MA 544 Final Project

By: Nick, Gokul, Samruth

Introduction

Purpose:

- Which type of recommendation system is best for suggesting new media to a user?

Dataset:

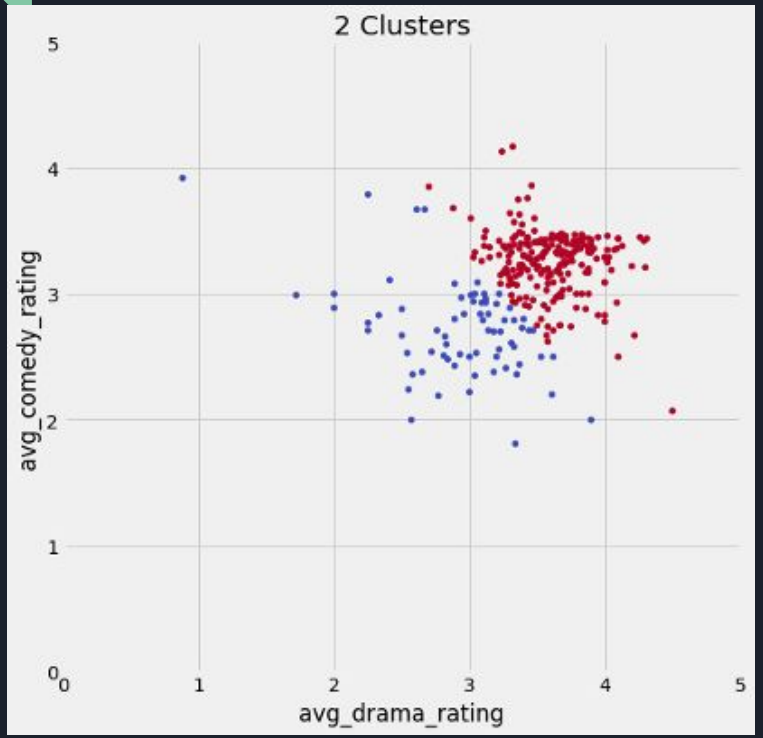
- The 100k MovieLens dataset will be used for this experiment
- https://grouplens.org/datasets/movielens/

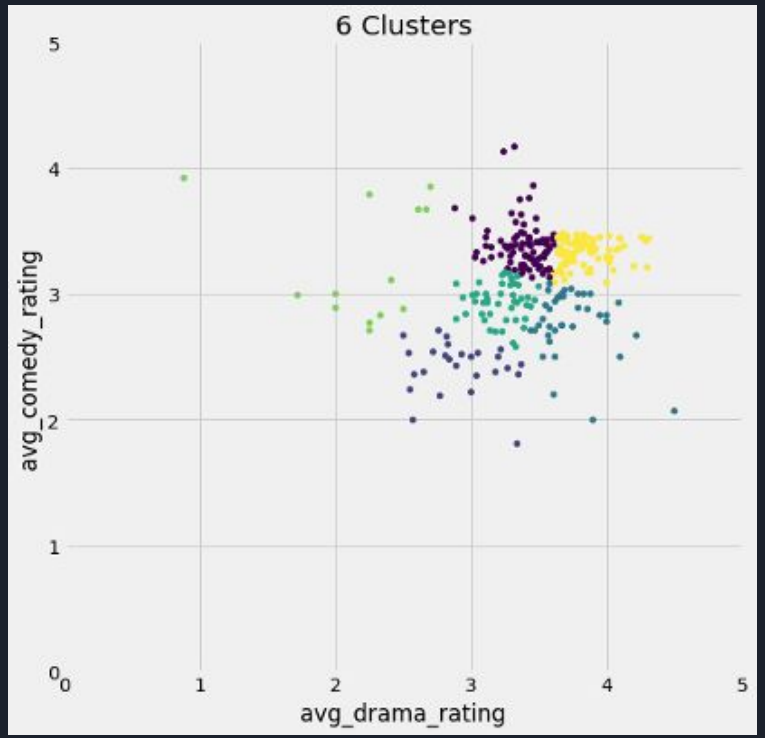
K - Means Clustering

K means Clustering is a method of vector quantization that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

K-Means: 2D Representation





K - Means Clustering System

Recommender Function via K-Means

All this information is useful, but it needs to be in a singular function. A function is created to input a user. The the function will cluster the data, find the user's groups and make reccomendations based on the group

```
def movie_recommender_k_means_cluster(user_movie_ratings, user_id= 143, cluster_number = 20):
    '''
    Finds recomendations for specific user based on number of clusters

    Parameters
    ----------
    user_movie_ratings : pandas
        data frame of users and movies row by col .
    user_id : int
        userid number for recommendatin.
    cluster_number : int, optional
        Number of clusters. The default is 20.

    Returns
    -------
    list_of_recommended_movies : list
        list of movies recommended.

    '''
    ## Variables
    user_idx = user_id -1
    pick_from_top_movies = 20
```

K- Means Clustering Results

```
# User 143, 15 Clusters
start_time = time.time()
movie_recommender_k_means_cluster(user_movie_ratings, user_id= 143, cluster_number = 15)
print(f'Total Time to Run: {time.time() - start_time}')
```

```
Users ratings for movies in cluster:
Kill Bill: Vol. 2 (2004)      5.0
Gone with the Wind (1939)     5.0
Finding Neverland (2004)      5.0
Forrest Gump (1994)           5.0
Clockwork Orange, A (1971)    5.0
Name: 142, dtype: float64
User 143 has 77 movie reviews

Users top ratings:

title
Kill Bill: Vol. 2 (2004)      5.0
Gone with the Wind (1939)     5.0
Finding Neverland (2004)      5.0
Forrest Gump (1994)           5.0
Clockwork Orange, A (1971)    5.0
Name: 143, dtype: float64


Movies Average Rating in Cluster not seen by user:

Ratatouille (2007)                                5.000000
African Queen, The (1951)                         4.833333
Cinema Paradiso (Nuovo cinema Paradiso) (1989)    4.812500
Psycho (1960)                                     4.750000
Stand by Me (1986)                                4.636364
dtype: float64

The list of recommended movies for user 143 is:

['Psycho (1960)', 'Third Man, The (1949)', 'Citizen Kane (1941)', 'Hoop Dreams (1994)']
Total Time to Run: 4.063762903213501
```

SVD Clustering

SVD is a matrix factorisation technique, which reduces the number of features of a dataset by reducing the space dimension from N-dimension to K-dimension (where K<N) using:

$$A = USV^T$$

Where A is a m x n utility matrix, U is a m x r orthogonal left singular matrix, S is a r x r diagonal matrix and V is a r x n diagonal right singular matrix

SVD Cluserting Code

In [ ]:
```
df_title = pd.read_csv('../input/movie_titles.csv', encoding = "ISO-8859-1", header = None, names = ['Movie_Id', 'Year', 'Na
df_title.set_index('Movie_Id', inplace = True)
print (df_title.head(10))
```

In [39]:
```
reader = Reader()

# get just top 100K rows for faster run time
data = Dataset.load_from_df(ratings_data[['userId', 'new_movie_id', 'rating']][:], reader)
#data.split(n_folds=3)

svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'])
```

Out[39]:
```
{'test_rmse': array([0.87007861, 0.87520837, 0.87542801, 0.87252011, 0.869551  ]),
 'test_mae': array([0.6684968 , 0.67203905, 0.6709185 , 0.66903059, 0.66902215]),
 'fit_time': (3.4100866317749023,
  3.498782157897949,
  3.44977068901062,
  3.565463066101074,
  3.4956393241882324),
 'test_time': (0.12617945671081543,
  0.12566661834716797,
  0.09174680709838867,
  0.08976411819458008,
  0.13264799118041992)}
```

In [66]:
```
f = ['count','mean']

df_movie_summary = ratings_data.groupby('movieId')['rating'].agg(f)
df_movie_summary.index = df_movie_summary.index.map(int)
movie_benchmark = round(df_movie_summary['count'].quantile(0.7),0)
drop_movie_list = df_movie_summary[df_movie_summary['count'] < movie_benchmark].index
```

In [67]:
```
def movie_recs(user):
    user_liked_movies = ratings_data[(ratings_data['userId'] == user) & (ratings_data['rating'] == 5)]
    user_liked_movies = user_liked_movies.set_index('movieId')
    user_liked_movies = pd.merge(movies_data, user_liked_movies, on="movieId")
    Movies_liked = user_liked_movies['title']

    user_liked_movie = movies_data.copy()
    user_liked_movie = user_liked_movie.reset_index()
    user_liked_movie = user_liked_movie[~user_liked_movie['movieId'].isin(drop_movie_list)]

    # getting full dataset
    data = Dataset.load_from_df(ratings_data[['userId', 'movieId', 'rating']], reader)

    trainset = data.build_full_trainset()
    svd.fit(trainset)

    user_liked_movie['Estimate_Rating'] = user_liked_movie['movieId'].apply(lambda x: svd.predict(user, x).est)

    user_liked_movie = user_liked_movie.drop('movieId', axis = 1)

    user_liked_movie = user_liked_movie.sort_values('Estimate_Rating', ascending=False)
    recommended_movies = user_liked_movie[['title','Estimate_Rating']]

    print('Movies The User Likes')
    print('')
    print(Movies_liked.head(20))
    print('')
    print('')
    print('Recommended Movies with estimated Ratings:')
    print('')
    print(recommended_movies.head(10))
```

SVD Clustering Results

```
In [69]: 1 movie_recs(user = 143)

Movies The User Likes

0              Forrest Gump (1994)
1           Sixteen Candles (1984)
2                  Shrek (2001)
3            Legally Blonde (2001)
4       Walk to Remember, A (2002)
5              Uptown Girls (2003)
6           Chasing Liberty (2004)
7            50 First Dates (2004)
8            13 Going on 30 (2004)
9       Bill Cosby, Himself (1983)
10                       Up (2009)
11                Fired Up (2009)
12             Hangover, The (2009)
13    (500) Days of Summer (2009)
14                  Tangled (2010)
Name: title, dtype: object


Recommended Movies with estimated Ratings:

                                                  title  Estimate_Rating
937     Seventh Seal, The (Sjunde inseglet, Det) (1957)         4.245808
512                           Beauty and the Beast (1991)         4.223449
6571                              Eastern Promises (2007)         4.192814
7467                                    Tangled (2010)         4.174376
3194                                     Shrek (2001)         4.164004
975                                Cool Hand Luke (1967)         4.158392
711                                     Notorious (1946)         4.132487
966                    Manchurian Candidate, The (1962)         4.123778
4076     Harry Potter and the Chamber of Secrets (2002)         4.100718
228   Like Water for Chocolate (Como agua para choco...         4.092270
```

Observations/Conclusion

In conclusion:

- SVD system is overall faster (on average it tends to take 4.5 seconds for this dataset),

- K-means is more customizable because you can change the cluster number which allows for more precision recommendations.