ASSIGNMENT 1C

Joshua Paterson, Kaiyun Yu
N10193197, n9889663

Assignment 1C

Contribution table

Author	Completed
Joshua Paterson – N10193197	Problem 1
Kaiyun Yu - N9889663	Problem 2

Problem 1

Dataset

The data for this problem comes from the MovieLens user rating dataset. The relevant files being used are rating.csv which contain user ratings from movies and movies.csv which relate is used to refence movie names. This dataset contains 100836 ratings of 9742 movies from 610 users.

Dataset Pre-processing

As the objective of the problem is to find movie recommendations for users the data in its current form will not be able to produce this. As users are the object that will be clustered with the information used to cluster them will be their ratings of varies movies. The data will be transformed to create a table where the row of the dataset will be each individual user and the columns will represent a specific movie rating. In figure 1 below a visual representation of the data can be seen.

It should be noted that a lot of the data is unrated or NaN as most users will not have rated every movie.

It was decided to use all movies and users in the model as it seemed unlikely that every user will have watched a high number of the 9742 movies available causing a sparse dataset.

```
# Merge the two tables then pivot so we have Users X Movies dataframe
ratings_title = pd.merge(ratings, movies[['movieId', 'title']], on='movieId' )
user_movie_ratings = pd.pivot_table(ratings_title, index='userId', columns= 'title', values='rating')
print('dataset dimensions: ', user_movie_ratings.shape, '\n\nSubset example:')
user_movie_ratings.iloc[:6, :10]
```

```
dataset dimensions:  (610, 9719)

Subset example:
```

title	'71 (2014)	'Hellboy': The Seeds of Creation (2004)	'Round Midnight (1986)	'Salem's Lot (2004)	'Til There Was You (1997)	'Tis the Season for Love (2015)	'burbs, The (1989)	'night Mother (1986)	(500) Days of Summer (2009)	*batteries not included (1987)
userId										
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 1: Data after Pre-processing

Model

To cluster this dataset the k-means model/algorithm was chosen to be used. This model was chosen because due to the fact the there are so many dimensions (movies) to this data it is hard to visualise, and it is therefore hard for me to know how this data should be classified due to this the simplest option was chosen being k-means. This method was also chosen to limit the number of movies recommendation per cluster if a method like GMM was chosen then multiple movies could belong to

many clusters if there is a lot of overlap movie recommendation could simply become the most highly rated and watched movies. k-means algorithms uses hard assignment and avoids this issue.

The algorithm uses all default values except for k or number of clusters. To find an optimal number of clusters for this problem a grid search like method was used creating 14 k-means models increasing the increment of cluster by 1 then calculating the Silhouette Score to determine how well users are clustered. A Silhouette Score measure of how similar an object is to its own cluster compared to other clusters. A higher value indicates that a user is more accurately depicted in a cluster on average. As seen in figure 2 below the Silhouette Score (Distortion) falls after 4 clusters however after running this grid search method a few times the location of this fall varies around 4 to 6 and therefore a k of 5 was chosen.

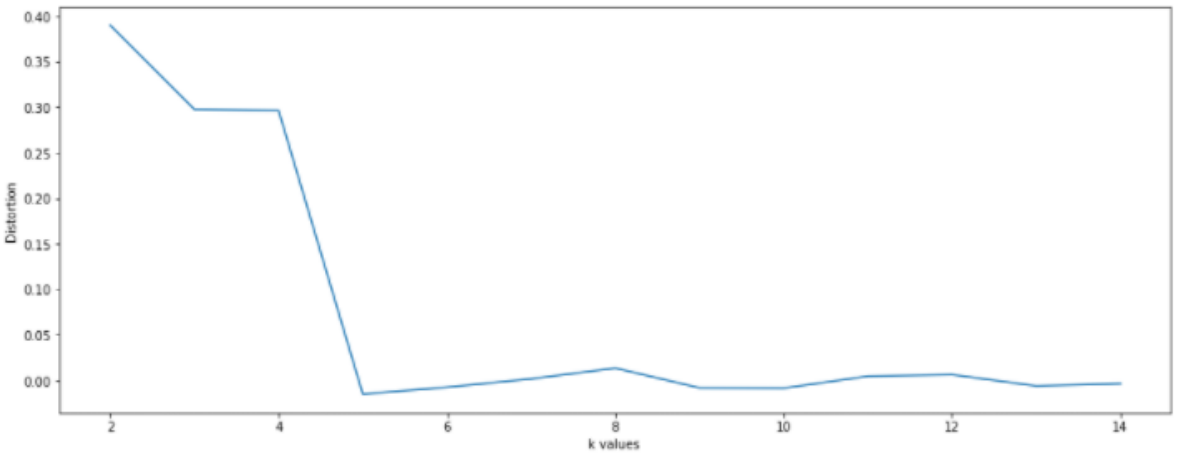


Figure 2: silhouette score of n clusters

How recommendations are formed

Using the model stated previously the model can generate predictions for users by locating the cluster of which and individual user belongs to then finding the highest rated movies that there cluster has rated that they themselves have not rated.

Results of the clustering

In figure 3 below it shows a visual of the resulting cluster that the model and dataset produce. Each of the plots in this figure shows the rating of movies (columns) in colour (5-star rating) by users (rows). Note that white means no rating given. There are cluster with both unproportionally huge and small group showing the groups are not sorted very evenly. To determine how well this model fits the problem an analysis of individuals will be given below.

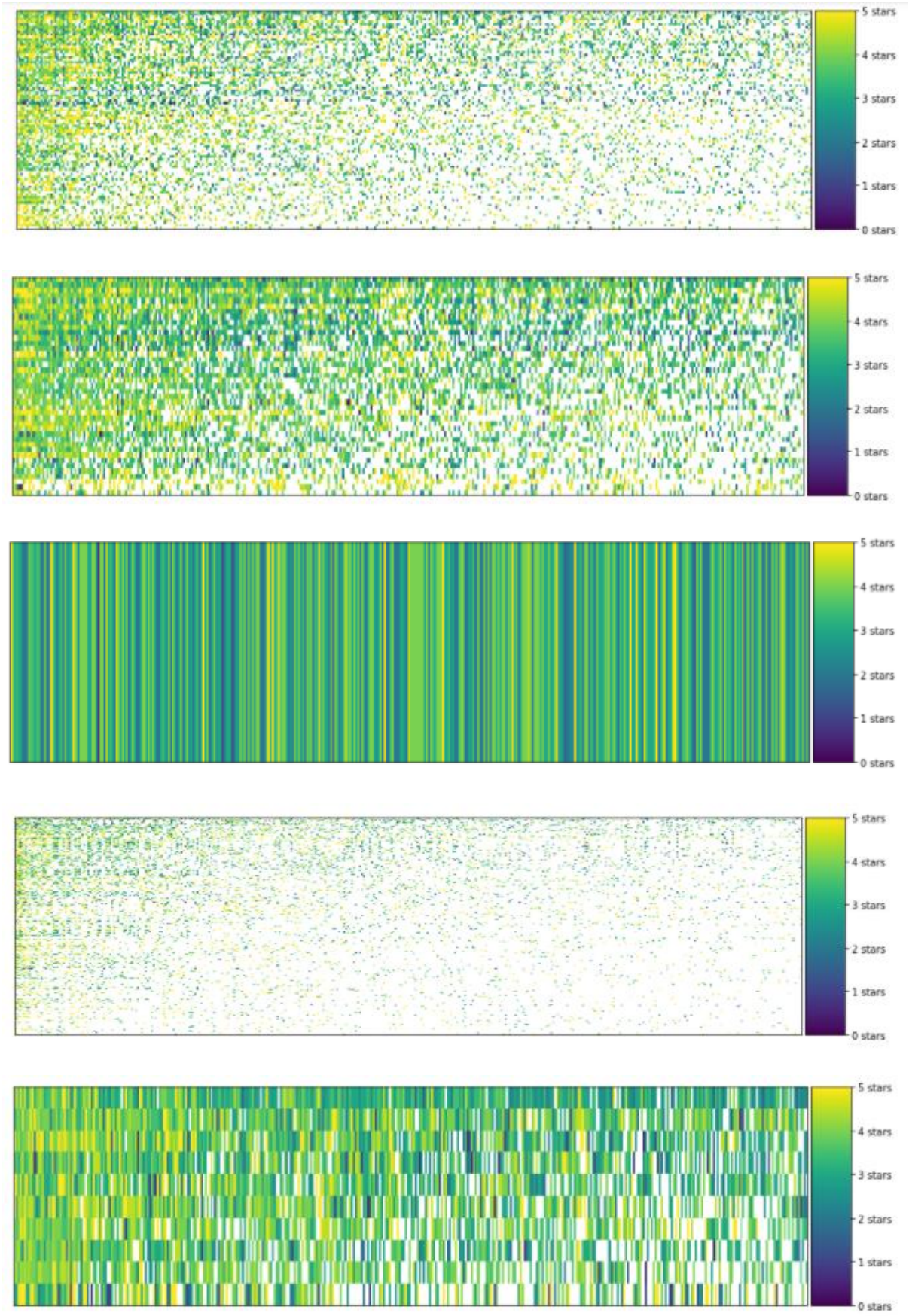


Figure 3: Resulting Clusters of Model.

Results/User Recommendations

User 4

In the below figure or figure 4 is shows users 4 most watched movies genres that was rated above 3. This user enjoys mostly drama, romance and comedy type movies.

```
Drama                                          15
Comedy                                         13
Comedy|Drama                                   10
Comedy|Drama|Romance                            6
Drama|Romance                                   6
                                               ..
Adventure|Comedy|Western                        1
Children|Comedy|Fantasy|Musical                 1
Action|Adventure|Mystery|Romance|Thriller       1
Adventure|Comedy                                1
Adventure|Fantasy|Musical                       1
```

Figure 4: User 4's count of genres rated above 3.

In Figure 5 below it shows the top 5 recommendations the model has produced, and as seen Drama or Comedy is in the movie Genres. This show for User 4 the cluster can group similar movies to what user 4 has shown a liking towards.

	movieId	title	genres
277	318	Shawshank Redemption, The (1994)	Crime|Drama
733	953	It's a Wonderful Life (1946)	Children|Drama|Fantasy|Romance
906	1204	Lawrence of Arabia (1962)	Adventure|Drama|War
929	1228	Raging Bull (1980)	Drama
1883	2502	Office Space (1999)	Comedy|Crime

Figure 5: User 4's Top 5 recommendations

User 42

In the below figure or figure 6 it shows users 42 most watched movies genres that was rated above 3. This user enjoys mostly Drama and Comedy type movies and seems very similar to User 4 however it should be noted that this user was sorted into a different cluster. This may be because this user has shown a liking to other genres where User 4 seemed to be exclusively into comedy drama and Romance.

```
Comedy                                 33
Drama                                  22
Comedy|Romance                         18
Drama|Romance                           9
Action|Adventure|Thriller               9
                                       ..
Comedy|Drama|War                        1
Mystery|Thriller                        1
Crime|Horror|Mystery|Thriller           1
Comedy|Crime|Drama|Thriller             1
Action|Adventure|Mystery|Thriller       1
```

Figure 6: User 42's count of genres rated above 3.

In Figure 7 below it shows the top 5 recommendations the model has produced, and as seen the movies in the drama genre does show however unlike User 4 this cluster seems to have more action and adventure movies included in the selection.

	movieId	title	genres
224	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi
701	919	Wizard of Oz, The (1939)	Adventure|Children|Fantasy|Musical
941	1242	Glory (1989)	Drama|War
971	1272	Patton (1970)	Drama|War
1291	1721	Titanic (1997)	Drama|Romance

Figure 7: User 42's Top 5 recommendations

User 314

In the below figure or figure 4 is shows users 314 most watched movies genres that was rated above 3. This user enjoys mostly drama, comedy, and romance type movies. This preference for these types of movies genres to the point where they are the only genres that the user has rated above 5 more then once is very similar to User 4 and the model was able to recognise this as they are in the same cluster.

```
Comedy|Romance                                    5
Drama                                            4
Comedy|Drama|Romance                             2
Children|Drama|Fantasy|Mystery                   1
Action|Adventure|Thriller                        1
Comedy                                           1
Drama|Romance                                    1
Adventure|Drama|IMAX                             1
Drama|Horror|Sci-Fi                              1
Action|Comedy|Sci-Fi                             1
Action|Adventure|Sci-Fi                          1
Crime|Drama                                      1
Action|Thriller                                  1
Action|Crime|Drama|War                           1
Adventure|Drama|Sci-Fi                           1
Comedy|Drama|Fantasy|Romance|Thriller            1
Crime|Mystery|Thriller                           1
Action|Drama|War                                 1
Comedy|Drama|Romance|War                         1
Adventure|Animation|Children|Drama|Musical|IMAX  1
Drama|War                                        1
Action|Sci-Fi                                    1
Comedy|Fantasy|Romance                           1
```

Figure 8: User 314's count of genres rated above 3.

In Figure 9 below it shows the top 5 recommendations the model has produced, and as seen movies in the drama and romance genres are frequently suggested for this user showing a reasonable recommendation list however this is one suggestion in this list where the User has shown no interest in its genres. This was likely because movies in clusters are sorted by most highly rated and will therefore have a bias for movies with high average rating in their cluster.

	movieId	title	genres
277	318	Shawshank Redemption, The (1994)	Crime|Drama
686	904	Rear Window (1954)	Mystery|Thriller
690	908	North by Northwest (1959)	Action|Adventure|Mystery|Romance|Thriller
975	1276	Cool Hand Luke (1967)	Drama
1917	2542	Lock, Stock & Two Smoking Barrels (1998)	Comedy|Crime|Thriller

Figure 9: User 314's Top 5 recommendations

Overall thoughts

Overall, I think the recommendation where reasonable as the model was able to recognise Users 4 and 314 with very similar tastes as they both clearly did not like much outside of 3 genres, they would only highly rate. User 34 also like those same genres but showed some interest in other genres and was able to be sorted into a group with a more diverse movie selection. As the model was able to easily categized similar users the rating of each group would allow the model to find recommendation of user with similar preference. However there where some issues as there was one group with one user assigned to it. This is an issue as this user would not be able to find recommendation due to this issue. To fix this in the future the number of k would need to be selected to find a number that would have more then 1 amount of people per cluster.

Problem 2

Description of the algorithm

We eliminated the bad data and normalized the data. In the colour classification problem, we also performed data enhancement operations such as image flipping.

We use ResNet34 as our method and added a classifier to the last layer to get the results we want. The reason for using this network is that it has been proven to be very effective in many scenarios on image classification problems. In order to speed up the training speed, here we use pytorch's pre-trained initialization parameters when training the network, the loss function uses the crossentropy loss function, the optimization algorithm uses the Adam algorithm, and the learning rate is set to automatically decrease.

Our algorithm basically achieves an accuracy of $80\%-90\%$ on these test sets within 50 to 100 iteration steps. Due to the small amount of data, this effect is still very successful. If you want better results, you need more data. Regarding semantic search, we can use CNN to first extract the features we want from the sentence, and then use our image classification algorithm for matching search.

Data has been resized to 256,128 sizes and random horizontal flip for increasing the data augmentation.

Results

In this section, we list the results of classifying different traits on the testing data set. The number in the upper right corner of each picture represents the recognition result of our algorithm for each picture.

Gender

There are totally 20 Epoch has been used in training. The best train accuracy is 0.998066 and best valid accuracy is 0.664894. Training was completed in 8 mins.

Figure 1: Results of gender classification.

Figure 2: Results of gender classification.

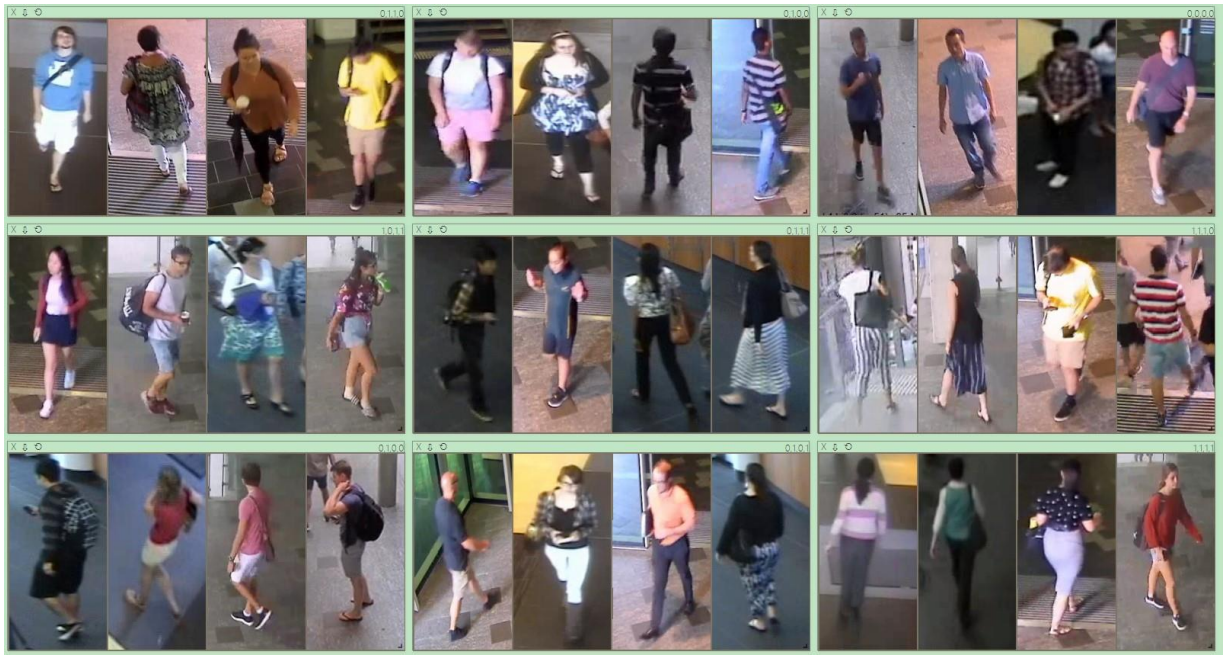


Figure 3: Results of gender classification.

Figure 4: Results of gender classification.

Figure 5: Results of gender classification.

Torso Clothing Type

There are totally 20 Epoch has been used in training. The best train accuracy is 1.00 and best valid accuracy is 0.544231. Training was completed in 8m 22s.

Figure 6: Results of torso clothing type classification.

Figure 7: Results of torso clothing type classification.

Figure 8: Results of torso clothing type classification.

Figure 9: Results of torso clothing type classification.

Figure 10: Results of torso clothing type classification.

Torso Clothing Colour

There are totally 20 Epoch has been used in training. The best train accuracy is 0.829457 and best valid accuracy is 0.474490. Training was completed in 8m 22s.

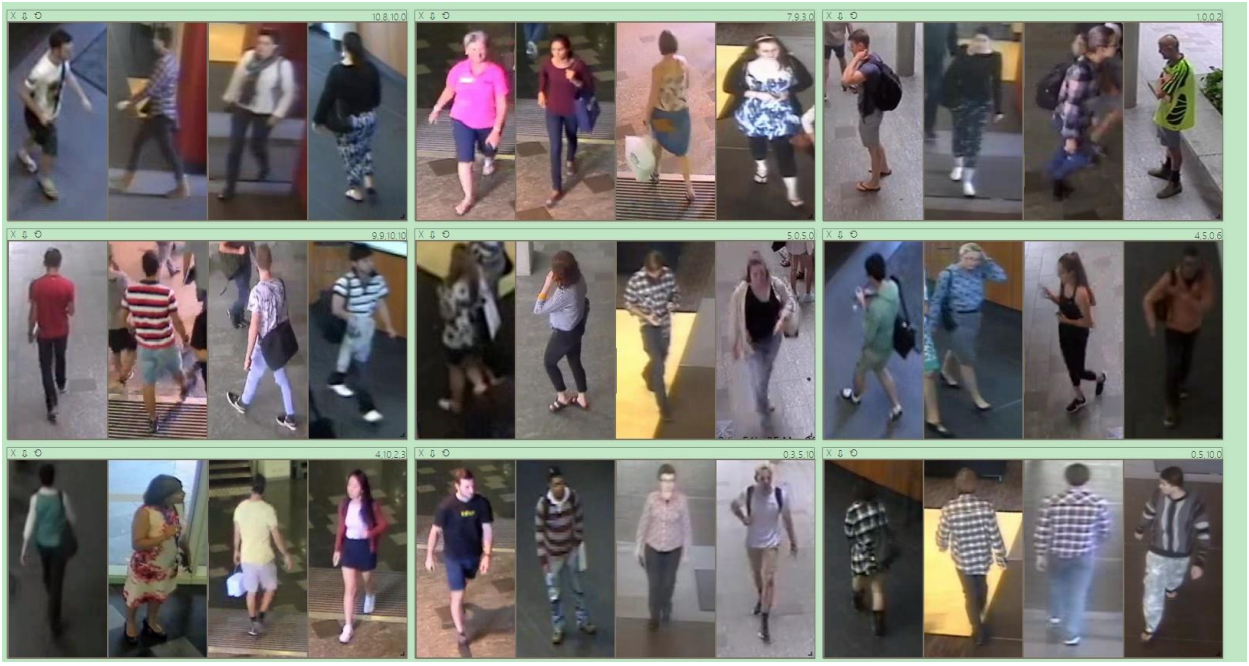


Figure 11: Results of torso clothing colour classification.

Figure 12: Results of torso clothing colour classification.

Figure 13: Results of torso clothing colour classification.

Figure 14: Results of torso clothing colour classification.

Figure 15: Results of torso clothing colour classification.

Leg Clothing Type

There are totally 20 Epoch has been used in training. The best train accuracy is 1.00 and best valid accuracy is 0.84. Training was completed in 15m 36s.

Figure 16: Results of leg clothing type classification.

Figure 17: Results of leg clothing type classification.

Figure 18: Results of leg clothing type classification.

Figure 19: Results of leg clothing type classification.

Figure 20: Results of leg clothing type classification.

Leg Clothing Colour

There are totally 20 Epoch has been used in training. The best train accuracy is 0.998066 and best valid accuracy is 0.664894. Training completed in 5m 36s.

Figure 21: Results of leg clothing colour classification.

Figure 22: Results of leg clothing colour classification.

Figure 23: Results of leg clothing colour classification.

Figure 24: Results of leg clothing colour classification.

Figure 25: Results of leg clothing colour classification.

Luggage

There are totally 20 Epoch has been used in training. The best train accuracy is 0.863462 and best valid accuracy is 0.647059. Training was completed in 8m 5s.

Figure 26: Results of luggage classification.

Figure 27: Results of luggage classification.

Figure 28: Results of luggage classification.

Figure 29: Results of luggage classification.

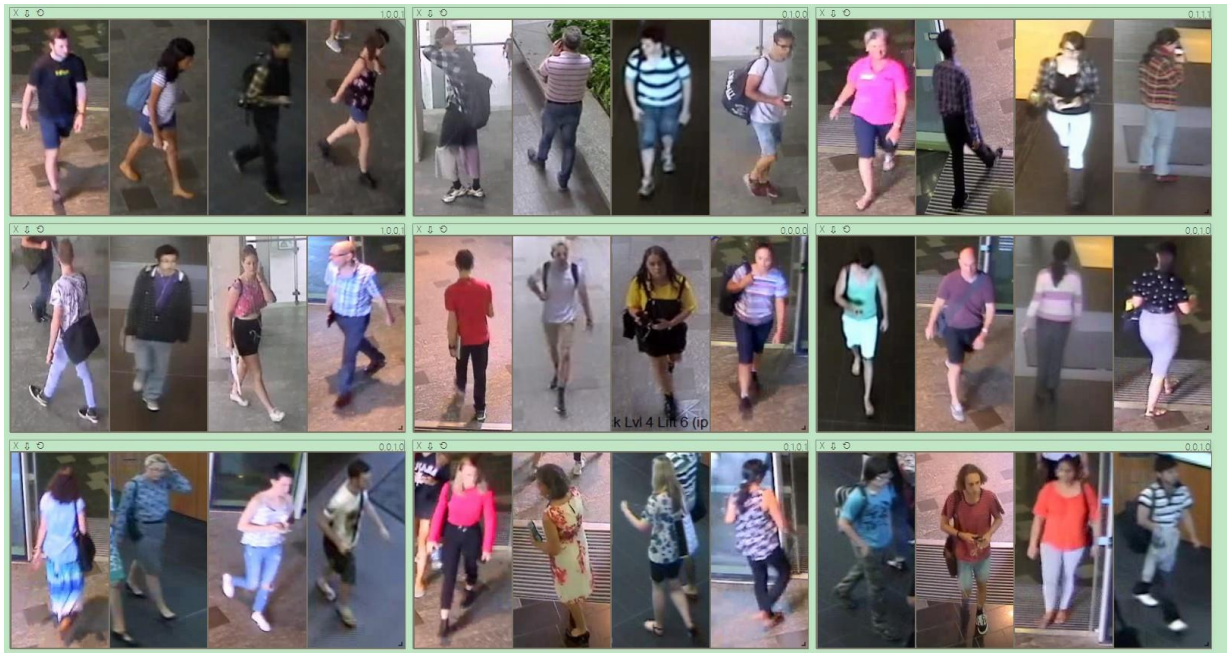


Figure 30: Results of luggage classification.

Problems and solution

In this model, training accuracy has achieved significant accuracy which achieved 1.0 accuracy. The major reason which causes the problem is in some data set there are too many classifications but with small amount data, random horizontal flip has been applied to the model for data augmentation. Moreover, the model has overfitting in some points, thus learning rate decay scheduler is using for reducing the overfitting problem. The outcome is significant and achieved satisfied result within small amount of data.

Code

In [1]:
```
# Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import csr_matrix
from mpl_toolkits.axes_grid1 import make_axes_locatable
from sklearn.cluster import KMeans
from sklearn.metrics import mean_squared_error
import itertools
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import KMeans
import helper
%matplotlib inline
```

In [2]:
```
# Import the Movies dataset
movies = pd.read_csv('data/Q1/movies.csv')
movies.head()
```

Out[2]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [3]:
```
# Import the ratings dataset
ratings = pd.read_csv('data/Q1/ratings.csv')
ratings.head()
```

Out[3]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

In [4]:
```
# Print the number of records and the total number of movies
print('The dataset contains: ', len(ratings), ' ratings of ', len(movies), ' movies.
```

```
The dataset contains:  100836  ratings of  9742  movies.
```

In [5]:
```
#Mvoiew level clustering
```

In [ ]:
```
```

In [6]:
```
# Merge the two tables then pivot so we have Users X Movies dataframe
ratings_title = pd.merge(ratings, movies[['movieId', 'title']], on='movieId' )
user_movie_ratings = pd.pivot_table(ratings_title, index='userId', columns= 'title',
print('dataset dimensions: ', user_movie_ratings.shape, '\n\nSubset example:')
user_movie_ratings.iloc[:6, :10]
```

```
dataset dimensions:  (610, 9719)
```

Subset example:

Out[6]:

title	'71 (2014)	'Hellboy': The Seeds of Creation (2004)	'Round Midnight (1986)	'Salem's Lot (2004)	'Til There Was You (1997)	'Tis the Season for Love (2015)	'burbs, The (1989)	'night Mother (1986)	(500) Days of Summer (2009)	*batteries no included (1987
userId										
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [7]:
```
n_movies = 30
n_users = 18
most_rated_movies_users_selection = helper.sort_by_rating_density(user_movie_ratings

print('dataset dimensions: ', most_rated_movies_users_selection.shape)
most_rated_movies_users_selection.head()
```

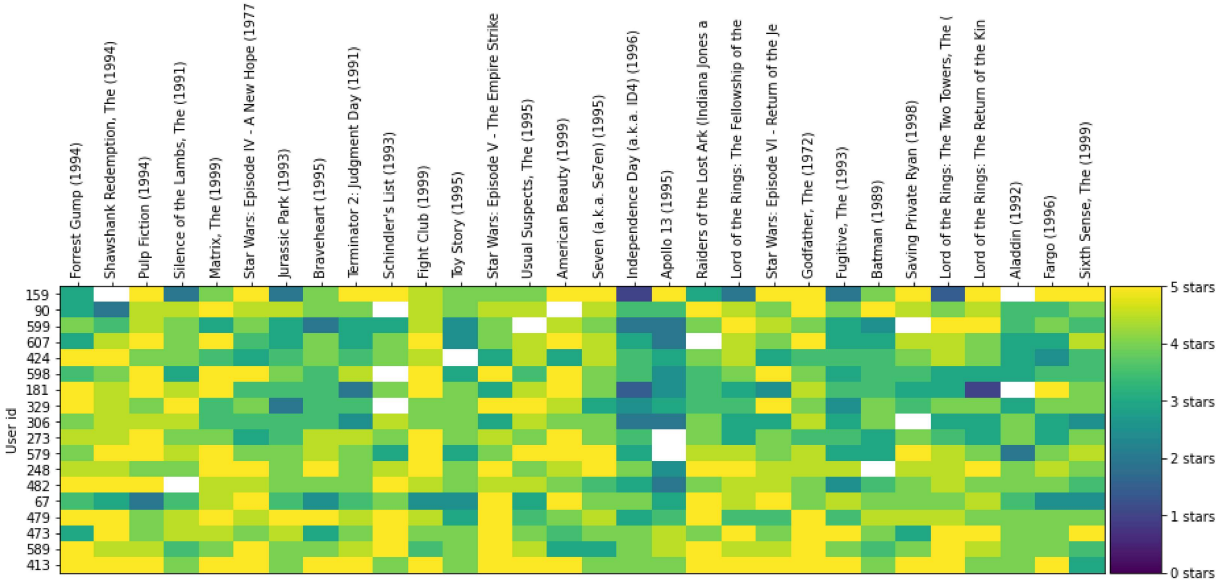
```
dataset dimensions:  (18, 30)
```

Out[7]:

title	Forrest Gump (1994)	Shawshank Redemption, The (1994)	Pulp Fiction (1994)	Silence of the Lambs, The (1991)	Matrix, The (1999)	Star Wars: Episode IV - A New Hope (1977)	Jurassic Park (1993)	Braveheart (1995)	Terminator 2: Judgment Day (1991)	Sch Lis
413	5.0	5.0	5.0	4.0	5.0	5.0	4.0	5.0	5.0	
589	5.0	4.5	4.5	3.5	4.0	5.0	4.0	4.0	4.5	
473	3.0	5.0	4.0	4.5	4.5	4.0	4.5	3.0	4.0	
479	5.0	5.0	4.0	4.5	5.0	4.5	5.0	5.0	4.5	
67	3.5	3.0	2.0	3.5	4.5	5.0	3.5	2.5	3.5	

5 rows × 30 columns

In [8]:
```
helper.draw_movies_heatmap(most_rated_movies_users_selection)
```



In [9]:
```
user_movie_ratings =  pd.pivot_table(ratings_title, index='userId', columns= 'title'
most_rated_movies_1k = helper.get_most_rated_movies(user_movie_ratings, 9719)
#most_rated_movies_1k = helper.get_most_rated_movies(user_movie_ratings, 5000)
#most_rated_movies_1k = helper.get_most_rated_movies(user_movie_ratings, 193609)
#most_rated_movies_1k = helper.get_most_rated_movies(user_movie_ratings, 50000)
print(user_movie_ratings.shape)
print(most_rated_movies_1k.shape)
```

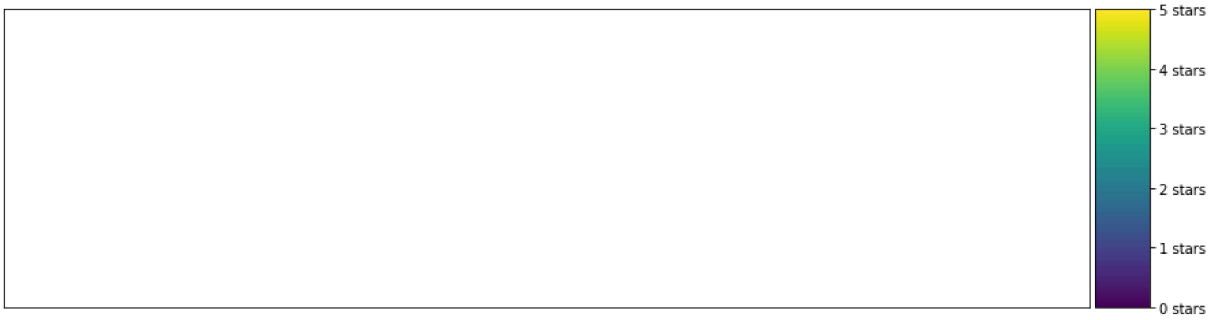
```
(610, 9719)
(610, 9719)
```

In [10]:
```
#sparse_ratings = csr_matrix(pd.SparseDataFrame(most_rated_movies_1k).to_coo())
#most_rated_movies_1k.fillna(0)
sp_arr = csr_matrix(most_rated_movies_1k.fillna(0))
sdf = pd.DataFrame.sparse.from_spmatrix(sp_arr)
sparse_ratings = sdf.sparse.to_coo()
```

In [11]:
```
# 20 clusters
#predictions = KMeans(n_clusters=20, algorithm='full').fit_predict(sparse_ratings)
predictions = KMeans(n_clusters=5, algorithm='full').fit_predict(sparse_ratings)

max_users = 70
max_movies = 50

clustered = pd.concat([most_rated_movies_1k.reset_index(), pd.DataFrame({'group':pre
```

In [12]:
```
# TODO: Pick a cluster ID from the clusters above
cluster_number = 5

# Let's filter to only see the region of the dataset with the most number of values
n_users = 75
n_movies = 300
cluster = clustered[clustered.group == cluster_number].drop(['index', 'group'], axis

cluster = helper.sort_by_rating_density(cluster, n_movies, n_users)
helper.draw_movies_heatmap(cluster, axis_labels=False)
```

```
C:\Users\User\Documents\GitHub\420_A1\Assignment 1C\Problem_1\helper.py:202: UserWar
ning: Attempting to set identical bottom == top == -0.5 results in singular transfor
mations; automatically expanding.
  heatmap = ax.imshow(most_rated_movies_users_selection,  interpolation='nearest', v
min=0, vmax=5, aspect='auto')
```



In [13]:
```
#cluster = clustered[clustered.group == 5].drop(['index', 'group'], axis=1)
#cluster = helper.sort_by_rating_density(cluster, n_movies, n_users)
#helper.draw_movies_heatmap(cluster, axis_labels=False)
```

In [14]:
```
cluster.fillna('').head()
```

Out[14]:

Forrest Gump (1994)	Tomboy (2011)	Tom and Jerry: A Nutcracker Tale (2007)	Them (Ils) (2006)	The Unauthorized Saved by the Bell Story (2014)	The Voices (2014)	The Void (2016)	The Wailing (2016)	The Wait (2015)	The Wild One (1953)	...

0 rows × 300 columns

In [15]:
```
# TODO: Fill in the name of the column/movie. e.g. 'Forrest Gump (1994)'
# Pick a movie from the table above since we're looking at a subset
#movie_name = 'Apollo 13 (1995)'

#cluster[movie_name].mean()
```

In [16]:
```
# The average rating of 20 movies as rated by the users in the cluster
cluster.mean().head(20)
```

Out[16]:
```
Forrest Gump (1994)                                 NaN
Tomboy (2011)                                       NaN
Tom and Jerry: A Nutcracker Tale (2007)             NaN
Them (Ils) (2006)                                   NaN
The Unauthorized Saved by the Bell Story (2014)     NaN
The Voices (2014)                                   NaN
The Void (2016)                                     NaN
The Wailing (2016)                                  NaN
The Wait (2015)                                     NaN
The Wild One (1953)                                 NaN
The Witch (2015)                                    NaN
The Wolfpack (2015)                                 NaN
The Wooden Horse (1950)                             NaN
The Year My Voice Broke (1987)                      NaN
Themroc (1973)                                      NaN
The Town that Dreaded Sundown (2014)                NaN
There Once Was a Dog (1982)                         NaN
Tom and Jerry: Shiver Me Whiskers (2006)            NaN
Ultimate Avengers 2 (2006)                          NaN
They (2002)                                         NaN
dtype: float64
```

In [17]:
```
cluster.fillna('').head()
```

Out[17]:

Forrest Gump (1994)	Tomboy (2011)	Tom and Jerry: A Nutcracker Tale (2007)	Them (Ils) (2006)	The Unauthorized Saved by the Bell Story (2014)	The Voices (2014)	The Void (2016)	The Wailing (2016)	The Wait (2015)	The Wild One (1953)	...

0 rows × 300 columns

In [18]:
```
#user_id = 2

# Get all this user's ratings
#user_2_ratings  = cluster.loc[user_id, :]
#user_2_ratings
```

In [19]:
```
# TODO: Pick a user ID from the dataset
# Look at the table above outputted by the command "cluster.fillna('').head()"
# and pick one of the user ids (the first column in the table)
#user_id = 4

# Get all this user's ratings
#user_2_ratings  = cluster.loc[user_id, :]

# Which movies did they not rate? (We don't want to recommend movies they've already
#user_2_unrated_movies =  user_2_ratings[user_2_ratings.isnull()]

# What are the ratings of these movies the user did not rate?
#avg_ratings = pd.concat([user_2_unrated_movies, cluster.mean()], axis=1, join='inne

# Let's sort by rating so the highest rated movies are presented first
#avg_ratings.sort_values(ascending=False)[:20]
```

In [20]:
```
# https://github.com/gouravaich/k-means-clustering-movie-ratings/blob/master/k-means
```

In [21]:
```
#print(len(avg_ratings.sort_values(ascending=False)))
```

In [22]:
```
#testing
```

In [23]:
```
#predictions 3
# Pick a cluster ID from the clusters above
#from random import randrange
#cluster_number = randrange(10)
# Let's filter to only see the region of the dataset with the most number of values
#n_users = 75
#n_movies = 300
#cluster = clustered[clustered.group == cluster_number].drop(['index', 'group'], axi
# Sort and print the cluster
#cluster = helper.sort_by_rating_density(cluster, n_movies, n_users)
#helper.draw_movies_heatmap(cluster, axis_labels=False)
```

In [24]:
```
#print(cluster_number)
```

In [ ]:
```

```

In [ ]:
```

```

In [25]:
```
# Print the ratings
#cluster.fillna('').head()
```

In [26]:
```
#movie_name = "Matrix, The (1999)"
#cluster[movie_name].mean()
```

In [27]:
```
# Pick a user ID from the dataset
#user_id = 4
# Get all this user's ratings
#user_2_ratings  = cluster.loc[user_id, :]
# Which movies did they not rate?
#user_2_unrated_movies =  user_2_ratings[user_2_ratings.isnull()]
# What are the ratings of these movies the user did not rate?
#avg_ratings = pd.concat([user_2_unrated_movies, cluster.mean()], axis=1, join='inne
# Let's sort by rating so the highest rated movies are presented first
#avg_ratings.sort_values(ascending=False)[:20]
```

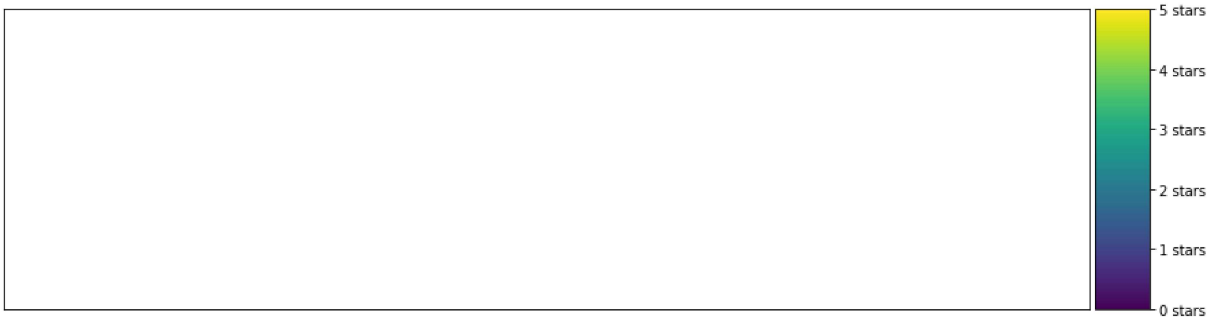
In [28]:
```
# Pick a user ID from the dataset
#user_id = 42
# Get all this user's ratings
#user_2_ratings  = cluster.loc[user_id, :]
# Which movies did they not rate?
#user_2_unrated_movies =  user_2_ratings[user_2_ratings.isnull()]
# What are the ratings of these movies the user did not rate?
#avg_ratings = pd.concat([user_2_unrated_movies, cluster.mean()], axis=1, join='inne
# Let's sort by rating so the highest rated movies are presented first
#avg_ratings.sort_values(ascending=False)[:20]
```

In [29]:
```
# Pick a user ID from the dataset
#user_id = 314
# Get all this user's ratings
#user_2_ratings  = cluster.loc[user_id, :]
# Which movies did they not rate?
#user_2_unrated_movies =  user_2_ratings[user_2_ratings.isnull()]
# What are the ratings of these movies the user did not rate?
#avg_ratings = pd.concat([user_2_unrated_movies, cluster.mean()], axis=1, join='inne
# Let's sort by rating so the highest rated movies are presented first
#avg_ratings.sort_values(ascending=False)[:20]
```

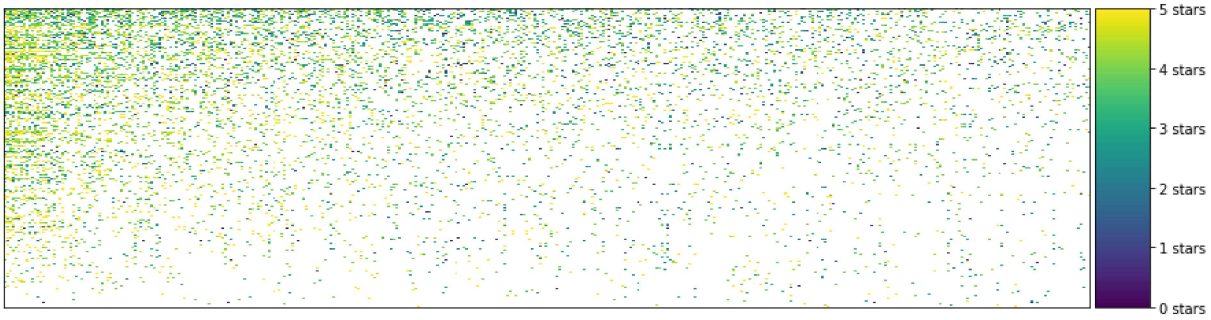
In [30]:
```
from random import randrange
while True:
    try:
    #predictions testing
        # Pick random cluster ID from the clusters above
        cluster_number = randrange(6)
        #cluster_number = 12
        # Let's filter to only see the region of the dataset with the most number of
        n_users = 400
        n_movies = 400
        cluster = clustered[clustered.group == cluster_number].drop(['index', 'group
        # Sort and print the cluster
        cluster = helper.sort_by_rating_density(cluster, n_movies, n_users)
        helper.draw_movies_heatmap(cluster, axis_labels=False)
        print(cluster_number)
        print("_________")

        # Pick a user ID from the dataset
        user_id = 4
        # Get all this user's ratings
        user_2_ratings  = cluster.loc[user_id, :]
        # Which movies did they not rate?
        user_2_unrated_movies =  user_2_ratings[user_2_ratings.isnull()]
        # What are the ratings of these movies the user did not rate?
```

```
        avg_ratings = pd.concat([user_2_unrated_movies, cluster.mean()], axis=1, joi
        # Let's sort by rating so the highest rated movies are presented first
        User_4_sugest = avg_ratings.sort_values(ascending=False)
        print(User_4_sugest[:20])
        break
    except:
        print("An exception occurred")
```

```
C:\Users\User\Documents\GitHub\420_A1\Assignment 1C\Problem_1\helper.py:202: UserWar
ning: Attempting to set identical bottom == top == -0.5 results in singular transfor
mations; automatically expanding.
  heatmap = ax.imshow(most_rated_movies_users_selection,  interpolation='nearest', v
min=0, vmax=5, aspect='auto')
```




```
5
_________
An exception occurred
```




```
1
_________
Shawshank Redemption, The (1994)                                    4.445
122
Cool Hand Luke (1967)                                               4.380
000
North by Northwest (1959)                                           4.370
370
Rear Window (1954)                                                  4.317
073
Casablanca (1942)                                                   4.295
918
Departed, The (2006)                                                4.294
643
Godfather: Part II, The (1974)                                      4.269
231
Chinatown (1974)                                                    4.258
621
One Flew Over the Cuckoo's Nest (1975)                              4.256
579
Dark Knight, The (2008)                                             4.255
319
Green Mile, The (1999)                                              4.254
717
American History X (1998)                                           4.250
000
Pulp Fiction (1994)                                                 4.244
526
Taxi Driver (1976)                                                  4.244
186
```

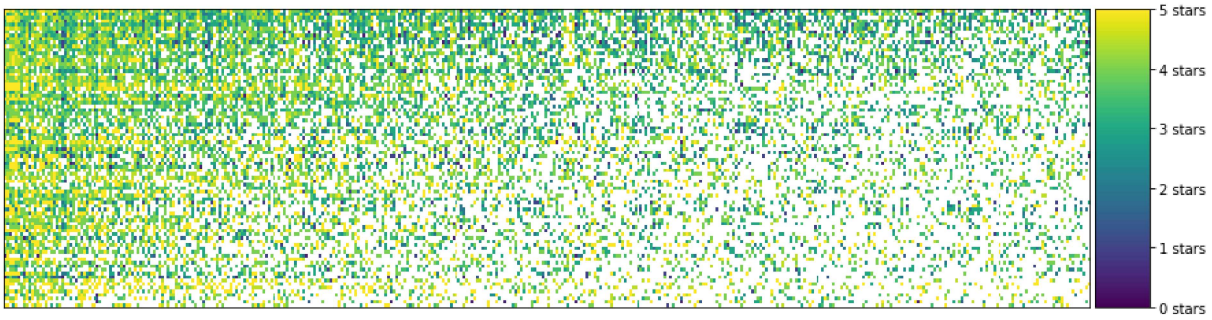
```
Goodfellas (1990)                                                    4.241
379
Usual Suspects, The (1995)                                           4.227
778
Singin' in the Rain (1952)                                           4.227
273
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)    4.216
981
To Kill a Mockingbird (1962)                                         4.214
286
Fight Club (1999)                                                    4.212
687
Name: 0, dtype: float64
```

In [31]:

```
while True:
    try:
    #predictions testing
        # Pick a cluster ID from the clusters above
        cluster_number = randrange(6)
        #cluster_number = 12
        # Let's filter to only see the region of the dataset with the most number of
        n_users = 400
        n_movies = 400
        cluster = clustered[clustered.group == cluster_number].drop(['index', 'group
        # Sort and print the cluster
        cluster = helper.sort_by_rating_density(cluster, n_movies, n_users)
        helper.draw_movies_heatmap(cluster, axis_labels=False)
        print(cluster_number)
        print("_________")

        # Pick a user ID from the dataset
        user_id = 42
        # Get all this user's ratings
        user_2_ratings  = cluster.loc[user_id, :]
        # Which movies did they not rate?
        user_2_unrated_movies =  user_2_ratings[user_2_ratings.isnull()]
        # What are the ratings of these movies the user did not rate?
        avg_ratings = pd.concat([user_2_unrated_movies, cluster.mean()], axis=1, joi
        # Let's sort by rating so the highest rated movies are presented first
        User_42_sugest = avg_ratings.sort_values(ascending=False)
        print(User_42_sugest[:20])
        break
    except:
        print("An exception occurred")
```

```
C:\Users\User\Documents\GitHub\420_A1\Assignment 1C\Problem_1\helper.py:202: UserWar
ning: Attempting to set identical bottom == top == -0.5 results in singular transfor
mations; automatically expanding.
  heatmap = ax.imshow(most_rated_movies_users_selection,  interpolation='nearest', v
min=0, vmax=5, aspect='auto')
```


```
5
_________
An exception occurred
```

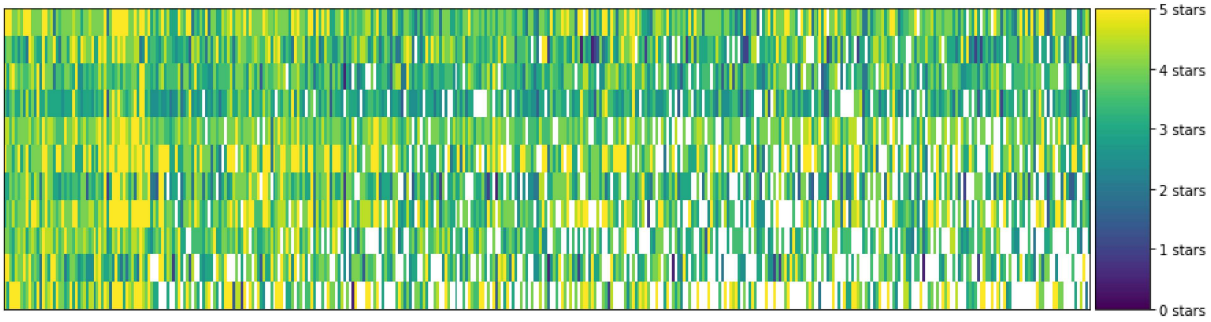



```
0
_________
Platoon (1986)                                          4.333333
Schindler's List (1993)                                 4.326923
Spirited Away (Sen to Chihiro no kamikakushi) (2001)    4.303571
Life Is Beautiful (La Vita è bella) (1997)              4.290323
Office Space (1999)                                     4.276596
Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)    4.237500
Departed, The (2006)                                    4.230769
Rear Window (1954)                                      4.209677
Shaun of the Dead (2004)                                4.202703
One Flew Over the Cuckoo's Nest (1975)                  4.197674
28 Days Later (2002)                                    4.187500
American History X (1998)                               4.180851
Eternal Sunshine of the Spotless Mind (2004)            4.174419
Forrest Gump (1994)                                     4.173077
Run Lola Run (Lola rennt) (1998)                        4.171875
Snatch (2000)                                           4.170455
Amadeus (1984)                                          4.166667
Citizen Kane (1941)                                     4.156250
Casino (1995)                                           4.153846
Boondock Saints, The (2000)                             4.148148
Name: 0, dtype: float64
```

In [32]:

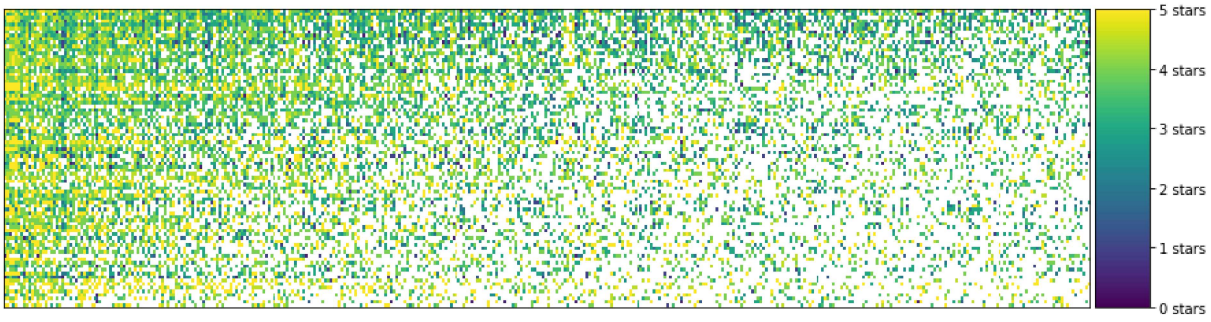
```
i = 0
while True:
    try:
    #predictions testing
        # Pick a cluster ID from the clusters above
        cluster_number = randrange(6)
        #cluster_number = i
        #cluster_number = 12
        # Let's filter to only see the region of the dataset with the most number of
        n_users = 400
        n_movies = 400
        cluster = clustered[clustered.group == cluster_number].drop(['index', 'group
        # Sort and print the cluster
        cluster = helper.sort_by_rating_density(cluster, n_movies, n_users)
        helper.draw_movies_heatmap(cluster, axis_labels=False)
        print(cluster_number)
        print("_________")

        # Pick a user ID from the dataset
        user_id = 314
        # Get all this user's ratings
        user_2_ratings  = cluster.loc[user_id, :]
        # Which movies did they not rate?
        user_2_unrated_movies =  user_2_ratings[user_2_ratings.isnull()]
        # What are the ratings of these movies the user did not rate?
        avg_ratings = pd.concat([user_2_unrated_movies, cluster.mean()], axis=1, joi
        # Let's sort by rating so the highest rated movies are presented first
        User_314_sugest = avg_ratings.sort_values(ascending=False)
        print(User_314_sugest[:20])
        break
    except:
```

```
        print("An exception occurred")
    i = i+1
```



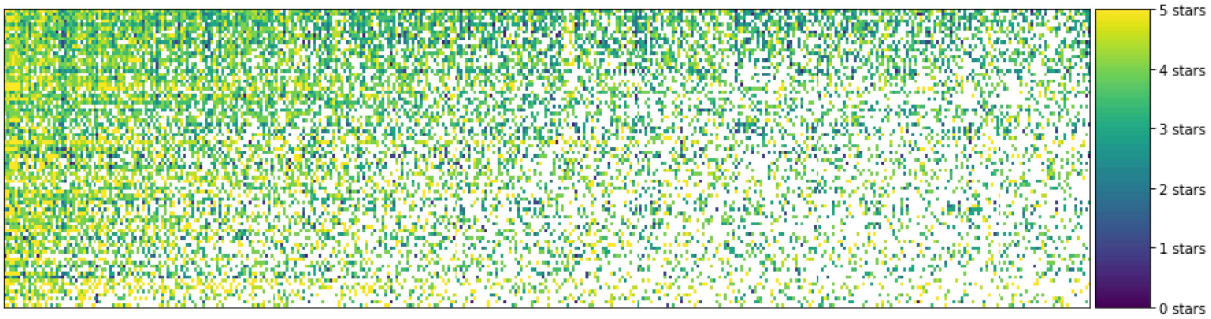
3

An exception occurred



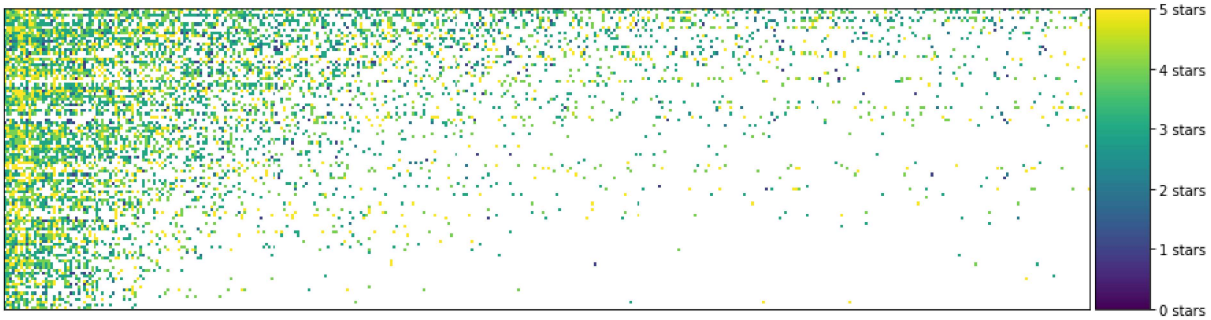
0

An exception occurred



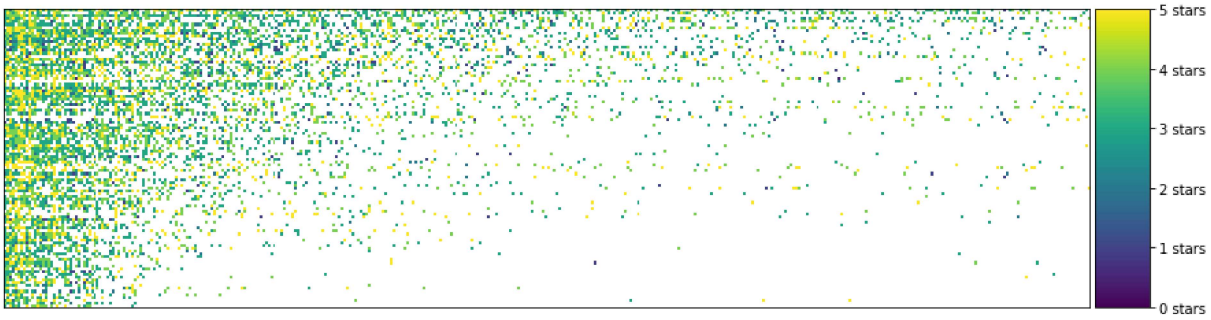
0

An exception occurred



4

An exception occurred



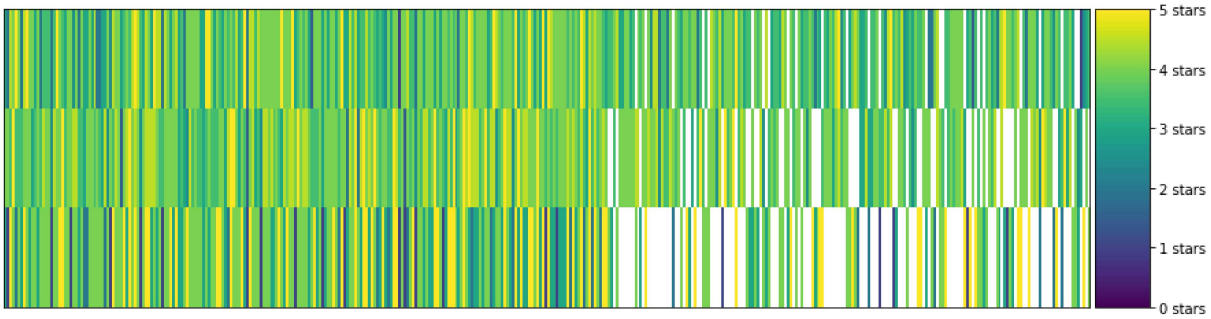
4

An exception occurred

```
C:\Users\User\Documents\GitHub\420_A1\Assignment 1C\Problem_1\helper.py:202: UserWar
ning: Attempting to set identical bottom == top == -0.5 results in singular transfor
mations; automatically expanding.
  heatmap = ax.imshow(most_rated_movies_users_selection,  interpolation='nearest', v
min=0, vmax=5, aspect='auto')
```

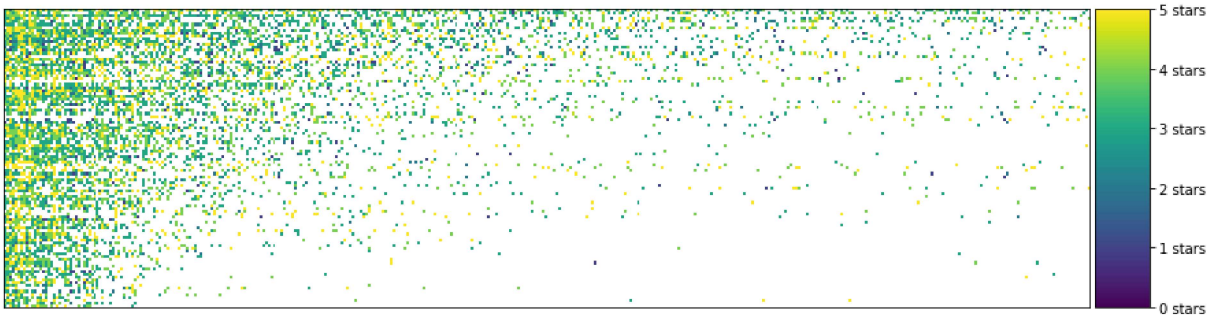
5

An exception occurred



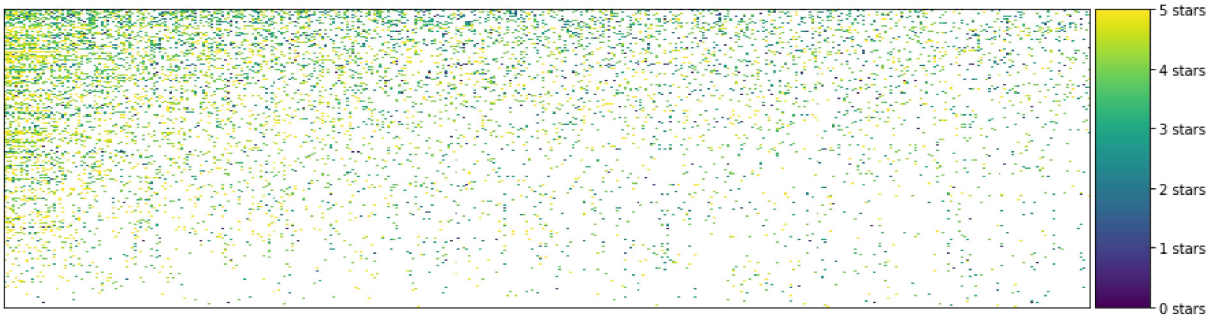
2

An exception occurred



4

An exception occurred




```
1
```

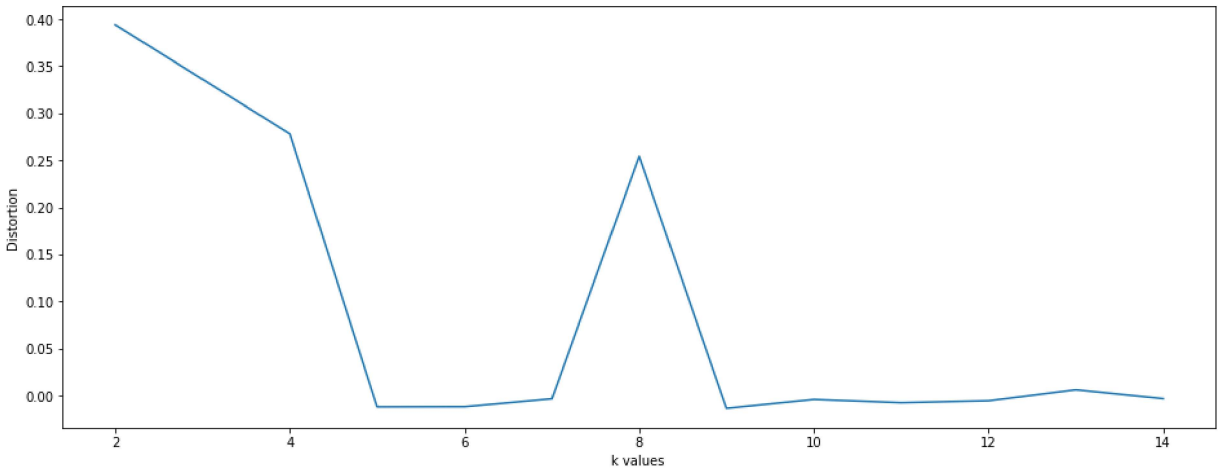
```
Cool Hand Luke (1967)                                                    4.380
000
North by Northwest (1959)                                                4.370
370
Rear Window (1954)                                                       4.317
073
Casablanca (1942)                                                        4.295
918
Departed, The (2006)                                                     4.294
643
Godfather: Part II, The (1974)                                           4.269
231
Chinatown (1974)                                                         4.258
621
One Flew Over the Cuckoo's Nest (1975)                                   4.256
579
Green Mile, The (1999)                                                   4.254
717
American History X (1998)                                                4.250
000
Pulp Fiction (1994)                                                      4.244
526
Taxi Driver (1976)                                                       4.244
186
Goodfellas (1990)                                                        4.241
379
Usual Suspects, The (1995)                                               4.227
778
Singin' in the Rain (1952)                                               4.227
273
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)    4.216
981
To Kill a Mockingbird (1962)                                             4.214
286
Lock, Stock & Two Smoking Barrels (1998)                                 4.204
545
Apocalypse Now (1979)                                                    4.202
128
Shining, The (1980)                                                      4.197
917
Name: 0, dtype: float64
```

In [33]:

```
# Calculate error values for all k values we're interested in
#possible_k_values = range(2, 100)
possible_k_values = range(2, 15)
errors_per_k = [helper.clustering_errors(k, sparse_ratings) for k in possible_k_valu
#uses silhouette_score
fig, ax = plt.subplots(figsize=(16, 6))
plt.plot(possible_k_values, errors_per_k)
plt.xlabel('k values ')
plt.ylabel('Distortion')
```

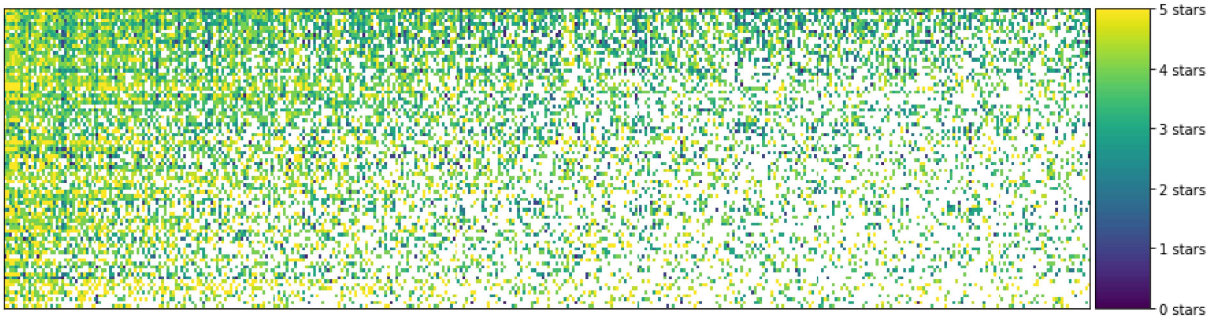
Out[33]:

```
Text(0, 0.5, 'Distortion')
```

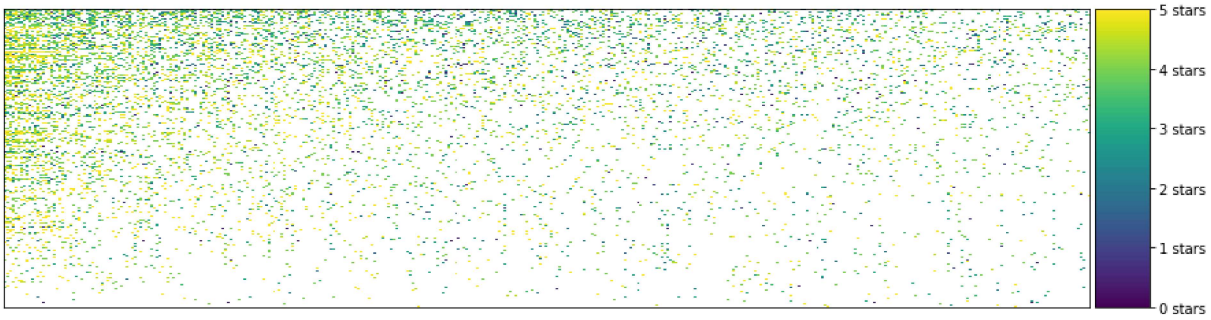


In [34]:
```
# Calculate error values for all k values we're interested in
#possible_k_values = range(2, 100)
#possible_k_values = range(2, 30)
#errors_per_k = [helper.clustering_errors(k, sparse_ratings) for k in possible_k_val
#uses silhouette_score
#fig, ax = plt.subplots(figsize=(16, 6))
#plt.plot(possible_k_values, errors_per_k)
#plt.xlabel('k values ')
#plt.ylabel('Distortion')
```

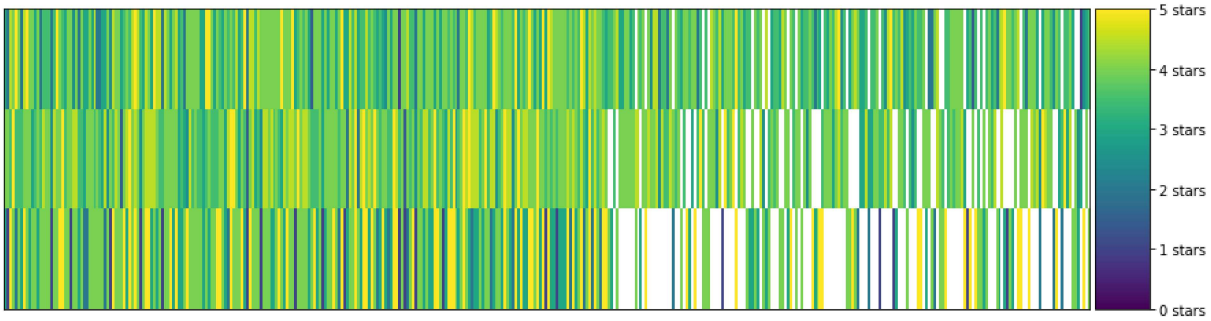
In [35]:
```
i = 0
while i<7:
    try:
    #predictions testing
        # Pick a cluster ID from the clusters above
        cluster_number = i
        #cluster_number = i
        #cluster_number = 12
        # Let's filter to only see the region of the dataset with the most number of
        n_users = 400
        n_movies = 400
        cluster = clustered[clustered.group == cluster_number].drop(['index', 'group
        # Sort and print the cluster
        cluster = helper.sort_by_rating_density(cluster, n_movies, n_users)
        helper.draw_movies_heatmap(cluster, axis_labels=False)
        print(cluster_number)
        print("_________")
    except:
        print("An exception occurred")
    i = i+1
```



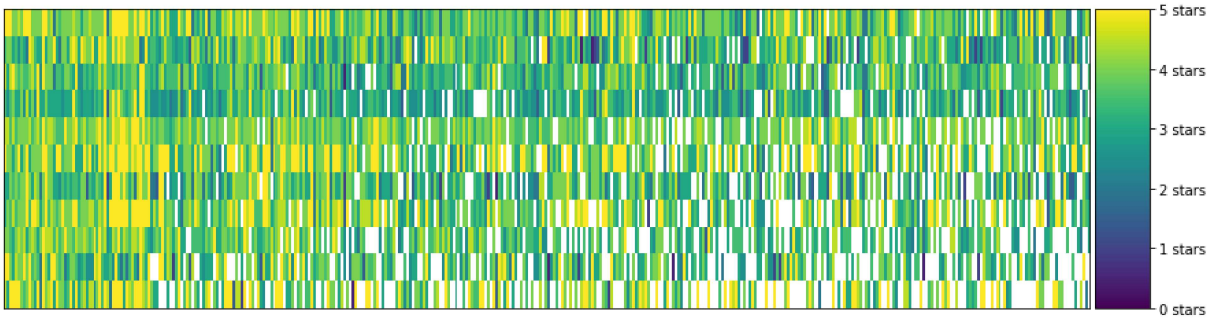

```
0
_________
```



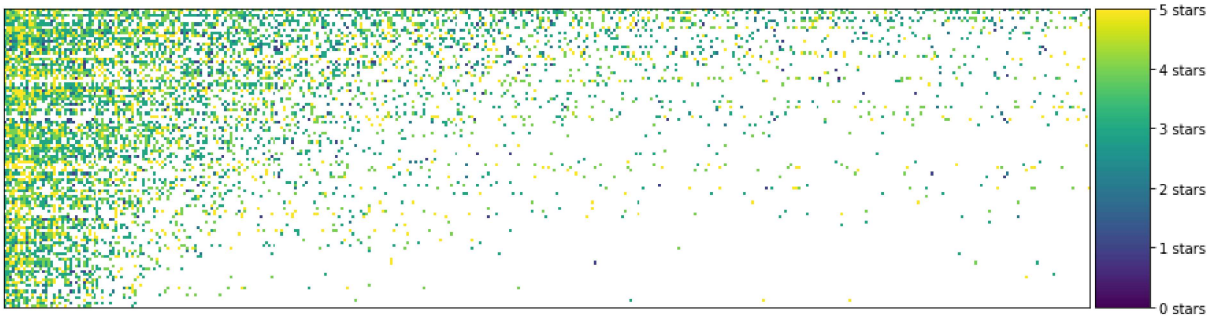
1



2



3

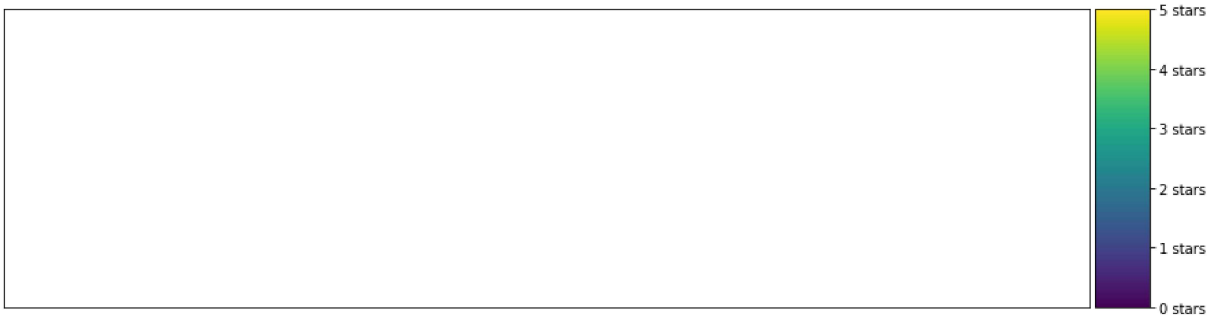


4

```
C:\Users\User\Documents\GitHub\420_A1\Assignment 1C\Problem_1\helper.py:202: UserWar
ning: Attempting to set identical bottom == top == -0.5 results in singular transfor
mations; automatically expanding.
  heatmap = ax.imshow(most_rated_movies_users_selection,  interpolation='nearest', v
min=0, vmax=5, aspect='auto')
```

5

```
C:\Users\User\Documents\GitHub\420_A1\Assignment 1C\Problem_1\helper.py:202: UserWar
ning: Attempting to set identical bottom == top == -0.5 results in singular transfor
mations; automatically expanding.
  heatmap = ax.imshow(most_rated_movies_users_selection,  interpolation='nearest', v
min=0, vmax=5, aspect='auto')
```



6

In [36]:
```
##user 4 most type of viewed movie
```

In [37]:
```
ratings_title = pd.merge(ratings, movies[['movieId', 'title','genres']], on='movieId

ratings_title = ratings_title[ratings_title['userId'] == 4]

#ratings_title.sort_values(by=['rating'], ascending=False).head()
ratings_title = ratings_title[ratings_title['rating'] > 3]
#
#ratings_title.head()
ratings_title['genres'].value_counts()
```

Out[37]:
```
Drama                                          15
Comedy                                         13
Comedy|Drama                                   10
Drama|Romance                                   6
Comedy|Drama|Romance                            6
                                               ..
Crime|Drama|Film-Noir                           1
Action|Adventure|Drama|Thriller|Western         1
Animation|Children|Comedy|Musical               1
Western                                         1
Adventure|Fantasy|Musical                       1
Name: genres, Length: 65, dtype: int64
```

In [38]:
```
#User_4_sugest = pd.merge(User_4_sugest, movies[['genres']], on='movieId' )
#User_4_sugest['genres'].value_counts()

print(User_4_sugest[:10])
```

```
Shawshank Redemption, The (1994)          4.445122
Cool Hand Luke (1967)                     4.380000
North by Northwest (1959)                 4.370370
Rear Window (1954)                        4.317073
Casablanca (1942)                         4.295918
Departed, The (2006)                      4.294643
Godfather: Part II, The (1974)            4.269231
Chinatown (1974)                          4.258621
One Flew Over the Cuckoo's Nest (1975)    4.256579
Dark Knight, The (2008)                   4.255319
Name: 0, dtype: float64
```

In [39]:
```
#ecommendations = movies[movies['title'] == 'Chinatown (1974)'
#                         or 'Citizen Kane (1941)'
#                         or 'Lawrence of Arabia (1962)'
#                         or 'Dr. Strangelove or: How I Learned to Stop Worrying and
```

```
#                           or 'CShawshank Redemption, The (1994)']


recom = movies[movies['title'].isin(['Raging Bull (1980)',
                                     'Lawrence of Arabia (1962)',
                                     'It\'s a Wonderful Life (1946)',
                                     'Shawshank Redemption, The (1994)',
                                     'Office Space (1999)' ])]
recom
```

Out[39]:

	movieId	title	genres
277	318	Shawshank Redemption, The (1994)	Crime|Drama
733	953	It's a Wonderful Life (1946)	Children|Drama|Fantasy|Romance
906	1204	Lawrence of Arabia (1962)	Adventure|Drama|War
929	1228	Raging Bull (1980)	Drama
1883	2502	Office Space (1999)	Comedy|Crime

In [40]:
```
ratings_title = pd.merge(ratings, movies[['movieId', 'title','genres']], on='movieId

ratings_title = ratings_title[ratings_title['userId'] == 42]

#ratings_title.sort_values(by=['rating'], ascending=False).head()
ratings_title = ratings_title[ratings_title['rating'] > 3]
#
#ratings_title.head()
ratings_title['genres'].value_counts()
```

Out[40]:
```
Comedy                       33
Drama                        22
Comedy|Romance               18
Action|Adventure|Thriller     9
Drama|Romance                 9
                             ..
Drama|Romance|Thriller        1
Drama|Fantasy|Thriller        1
Comedy|Drama|War              1
Romance                       1
Action|Drama|Western          1
Name: genres, Length: 98, dtype: int64
```

In [41]:
```
print(User_42_sugest[:10])
```

```
Platoon (1986)                                          4.333333
Schindler's List (1993)                                 4.326923
Spirited Away (Sen to Chihiro no kamikakushi) (2001)    4.303571
Life Is Beautiful (La Vita è bella) (1997)              4.290323
Office Space (1999)                                     4.276596
Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)    4.237500
Departed, The (2006)                                    4.230769
Rear Window (1954)                                      4.209677
Shaun of the Dead (2004)                                4.202703
One Flew Over the Cuckoo's Nest (1975)                  4.197674
Name: 0, dtype: float64
```

In [42]:
```
recom = movies[movies['title'].isin(['Glory (1989)',
                                     'Star Wars: Episode IV - A New Hope (1977)',
                                     'Patton (1970)',
                                     'Wizard of Oz, The (1939)',
                                     'Titanic (1997)' ])]
recom
```

Out[42]:

	movieId	title	genres
224	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi
701	919	Wizard of Oz, The (1939)	Adventure|Children|Fantasy|Musical
941	1242	Glory (1989)	Drama|War
971	1272	Patton (1970)	Drama|War
1291	1721	Titanic (1997)	Drama|Romance

In [43]:
```
ratings_title = pd.merge(ratings, movies[['movieId', 'title','genres']], on='movieId

ratings_title = ratings_title[ratings_title['userId'] == 314]

#ratings_title.sort_values(by=['rating'], ascending=False).head()
ratings_title = ratings_title[ratings_title['rating'] > 3]
#
#ratings_title.head()
ratings_title['genres'].value_counts()
```

Out[43]:
```
Comedy|Romance                                5
Drama                                        4
Comedy|Drama|Romance                         2
Comedy|Fantasy|Romance                       1
Drama|Horror|Sci-Fi                          1
Comedy|Drama|Romance|War                     1
Action|Thriller                              1
Action|Drama|War                             1
Comedy                                       1
Adventure|Drama|Sci-Fi                       1
Children|Drama|Fantasy|Mystery               1
Drama|War                                    1
Drama|Romance                                1
Action|Adventure|Thriller                    1
Crime|Mystery|Thriller                       1
Action|Adventure|Sci-Fi                      1
Crime|Drama                                  1
Action|Sci-Fi                                1
Comedy|Drama|Fantasy|Romance|Thriller        1
Action|Comedy|Sci-Fi                         1
Action|Crime|Drama|War                       1
Adventure|Animation|Children|Drama|Musical|IMAX    1
Adventure|Drama|IMAX                         1
Name: genres, dtype: int64
```

In [44]:
```
print(User_314_sugest[:10])
```

```
Cool Hand Luke (1967)                    4.380000
North by Northwest (1959)                4.370370
Rear Window (1954)                       4.317073
Casablanca (1942)                        4.295918
Departed, The (2006)                     4.294643
Godfather: Part II, The (1974)           4.269231
Chinatown (1974)                         4.258621
One Flew Over the Cuckoo's Nest (1975)   4.256579
Green Mile, The (1999)                   4.254717
American History X (1998)                4.250000
Name: 0, dtype: float64
```

In [45]:
```
recom = movies[movies['title'].isin(['Cool Hand Luke (1967)',
                                     'Shawshank Redemption, The (1994)',
                                     'Lock, Stock & Two Smoking Barrels (1998)',
                                     'North by Northwest (1959)',
                                     'Rear Window (1954)' ])]
recom
```

Out[45]:

	movieId	title	genres
277	318	Shawshank Redemption, The (1994)	Crime|Drama
686	904	Rear Window (1954)	Mystery|Thriller
690	908	North by Northwest (1959)	Action|Adventure|Mystery|Romance|Thriller
975	1276	Cool Hand Luke (1967)	Drama
1917	2542	Lock, Stock & Two Smoking Barrels (1998)	Comedy|Crime|Thriller

In [ ]:

```
                        loss.backward()
                        # scheduler.step(loss)
                        optimizer.step()

                        # loss_meter.add(loss.item())
                        # confusion_matrix.add(outputs.detach(), labels.detach())

                # statistics
                # cacul `running_loss` and `running_corrects`
                running_loss += loss.item() * inputs.size(0)
                # torch.sum(preds == labels.data)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase == 'train' and epoch_acc > best_train_acc:
                temp = epoch_acc
            if phase == 'val' and epoch_acc > 0 and epoch_acc < temp:
                best_train_acc = temp
                best_val_acc = epoch_acc
                best_iteration = epoch
                best_model_wts = copy.deepcopy(model.state_dict())

    # caculator the time
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best epoch: {:4f}'.format(best_iteration))
    print('Best train Acc: {:4f}'.format(best_train_acc))
    print('Best val Acc: {:4f}'.format(best_val_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model


if __name__ == '__main__':
    model_train = train_model(model_conv, criterion,
                              optimizer_conv, exp_lr_scheduler)
    torch.save(
        model_train, 'GenderTest.pkl')

```

```python
def visualize(data, preds):
    viz = visdom.Visdom(env='main')
    # print(data.size()) #torch.Size([4, 3, 224, 224])
    out = make_grid(data)
    # print(out.size()) #torch.Size([3, 228, 906])
    #caculator std,mean correctly
    inp = torch.transpose(out, 0, 2)
    # print(inp.size()) #return torch.Size([906, 228, 3])
    mean = torch.FloatTensor([0.485, 0.456, 0.406])
    std = torch.FloatTensor([0.229, 0.224, 0.225])
    inp = std * inp + mean
    # transoise data
    inp = torch.transpose(inp, 0, 2)
    # print(inp.size()) #returntorch.Size([3, 228, 906])

    # set batch size as 4
    viz.images(inp, opts=dict(title='{},{},{},{}'.format(
        preds[0].item(), preds[1].item(), preds[2].item(), preds[3].item())))

    # viz.images(inp, opts=dict(title='{}'.format(preds[0].item())))


def self_dataset():
    data_test_root = 'Test_Data\Originals'
    test_data = GenderData(data_test_root)
    dataloaders = data.DataLoader(
        test_data, batch_size=4, shuffle=True, num_workers=0)
    for inputs in dataloaders:
        inputs = inputs.to(device)   # data inputs
        outputs = model_test(inputs)
        _, preds = torch.max(outputs, 1)
        visualize(inputs, preds)


if __name__ == '__main__':
    # device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    device = torch.device('cpu')
    print(device)

    model_test = torch.load(
        'GenderTest.pkl')

    model_test.to(device)
    model_test.eval()

    dataloaders = self_dataset()

```

```
    img = cv2.imread(pngfile)
    if dict[os.path.basename(pngfile)] == '0':
        imgpath = os.path.join(
            'gender_data\\train\\0', os.path.basename(pngfile))
        cv2.imwrite(imgpath, img)
    elif dict[os.path.basename(pngfile)] == '1':
        imgpath = os.path.join(
            'gender_data\\train\\1', os.path.basename(pngfile))
        cv2.imwrite(imgpath, img)
    # elif dict[os.path.basename(pngfile)] == '-1':
    #     imgpath = os.path.join(
    #         'Train_Data\gender_data\\train\\-1', os.path.basename(pngfile))

    k = k+1

print(k)
# exit()

names = []
gender = []
with open('Test_Data\Test.csv') as f:
    f_csv = csv.reader(f)
    headers = next(f_csv)
    for row in f_csv:
        names.append(row[0])
        gender.append(row[1])
dict = {}

for j in range(len(names)):
    dict[names[j]] = gender[j]

k = 0
for pngfile in glob.glob('Test_Data\Originals\*.png'):
    img = cv2.imread(pngfile)
    if dict[os.path.basename(pngfile)] == '0':
        imgpath = os.path.join(
            'gender_data\\val\\0', os.path.basename(pngfile))
        cv2.imwrite(imgpath, img)
    elif dict[os.path.basename(pngfile)] == '1':
        imgpath = os.path.join(
            'gender_data\\val\\1', os.path.basename(pngfile))
        cv2.imwrite(imgpath, img)
    # elif dict[os.path.basename(pngfile)] == '-1':
    #     imgpath = os.path.join(
    #         'gender_data\\val\\-1', os.path.basename(pngfile))

    k = k+1

print(k)
exit()

```