859–864, 2019, doi: 10.1109/ICMLA.2019.00149.

[10] A. Gambi, T. Huynh, and G. Fraser, “Automatically reconstructing car crashes from police reports for testing self-driving cars,” *Proc. - 2019 IEEE/ACM 41st Int. Conf. Softw. Eng. Companion, ICSE-Companion 2019*, pp. 290–291, 2019, doi: 10.1109/ICSE-Companion.2019.00119.

APPENDIX

This page contains all the code that are mainly the source of how the algorithm will behave, this is originally a jupyter notebook file and converted to a python file for easier interpretation, however the code is expected to run only on notebook with a cell environment (I tried to run the code in normal python .py file and it will stuck on movie info since InteractiveShell is not present) this is the whole code for this project, it is expected to work with normal anaconda distribution package, the code works even better on Ubuntu Linux in my own experience.

Coding Euclidean Distance Full Code

```
1. #!/usr/bin/env python
```

```
2. # coding: utf-8


3. # In[1]:


4. # Importing packages
5. from math import sqrt
6. import pandas as pd
7. import numpy as np
8. from matplotlib import pyplot as plt


9. # I need to get more than one output Line
10.      from IPython.core.interactiveshell import InteractiveShell
11.      InteractiveShell.ass_node_interactivity = "all"


12.      # In[2]:


13.      # Getting the Dataset
14.      movies = pd.read_csv('ml-latest/movies.csv')
15.      movies.head()


16.      # In[3]:


17.      ratings=pd.read_csv('ml-latest-
   small/ratings.csv',usecols=['userId','movieId','rating'])
18.      ratings.head()


19.      # In[4]:


20.      # Reading users dataset into a pandas dataframe object.
21.      u_cols = ['user_Id', 'age', 'sex', 'occupation', 'zip_code']
22.      users = pd.read_csv('ml-latest-
   small/users.dat', sep='::', names=u_cols,
23.      encoding='latin-1')


24.      # In[5]:


25.      users.head()


26.      # In[6]:


27.      tags = pd.read_csv('ml-latest-small/tags.csv')
28.      tags.head()
```

```
# In[7]:

del tags['timestamp']

# In[8]:

movies.info()

# In[9]:

ratings['rating'].describe(include='all')

# Most user rate at 3.5 Star

# In[10]:

ratings.groupby('rating')['movieId'].nunique()
ratings.hist(column='rating',figsize=(10,10),bins=5,grid=False)

# In[11]:

tag_counts = tags['tag'].value_counts()
tag_counts[:15]
tag_counts[:15].plot(kind='pie', figsize=(10,10),autopct='%1.1f%%
')

# ## Counting Number of movies
# Total movies = 58098
# ***

# ## Removing movies with no genres
# genre_filter= (movies['genres'] == '(no genres listed) ')
# ***

# In[12]:

genre_filter = (movies['genres']== '(no genres listed) ')

# In[13]:
```

```
52.      movies=movies[~genre_filter]
53.      movies=movies.reset_index(drop=True) ## Reset / Re Index DataFram
   e


54.      # In[14]:


55.      genres_count= {}
56.      for row in range(movies['movieId'].count()):
57.      for genre in movies['genres'][row].split("|"):
58.      if(genre !=''):
59.      genres_count[genre]= genres_count.get(genre,0)+1
60.      genres_count


61.      # counting most watched movies type /genres


62.      # In[15]:


63.      fig, ax= plt.subplots(figsize=(15,10))
64.      ax.barh(range(len(genres_count)),genres_count.values())
65.      plt.yticks(range(len(genres_count)),list(genres_count.keys()))
66.      plt.xlabel('Movie Count')
67.      plt.title("Genre Popularity")
68.      for i, v in enumerate(genres_count.values()):
69.      ax.text(v + 20, i + .10, v, color='blue', fontweight='bold')


70.      # In[16]:


71.      # Getting series of list by applying split operation.
72.      movies.genres = movies.genres.str.split('|')


73.      # Getting distinc genre types for generating columns of genre typ
   e
74.      genre_columns = list(set([j for i in movies['genres'].tolist() fo
   r j in i]))


75.      # Iterating over every list to create and fill values into column
   s
76.      for j in genre_columns:
77.      movies[j] = 0
78.      for i in range(movies.shape[0]):
79.      for j in genre_columns:
80.      movies.loc[i,j] = 1


81.      # In[17]:
```

```
82.      movies.head()

83.      # In[18]:

84.      ratings.head()

85.      # In[19]:

86.      ratings.shape

87.      # In[20]:

88.      #Function to get the rating given by a user to a movie.
89.      def get_rating(userid,movieid):
90.      return (ratings.loc[(ratings.userId==userid) & (ratings.movieId =
   = movieid),'rating'].iloc[0])

91.      # Function to get the list of all movie ids the specified user ha
   s rated.
92.      def get_movieids(userid):
93.      return (ratings.loc[(ratings.userId==userid),'movieId'].tolist())

94.      # Function to get the movie titles against the movie id.
95.      def get_movie_title(movieid):
96.      return (movies.loc[(movies.movieId == movieid),'title'].iloc[0])

97.      def get_ID_Title(userid): # Getting a list of rated movies by a s
   pecific user
98.      return (movies.loc[(ratings.userId == userid),'title'].tolist())

99.      # In[46]:

100.     get_rating(1,1)

101.     # In[21]:

102.     get_ID_Title(11)

103.     # In[22]:
```

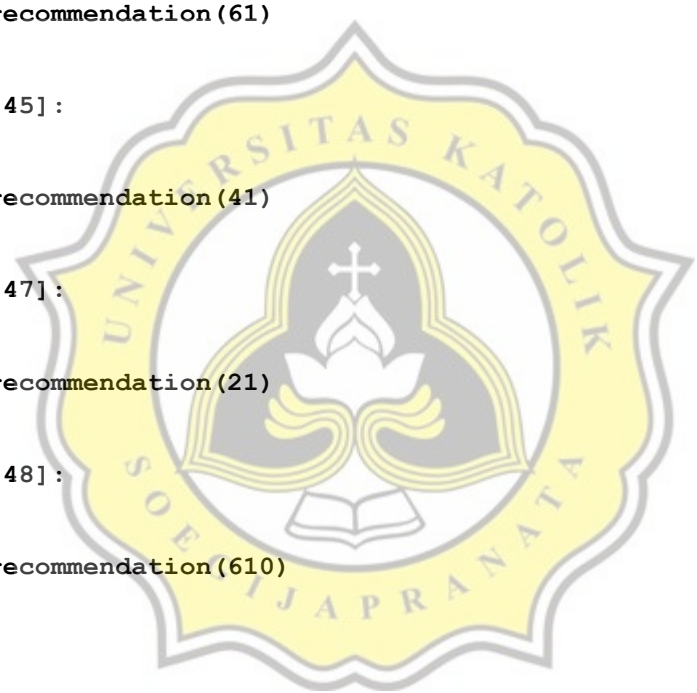
```
get_movieids(11)


# ## The following is the
# ## Formula for Euclidean Distance
#
# $$ d1(p,q) = \sqrt{(q1-p1)^2} $$
#
# ***


# In[23]:


def euclidean_distance(user1,user2):
# Getting details of person 1 and person 2
df_first= ratings.loc[ratings['userId']==user1]
df_second= ratings.loc[ratings.userId==user2]


# finding similar movies for person 1 and 2 LATER THIS WILL BE
# seperrated by rating_x and rating_y


df= pd.merge(df_first,df_second,how='inner',on='movieId')


# if no similar movie are found, return 0 (NO SIMILARITY)
if(len(df)==0): return 0


# sum of squared difference between ratings
distance=pow((df['rating_x']-df['rating_y']),2)
total_euclidean = sum(distance)
return 1/(1+total_euclidean)


euclidean_distance(1,21)


# In[24]:


euclidean_distance(1,9)


# In[25]:


euclidean_distance(1,21)


# In[26]:


euclidean_distance(1,310)
```

```
# In[27]:

euclidean_distance(11,41)

# In[28]:

euclidean_distance(61,89)

# In[29]:

euclidean_distance(61,89)

# In[30]:

euclidean_distance(23,86)

# In[31]:

euclidean_distance(1,85)

# In[32]:

euclidean_distance(1,77)

# In[33]:

euclidean_distance(1,53)

# In[34]:

euclidean_distance(1,44)

# In[35]:

def most_similar_users_(user1,number_of_users,metric='euclidean_d
istance'):
    '''
```

```
151.     user1 : Targeted User
152.     number_of_users : number of most similar users you want to user1.
153.     metric : metric to be used to calculate inter-
   user similarity score. ('manhattan' or else)
154.     '''
155.     # Getting distinct user ids.
156.     user_ids = ratings.userId.unique().tolist()


157.     # Getting similarity score between targeted and every other user
   in the list(or subset of the list).
158.     if(metric == 'manhattan_distance'):
159.     # similarity_score = [(pearson_correlation_score(user1,nth_user),
   nth_user) for nth_user in user_ids[:100] if nth_user != user1]
160.     similarity_score = [(manhattan_distance(user1,nth_user),nth_user)
    for nth_user in user_ids[:100] if nth_user != user1]
161.     else:
162.     similarity_score = [(euclidean_distance(user1,nth_user),nth_user)
    for nth_user in user_ids[:100] if nth_user != user1]


163.     # Sorting in descending order.
164.     similarity_score.sort()
165.     similarity_score.reverse()


166.     # Returning the top most 'number_of_users' similar users.
167.     return similarity_score[:number_of_users]


168.     # In[36]:


169.     most_similar_users_(23,20)


170.     # In[37]:


171.     most_similar_users_(1,20) # 20 Most similar user with user 1


172.     # In[38]:


173.     def get_recommendation(userid):
174.     user_ids = ratings.userId.unique().tolist()
175.     total = {}
176.     similariy_sum = {}


177.     # Iterating over subset of user ids.
178.     for user in user_ids[:100]:


179.     # not comparing the user to itself (obviously!)
```

```
180.     if user == userid:
181.     continue


182.     # Getting similarity score between the users.
183.     # score = pearson_correlation_score(userid,user)
184.     score = euclidean_distance(userid,user)


185.     # not considering users having zero or less similarity score.
186.     if score <= 0:
187.     continue


188.     # Getting weighted similarity score and sum of similarities betwe
   en both the users.
189.     for movieid in get_movieids(user):
190.     # Only considering not watched/rated movies
191.     if movieid not in get_movieids(userid) or get_rating(userid,movie
   id) == 0:
192.     total[movieid] = 0
193.     total[movieid] += get_rating(user,movieid) * score
194.     similariy_sum[movieid] = 0
195.     similariy_sum[movieid] += score


196.     # Normalizing ratings
197.     ranking = [(tot/similariy_sum[movieid],movieid) for movieid,tot i
   n total.items()]
198.     ranking.sort()
199.     ranking.reverse()


200.     # Getting movie titles against the movie ids.
201.     recommendations = [get_movie_title(movieid) for score,movieid in
   ranking]
202.     return recommendations[:20]


203.     # In[39]:


204.     get_recommendation(1)


205.     # In[40]:


206.     get_recommendation(9)


207.     # In[41]:


208.     get_recommendation(2)
```

```
# In[42]:
get_recommendation(3)
# In[43]:
get_recommendation(11)
# In[44]:
get_recommendation(61)
# In[45]:
get_recommendation(41)
# In[47]:
get_recommendation(21)
# In[48]:
get_recommendation(610)
```



Coding Manhattan Distance Full Code

```
1. #!/usr/bin/env python
2. # coding: utf-8
```

```
# In[1]:



# Importing packages
from math import sqrt
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt


# I need to get more than one output Line
    from IPython.core.interactiveshell import InteractiveShell
    InteractiveShell.ass_node_interactivity = "all"



    # In[2]:



    # Getting the Dataset
    movies = pd.read_csv('ml-latest/movies.csv')
    movies.head()



    # In[3]:



    ratings=pd.read_csv('ml-latest-
small/ratings.csv',usecols=['userId','movieId','rating'])
    ratings.head()



    # In[4]:



    # Reading users dataset into a pandas dataframe object.
    u_cols = ['user_Id', 'age', 'sex', 'occupation', 'zip_code']
    users = pd.read_csv('ml-latest-small/users.dat', sep='::',
names=u_cols,
    encoding='latin-1')



    # In[5]:
```

```
25.     users.head()


26.     # In[6]:


27.     tags = pd.read_csv('ml-latest-small/tags.csv')
28.     tags.head()


29.     # In[7]:


30.     del tags['timestamp']


31.     # In[8]:


32.     movies.info()


33.     # In[9]:


34.     ratings['rating'].describe(include='all')


35.     # Most user rate at 3.5 Star


36.     # In[10]:


37.     ratings.groupby('rating')['movieId'].nunique()
38.     ratings.hist(column='rating',figsize=(10,10),bins=5,grid=False)
```

```
39.      # In[11]:


40.      tag_counts = tags['tag'].value_counts()
41.      tag_counts[:15]
42.      tag_counts[:15].plot(kind='pie',
   figsize=(10,10),autopct='%1.1f%%')


43.      # ## Counting Number of movies
44.      # Total movies = 58098
45.      # ***


46.      # ## Removing movies with no genres
47.      # genre_filter= (movies['genres'] == '(no genres listed) ')
48.      # ***


49.      # In[12]:


50.      genre_filter = (movies['genres']== '(no genres listed) ')


51.      # In[13]:


52.      movies=movies[~genre_filter]
53.      movies=movies.reset_index(drop=True) ## Reset / Re Index
   DataFrame


54.      # Counting genres


55.      # In[14]:


56.      genres_count= {}
57.      for row in range(movies['movieId'].count()):
58.      for genre in movies['genres'][row].split("|"):
59.      if(genre !=''):
60.      genres_count[genre]= genres_count.get(genre,0)+1
```

```
61.      genres_count


62.      # counting most watched movies type /genres

63.      # In[15]:


64.      fig, ax= plt.subplots(figsize=(15,10))
65.      ax.barh(range(len(genres_count)),genres_count.values())
66.      plt.yticks(range(len(genres_count)),list(genres_count.keys()))
67.      plt.xlabel('Movie Count')
68.      plt.title("Genre Popularity")
69.      for i, v in enumerate(genres_count.values()):
70.      ax.text(v + 20, i + .10, v, color='blue', fontweight='bold')


71.      # In[16]:


72.      # Getting series of lists by applying split operation.
73.      movies.genres = movies.genres.str.split('|')

74.      # Getting distinct genre types for generating columns of genre
   type.
75.      genre_columns = list(set([j for i in movies['genres'].tolist()
   for j in i]))

76.      # Iterating over every list to create and fill values into
   columns.
77.      for j in genre_columns:
78.      movies[j] = 0
79.      for i in range(movies.shape[0]):
80.      for j in genre_columns:
81.      if(j in movies['genres'].iloc[i]):
82.      movies.loc[i,j] = 1


83.      # ## Showing Genre that matched the movie
84.      # in a data frame
85.      # ***

86.      # In[17]:
```

```
movies.head()

# In[18]:

ratings.head()

# In[19]:

ratings.shape

# In[20]:

#Function to get the rating given by a user to a movie.
def get_rating(userid,movieid):
return (ratings.loc[(ratings.userId==userid) & (ratings.movieId
== movieid),'rating'].iloc[0])

# Function to get the list of all movie ids the specified user
has rated.
def get_movieids(userid):
return (ratings.loc[(ratings.userId==userid),'movieId'].tolist())

# Function to get the movie titles against the movie id.
def get_movie_title(movieid):
return (movies.loc[(movies.movieId == movieid),'title'].iloc[0])

def get_ID_Title(userid): # Getting a list of rated movies by a
specific user
return (movies.loc[(ratings.userId == userid),'title'].tolist())

# In[21]:
```

```
get_rating(1,1)

# In[22]:

get_movieids(1)

# In[23]:

get_ID_Title(1)

# In[24]:

get_ID_Title(11)

# In[25]:

get_movieids(11)

# In[26]:

get_movieids(1)

# In[27]:

get_ID_Title(1)
```

```
# In[28]:


get_movie_title(1)


# In[29]:


get_movie_title(2)


# ## The following is the
# ## Formula for Euclidean Distance
#
# $$ d1(p,q) = \sqrt{(q1-p1)^2} $$
#
# ***

# In[30]:


def euclidean_distance(user1,user2):
# Getting details of person 1 and person 2
df_first= ratings.loc[ratings['userId']==user1]
df_second= ratings.loc[ratings.userId==user2]

# finding similar movies for person 1 and 2 LATER THIS WILL BE
# seperrated by rating_x and rating_y

df= pd.merge(df_first,df_second,how='inner',on='movieId')

# if no similar movie are found, return 0 (NO SIMILARITY)
if(len(df)==0): return 0

# sum of squared difference between ratings
distance=pow((df['rating_x']-df['rating_y']),2)
total_euclidean = sum(distance)
return 1/(1+total_euclidean)
```

```
142.    euclidean_distance(1,21)


143.    # In[31]:


144.    euclidean_distance(1,9)


145.    # ## The following is the
146.    # ## Formula for Manhattan Distance
147.    #
148.    # $$ d1(p,q) =||p-q|| i = \sum_{i=1}^n|pi-qi| $$
149.    # ***

150.    # In[32]:


151.    def manhattan_distance(user1,user2):
152.    df_first= ratings.loc[ratings['userId']==user1]
153.    df_second= ratings.loc[ratings.userId==user2]

154.    # finding similar movies for person 1 and 2 LATER THIS WILL BE
155.    # seperrated by rating_x and rating_y

156.    df= pd.merge(df_first,df_second,how='inner',on='movieId')

157.    # if no similar movie are found, return 0 (NO SIMILARITY)
158.    if(len(df)==0): return 0

159.    # The Manhattan Distance
160.    sum1_square = sum(df['rating_x'])
161.    sum2_square = sum(df['rating_y'])

162.    sum_absolute=sum((abs(df['rating_x']-df['rating_y'])))
163.    return 1/(1+sum_absolute)


164.    #     return sum(abs(a-b) for a,b in zip(x,y))
```

```
manhattan_distance(1,9)

# In[33]:

manhattan_distance(1,21)

# In[34]:

manhattan_distance(1,310)

# In[35]:

manhattan_distance(11,41)

# In[36]:

manhattan_distance(61,89)

# In[37]:

manhattan_distance(23,86)

# In[38]:

manhattan_distance(1,85)
```

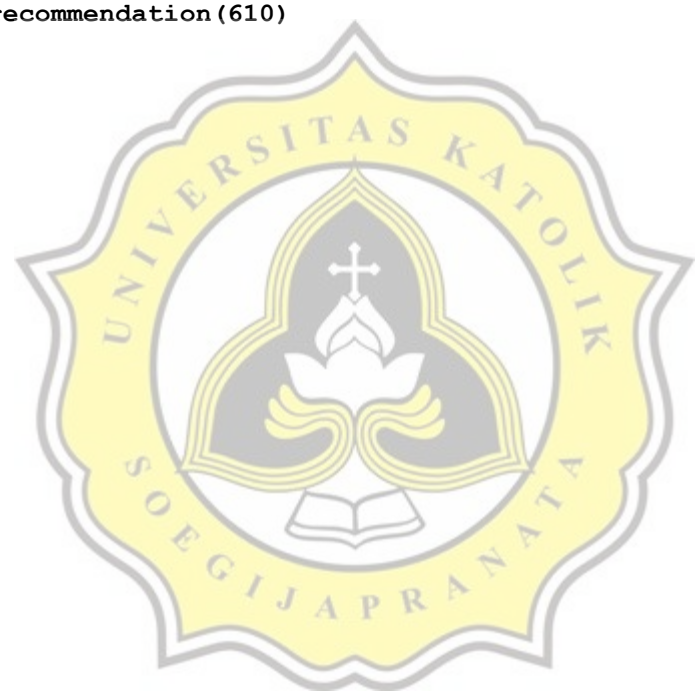
```
# In[39]:


manhattan_distance(1,77)


# In[40]:


manhattan_distance(1,53)


# In[41]:


manhattan_distance(1,44)


# In[42]:


def most_similar_users_(user1,number_of_users,metric='manhattan_distance'):
    '''
    user1 : Targeted User
    number_of_users : number of most similar users you want to user1.
    metric : metric to be used to calculate inter-user similarity score. ('manhattan' or else)
    '''
    # Getting distinct user ids.
    user_ids = ratings.userId.unique().tolist()

    # Getting similarity score between targeted and every other user in the list(or subset of the list).
    if(metric == 'manhattan_distance'):
    # similarity_score = [(pearson_correlation_score(user1,nth_user),nth_user) for nth_user in user_ids[:100] if nth_user != user1]
    similarity_score = [(manhattan_distance(user1,nth_user),nth_user) for nth_user in user_ids[:100] if nth_user != user1]
    else:
```

```
198.     similarity_score =
   [(euclidean_distance_score(user1,nth_user),nth_user) for nth_user in
   user_ids[:100] if nth_user != user1]


199.     # Sorting in descending order.
200.     similarity_score.sort()
201.     similarity_score.reverse()


202.     # Returning the top most 'number_of_users' similar users.
203.     return similarity_score[:number_of_users]



204.     # In[43]:



205.     most_similar_users_(23,20)



206.     # In[44]:



207.     most_similar_users_(1,20) # 20 Most similar user with user 1



208.     # In[45]:



209.     def get_recommendation(userid):
210.     user_ids = ratings.userId.unique().tolist()
211.     total = {}
212.     similariy_sum = {}


213.     # Iterating over subset of user ids.
214.     for user in user_ids[:100]:


215.     # not comparing the user to itself (obviously!)
216.     if user == userid:
217.     continue


218.     # Getting similarity score between the users.
219.     # score = pearson_correlation_score(userid,user)
220.     score = manhattan_distance(userid,user)
```

```
221.     # not considering users having zero or less similarity score.
222.     if score <= 0:
223.     continue


224.     # Getting weighted similarity score and sum of similarities
   between both the users.
225.     for movieid in get_movieids(user):
226.     # Only considering not watched/rated movies
227.     if movieid not in get_movieids(userid) or
   get_rating(userid,movieid) == 0:
228.     total[movieid] = 0
229.     total[movieid] += get_rating(user,movieid) * score
230.     similariy_sum[movieid] = 0
231.     similariy_sum[movieid] += score


232.     # Normalizing ratings
233.     ranking = [(tot/similariy_sum[movieid],movieid) for movieid,tot
   in total.items()]
234.     ranking.sort()
235.     ranking.reverse()


236.     # Getting movie titles against the movie ids.
237.     recommendations = [get_movie_title(movieid) for score,movieid in
   ranking]
238.     return recommendations[:20]



239.     # In[46]:



240.     get_recommendation(1)



241.     # In[47]:



242.     get_recommendation(85)



243.     # In[48]:
```

```
get_recommendation(77)

# In[49]:

get_recommendation(9)

# In[50]:

get_recommendation(2)

# In[51]:

get_recommendation(3)

# In[52]:

get_recommendation(11)

# In[49]:

get_recommendation(61)

# In[50]:

get_recommendation(41)
```

```
# In[51]:


get_recommendation(21)


# In[52]:


get_recommendation(610)
```




UNIVERSITAS KATOLIK
SOEGIJAPRANATA

PLAGIARISM CHECK.ORG

2.09% PLAGIARISM APPROXIMATELY | **12.67%** IN REFERENCES

Report #13366305

INTRODUCTION Background Movie Recommender system is a very important algorithm for streaming services such as YouTube, Netflix, Amazon Prime and many more. The demand for users to keep using the service is high and to do that the company have to implement a method such that the users will get a recommendation of preferred movie. Recommender System is widely utilized in today s standard starting from e-commerce, games store such as Valve Steam and streaming services such as YouTube and Netflix. Streaming services have been a huge popularity for the last 5 years, starting from TV shows to Documentary even a Blockbuster movie. The advancement of technology and internet allows this magnificent works to become a reality where one can watch a movie and then the services will recommend another similar movie keeping the user in the platform as long as they can so the user will keep subscribing and continously enjoying the movie type of their favorite. A prediction algorithm that will correctly guess