     101    10.0
5        1     713     8.0
6        1   36032     8.0
7        1     656     5.0
8        1    1485    10.0
9        1   17901     6.0
```

[43]:
```python
animes_df.head(10)
```

[43]:
```
   mal_id                                             genres  \
0       1  ['Action', 'Adventure', 'Comedy', 'Drama', 'Sc…
1     100  ['Comedy', 'Drama', 'Fantasy', 'Magic', 'Roman…
2    1000  ['Action', 'Sci-Fi', 'Adventure', 'Space', 'Dr…
3   10003         ['Comedy', 'Dementia', 'Horror', 'Seinen']
4   10005         ['Action', 'Adventure', 'Mecha', 'Sci-Fi']
5    1001                  ['Adventure', 'Drama', 'Shounen']
6   10012               ['Comedy', 'Parody', 'Supernatural']
7   10014                            ['Drama', 'Historical']
8   10015           ['Action', 'Fantasy', 'Game', 'Shounen']
9   10016                         ['Comedy', 'Martial Arts']

                                      title
0                              Cowboy Bebop
1    Shin Shirayuki-hime Densetsu Prétear
2           Uchuu Kaizoku Captain Herlock
3       Kago Shintarou Anime Sakuhin Shuu
4  Tetsujin 28-gou: Hakuchuu no Zangetsu
5                Tide-Line Blue: Kyoudai
6                      Carnival Phantasm
7                      Shouwa Monogatari
8                        Yu Gi Oh! Zexal
```

```
9                         Kizuna Ichigeki
```

You can merge the two dataframe on a common column mal_id to obtain the records of user_data_df concatenated with the corresponding details of the movie from the animes_df.

[44]:
```python
df = pd.merge(user_data_df, animes_df[['mal_id', 'title']], on='mal_id')
df.tail(20)
```

[44]:
```
        user_id  mal_id  rating  \
931731     2193    3838     8.0
931732     1604    2758     5.0
931733     2092    2758     6.0
931734     2193    2758     5.0
931735     1823   35516     1.0
931736     2092   35516     6.0
931737     1858   40496     9.0
931738     1893   28813     6.0
931739     2092   28813     7.0
931740     2116   28813     8.0
931741     2193   28813     7.0
931742     1951   38347     5.0
931743     2092   38347     6.0
931744     2052    4723     7.0
931745     2193    4723     8.0
931746     2092   37896     7.0
931747     2193   37896     5.0
931748     2092   42044     6.0
931749     2092   41528     6.0
931750     2092   38490     6.0

                                                    title
931731                            Himitsu no Akko-chan 2
931732                           Shippuu! Iron Leaguer
931733                           Shippuu! Iron Leaguer
931734                           Shippuu! Iron Leaguer
931735                                          Dappys
931736                                          Dappys
931737  Maou Gakuin no Futekigousha: Shijou Saikyou no…
931738                      Bamboo Blade: Fanfu-Fufe-Fo
931739                      Bamboo Blade: Fanfu-Fufe-Fo
931740                      Bamboo Blade: Fanfu-Fufe-Fo
931741                      Bamboo Blade: Fanfu-Fufe-Fo
931742              KisKis! Wo de Nanyou Shi Bohe Tang
931743              KisKis! Wo de Nanyou Shi Bohe Tang
931744                          Seishun Anime Zenshuu
931745                          Seishun Anime Zenshuu
931746                             Ling Yu 6th Season
931747                             Ling Yu 6th Season
```

```
931748         Minegishi-san wa Ootsu-kun ni Tabesasetai
931749                      Xing Chen Bian: Yu Li Cang Hai
931750                                           Xixing Ji
```

[45]:
```
df.shape
```

[45]:
```
(931751, 4)
```

Ensure there are no duplicate records for any given combination of user_id and title

[46]:
```
df = df.drop_duplicates(['user_id','title'])
```

[47]:
```
df.tail(20)
```

[47]:
```
        user_id  mal_id  rating  \
931731     2193    3838     8.0
931732     1604    2758     5.0
931733     2092    2758     6.0
931734     2193    2758     5.0
931735     1823   35516     1.0
931736     2092   35516     6.0
931737     1858   40496     9.0
931738     1893   28813     6.0
931739     2092   28813     7.0
931740     2116   28813     8.0
931741     2193   28813     7.0
931742     1951   38347     5.0
931743     2092   38347     6.0
931744     2052    4723     7.0
931745     2193    4723     8.0
931746     2092   37896     7.0
931747     2193   37896     5.0
931748     2092   42044     6.0
931749     2092   41528     6.0
931750     2092   38490     6.0

                                                title
931731                             Himitsu no Akko-chan 2
931732                              Shippuu! Iron Leaguer
931733                              Shippuu! Iron Leaguer
931734                              Shippuu! Iron Leaguer
931735                                             Dappys
931736                                             Dappys
931737  Maou Gakuin no Futekigousha: Shijou Saikyou no…
931738                         Bamboo Blade: Fanfu-Fufe-Fo
931739                         Bamboo Blade: Fanfu-Fufe-Fo
931740                         Bamboo Blade: Fanfu-Fufe-Fo
931741                         Bamboo Blade: Fanfu-Fufe-Fo
```

```
931742                 KisKis! Wo de Nanyou Shi Bohe Tang
931743                 KisKis! Wo de Nanyou Shi Bohe Tang
931744                              Seishun Anime Zenshuu
931745                              Seishun Anime Zenshuu
931746                                Ling Yu 6th Season
931747                                Ling Yu 6th Season
931748          Minegishi-san wa Ootsu-kun ni Tabesasetai
931749                      Xing Chen Bian: Yu Li Cang Hai
931750                                          Xixing Ji
```

Association algorithms need data in a format such that the userId forms the index, the columns are the movie titles and the values can be 1 or 0 depending on whether that user has watched the movie of the corresponding column. The resulting data is like a user's watchlist, for each userId, having 1 in columns of the movies that the user has watched and 0 otherwise.

[48]:
```
df_pivot = df.pivot(index='user_id', columns='title', values='rating').fillna(0)
```

[49]:
```
df_pivot.head()
```

[49]:
```
title    "0"  "Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi  \
user_id
1        0.0                                       0.0
2        0.0                                       0.0
3        0.0                                       0.0
4        0.0                                       0.0
5        0.0                                       0.0

title    "Bungaku Shoujo" Memoire  "Bungaku Shoujo" Movie  \
user_id
1                             0.0                     0.0
2                             0.0                     0.0
3                             0.0                     0.0
4                             0.0                     0.0
5                             0.0                     0.0

title    "Calpis" Hakkou Monogatari  "Eiji"  "Eiyuu" Kaitai  \
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

title    "Kiss Dekiru Gyoza" x Mameshiba Movie  "Parade" de Satie  \
user_id
1                                          0.0                0.0
2                                          0.0                0.0
3                                          0.0                0.0
```

```
4                                             0.0                  0.0
5                                             0.0                  0.0

title    "R100" x Mameshiba Original Manners  …  s.CRY.ed Alteration I: Tao  \
user_id                                       …
1                                        0.0  …                         0.0
2                                        0.0  …                         0.0
3                                        0.0  …                         0.0
4                                        0.0  …                         0.0
5                                        0.0  …                         0.0

title    s.CRY.ed Alteration II: Quan  the FLY BanD!  xxxHOLiC  xxxHOLiC Kei  \
user_id
1                                 0.0            0.0       0.0           0.0
2                                 0.0            0.0       9.0           9.0
3                                 0.0            0.0       0.0           0.0
4                                 0.0            0.0       0.0           0.0
5                                 0.0            0.0       0.0           0.0

title    xxxHOLiC Movie: Manatsu no Yoru no Yume  xxxHOLiC Rou  \
user_id
1                                            0.0           0.0
2                                            7.0           9.0
3                                            0.0           0.0
4                                            0.0           0.0
5                                            0.0           0.0

title    xxxHOLiC Shunmuki  ēlDLIVE
user_id
1                      0.0      0.0  0.0
2                      9.0      0.0  0.0
3                      0.0      0.0  0.0
4                      0.0      0.0  0.0
5                      0.0      4.0  0.0

[5 rows x 11334 columns]
```

[50]:
```python
def encode_ratings(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

df_pivot = df_pivot.applymap(encode_ratings)
```

[51]:
```python
df_pivot.head()
```

[51]:
```
title    "0"  "Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi  \
user_id
1          0                                          0
2          0                                          0
3          0                                          0
4          0                                          0
5          0                                          0

title    "Bungaku Shoujo" Memoire  "Bungaku Shoujo" Movie  \
user_id
1                               0                       0
2                               0                       0
3                               0                       0
4                               0                       0
5                               0                       0

title    "Calpis" Hakkou Monogatari  "Eiji"  "Eiyuu" Kaitai  \
user_id
1                                 0       0                0
2                                 0       0                0
3                                 0       0                0
4                                 0       0                0
5                                 0       0                0

title    "Kiss Dekiru Gyoza" x Mameshiba Movie  "Parade" de Satie  \
user_id
1                                            0                  0
2                                            0                  0
3                                            0                  0
4                                            0                  0
5                                            0                  0

title    "R100" x Mameshiba Original Manners  …  s.CRY.ed Alteration I: Tao  \
user_id                                       …
1                                          0  …                           0
2                                          0  …                           0
3                                          0  …                           0
4                                          0  …                           0
5                                          0  …                           0

title    s.CRY.ed Alteration II: Quan  the FLY BanD!  xxxHOLiC  xxxHOLiC Kei  \
user_id
1                                   0              0         0             0
2                                   0              0         1             1
3                                   0              0         0             0
4                                   0              0         0             0
5                                   0              0         0             0
```

```
title    xxxHOLiC Movie: Manatsu no Yoru no Yume  xxxHOLiC Rou  \
user_id
1                                              0             0
2                                              1             1
3                                              0             0
4                                              0             0
5                                              0             0

title    xxxHOLiC Shunmuki  ēlDLIVE
user_id
1                        0        0  0
2                        1        0  0
3                        0        0  0
4                        0        0  0
5                        0        1  0

[5 rows x 11334 columns]
```

[52]:
```
frequent_items = apriori(df_pivot, min_support=0.4, use_colnames=True)
frequent_items.head()
```

[52]:
```
    support                                          itemsets
0  0.506150                                   (Akame ga Kill!)
1  0.609112                                    (Angel Beats!)
2  0.524374  (Ano Hi Mita Hana no Namae wo Bokutachi wa Mad…
3  0.529385                                         (Another)
4  0.481549                             (Ansatsu Kyoushitsu)
```

[53]:
```
frequent_items_fp = fpgrowth(df_pivot, min_support=0.4, use_colnames=True)
frequent_items_fp.head()
```

[53]:
```
    support                itemsets
0  0.783144     (Shingeki no Kyojin)
1  0.744419  (Boku no Hero Academia)
2  0.739408          (One Punch Man)
3  0.725740         (Kimi no Na wa.)
4  0.722551            (Death Note)
```

[54]:
```
%timeit apriori(df_pivot, min_support=0.4)
```

```
1.63 s ± 115 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[55]:
```
%timeit fpgrowth(df_pivot, min_support=0.4)
```

```
2.47 s ± 108 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[56]:
```
rules = association_rules(frequent_items, metric="lift", min_threshold=1)
rules.head()
```

[56]:
```
                              antecedents                             consequents  \
0                          (Akame ga Kill!)              (Boku no Hero Academia)
1                   (Boku no Hero Academia)                      (Akame ga Kill!)
2                          (Akame ga Kill!)   (Boku no Hero Academia 2nd Season)
3        (Boku no Hero Academia 2nd Season)                      (Akame ga Kill!)
4                          (Akame ga Kill!)                         (Death Note)

   antecedent support  consequent support   support  confidence      lift  \
0            0.506150            0.744419  0.441913    0.873087  1.172844
1            0.744419            0.506150  0.441913    0.593635  1.172844
2            0.506150            0.674715  0.402733    0.795680  1.179282
3            0.674715            0.506150  0.402733    0.596894  1.179282
4            0.506150            0.722551  0.428246    0.846085  1.170968

   leverage  conviction
0  0.065125    2.013832
1  0.065125    1.215287
2  0.061226    1.592032
3  0.061226    1.225111
4  0.062526    1.802606
```

Let's sort the result by descending order of lift. So that the most likely movie that the user will watch is recommended first.

[57]:
```
result_df = rules.sort_values(by=['lift'], ascending=False)
result_df.head()
```

[57]:
```
                                          antecedents  \
1141                  (Shokugeki no Souma: Ni no Sara)
1140                              (Shokugeki no Souma)
5068                   (Shingeki no Kyojin, Fate/Zero)
5073                            (Fate/Zero 2nd Season)
1172  (Yahari Ore no Seishun Love Comedy wa Machigat…

                                          consequents  antecedent support  \
1141                              (Shokugeki no Souma)            0.419590
1140                  (Shokugeki no Souma: Ni no Sara)            0.482916
5068                            (Fate/Zero 2nd Season)            0.449203
5073                   (Shingeki no Kyojin, Fate/Zero)            0.452392
1172  (Yahari Ore no Seishun Love Comedy wa Machigat…           0.489294

      consequent support   support  confidence      lift  leverage  conviction
1141            0.482916  0.413667    0.985885  2.041526  0.211041   36.633433
1140            0.419590  0.413667    0.856604  2.041526  0.211041    4.047596
5068            0.452392  0.411390    0.915822  2.024399  0.208174    6.505322
```

```
5073              0.449203  0.411390    0.909366  2.024399  0.208174    6.077130
1172              0.424601  0.418679    0.855680  2.015254  0.210924    3.986956
```

[58]:
```python
recomm_df = result_df[result_df['antecedents'].apply(lambda x: len(x) ==1 and␣
 ↪next(iter(x)) == 'Death Note')]
recomm_df.head()
```

[58]:
```
         antecedents                                        consequents  \
13556  (Death Note)  (Fullmetal Alchemist: Brotherhood, Shingeki no…
4532   (Death Note)    (Fullmetal Alchemist: Brotherhood, Tokyo Ghoul)
14172  (Death Note)     (Steins;Gate, Shingeki no Kyojin, Tokyo Ghoul)
4725   (Death Note)                         (Mirai Nikki, One Punch Man)
4521   (Death Note)    (Steins;Gate, Fullmetal Alchemist: Brotherhood)

       antecedent support  consequent support   support  confidence      lift  \
13556            0.722551            0.462415  0.415034    0.574401  1.242178
4532             0.722551            0.489294  0.438269    0.606557  1.239659
14172            0.722551            0.449658  0.401367    0.555485  1.235350
4725             0.722551            0.453303  0.402733    0.557377  1.229591
4521             0.722551            0.485194  0.430524    0.595839  1.228043

       leverage  conviction
13556  0.080916    1.263127
4532   0.084729    1.298045
14172  0.076466    1.238074
4725   0.075199    1.235130
4521   0.079947    1.273764
```

[59]:
```python
recomm_df = recomm_df[recomm_df['lift'] > 1]
recomm_df.head()
```

[59]:
```
         antecedents                                        consequents  \
13556  (Death Note)  (Fullmetal Alchemist: Brotherhood, Shingeki no…
4532   (Death Note)    (Fullmetal Alchemist: Brotherhood, Tokyo Ghoul)
14172  (Death Note)     (Steins;Gate, Shingeki no Kyojin, Tokyo Ghoul)
4725   (Death Note)                         (Mirai Nikki, One Punch Man)
4521   (Death Note)    (Steins;Gate, Fullmetal Alchemist: Brotherhood)

       antecedent support  consequent support   support  confidence      lift  \
13556            0.722551            0.462415  0.415034    0.574401  1.242178
4532             0.722551            0.489294  0.438269    0.606557  1.239659
14172            0.722551            0.449658  0.401367    0.555485  1.235350
4725             0.722551            0.453303  0.402733    0.557377  1.229591
4521             0.722551            0.485194  0.430524    0.595839  1.228043

       leverage  conviction
13556  0.080916    1.263127
```

```
4532    0.084729    1.298045
14172   0.076466    1.238074
4725    0.075199    1.235130
4521    0.079947    1.273764
```

[60]:
```
anime_rec = recomm_df['consequents'].values

anime_rec_list = []
for rec in anime_rec:
    for title in rec:
        if title not in anime_rec_list:
            anime_rec_list.append(title)
```

The top 5 anime that the user is most likely to watch can be obtained

[61]:
```
anime_rec_list[:5]
```

[61]:
```
['Fullmetal Alchemist: Brotherhood',
 'Shingeki no Kyojin',
 'Tokyo Ghoul',
 'Steins;Gate',
 'Mirai Nikki']
```

1.6 Singular Value Decomposition (SVD)

Followed this tutorial: https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-part-2-k-nearest-neighbors-and-matrix-c04b3c2ef55c

[3]:
```
# Imports and process needed datasets
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
import sklearn
from sklearn.decomposition import TruncatedSVD
import matplotlib.pyplot as plt

user_rating_data = './datasets/user_score_data.csv'
df = pd.read_csv(user_rating_data)
user_rating_df = df[['user_id', 'mal_id', 'rating']].copy()

anime_info_data = './datasets/anime_data.csv'
anime_df = pd.read_csv(anime_info_data)
columns = ['aired_from', 'aired_to', 'duration', 'episodes', 'genres',␣
↪'popularity', 'premiered', 'rank', 'rating', 'score', 'scored_by', 'source',␣
↪'status', 'studios', 'synopsis', 'title', 'type']
anime_df = anime_df.drop(columns, axis=1)
anime_df = anime_df.dropna()
```

Combine datasets and group by title to get total rating count for each show

[4]:
```
combine_user_anime = pd.merge(user_rating_df, anime_df, on='mal_id')
total_ratings = (combine_user_anime.
                    groupby(by = ['title_english'])['rating'].
                count().
                reset_index().
                rename(columns = {'rating' : 'totalRatingCount'})
                [['title_english', 'totalRatingCount']]
                )
total_ratings.head()
```

[4]:

	title_english	totalRatingCount
0	"Parade" de Satie	14
1	"Star"t	15
2	-OutsideR:RequieM-	17
3	.Koni-chan	9
4	.hack//G.U. Trilogy	49

Narrow the dataset down to anime that have been rated a certain number of times

[5]:
```
userRatings_with_totalRatings = combine_user_anime.merge(total_ratings,␣
 ↪left_on='title_english', right_on='title_english')
userRatings_with_totalRatings.head(40)

popularity_threshold = 100 # this can be changed to narrow the scope of our data
ratings_top_anime = userRatings_with_totalRatings.query('totalRatingCount >=␣
 ↪@popularity_threshold')
n = len(pd.unique(ratings_top_anime['title_english']))
print("Number of unique anime to be used: ", n)
```

```
Number of unique anime to be used:  1710
```

Convert to 2D Matrix and transpose

[6]:
```
ratings_top_anime_pivot = ratings_top_anime.pivot_table(index = 'user_id',␣
 ↪columns='title_english', values='rating', aggfunc=np.sum).fillna(0)
transposed_ratings = ratings_top_anime_pivot.values.T
ratings_top_anime_pivot.head()
```

[6]:

title_english	.hack//Sign	07-Ghost	11eyes	5 Centimeters Per Second	\
user_id					
1	0.0	0.0	0.0	10.0	
2	0.0	0.0	9.0	8.0	
3	0.0	0.0	0.0	7.0	
4	0.0	6.0	0.0	0.0	
5	0.0	0.0	0.0	0.0	

title_english	7 Seeds	91 Days	91 Days: Brief Candle	\
user_id				

```
1                 0.0      0.0                     0.0
2                 0.0      9.0                     0.0
3                 0.0      8.0                     0.0
4                 0.0      0.0                     0.0
5                 0.0      8.0                     0.0

title_english  91 Days: Shoal of Time/All Our Yesterdays/Tomorrow and Tomorrow
\
user_id
1                                                              0.0
2                                                              6.0
3                                                              0.0
4                                                              0.0
5                                                              0.0

title_english  A Bridge to the Starry Skies  A Centaur's Life  …  \
user_id                                                        …
1                                       0.0               0.0  …
2                                       0.0               0.0  …
3                                       0.0               0.0  …
4                                       0.0               0.0  …
5                                       0.0               0.0  …

title_english  the Garden of sinners Chapter 2: Murder Speculation Part A  \
user_id
1                                                              0.0
2                                                              0.0
3                                                              0.0
4                                                              0.0
5                                                              0.0

title_english  the Garden of sinners Chapter 3: Remaining Sense of Pain  \
user_id
1                                                              0.0
2                                                              0.0
3                                                              0.0
4                                                              0.0
5                                                              0.0

title_english  the Garden of sinners Chapter 4: The Hollow Shrine  \
user_id
1                                                              0.0
2                                                              0.0
3                                                              0.0
4                                                              0.0
5                                                              0.0
```

```
title_english  the Garden of sinners Chapter 5: Paradox Paradigm  \
user_id
1                                                              0.0
2                                                              0.0
3                                                              0.0
4                                                              0.0
5                                                              0.0

title_english  the Garden of sinners Chapter 6: Oblivion Recording  \
user_id
1                                                              0.0
2                                                              0.0
3                                                              0.0
4                                                              0.0
5                                                              0.0

title_english  the Garden of sinners Chapter 7: Murder Speculation Part B  \
user_id
1                                                              0.0
2                                                              0.0
3                                                              0.0
4                                                              0.0
5                                                              0.0

title_english  the Garden of sinners Chapter 8: The Final Chapter  \
user_id
1                                                              0.0
2                                                              0.0
3                                                              0.0
4                                                              0.0
5                                                              0.0

title_english  the Garden of sinners Remix -Gate of seventh heaven-  \
user_id
1                                                              0.0
2                                                              0.0
3                                                              0.0
4                                                              0.0
5                                                              0.0

title_english  tsuritama  xxxHOLiC
user_id
1                    0.0       0.0
2                    9.0       9.0
3                    0.0       0.0
4                    8.0       0.0
5                    0.0       0.0
```

[5 rows x 1710 columns]

1.6.1 Find the best model by calculating RMSE for different number of latent variables

[7]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt

def svd_rsme(data, n_latent_var):
    # Split data
    train, test = train_test_split(data, test_size = 0.2, random_state=5)
    test_transposed = test.values.T
    train_transposed = train.values.T
    transposed_ratings = data.values.T

    # Run model on data
    SVD = TruncatedSVD(n_components=n_latent_var, random_state=17)
    test_matrix = SVD.fit_transform(test_transposed)
    train_matrix = SVD.fit_transform(train_transposed)
    true_matrix = SVD.fit_transform(transposed_ratings)

    # Return RSME
    rmse = sqrt(mean_squared_error(true_matrix, train_matrix))
    return rmse
```

[8]:
```
RSME_list = []
for i in range(20):
    rsme = svd_rsme(ratings_top_anime_pivot, i+1)
    RSME_list.append([i+1, rsme])

# Display RSME Dataframe
RSME_DF = pd.DataFrame(RSME_list, columns=['Latent_Var_Num', 'RSME'])
RSME_DF= RSME_DF.style.set_caption("Latent Variable RSME Comparison")
RSME_DF
```

[8]: <pandas.io.formats.style.Styler at 0x7fd9387e1438>

1.6.2 Best model is when the number of latent variables = 3

[67]:
```
import warnings
warnings.filterwarnings("ignore", category = RuntimeWarning)

SVD = TruncatedSVD(n_components=3, random_state=17)
matrix = SVD.fit_transform(transposed_ratings)
matrix.shape
```

[67]: (1710, 3)

1.6.3 Calculate Pearson R Correlation Coefficient (PCC)

[68]:
```
# Correlation Coefficient
corr = np.corrcoef(matrix)
corr.shape
```

[68]: (1710, 1710)

1.6.4 Recommendations based on PCC of SVD Model - Random Choice

[85]:
```
import collections

anime_titles = ratings_top_anime_pivot.columns
anime_titles_list = list(anime_titles)

# Pick random anime
title_chosen = np.random.choice(anime_titles_list)
#print('Recommendations for: ', title_chosen)

# Get its index and correlation coefficient
title_index = anime_titles_list.index(title_chosen)
corr_title = corr[title_index]

# Store the highly correlated titles in a dictionary
ranking = {}
for i in range(len(anime_titles)):
    if 0.9 < corr_title[i] < 1.0:
        ranking[corr_title[i]] = anime_titles[i]

# Sort list in descending order and display final ranking df
ranking = collections.OrderedDict(sorted(ranking.items(),reverse=True))
print(len(ranking))
list_num = 10
if len(ranking) < 10:
    list_num = len(ranking)

ranked_list = []
for j in range(list_num):
    ranked_title = list(ranking.values())[j]
    ranked_coef = list(ranking.keys())[j]
    ranked_list.append([j+1, ranked_title, ranked_coef])

# Display Final Dataframe
ranked_df = pd.DataFrame(ranked_list, columns=['Rank', 'Anime Title', 'R␣
↪Correlation'])
```

```
ranked_df= ranked_df.style.set_caption("Recommendations for '" + title_chosen␣
↪+"'")
ranked_df
```

982

[85]: <pandas.io.formats.style.Styler at 0x7fd8f83a2dd8>

1.6.5 Recommendations - Input Title 'Bleach'

[86]:
```
# Type in title
title_chosen = "Bleach"

# Get its index and correlation coefficient
title_index = anime_titles_list.index(title_chosen)
corr_title = corr[title_index]

# Store the highly correlated titles in a dictionary
ranking = {}
for i in range(len(anime_titles)):
    if 0.9 < corr_title[i] < 1.00000000000000:
        ranking[corr_title[i]] = anime_titles[i]

# Sort list in descending order
ranking = collections.OrderedDict(sorted(ranking.items(),reverse=True))
ranked_list = []
for j in range(10):
    ranked_title = list(ranking.values())[j]
    ranked_coef = list(ranking.keys())[j]
    ranked_list.append([j+1, ranked_title, ranked_coef])

# Display Final Dataframe
ranked_df = pd.DataFrame(ranked_list, columns=['Rank', 'Anime Title', 'R␣
↪Correlation'])
ranked_df= ranked_df.style.set_caption("Recommendations for '" + title_chosen␣
↪+"'")
ranked_df
```

[86]: <pandas.io.formats.style.Styler at 0x7fd91850e240>

1.7 Alternating Least Squares

Followed this tutorial: https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisation-ebe1ad2e7679

1.7.1 Load and prepare data

[1]:
```
import pandas as pd
import numpy as np
from pyspark.sql.functions import col, explode
from pyspark import SparkContext
from pyspark.sql import SparkSession

sc = SparkContext
# sc.setCheckpointDir('checkpoint')
spark = SparkSession.builder.appName('Recommendations').getOrCreate()

ratings = spark.read.csv('./datasets/user_score_data.csv',header=True)
anime = spark.read.csv('./datasets/anime_data.csv', header=True)
```

```
/opt/anaconda3/envs/myenv/lib/python3.6/site-packages/pyspark/context.py:238:
FutureWarning: Python 3.6 support is deprecated in Spark 3.2.
  FutureWarning
```

[2]:
```
ratings = ratings.\
    withColumn('user_id', col('user_id').cast('integer')).\
    withColumn('mal_id', col('mal_id').cast('integer')).\
    withColumn('rating', col('rating').cast('float')).\
    drop('_c0')

anime = anime.\
    withColumn('mal_id', col('mal_id').cast('integer')).\
    drop('aired_from', 'aired_to', 'duration', 'episodes', 'genres',␣
 ↪'popularity', 'premiered', 'rank', 'rating', 'score', 'scored_by', 'source',␣
 ↪'status', 'studios', 'synopsis', 'title', 'type')
```

1.7.2 Calculate Sparsity

[3]:
```
# Count the total number of ratings in the dataset
numerator = ratings.select("rating").count()

# Count the number of distinct user_id and distinct mal_id
num_users = ratings.select("user_id").distinct().count()
num_anime = ratings.select("mal_id").distinct().count()

# Set the denominator equal to the number of users multiplied by the number of␣
 ↪anime
denominator = num_users * num_anime

# Divide the numerator by the denominator
sparsity = (1.0 - (numerator *1.0)/denominator)*100
print("The ratings dataframe is ", "%.2f" % sparsity + "% empty.")
```

The ratings dataframe is 97.31% empty.

1.7.3 Interpret Ratings

[4]:
```
# Group data by user_id, count ratings
user_id_ratings = ratings.groupBy("user_id").count().orderBy('count',␣
↪ascending=False)
user_id_ratings.show()
```

```
+-------+-----+
|user_id|count|
+-------+-----+
|   2193|14025|
|   2092|13991|
|   1473|10990|
|    358| 9583|
|   1018| 8405|
|    584| 4993|
|   1539| 4858|
|   1755| 4381|
|   1604| 4243|
|    128| 3597|
|    515| 3595|
|    896| 3352|
|   1406| 3300|
|   1432| 3206|
|    834| 3157|
|   1661| 3024|
|   1823| 3009|
|   1534| 2649|
|   1515| 2603|
|    837| 2522|
+-------+-----+
only showing top 20 rows
```

1.7.4 Build Out an ALS Model

[5]:
```
from pyspark.ml.recommendation import ALS

# Create test and train set
(train, test) = ratings.randomSplit([0.8, 0.2], seed = 1234)

# Create ALS model
als = ALS(userCol="user_id", itemCol="mal_id", ratingCol="rating", nonnegative␣
↪= True, implicitPrefs = False, coldStartStrategy="drop")
```

1.7.5 Tune ALS Model - RMSE Evaluator

[6]:
```
# Import the requisite items
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# Add hyperparameters and their respective values to param_grid
param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [10, 50, 100, 150]) \
            .addGrid(als.regParam, [.01, .05, .1, .15]) \
            .build()

# Define evaluator as RMSE and print length of evaluator
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",␣
 ↪predictionCol="prediction")

print("Num models to be tested: ", len(param_grid))
```

```
Num models to be tested:  16
```

1.7.6 Build Cross Validation Pipeline

[7]:
```
# Build cross validation using CrossValidator
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid,␣
 ↪evaluator=evaluator, numFolds=5)

# Confirm cv was built
print(cv)
```

```
CrossValidator_630608288769
```

1.7.7 Build Model and Best Model Parameters

[8]:
```
#Fit cross validator to the 'train' dataset
model = cv.fit(train)

#Extract best model from the cv model above
best_model = model.bestModel

# View the predictions
test_predictions = best_model.transform(test)
RMSE = evaluator.evaluate(test_predictions)
print("RMSE of Best ALS Model:", RMSE)
```

```
RMSE of Best ALS Model: 1.7495747228788467
```

1.7.8 Make Recommendations

[12]:
```
# Generate n Recommendations for all users
nrecommendations = best_model.recommendForAllUsers(10)
nrecommendations.limit(10).show()
```

```
+-------+--------------------+
|user_id|     recommendations|
+-------+--------------------+
|     12|[{3297, 9.284231}…|
|     22|[{820, 9.421549},…|
|     26|[{33050, 9.924782…|
|     27|[{33050, 10.31628…|
|     28|[{33050, 10.57829…|
|     31|[{33050, 9.716631…|
|     34|[{28977, 9.537582…|
|     44|[{820, 9.650627},…|
|     53|[{33050, 9.398941…|
|     65|[{33050, 9.364338…|
+-------+--------------------+
```

[13]:
```
nrecommendations = nrecommendations\
    .withColumn("rec_exp", explode("recommendations"))\
    .select('user_id', col("rec_exp.mal_id"), col("rec_exp.rating"))

nrecommendations.limit(10).show()
```

```
+-------+------+--------+
|user_id|mal_id|  rating|
+-------+------+--------+
|     12|  3297|9.284231|
|     12| 17074|9.276755|
|     12| 36862|9.200462|
|     12|  2921|9.143925|
|     12|   283|9.067277|
|     12|   820|9.005661|
|     12|    26| 8.99276|
|     12|    32|8.990191|
|     12| 33095|8.977576|
|     12|    30|8.892254|
+-------+------+--------+
```

1.7.9 Display Top 5 Recommendations for user_id = 65

[14]:
```
nrecommendations.join(anime, on='mal_id').filter('user_id = 65').limit(5).
↪show(truncate=False)
```

```
+------+-------+--------+-------------------------------------------------+
|mal_id|user_id|rating  |title_english                                    |
+------+-------+--------+-------------------------------------------------+
|33050 |65     |9.364338|Fate/stay night: Heaven's Feel - III. Spring Song|
|820   |65     |9.204607|Legend of the Galactic Heroes                    |
|35180 |65     |9.127725|March Comes In Like A Lion 2nd Season            |
|35247 |65     |9.103279|Owarimonogatari Second Season                    |
|17074 |65     |9.00018 |Monogatari Series: Second Season                 |
+------+-------+--------+-------------------------------------------------+
```