Milestone2_01_store_tmdb_complete_csv

April 12, 2017

1 TMDB

We have to breakdown to multiple files for a group of id, or else memory is not enough.

```
In [3]: import tmdbsimple as tmdb
        tmdb.API_KEY = "71e259894a515060876bab2a33d6bdc9"
```

```
In [2]: from imdb import IMDb
        import pandas as pd
        from PIL import Image
        from StringIO import StringIO
        import requests
        import os
        import time
        from shutil import copyfile
        import csv
```

```
In [3]: dir_python_notebook = os.getcwd()
        dir_movie_project = os.path.abspath(os.path.join(dir_python_notebook, os.pa
        dir_data = os.path.join(dir_movie_project, 'data')
```

2 Load data

```
In [26]: # get column names to retrieve information
         tmdb_movie_id = tmdb.Movies(2)
         movie_info = tmdb_movie_id.info()
         tmdb_info_column  = tmdb_movie_id.info().keys()
         tmdb_info_column  = [str(c) for c in tmdb_info_column]
         tmdb_info_column.sort()
```

```
In [27]: latest_r = requests.get('https://api.themoviedb.org/3/movie/latest?api_key
         latest_id = latest_r.json()['id']
         latest_id
```

```
Out[27]: 451100
```

In [46]:
```
id_start = 220001
id_range = 20000
id_end = id_start + id_range -1
print(id_end)
i_list = range(id_start,id_end+1)
print(i_list[-1])
```

```
240000
240000
```

In [ ]:
```
# if we already have a movie data file, we can just continue appending it i
while id_start < latest_id:
    id_end = id_start + id_range - 1
    i_list = range(id_start,id_end + 1)

    movies = pd.DataFrame(columns=tmdb_info_column)
    new_filename = str(dir_data)+'\\drv_tmdb_movie_details_'+str(id_start)+

    if os.path.isfile(new_filename):
        movies = pd.read_csv(new_filename,index_col=0, sep='\t', encoding='
        if len(movies['id']) >0:
            movie_ids = movies['id'].tolist()
            tmdb_info_column = movies.columns
            i_list = [x for x in i_list if x not in movie_ids]
            movies = pd.DataFrame(columns=tmdb_info_column)
    else:
        movies.to_csv(new_filename,header =tmdb_info_column, sep='\t', enco

    for i in i_list:
        #skip the non-existing movie ids
        if (i % 40 == 0):
            # make sure we do not hit API limit
            time.sleep(12)
            movies.to_csv(new_filename,mode = 'a',header=False, sep='\t', e
            movies = pd.DataFrame(columns=tmdb_info_column)
        try:
            tmdb_movies = tmdb.Movies(i)
            info = tmdb_movies.info()

            movie_details = []
            for c in tmdb_info_column:
                if info.has_key(c):
                    if info[c] is not None:
                        if c in ['genres','spoken_languages','production_co
                            movie_details.append([d['name'] for d in info[c
                        elif c in ['belongs_to_collection']:
                            movie_details.append(info[c]['name'])
```

```
                            else:
                                movie_details.append(info[c])
                        else:
                            movie_details.append(None)
                    else:
                            movie_details.append(None)
                movies.loc[i] = movie_details
            except Exception:
                continue

        movies.to_csv(new_filename,mode = 'a',header=False, sep='\t', encoding=
        movies = pd.DataFrame(columns=tmdb_info_column)
        id_start = id_start + id_range
```

In [34]:
```
id_start = 200001
id_range = 20000

while id_start < 220000:
    id_end = id_start + id_range - 1
    filename_json = str(dir_data)+'\\drv_tmdb_movie_details_'+str(id_start
    filename_csv = str(dir_data)+'\\drv_tmdb_movie_details_'+str(id_start)
    if os.path.isfile(filename_json):
        movies = pd.read_json(filename_json)
        tmdb_info_column = movies.columns
        movies.to_csv(filename_csv,header =tmdb_info_column, sep='\t', enc
    id_start = id_start + id_range
```

In [5]:
```
tmdb_filename = str(dir_data)+'\\drv_tmdb_movie_details_280001_300000.csv'
tmdb_movies = pd.read_csv(tmdb_filename,index_col=0, sep='\t', encoding='ut
```

In [6]:
```
max(tmdb_movies['id'])
```

Out[6]: 300000.0

In [ ]:

Milestone2_02_store_imdb_complete_csv

April 12, 2017

1 IMDB

We have to breakdown to multiple files for a group of id, or else memory is not enough.

```
In [1]: import tmdbsimple as tmdb
        tmdb.API_KEY = "71e259894a515060876bab2a33d6bdc9"

In [2]: import imdb as ib
        from imdb import IMDb
        import pandas as pd
        from PIL import Image
        from StringIO import StringIO
        import requests
        import os
        import time
        from shutil import copyfile
        import types
        import numpy as np
        import csv

In [3]: dir_python_notebook = os.getcwd()
        dir_movie_project = os.path.abspath(os.path.join(dir_python_notebook, os.pa
        dir_data = os.path.join(dir_movie_project, 'data')
```

2 Load data

```
In [4]: def get_imdb_columns():
            imdb = IMDb()
            #get a movie by id
            imdb_movie = imdb.get_movie('0325980')
            #access attributes of the movie by dictionary keys
            imdb.update(imdb_movie)
            imdb_info_column = imdb_movie.keys_alias.values()
            imdb_info_column = list(set(imdb_info_column))
            if 'imdb_id' not in imdb_info_column:
                imdb_info_column.append('imdb_id')
            if 'tmdb_id' not in imdb_info_column:
```

```
            imdb_info_column.append('tmdb_id')
        imdb_info_column.sort()
        return(imdb_info_column)
```

In [5]:
```
get_imdb_columns()
```

Out[5]:
```
['airing',
 'akas',
 'amazon reviews',
 'art direction',
 'assistant director',
 'cast',
 'casting director',
 'certificates',
 'cinematographer',
 'color info',
 'costume department',
 'costume designer',
 'countries',
 'cover url',
 'creator',
 'director',
 'distributors',
 'editor',
 'faqs',
 'full-size cover url',
 'genres',
 'guests',
 'imdb_id',
 'languages',
 'make up',
 'merchandising links',
 'misc links',
 'miscellaneous companies',
 'miscellaneous crew',
 'non-original music',
 'number of episodes',
 'number of seasons',
 'original music',
 'parents guide',
 'photo sites',
 'plot',
 'producer',
 'production companies',
 'production manager',
 'rating',
 'runtimes',
 'set decoration',
```

```
         'sound clips',
         'sound crew',
         'special effects',
         'special effects companies',
         'stunt performer',
         'tmdb_id',
         'video clips',
         'visual effects',
         'writer']
```

```
In [6]: def get_movie_details(imdb_info_column,imdb_id, tmdb_id):
            movie_details = []
            try:
                imdb = IMDb()
                imdb_movie = imdb.get_movie(imdb_id)
                imdb.update(imdb_movie)
                if (len(imdb_movie.keys()) > 0) :
                    movie_details = []
                    for c in imdb_info_column:
                        if imdb_movie.has_key(c):
                            info_field = imdb_movie[c]
                            if info_field is not None:
                                if type(info_field) is list:
                                    if isinstance(info_field[0], ib.Person.Person):
                                        info_list = []
                                        for item in info_field:
                                            info_list.append(item.getID())
                                        movie_details.append(info_list)
                                    elif isinstance(info_field[0], ib.Company.Compa
                                        info_list = []
                                        for item in info_field:
                                            info_list.append(item.getID())
                                        movie_details.append(info_list)
                                    else:
                                        movie_details.append(info_field)
                                else:
                                    movie_details.append(info_field)
                            else:
                                    movie_details.append(None)

                        else:
                            if c == "imdb_id":
                                movie_details.append(imdb_id)
                            elif c == "tmdb_id":
                                movie_details.append(tmdb_id)
                            else:
                                movie_details.append(None)
                else:
```

```
                invalid_imdb_ids.append(i)
                #movies.head(5)
            return movie_details
        except Exception:
            return movie_details
```

In [7]:
```
latest_id = 300000
```

In [8]:
```
imdb_info_column = get_imdb_columns()
imdb_invalid_id_filename = str(dir_data)+'\\drv_imdb_movie_invalid_id.json'

id_start = 280001
id_range = 20000

while id_start < latest_id:
    id_end = id_start + id_range - 1
    tmdb_filename = str(dir_data)+'\\drv_tmdb_movie_details_'+str(id_start)
    imdb_filename = str(dir_data)+'\\drv_imdb_movie_info_'+str(id_start)+'_

    movies = pd.DataFrame(columns=imdb_info_column)
    invalid_ids =  pd.DataFrame(columns= ['imdb_id','tmdb_id'])
    count = 0

    tmdb_movie = pd.read_json(tmdb_filename)

    # only load the one with imdb_id
    tmdb_movie = tmdb_movie[tmdb_movie['imdb_id'].notnull()]
    tmdb_ids = tmdb_movie['id'].tolist()

    # if already loaded, no need to reload
    if os.path.isfile(imdb_filename):
        movies = pd.read_csv(imdb_filename,index_col=0, sep='\t', encoding=
        loaded_tmdb_ids = movies['tmdb_id'].tolist()
        if len(loaded_tmdb_ids) >0:
            tmdb_ids = [x for x in tmdb_ids if x not in loaded_tmdb_ids]
        imdb_info_column = movies.columns
        movies = pd.DataFrame(columns=imdb_info_column)
    else:
        movies.to_csv(imdb_filename,header =imdb_info_column, sep='\t', enc

    # we don't want to waste our effort to load invalid imdb_id
    if os.path.isfile(imdb_invalid_id_filename):
        invalid_imdb_id_df = pd.read_json(imdb_invalid_id_filename)
        invalid_tmdb_ids = invalid_imdb_id_df['tmdb_id'].tolist()
        if len(invalid_tmdb_ids) >0:
            tmdb_ids = [x for x in tmdb_ids if x not in invalid_tmdb_ids]

    for tmdb_id in tmdb_ids:
```

```
                imdb_id = tmdb_movie[tmdb_movie['id']==tmdb_id]['imdb_id'].tolist()
                if imdb_id is not None:
                    imdb_id = str(imdb_id.replace('tt',''))
                if imdb_id is None or imdb_id =='':
                    invalid_ids = invalid_ids.append({'imdb_id': imdb_id, 'tmdb_id'

                else:
                    movie_details = get_movie_details(imdb_info_column,imdb_id, tmc
                    if len(movie_details) > 0 :
                        movies.loc[tmdb_id] = movie_details
                    else:
                        invalid_ids = invalid_ids.append({'imdb_id': imdb_id, 'tmdb

                count += 1
                if (count % 50 == 0):
                    movies.to_csv(imdb_filename,mode = 'a',header=False, sep='\t',
                    invalid_ids.to_json(path_or_buf= imdb_invalid_id_filename)
                    movies = pd.DataFrame(columns=imdb_info_column)

            movies.to_csv(imdb_filename,mode = 'a',header=False, sep='\t', encoding
            invalid_ids.to_json(path_or_buf= imdb_invalid_id_filename)
            id_start = id_start + id_range
```

In [9]:

```
# load one file as example
imdb_filename = str(dir_data)+'\\drv_imdb_movie_info_20001_40000.csv'
imdb_movies = pd.read_csv(imdb_filename,index_col=0, sep='\t', encoding='ut
imdb_movies.head(5)['tmdb_id']
```

Out[9]:

```
20002.0    20002.0
20003.0    20003.0
20004.0    20004.0
20006.0    20006.0
20007.0    20007.0
Name: tmdb_id, dtype: float64
```

In [ ]:

Milestone2_03_get_response_variable

April 12, 2017

The notebook is used to create response variable.

```
In [5]: import tmdbsimple as tmdb
        tmdb.API_KEY = "71e259894a515060876bab2a33d6bdc9"
```

```
In [6]: import imdb as ib
        from imdb import IMDb
        import pandas as pd
        from PIL import Image
        from StringIO import StringIO
        import requests
        import os
        import time
        from shutil import copyfile
        import types
        import numpy as np
```

```
In [7]: dir_python_notebook = os.getcwd()
        dir_movie_project = os.path.abspath(os.path.join(dir_python_notebook, os.pa
        dir_data = os.path.join(dir_movie_project, 'data')
```

1 Load data

```
In [27]: tmdb_filename = str(dir_data)+'\\drv_tmdb_movie_details_1_20000.json'
         imdb_filename = str(dir_data)+'\\drv_imdb_movie_info_1_20000.json'
         tmdb_movies = pd.read_json(tmdb_filename)
         imdb_movies = pd.read_json(imdb_filename)
```

```
In [9]: imdb_genres = {12: 'Adventure',
         14: 'Fantasy',
         16: 'Animation',
         18: 'Drama',
         27: 'Horror',
         28: 'Action',
         35: 'Comedy',
         36: 'History',
         37: 'Western',
```

```
53: 'Thriller',
80: 'Crime',
99: 'Documentary',
878: 'Sci-Fi',
9648: 'Mystery',
10402: 'Music',
10749: 'Romance',
10751: 'Family',
10752: 'War',
10770: 'TV Movie',
1: 'Adult',
2: 'Biography',
3: 'Film Noir',
4: 'Game-Show',
5: 'Musical',
6: 'News',
7: 'Reality-TV',
8: 'Short',
9: 'Sport',
10: 'Talk-Show'}
```

In [10]:
```
items = imdb_genres.items()
imdb_genre_df = pd.DataFrame({'keys': [i[0] for i in items], 'values': [i
imdb_genre_df.columns = ['id','genres']
```

In [11]:
```
imdb_genre_df
```

Out[11]:

	id	genres
0	10752	War
1	1	Adult
2	2	Biography
3	3	Film Noir
4	4	Game-Show
5	5	Musical
6	6	News
7	7	Reality-TV
8	8	Short
9	9	Sport
10	10	Talk-Show
11	12	Adventure
12	14	Fantasy
13	16	Animation
14	10770	TV Movie
15	27	Horror
16	28	Action
17	10402	Music
18	35	Comedy
19	36	History

```
20     37      Western
21   9648      Mystery
22     53     Thriller
23     80        Crime
24     99  Documentary
25     18        Drama
26    878       Sci-Fi
27  10749      Romance
28  10751       Family
```

2 Format data

```
In [12]: def get_genre(tmdb_movies ,key):
             tmdb_genre = tmdb_movies[tmdb_movies[key].notnull()][key].tolist()
             tmdb_genre_set = set()
             for g in tmdb_genre:
                 if g is not None:
                     tmdb_genre_set = tmdb_genre_set.union(set(g))
             tmdb_genre = list(tmdb_genre_set)
             tmdb_genre.sort()
             return(tmdb_genre)
```

```
In [13]: def get_genere_num (row, column_name):
             if row[column_name] is None :
                 return 0
             else:
                 return len(row[column_name])
```

```
In [14]: def is_genre (row, column_name, genre):
             """check if that movie is in this genre as a movie can have more than
             if row[column_name] is None :
                 return 0
             else:
                 if genre in row[column_name] :
                     return 1
                 else:
                     return 0
```

```
In [22]: def get_all_genre (row, column_name, imdb_genre):
             """check a vector for all movie genres"""
             s = ""
             for g in imdb_genre:
                 if row[column_name] is None :
                     s = s +"0"
                 else:
                     if g in row[column_name] :
                         s = s +"1"
                     else:
```

```
                s = s +"0"
        return s
```

```
In [34]: imdb_genre = imdb_genre_df['genres'].tolist()
         imdb_genre.sort()
         imdb_movies[u'genre_num'] = imdb_movies.apply(lambda row: get_genere_num(r

         for g in imdb_genre:
             imdb_movies[g] = imdb_movies.apply(lambda row: is_genre(row,u'genres',
```

```
In [35]: imdb_movies['all_genre'] = imdb_movies.apply(lambda row: get_all_genre(row
```

```
In [50]: imdb_movies['all_genre'] = imdb_movies['all_genre'].astype('str')
```

```
In [51]: imdb_response_var_filename = str(dir_data)+'\\drv_imdb_response_variable_1
         imdb_movies[['imdb_id','tmdb_id','all_genre']+imdb_genre].to_json(path_or_
```

```
In [ ]:
```

Milestone2_data_impuation

April 12, 2017

```
In [4]: %matplotlib inline
        import pandas as pd
        import numpy as np
        import json
        import matplotlib.pyplot as plt
        import seaborn.apionly as sns
        import re
```

1 Data Processing and Description

In this part, we are working on processing the messy and unstrucured IMDB dataset. We used 10,000 IMDB subsets. The original IMDB subset consists of 51 variables. There are 49 categorical variables and 2 numerical variables. The cell of the data is a list of the data so we should address this issue in our data wrangling process.

1.1 Load JSON File

```
In [326]: with open('/Users/Victoria_G/Desktop/CS109B/data/drv_imdb_movie_info_1_20
              data = json.load(f)
          imdb_movies=pd.DataFrame(data)

In [327]: imdb_movies.columns

Out[327]: Index([u'airing', u'akas', u'amazon reviews', u'art direction',
                 u'assistant director', u'cast', u'casting director', u'certificate
                 u'cinematographer', u'color info', u'costume department',
                 u'costume designer', u'countries', u'cover url', u'creator',
                 u'director', u'distributors', u'editor', u'faqs',
                 u'full-size cover url', u'genres', u'guests', u'imdb_id', u'langua
                 u'make up', u'merchandising links', u'misc links',
                 u'miscellaneous companies', u'miscellaneous crew',
                 u'non-original music', u'number of episodes', u'number of seasons'
                 u'original music', u'parents guide', u'photo sites', u'plot',
                 u'producer', u'production companies', u'production manager', u'rat
                 u'runtimes', u'set decoration', u'sound clips', u'sound crew',
                 u'special effects', u'special effects companies', u'stunt performe
                 u'tmdb_id', u'video clips', u'visual effects', u'writer'],
                dtype='object')
```

2 Change None value to NaN

Since there are None and NaN in the data, so in this step, we transferred the None to NaN first and then do the imputation.

```
In [328]: for c in imdb_movies.columns:
              imdb_movies[c].fillna(value=np.nan, inplace=True)
```

3 Delete rows with missing greater than 50%

As for the missing value analysis, before examine the missing rate for the columns, we should examine the missing rate for each row first. The reason is that there might be rows that have so much missing that are not informative. We found that for movies that are not very famous, there usually are very few information about it on the IMDB website. So we believe deleting these movies would be appropriate. In our datasets, there are so many categorical variables and most of variables are not apppropriate to impute. For example, we should not impute the missing value for the casting and the director.

```
In [329]: miss=[]
          for i in range(len(imdb_movies)):
              miss.append(imdb_movies.ix[i,:].isnull().values.ravel().sum()/float(l
          miss
          imdb_movies['row_missing']=miss

In [330]: imdb_movies=imdb_movies[imdb_movies['row_missing']<0.5]
```

4 Missing Rate

After removing the rows with missing rate greater than 0.5 and then we move on to explore the missing rate for each columns. We found that there are 15 variables have missing rate greater than 99%. In total, there are 16 variables having missing rate greater than 50%. We belive that it would not be appropriate to impute the columns with missing values greater than 50%. So we deleted these variables.

```
In [331]: acc=[]
          for c in imdb_movies.columns:
              acc.append(imdb_movies[c].isnull().values.ravel().sum()/float(len(imd
          df_missing= pd.DataFrame({'Columns': list(imdb_movies.columns), 'Missing'
          df_missing=df_missing.sort_values(by='Missing',ascending=False)
          df_missing

Out[331]:              Columns   Missing
          0             airing  1.000000
          18              faqs  1.000000
          48       video clips  1.000000
          44   special effects  1.000000
          42       sound clips  1.000000
          34       photo sites  1.000000
```

33	parents guide	1.000000
29	non-original music	1.000000
25	merchandising links	1.000000
21	guests	1.000000
26	misc links	1.000000
2	amazon reviews	1.000000
14	creator	0.999901
30	number of episodes	0.998915
31	number of seasons	0.993882
45	special effects companies	0.556740
41	set decoration	0.403098
49	visual effects	0.353858
46	stunt performer	0.326327
3	art direction	0.323268
6	casting director	0.319518
10	costume department	0.255477
11	costume designer	0.186797
27	miscellaneous companies	0.154529
24	make up	0.118709
38	production manager	0.090685
4	assistant director	0.068877
32	original music	0.058417
28	miscellaneous crew	0.033945
8	cinematographer	0.032268
35	plot	0.030985
7	certificates	0.025459
1	akas	0.023584
17	editor	0.012631
43	sound crew	0.011940
36	producer	0.010361
16	distributors	0.008585
37	production companies	0.007302
13	cover url	0.007006
19	full-size cover url	0.007006
50	writer	0.005131
40	runtimes	0.003454
9	color info	0.002171
23	languages	0.001776
20	genres	0.001579
5	cast	0.000395
39	rating	0.000296
15	director	0.000099
47	tmdb_id	0.000000
12	countries	0.000000
22	imdb_id	0.000000
51	row_missing	0.000000

5 Delete rows with genre missing

According to the missing value table, we found that some movies do not have the genre labeled. Since our goal is to train models to predict, if there is no genre information, the we believe these movies are not informative.

```
In [332]: imdb_movies=imdb_movies[imdb_movies['genres'].isnull()==False]
```

6 Delete columns with missing rate greater than 0.5

```
In [333]: d=df_missing[df_missing['Missing']>0.5]['Columns'].values
          for x in d:
              if x != 'special effects companies':
                  del imdb_movies[x]

In [334]: imdb_movies.columns

Out[334]: Index([                     u'akas',                u'art direction',
                          u'assistant director',                       u'cast',
                            u'casting director',               u'certificates',
                             u'cinematographer',                 u'color info',
                          u'costume department',           u'costume designer',
                                   u'countries',                  u'cover url',
                                    u'director',               u'distributors',
                                      u'editor',        u'full-size cover url',
                                      u'genres',                    u'imdb_id',
                                   u'languages',                    u'make up',
                     u'miscellaneous companies',         u'miscellaneous crew',
                              u'original music',                       u'plot',
                                    u'producer',       u'production companies',
                          u'production manager',                     u'rating',
                                    u'runtimes',             u'set decoration',
                                  u'sound crew', u'special effects companies',
                             u'stunt performer',                    u'tmdb_id',
                              u'visual effects',                     u'writer',
                                 u'row_missing'],
                dtype='object')
```

7 Delete aka, cover url columns

We found that akas are the different titles of the movies, which is similar with the id column. So we belive it would not be informative in genre prediction. In addition, the cover and full-size cover url are not informative to the genre prediction so we decide to drop these columns, too.

```
In [335]: del imdb_movies['akas']

In [336]: del imdb_movies['cover url']
          del imdb_movies['full-size cover url']
```

8 Data Processing & Data Imputation

After we remove all the missing values, we begin to process the format of the data. Since the value of each cell is a list, so we decide to generate new columns to keep top k of the list.

8.1 1. Certificate

As for the certificate of the movie, we decided to use the US certficate system. We generated two new columns. We found that most of the movies are R and PG. So we generated two columns indicating whether this movie is R and PG.

```
In [337]: # for null value data
          imdb_movies['certificates'].fillna('Unrated',inplace=True)
          # for not null value data
          certificate=[]
          for i in range(len(imdb_movies)):
              if imdb_movies['certificates'][i]!='Unrated':
                  c=re.search('USA:(\w*)',str(imdb_movies['certificates'][i]))
                  if c!=None:
                      certificate.append(c.group(1))
                  else:
                      certificate.append('Unrated')
              else:
                  certificate.append('Unrated')
```

```
In [338]: imdb_movies['certificates_new']=certificate
```

```
In [339]: # create new certificate column --- R
          cer_r=[]
          for c in certificate:
              cer_r.append(int(c=='R'))
          cer_r
          imdb_movies['certificates_R']=cer_r

          # create new certificate column --- PG
          cer_pg=[]
          for c in certificate:
              cer_pg.append(int(c=='PG'))
          imdb_movies['certificates_PG']=cer_pg
```

8.2 2. Other Variables

For other covariates, we decided to pick the first element in the columns and generate new columns.

```
In [340]: L=['art direction','assistant director','casting director','cinematograph
             'costume designer','countries','director','distributors','editor','lang
              'miscellaneous crew', 'original music','producer','production companie
              'set decoration','sound crew','stunt performer','visual effects','writ
```

```
for l in L:
    temp=[]
    imdb_movies[l].fillna('0',inplace=True)
    for i in range(len(imdb_movies)):
        if imdb_movies[l][i] != '0':
            temp.append(str(imdb_movies[l][i][0]))
        else:
            temp.append('0')
    imdb_movies[l+'_1']=temp
```

8.3 3. cast

As for the casting, according to the IMDB website, the list of the casting is ordered by the importance of the characters. So we decide to choose the top 4 casting.

```
In [341]: temp1=[]
          temp2=[]
          temp3=[]
          temp4=[]
          imdb_movies['cast'].fillna('0',inplace=True)
          for i in range(len(imdb_movies)):

              if imdb_movies['cast'][i] != '0':
                  temp1.append(str(imdb_movies['cast'][i][0]))
                  if len(imdb_movies['cast'][i]) >=2:
                      temp2.append(str(imdb_movies['cast'][i][1]))
                  else:
                      temp2.append('0')

                  if len(imdb_movies['cast'][i]) >=3:
                      temp3.append(str(imdb_movies['cast'][i][2]))
                  else:
                      temp3.append('0')

                  if len(imdb_movies['cast'][i]) >=4:
                      temp4.append(str(imdb_movies['cast'][i][3]))
                  else:
                      temp4.append('0')
              else:
                  temp1.append('0')
                  temp2.append('0')
                  temp3.append('0')
                  temp4.append('0')

          imdb_movies['cast_1']=temp1
          imdb_movies['cast_2']=temp2
          imdb_movies['cast_3']=temp3
          imdb_movies['cast_4']=temp4
```

8.4 4. runtimes

As for the runtimes, we found that each country may have different runtimes so we decided to use regular expression to extract the runtimes for each countries and then calculate the average.

```
In [342]: # average run time
          imdb_movies['runtimes'].fillna(-1,inplace=True)
          L=[]
          for row in imdb_movies['runtimes']:
              if row ==-1:
                  L.append(np.nan)
                  continue
              for e in row:
                  number=[]
                  number.extend(map(int,re.findall('\d+',e.encode('utf-8'))))
              L.append(1.0* sum(number)/len(number))
          imdb_movies['runtimes_avg']=L
```

```
In [343]: imdb_movies.columns
```

```
Out[343]: Index([             u'art direction',        u'assistant director',
                                     u'cast',          u'casting director',
                             u'certificates',          u'cinematographer',
                               u'color info',        u'costume department',
                         u'costume designer',                u'countries',
                                 u'director',             u'distributors',
                                   u'editor',                   u'genres',
                                  u'imdb_id',                u'languages',
                                  u'make up',   u'miscellaneous companies',
                       u'miscellaneous crew',           u'original music',
                                     u'plot',                 u'producer',
                     u'production companies',       u'production manager',
                                   u'rating',                 u'runtimes',
                           u'set decoration',               u'sound crew',
               u'special effects companies',          u'stunt performer',
                                  u'tmdb_id',           u'visual effects',
                                   u'writer',              u'row_missing',
                         u'certificates_new',           u'certificates_R',
                          u'certificates_PG',          u'art direction_1',
                     u'assistant director_1',       u'casting director_1',
                        u'cinematographer_1',     u'costume department_1',
                       u'costume designer_1',              u'countries_1',
                               u'director_1',           u'distributors_1',
                                 u'editor_1',              u'languages_1',
                                u'make up_1', u'miscellaneous companies_1',
                     u'miscellaneous crew_1',         u'original music_1',
                               u'producer_1',   u'production companies_1',
                     u'production manager_1',         u'set decoration_1',
                             u'sound crew_1',        u'stunt performer_1',
```

```
                         u'visual effects_1',                              u'writer_1',
                                  u'cast_1',                                u'cast_2',
                                  u'cast_3',                                u'cast_4',
                             u'runtimes_avg'],
              dtype='object')
```

```
In [344]: imdb_movies_new = imdb_movies.ix[:,'certificates_R':'runtimes_avg']
          imdb_movies_new['rating']=imdb_movies['rating']
```

8.5 5. Change catagorical to relative frequency

After we did all the data processing, we have to transfer the categorcial data to appropriate format for the modeling part. We decided to covert each categorical variables to relative frequency and the do the modeling part.

```
In [345]: for col in imdb_movies_new.columns[2:29]:
              L=[]
              s=imdb_movies_new[col].value_counts()
              for row in imdb_movies_new[col]:
                  L.append(float(s[row])/len(imdb_movies_new))
              imdb_movies_new[col]=L
```

8.6 5. genres

In this part, we implemented the one hot encoding for the genre part. We generated several columns and each columns contains the binary indicator indicating whether this movie belongs to each genre.

```
In [309]: imdb_movies.to_csv('imdb_imputed.csv')
          imdb_movies_new.to_csv('imdb_imputed_for_cluster.csv')
```

```
In [311]: imdb_movies['genres']
```

```
Out[311]: 100                                                  [Comedy, Crime]
          10001                                              [Comedy, History]
          10002                                        [Crime, Drama, Romance]
          10003             [Action, Adventure, Romance, Sci-Fi, Thriller]
          10004                [Drama, Fantasy, Horror, Mystery, Thriller]
          10005                                             [Action, Thriller]
          10006                                                 [Crime, Drama]
          10007                                    [Action, Horror, Thriller]
          10008                                   [Horror, Mystery, Thriller]
          10009            [Animation, Adventure, Comedy, Family, Fantasy]
          10010            [Animation, Adventure, Comedy, Family, Fantasy]
          10011                                             [Horror, Thriller]
          10012                                               [Comedy, Horror]
          10013                              [Comedy, Drama, Fantasy, Romance]
          10014                                             [Horror, Thriller]
          10015                                  [Action, Comedy, Drama, War]
```

```
10016                           [Action, Horror, Sci-Fi]
10017          [Action, Horror, Romance, Sci-Fi, Thriller]
10018                                    [Comedy, Drama]
10019                         [Comedy, Fantasy, Romance]
10020       [Animation, Family, Fantasy, Musical, Romance]
10021                                  [Comedy, Romance]
10022            [Action, Comedy, Drama, Family, Thriller]
10023                                    [Comedy, Crime]
10024                                            [Drama]
10025                         [Comedy, Fantasy, Romance]
10026                    [Action, Comedy, Crime, Thriller]
10027                     [Action, Crime, Drama, Thriller]
10028                             [Drama, Music, Romance]
10029                          [Comedy, Crime, Thriller]
                               ...
9965                                            [Comedy]
9966                          [Horror, Mystery, Thriller]
9967                       [Action, Drama, Thriller, War]
9968                              [Comedy, Crime, Drama]
9969                                    [Comedy, Family]
9971                                     [Comedy, Drama]
9972                            [Crime, Drama, Thriller]
9973                         [Adventure, Comedy, Family]
9974                                     [Comedy, Drama]
9975              [Animation, Adventure, Comedy, Family]
9976                                    [Comedy, Horror]
9977                            [Comedy, Drama, Romance]
9978               [Action, Adventure, Family, Thriller]
9980                    [Action, Comedy, Horror, Sci-Fi]
9981                    [Comedy, Family, Romance, Sport]
9982        [Animation, Adventure, Comedy, Family, Sci-Fi]
9985                                            [Comedy]
9986                           [Comedy, Family, Fantasy]
9987                 [Horror, Mystery, Sci-Fi, Thriller]
9988                                            [Comedy]
9989                    [Action, Crime, Drama, Thriller]
9990                    [Action, Crime, Drama, Thriller]
9991                       [Comedy, Crime, Drama, Sport]
9992             [Animation, Adventure, Family, Fantasy]
9993                   [Crime, Drama, Romance, Thriller]
9994    [Animation, Adventure, Family, Music, Mystery,...
9995                       [Crime, Drama, Action, Music]
9997                 [Action, Fantasy, Horror, Thriller]
9998                        [Adventure, Action, Fantasy]
9999                                      [Crime, Drama]
Name: genres, dtype: object
```

```
In [313]: L=[]
```

```
        for i in imdb_movies['genres']:
            L.extend(i)
        L=list(set(L))
```

```
In [314]: for l in L:
              col = np.zeros((len(imdb_movies)))
              for i in range(len(imdb_movies)):
                  for e in imdb_movies['genres'][i]:
                      if e ==l:
                          col[i]=1
              imdb_movies[l]=col
```

```
/Users/Victoria_G/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:7: Set
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/i
```

```
In [316]: imdb_movies.columns
```

```
Out[316]: Index([             u'art direction',        u'assistant director',
                                       u'cast',          u'casting director',
                               u'certificates',          u'cinematographer',
                                 u'color info',        u'costume department',
                           u'costume designer',                u'countries',
                                   u'director',             u'distributors',
                                     u'editor',                   u'genres',
                                    u'imdb_id',                u'languages',
                                    u'make up',   u'miscellaneous companies',
                         u'miscellaneous crew',           u'original music',
                                       u'plot',                 u'producer',
                       u'production companies',       u'production manager',
                                     u'rating',                 u'runtimes',
                             u'set decoration',               u'sound crew',
                            u'stunt performer',                  u'tmdb_id',
                              u'visual effects',                  u'writer',
                                u'row_missing',         u'certificates_new',
                             u'certificates_R',          u'certificates_PG',
                            u'art direction_1',     u'assistant director_1',
                         u'casting director_1',        u'cinematographer_1',
                       u'costume department_1',      u'costume designer_1',
                                u'countries_1',               u'director_1',
                             u'distributors_1',                 u'editor_1',
                                u'languages_1',                u'make up_1',
                  u'miscellaneous companies_1',     u'miscellaneous crew_1',
                           u'original music_1',               u'producer_1',
                     u'production companies_1',     u'production manager_1',
                           u'set decoration_1',             u'sound crew_1',
```

```
                  u'stunt performer_1',          u'visual effects_1',
                           u'writer_1',                    u'cast_1',
                             u'cast_2',                    u'cast_3',
                             u'cast_4',              u'runtimes_avg',
                             u'Sci-Fi',                     u'Crime',
                            u'Romance',                 u'Animation',
                              u'Music',                     u'Adult',
                             u'Comedy',                       u'War',
                             u'Horror',                 u'Film-Noir',
                            u'Western',                  u'Thriller',
                          u'Adventure',                   u'Mystery',
                              u'Short',                     u'Drama',
                             u'Action',               u'Documentary',
                            u'Musical',                   u'History',
                             u'Family',                   u'Fantasy',
                              u'Sport',                 u'Biography'],
         dtype='object')
```

In [323]:
```
df_impute= pd.read_csv('imdb_imputed_byRF_for_cluster_2.csv')
for col in imdb_movies.columns[64:]:
    df_impute[col]=imdb_movies[col].values
df_impute.head()
```

Out[323]:
```
   Unnamed: 0      X  certificates_R  certificates_PG  art.direction_1  \
0           1    100               1                0         0.322495
1           2  10001               0                1         0.027673
2           3  10002               1                0         0.212394
3           4  10003               0                1         0.019767
4           5  10004               1                0         0.104764

   assistant.director_1  casting.director_1  cinematographer_1  \
0              0.042103            0.010279           0.046254
1              0.093694            0.319431           0.165250
2              0.024906            0.006523           0.016308
3              0.024906            0.014133           0.025301
4              0.355406            0.017395           0.049812

   costume.department_1  costume.designer_1    ...      Short  Drama  Act
0              0.307966            0.028662    ...        0.0    0.0
1              0.307966            0.172860    ...        0.0    0.0
2              0.012453            0.005535    ...        0.0    1.0
3              0.126112            0.007116    ...        0.0    0.0
4              0.126112            0.040028    ...        0.0    1.0

   Documentary  Musical  History  Family  Fantasy  Sport  Biography
0          0.0      0.0      0.0     0.0      0.0    0.0        0.0
1          0.0      0.0      1.0     0.0      0.0    0.0        0.0
2          0.0      0.0      0.0     0.0      0.0    0.0        0.0
```

```
3          0.0       0.0       0.0      0.0       0.0     0.0         0.0
4          0.0       0.0       0.0      0.0       1.0     0.0         0.0

[5 rows x 57 columns]
```

```
In [325]: df_impute.to_csv('imdb_imputed_byRF_for_cluster_with_response.csv',index=
```

Milestone2_05_processed_imdb_imputed_file

April 12, 2017

In [1]:
```
from imdb import IMDb
import pandas as pd
from PIL import Image
from StringIO import StringIO
import requests
import os
import time
from shutil import copyfile
import csv
```

In [2]:
```
dir_python_notebook = os.getcwd()
dir_movie_project = os.path.abspath(os.path.join(dir_python_notebook, os.pa
dir_data = os.path.join(dir_movie_project, 'data')
```

In [8]:
```
imdb_filename = str(dir_data)+'\\imdb_imputed.csv'
imdb_movies = pd.read_csv(imdb_filename,index_col=0, sep=',', encoding='utf
```

In [9]:
```
imdb_movies.head(5)
```

Out[9]:
```
                                                  art direction  \
100.0                                                         0
10001.0  [u'0264966', u'0316904', u'0383595', u'0550026']
10002.0                                            [u'0413541']
10003.0  [u'0144261', u'0658390', u'0724682', u'0866777']
10004.0                                            [u'0003488']

                                              assistant director  \
100.0                 [u'0123407', u'0717230', u'0006859']
10001.0  [u'0267087', u'0382238', u'0680614', u'0731968...
10002.0               [u'0013936', u'0110556', u'0179192']
10003.0  [u'0068925', u'0072593', u'0250856', u'0250856...
10004.0               [u'0551862', u'0581688', u'0815340']

                                                        cast  \
100.0    [u'0002076', u'0002077', u'0602941', u'0005458...
10001.0  [u'0000635', u'0494189', u'0397398', u'0933957...
10002.0  [u'0001364', u'0879154', u'0000323', u'0001059...
10003.0  [u'0000174', u'0000223', u'0784884', u'0631792...
```

```
10004.0  [u'0000643', u'0001836', u'0001272', u'0001164...

                casting director  \
100.0    [u'0288911', u'0005363']
10001.0                         0
10002.0              [u'0007109']
10003.0  [u'0505059', u'0689691']
10004.0              [u'0470948']

                                                 certificates cinematographer
100.0    [u'Argentina:13', u'Australia:MA15+', u'Brazil...    [u'0362165']
10001.0  [u'Australia:PG', u'Finland:K-8', u'Germany:12...    [u'0201372']
10002.0  [u'Argentina:18', u'Australia:M', u'Finland:K-...    [u'0695536']
10003.0  [u'Argentina:13', u'Australia:M', u'Brazil:12'...    [u'0005793']
10004.0  [u'Argentina:16', u'Australia:MA15+', u'Denmar...    [u'0003791']

                       color info  \
100.0                 [u'Color']
10001.0               [u'Color']
10002.0  [u'Color::(Technicolor)']
10003.0    [u'Color::(Rankcolor)']
10004.0               [u'Color']

                                      costume department  \
100.0                           [u'0989056', u'0788235']
10001.0              [u'0100748', u'1021483', u'0372320']
10002.0                         [u'0196382', u'0788827']
10003.0  [u'0183262', u'0232875', u'0412130', u'0456586...
10004.0              [u'0124203', u'0305128', u'0568764']

                costume designer       countries    ...     \
100.0               [u'0171871']         [u'UK']    ...
10001.0  [u'0100748', u'0915204']  [u'Australia']    ...
10002.0             [u'0296220']         [u'UK']    ...
10003.0             [u'0829641']        [u'USA']    ...
10004.0             [u'0624703']        [u'USA']    ...

        production manager_1 set decoration_1 sound crew_1 stunt performer_
100.0                92061.0         405176.0      30552.0          337040.
10001.0              55373.0              0.0      66740.0           49882.
10002.0             115536.0              0.0      60952.0          286157.
10003.0              97161.0         949952.0      10057.0          132625.
10004.0             665655.0         130700.0     123638.0            7200.

         visual effects_1  writer_1  cast_1    cast_2    cast_3    cast_4
100.0             91100.0    5363.0  2076.0    2077.0  602941.0    5458.0
10001.0          184789.0  730000.0   635.0  494189.0  397398.0  933957.0
10002.0               0.0    1403.0  1364.0  879154.0     323.0    1059.0
```

```
10003.0          463671.0  153546.0    174.0     223.0  784884.0  631792.0
10004.0           84624.0     175.0    643.0    1836.0    1272.0    1164.0

[5 rows x 63 columns]
```

```
In [22]: imdb_genres = ['Action','Adult','Adventure','Animation','Biography','Comed
                        'Fantasy','Film Noir','Game-Show','History','Horror','Musica
                        'Romance','Sci-Fi','Short','Sport','Talk-Show','Thriller','W
```

```
In [24]: len(imdb_genres)
```

```
Out[24]: 28
```

```
In [13]: def get_genre(tmdb_movies ,key):
             tmdb_genre = tmdb_movies[tmdb_movies[key].notnull()][key].tolist()
             tmdb_genre_set = set()
             for g in tmdb_genre:
                 if g is not None:
                     tmdb_genre_set = tmdb_genre_set.union(set(g))
             tmdb_genre = list(tmdb_genre_set)
             tmdb_genre.sort()
             return(tmdb_genre)
```

```
In [14]: def get_genere_num (row, column_name):
             if row[column_name] is None :
                 return 0
             else:
                 return len(row[column_name])
```

```
In [15]: def is_genre (row, column_name, genre):
             """check if that movie is in this genre as a movie can have more than
             if row[column_name] is None :
                 return 0
             else:
                 if genre in row[column_name] :
                     return 1
                 else:
                     return 0
```

```
In [16]: def get_all_genre (row, column_name, imdb_genre):
             """check a vector for all movie genres"""
             s = ""
             for g in imdb_genre:
                 if row[column_name] is None :
                     s = s +"0"
                 else:
                     if g in row[column_name] :
                         s = s +"1"
                     else:
                         s = s +"0"
             return s
```

In [23]:
```
imdb_genre = imdb_genres
imdb_genre.sort()
imdb_movies[u'genre_num'] = imdb_movies.apply(lambda row: get_genere_num(r

for g in imdb_genre:
    imdb_movies[g] = imdb_movies.apply(lambda row: is_genre(row,u'genres',
```

In [25]:
```
imdb_movies['all_genre'] = imdb_movies.apply(lambda row: get_all_genre(row
```

In [26]:
```
imdb_movies['all_genre'] = imdb_movies['all_genre'].astype('str')
```

In [32]:
```
imdb_filename = str(dir_data)+'\\imdb_imputed_processed.csv'
imdb_movies.to_csv(imdb_filename, sep=',', encoding='utf-8')
```

In [ ]:
```
imdb_filename = str(dir_data)+'\\imdb_imputed_processed.json'
imdb_movies.to_json(path_or_buf= imdb_filename)
```

In [30]:
```
imdb_movies.columns
```

Out[30]:
```
Index([            u'art direction',         u'assistant director',
                            u'cast',           u'casting director',
                    u'certificates',            u'cinematographer',
                      u'color info',         u'costume department',
                u'costume designer',                  u'countries',
                        u'director',               u'distributors',
                          u'editor',                     u'genres',
                         u'imdb_id',                  u'languages',
                         u'make up',    u'miscellaneous companies',
              u'miscellaneous crew',             u'original music',
                            u'plot',                   u'producer',
            u'production companies',         u'production manager',
                          u'rating',                   u'runtimes',
                  u'set decoration',                 u'sound crew',
                 u'stunt performer',                    u'tmdb_id',
                  u'visual effects',                     u'writer',
                     u'row_missing',           u'certificates_new',
                  u'certificates_R',            u'certificates_PG',
                 u'art direction_1',       u'assistant director_1',
              u'casting director_1',          u'cinematographer_1',
            u'costume department_1',         u'costume designer_1',
                     u'countries_1',                 u'director_1',
                  u'distributors_1',                   u'editor_1',
                     u'languages_1',                  u'make up_1',
       u'miscellaneous companies_1',       u'miscellaneous crew_1',
                u'original music_1',                 u'producer_1',
          u'production companies_1',       u'production manager_1',
                u'set decoration_1',               u'sound crew_1',
               u'stunt performer_1',           u'visual effects_1',
                        u'writer_1',                     u'cast_1',
```

```
                                              u'cast_2',                     u'cast_3',
                                              u'cast_4',                  u'genre_num',
                                              u'Action',                     u'Adult',
                                           u'Adventure',                 u'Animation',
                                           u'Biography',                    u'Comedy',
                                               u'Crime',               u'Documentary',
                                               u'Drama',                    u'Family',
                                             u'Fantasy',                u'Film Noir',
                                           u'Game-Show',                   u'History',
                                              u'Horror',                     u'Music',
                                             u'Musical',                   u'Mystery',
                                                u'News',                u'Reality-TV',
                                             u'Romance',                    u'Sci-Fi',
                                               u'Short',                     u'Sport',
                                           u'Talk-Show',                  u'Thriller',
                                                 u'War',                   u'Western',
                                           u'all_genre'],
                       dtype='object')
```

```
In [ ]:
```

CS 109B: Final Project - Milestone 2:

April 6, 2017

```
knitr::opts_chunk$set(echo = TRUE)

set.seed(109) # Set seed for random number generator
#load packages
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(cluster)
library(factoextra)
library(mclust)
```

```
## Package 'mclust' version 5.2.3
## Type 'citation("mclust")' for citing this R package in publications.
```

```
library(corrplot)
library(dbscan)
library(MASS)
```

```
##
## Attaching package: 'MASS'
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(ggfortify)
library(NbClust)
library(caret)
```

```
## Loading required package: lattice
```

```
library('e1071')
library(tidyr)
```

Question: What does your choice of Y look like?

In IMDB data set, we have total 28 genres. However, it has been observared that some of these gernes show a high correlation. For instance, Thriller movie trends to associate closely with Horror movie. Also, some genres have very small number of movies, using clustering method helps us in reducing the number of genres. In specific, we intend to perform clustering on closely related genres (based on avialable attributes), so that we will end up with grouping together movie genres which show high similarity based on movie attributes. After clustering, we will have certain number of clusters, then we assign the top combination of genres in each cluster as the response variable for that cluster. For example, if the most popular multiple-label genre for cluster1 is Romance, Drama and Action, then we will regard the movies in cluster 1 have genre label: Romance, Drama and Action.

Load Data

```
# load data
data.df <- read.csv("/Users/Victoria_G/Desktop/CS109B/proj/imdb_imputed_byRF_for_cluster_with_response.c

#since most columns are categorical (dummy coding will be too many columns),
#we used the relative frequency for later clustering
relative_req <- function(x){
  if(class(x) == "numeric"){
    tab <- table(x)/nrow(data.val)
    ind <- match(x, as.numeric(names(tab)))
    unname(tab[ind])
  }
  else{
    x <- as.character(x)
    tab <- table(x)/nrow(data.val)
    ind <- match(x, as.character(names(tab)))
    unname(tab[ind])
  }
}
```

Clustering

Considering that we will have more than 200,000 movies in the whole data set, we would like to choose clustering methods that are suitable for very large data sets. (The following three clustering method were covered in the lectures, so we won't explain them in details here).

K-means: able to handle very large data set; good for general-purpose; not very uneven cluster size; will not have too many clusters.

PAM: less suitable for very large data set; minimize the average dissimilarity of objects to their closest selected object.

DBSCAN: able to handle very large data set; usually generate uneven cluster sizes

We would like to compare the performance of clustering based on their silhouette plot and whether the number of observations in clusters is reasonable.

```
#convert all categorical variables to relative frequency
data.val <- data.df %>% dplyr::select(c(1:32)) %>% mutate('certificate' = ifelse(certificates_R == 1, "R
data.val[,'certificate'] <- relative_req(data.val[,'certificate'])
data.val[,4:ncol(data.val)] <- scale(data.val[,4:ncol(data.val)], center = F)
data.scaled <- data.val[,4:ncol(data.val)]
head(data.scaled)
```

```
##   art.direction_1 assistant.director_1 casting.director_1
## 1      1.51797484            0.1876567         0.05608820
```

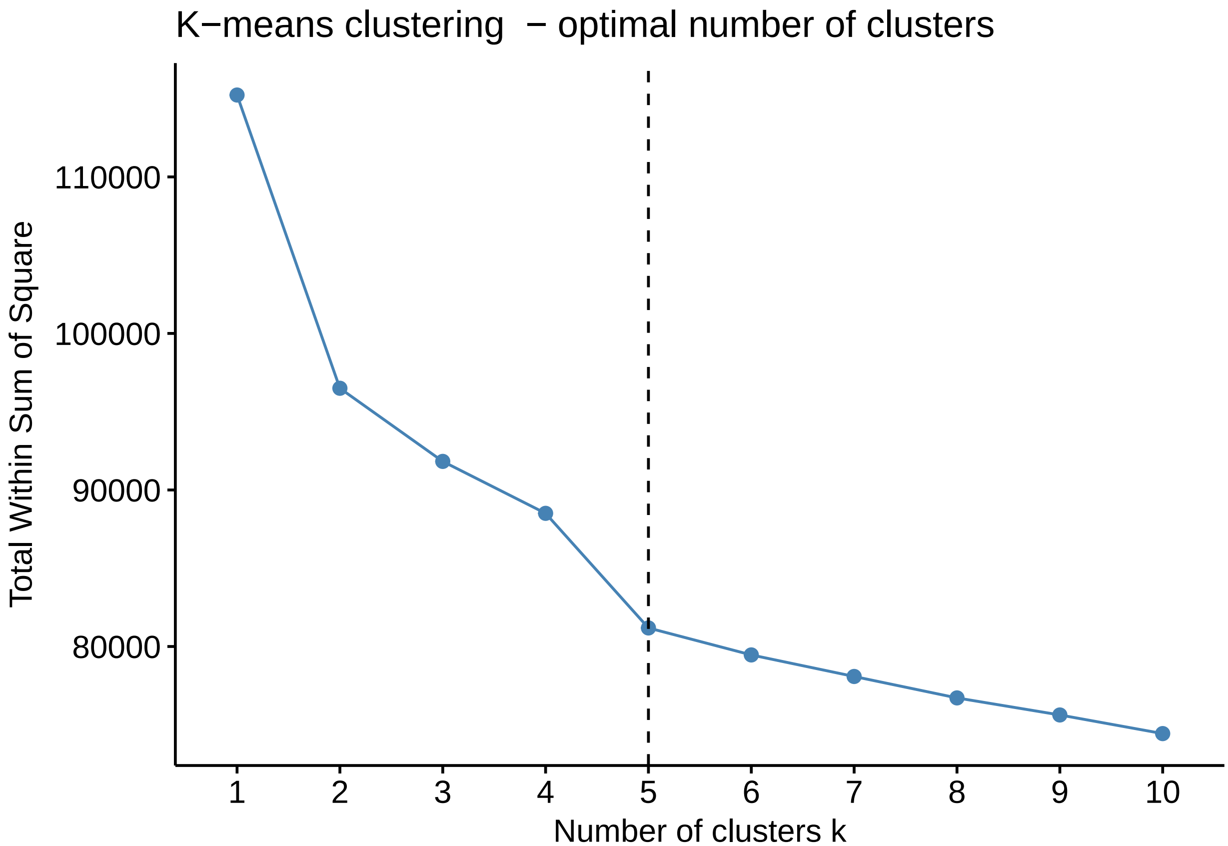
```
## 2      0.13025834           0.4176022         1.74304874
## 3      0.99973274           0.1110082         0.03559443
## 4      0.09304167           0.1110082         0.07712128
## 5      0.49312085           1.5840690         0.09491850
## 6      0.49312085           0.2731153         1.74304874
##   cinematographer_1 costume.department_1 costume.designer_1 countries_1
## 1         0.5595489           1.39871601         0.25060132   0.2268350
## 2         1.9990721           1.39871601         1.51138522   0.0360135
## 3         0.1972769           0.05655912         0.04839198   0.2268350
## 4         0.3060780           0.57277331         0.06221826   1.3582590
## 5         0.6025911           0.57277331         0.34997770   1.3582590
## 6         0.5105286           1.39871601         1.51138522   1.3582590
##   director_1 distributors_1  editor_1 languages_1  make.up_1
## 1  0.2952305      0.8508000 0.3643082    1.177706 1.73643930
## 2  1.6426925      0.2873316 1.8233400    1.177706 0.21721653
## 3  0.1605924      1.2314211 0.2014939    1.177706 0.09244632
## 4  0.1568074      0.7220606 0.2104892    1.177706 0.04396049
## 5  0.3163183      0.1604579 0.4749499    1.177706 0.04137458
## 6  0.8521670      0.1455316 0.8554496    1.177706 0.38788666
##   miscellaneous.companies_1 miscellaneous.crew_1 original.music_1
## 1                1.76276953           0.17605898       0.33015214
## 2                0.27840915           1.33887278       0.91102850
## 3                1.76276953           0.10961248       0.18947862
## 4                0.08938399           0.02400804       0.05550384
## 5                1.76276953           1.33887278       0.12440515
## 6                0.07399822           0.17605898       0.47082566
##   producer_1 production.companies_1 production.manager_1 set.decoration_1
## 1  1.5460773             0.07600071            0.4342688       0.54087330
## 2  1.5460773             0.17958686            1.5712088       1.53700050
## 3  0.3031844             0.16889046            0.6582964       1.53700050
## 4  1.5460773             0.16044594            1.5712088       0.09504887
## 5  1.5460773             1.75139414            0.4342688       0.09052273
## 6  1.5460773             0.12610488            1.5712088       1.53700050
##   sound.crew_1 stunt.performer_1 visual.effects_1  writer_1    cast_1
## 1   0.35871148         0.1970328       0.03548261 0.1872223 0.6421119
## 2   0.17234643         0.1182197       0.01335816 1.3878998 1.6394246
## 3   0.09070865         0.1689557       1.49444388 0.1872223 0.2323743
## 4   0.14018609         0.1024571       0.21039098 1.3878998 0.0995890
## 5   0.35871148         0.1024571       0.23418520 0.0108452 0.3883971
## 6   0.64815454         0.3851992       0.92713963 0.5548174 1.6394246
##     cast_2     cast_3    cast_4 runtimes_avg    rating certificate
## 1 0.463299 1.30959079 0.9153026    0.9233359 1.2631433   0.9225600
## 2 1.401337 1.30959079 0.2287475    0.7001964 0.7702093   0.6962927
## 3 0.463299 0.08583678 0.4576524    0.8002245 1.1399098   0.9225600
## 4 0.463299 0.43057582 0.2287475    0.8925581 0.9550595   0.6962927
## 5 1.401337 0.08583678 2.0594314    1.0079750 0.8164219   0.9225600
## 6 1.401337 0.08583678 0.4576524    0.7386687 0.6931884   0.9225600
```

Kmeans

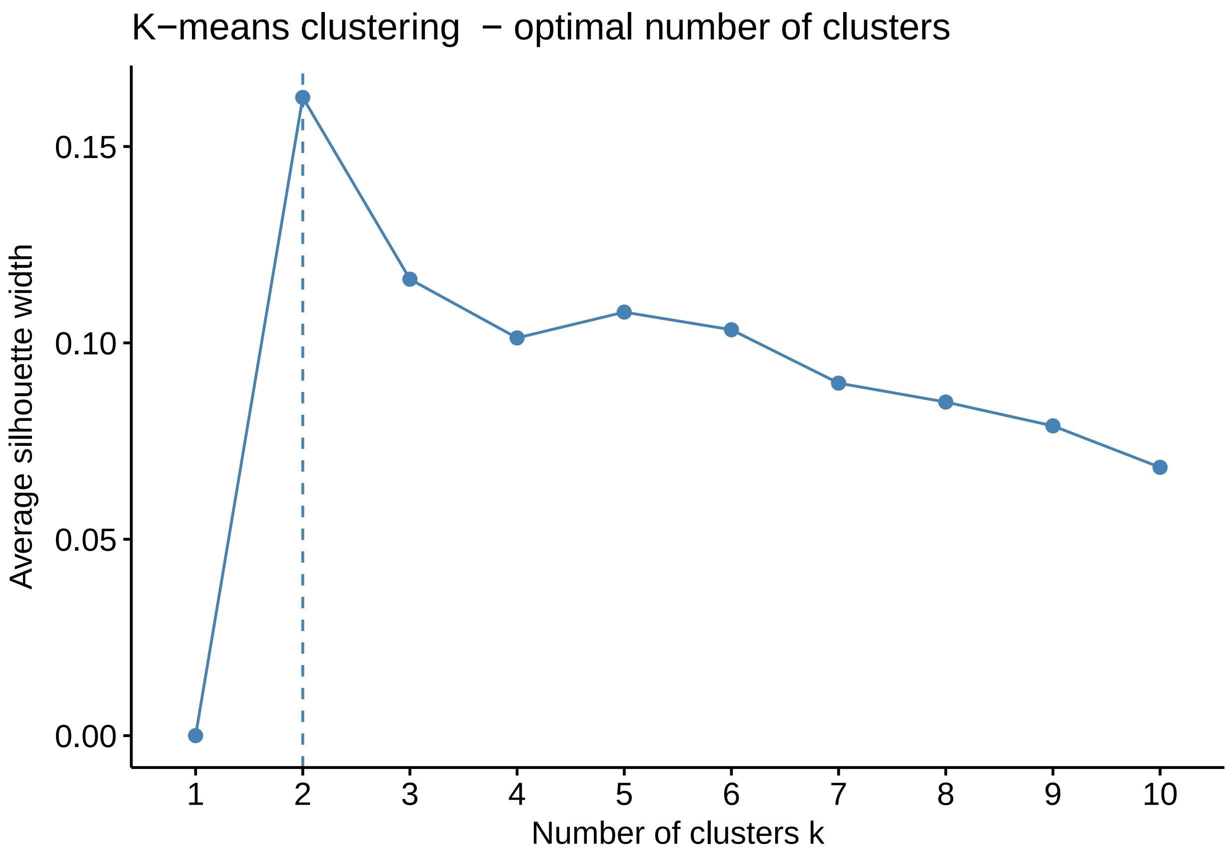
```r
#choose optimal number of clusters

# elbow plots
fviz_nbclust(data.scaled, kmeans,iter.max=30, method="wss", nstart = 5) +
  ggtitle("K-means clustering  - optimal number of clusters") +
```

```
  geom_vline(xintercept=5, linetype=2)
```




```
# average silhouette widths
fviz_nbclust(data.scaled, kmeans,method="silhouette", nstart = 5) +
  ggtitle("K-means clustering  - optimal number of clusters")
```

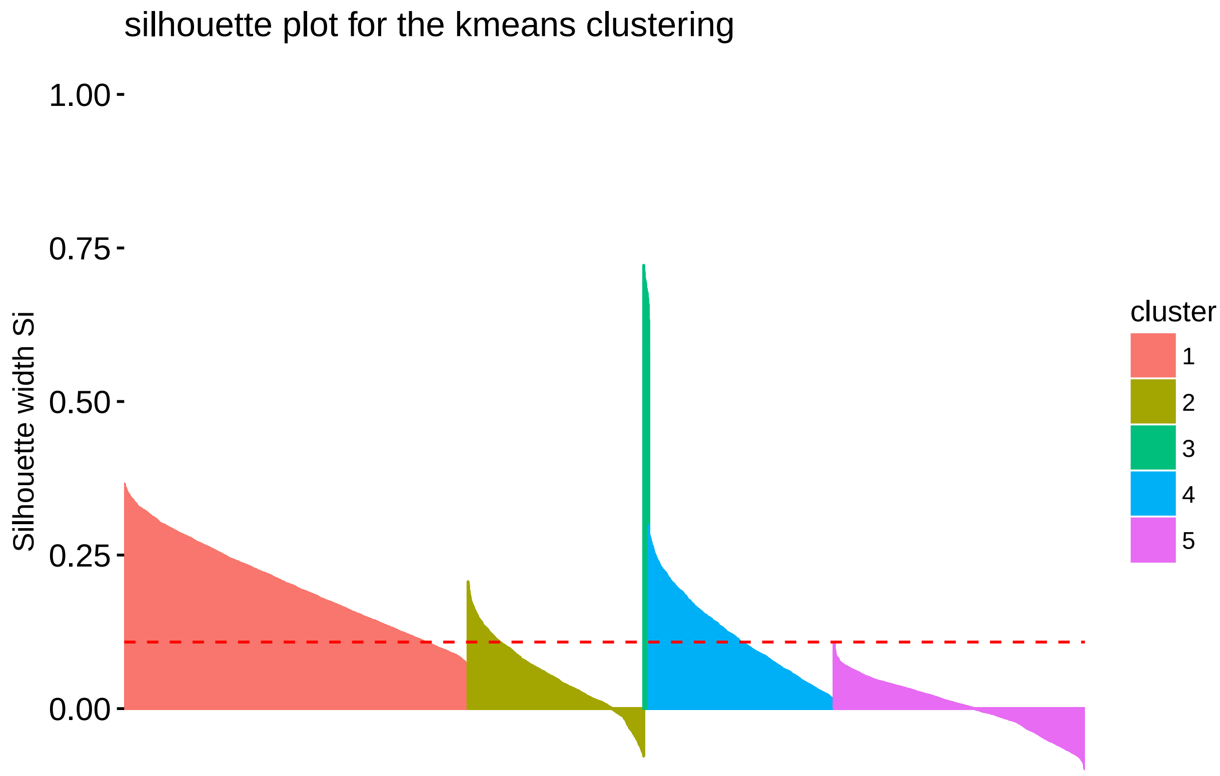


Since the average silhouette widths give too few clusters, we will like to choose the optimal number of clusters basde on the elbow plots.

```
cluster <- 5
data.km <- kmeans(data.scaled, cluster, nstart = 5)
```

```
# silhouette plot
sil_kmeans <- silhouette(data.km$cluster, dist = daisy(data.scaled))
fviz_silhouette(sil_kmeans) + ggtitle("silhouette plot for the kmeans clustering")
```

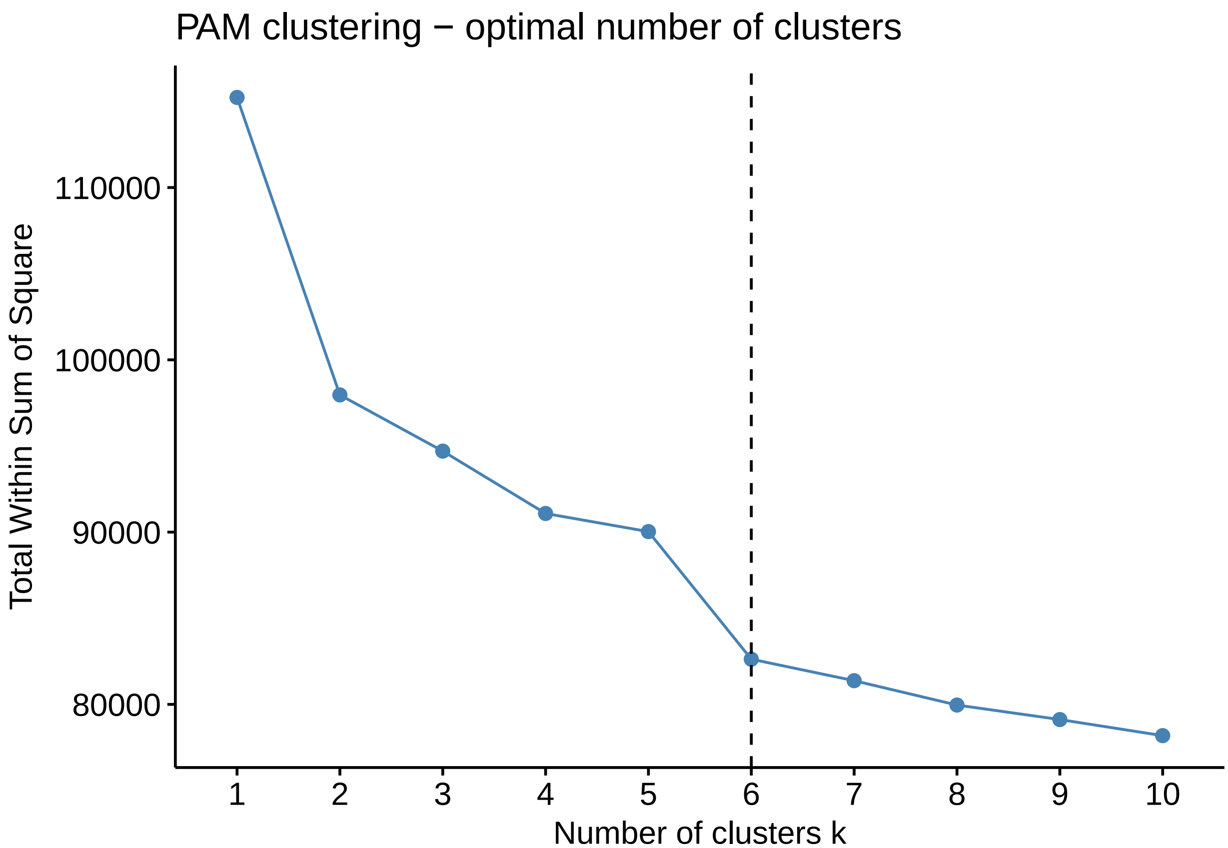
```
##   cluster size ave.sil.width
## 1       1 3621          0.20
## 2       2 1850          0.05
## 3       3   53          0.68
## 4       4 1953          0.12
## 5       5 2641          0.00
```



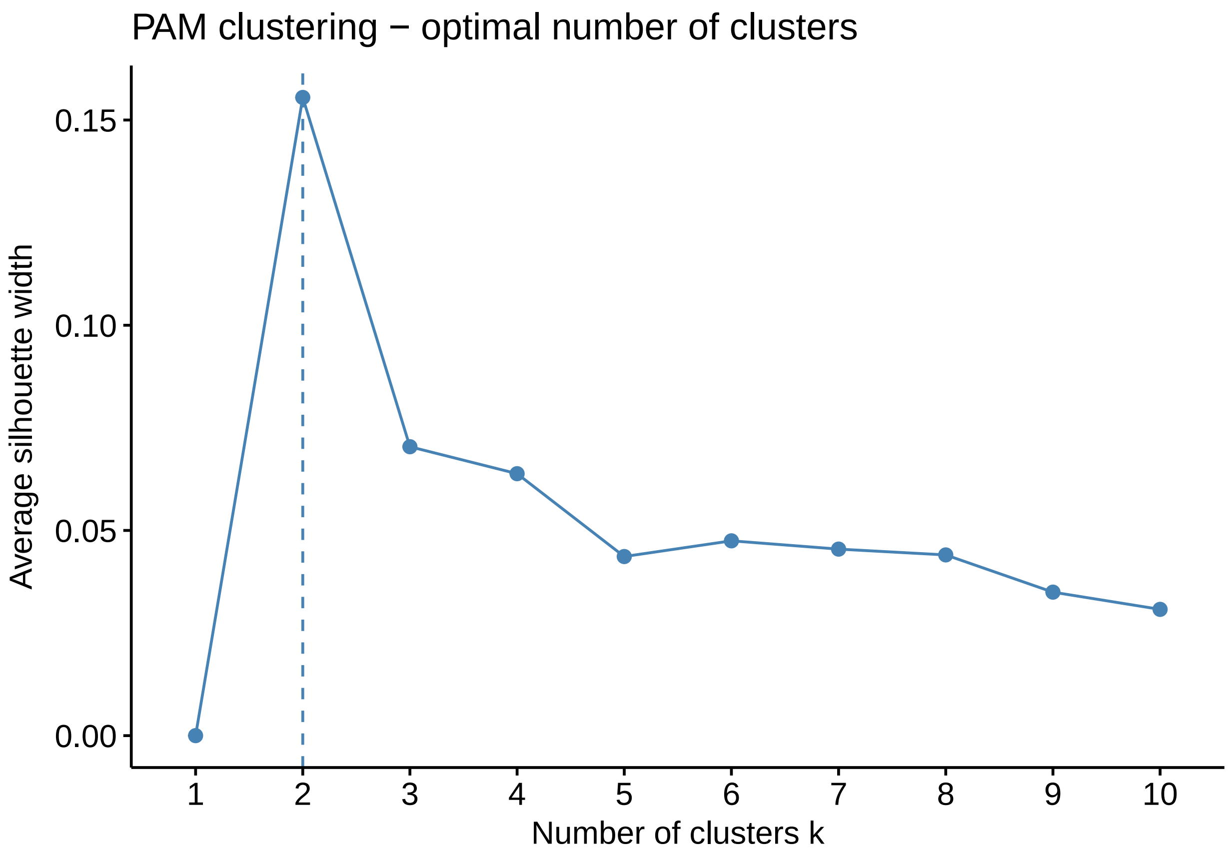
Partitioning around medoids (PAM)

```
# PAM
# find optimal cluster

# elbow plots
fviz_nbclust(data.scaled, pam, method="wss") +
  ggtitle("PAM clustering - optimal number of clusters") +
  geom_vline(xintercept=6,linetype=2)
```



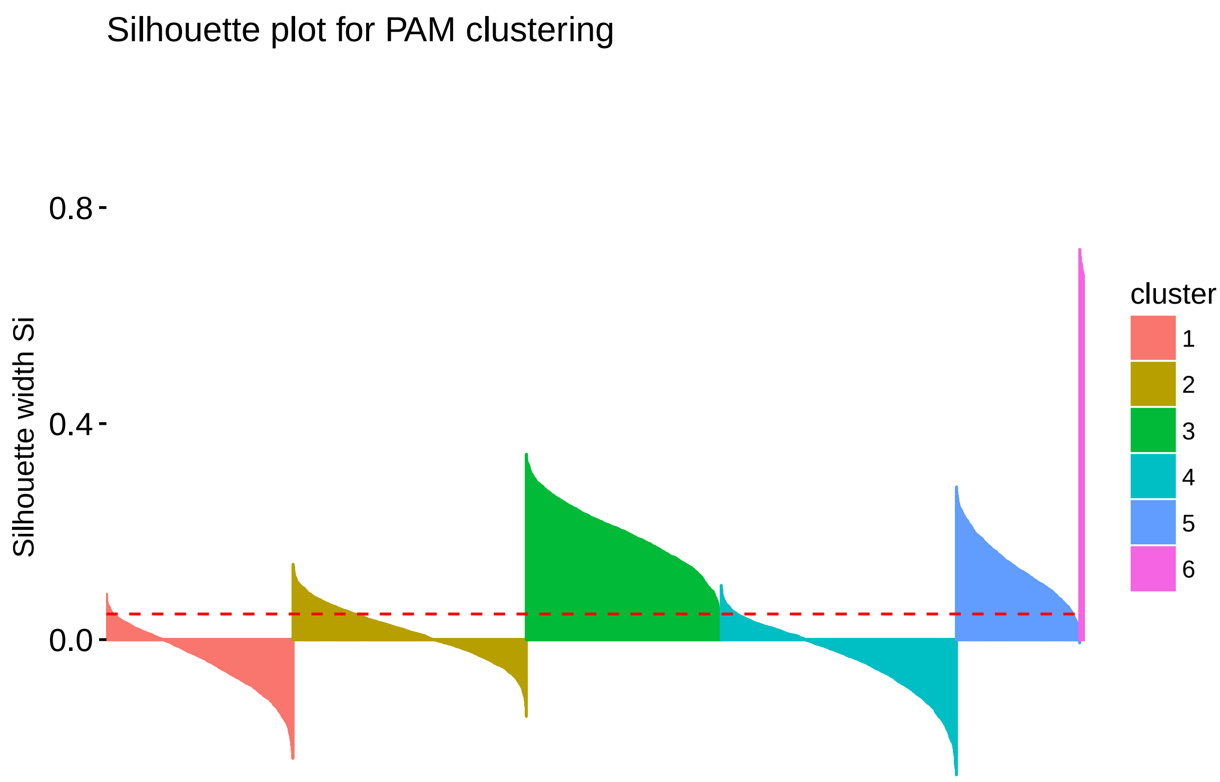

```
# average silhouette widths
fviz_nbclust(data.scaled,pam,method="silhouette") +
  ggtitle("PAM clustering - optimal number of clusters")
```




```
cluster <- 6
data.pam = pam(data.scaled, k=cluster)
```

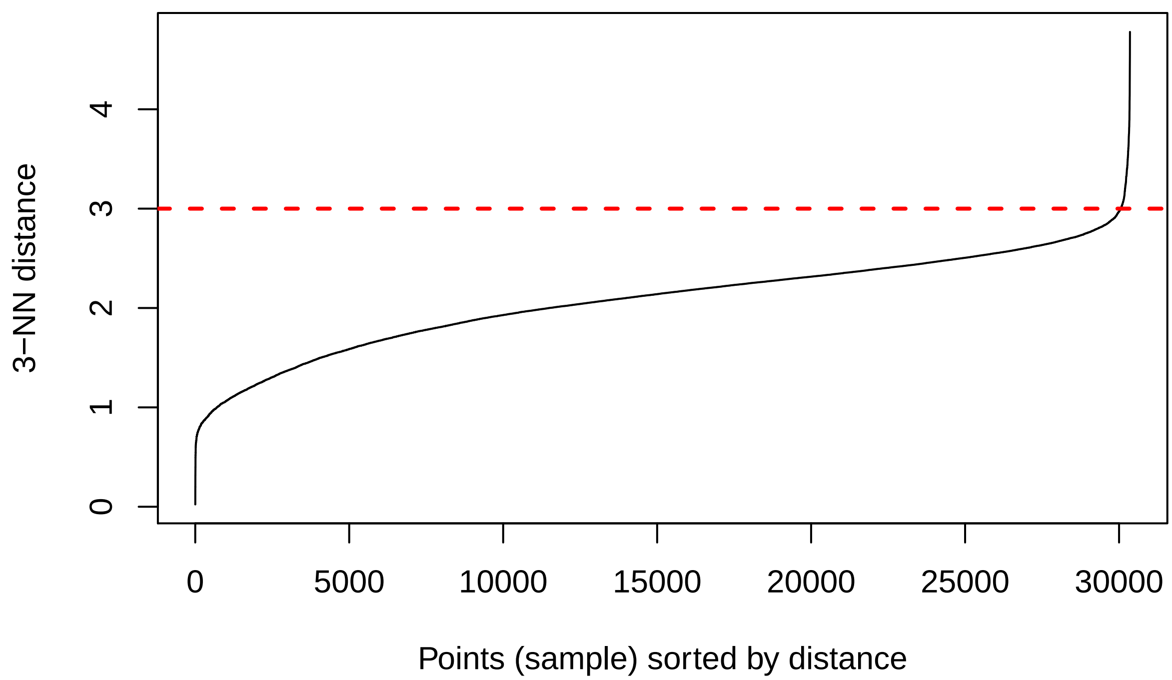
```
# silhouette plot
fviz_silhouette(silhouette(data.pam),
  main="Silhouette plot for PAM clustering")
```

```
##   cluster size ave.sil.width
## 1       1 1929         -0.04
## 2       2 2414          0.01
## 3       3 2014          0.20
## 4       4 2434         -0.03
## 5       5 1274          0.13
## 6       6   53          0.68
```



Density-based clustering

```r
kNNdistplot(data.scaled,k=3)
abline(3.0,0,lty=2,lwd=2,col="red") # added after seeing kNN plot
```




```r
#knees around 3.0
```

```r
data.db = dbscan(data.scaled, eps=3, minPts = 3)

data.db.df <- as.data.frame(data.db$cluster)
table(data.db.df)
```

```
## data.db.df
##    0    1    2    3
##   73 9999   38    8
```

```r
colnames(data.db.df)[1] <- "cluster"

#outliers
data.db.df %>%
  filter(cluster == 0)
```

```
##    cluster
## 1        0
## 2        0
## 3        0
## 4        0
## 5        0
## 6        0
## 7        0
## 8        0
## 9        0
## 10       0
## 11       0
## 12       0
## 13       0
## 14       0
## 15       0
## 16       0
## 17       0
## 18       0
## 19       0
## 20       0
## 21       0
## 22       0
## 23       0
## 24       0
## 25       0
## 26       0
## 27       0
## 28       0
## 29       0
## 30       0
## 31       0
## 32       0
## 33       0
## 34       0
## 35       0
## 36       0
## 37       0
## 38       0
## 39       0
```

```
## 40          0
## 41          0
## 42          0
## 43          0
## 44          0
## 45          0
## 46          0
## 47          0
## 48          0
## 49          0
## 50          0
## 51          0
## 52          0
## 53          0
## 54          0
## 55          0
## 56          0
## 57          0
## 58          0
## 59          0
## 60          0
## 61          0
## 62          0
## 63          0
## 64          0
## 65          0
## 66          0
## 67          0
## 68          0
## 69          0
## 70          0
## 71          0
## 72          0
## 73          0
```

Clustering comparison:

Since we would like to compare the performance of clustering based on their silhouette plot and whether the number of observations in clusters is reasonable, K-means out-performs PAM based on silhouette plot, and K-means gives much more reasonable number of observations in each cluster. Although the cluster distributin in K-means is still imbalanced (as we would expected), it is in a reasonable range. However, for the result we get from DBSCAN, we usually ends up with one cluster with too many observations than a reasonable expection (even choosing different parameters). Thus, we would like use K-means as the final clustering methods to generate response variables.

Process response variable from clustering result:

We proposed two methods to process response variables

(1) In each cluster, compute the occurance of each individual genre, then choose top three genres from them. Combine the top three genres from each cluster as the new label for that cluster. For example,

the top three populat genres in cluster 1 are Romance, Drama and Action, then the new label for cluster 1 is Romance, Drama, Action, together as the respones variable. However, this processing has two drawbacks. The first one is that it is very likely more than one cluster having the same top three genres. Also, some genres are more popular than others in the whole data set, so it will be more likely to be in the top three genres in each cluster.

(2) Thus, we proposed an alternative solution, by choosing the top one genre combination from each cluster as the response variable. It is very unlikely to have the same genre combination for different clusters using this method, and it will be consistent with the results based on the clustering. The main goal of clustering is to merge some similar genres into others, by using this method, we can satisify our inital goal.

```
#(1) compute each cluster's top three genres
data.df <- data.df %>% dplyr::mutate(cluster_response = data.km$'cluster')
for (i in 1:length(unique(data.df$cluster_response))){
  df1 <- data.df %>% filter(cluster_response == i) %>%
    dplyr::select(c(Sci.Fi:Biography))
  sort(colSums(df1))
}

genres <- colnames(data.df)[33:56]
genre_list = c("Sci.Fi","Crime", "Romance", "Animation", "Music",
"Adult", "Comedy", "War", "Horror", "Film.Noir", "Western", "Thriller",
"Adventure","Mystery","Short", "Drama", "Action", "Documentary", "Musical","History","Family","Fantasy"
data.df$genrecomb <- do.call(paste0, data.df[genre_list])

#(2) compute each cluster's top one genre combination
for (i in 1:length(unique(data.df$cluster_response))){
  df2 <- data.df %>% tidyr::unite(col =genres, Sci.Fi:Biography, sep = ",")
  df3 <- df2 %>% filter(cluster_response == i)
  print(genre_list[strsplit(df2[which.max(relative_req(df3$genres)),'genres'],",")[[1]]=="1"])
}
```

```
## [1] "Horror"   "Thriller" "Drama"
## [1] "Romance" "Music"   "Drama"
## [1] "Animation" "Comedy"    "Adventure" "Family"    "Fantasy"
## [1] "Romance" "Comedy"  "Fantasy"
## [1] "Horror"   "Thriller" "Action"
```

For our sample data, we end up with five clusters, and we figure out the genre combination for each cluster as following. Then we assign the result to the original data set.

```
map <-data.frame(cluster_response = c(1:5), genres_comb = c("Horror, Thriller, Drama",
                                                            "Romance, Music, Drama",
                                                            "Animation, Comedy, Adventure, Family, Fanta
                                                            "Romance, Comedy, Fantasy",
                                                            "Horror, Thriller, Action"))
```

Output the original data set with new response variable as a csv file

```
df5 <- left_join(data.df %>% dplyr::select(-ncol(data.df)), map, by = "cluster_response" )
write.csv(x = df5, file = "imdb_cluster_result.csv",row.names = F)
```

Milestone2_07_create_training_set

April 12, 2017

We will have to create our training set by merging IMDB and TMDB data set. 1. IMDB movies have a fewer movies number than TMDB. Since our eventual goal is use poster from TMDB for prediction as well, we will only consider movies that are in both IMDB and TMDB data set. Please note the example below only contains movies with TMDB ID from 1 to 20000.

```
In [1]: import pandas as pd
        from PIL import Image
        from StringIO import StringIO
        import requests
        import os
        import time
        from shutil import copyfile
        import csv
```

```
In [2]: dir_python_notebook = os.getcwd()
        dir_movie_project = os.path.abspath(os.path.join(dir_python_notebook, os.pa
        dir_data = os.path.join(dir_movie_project, 'data')
```

1 Load Data

```
In [53]: tmdb_filename = str(dir_data)+'\\drv_tmdb_movie_details_1_20000.csv'
         tmdb_movies = pd.read_csv(tmdb_filename,index_col=0, sep='\t', encoding='u
```

```
In [54]: imdb_filename = str(dir_data)+'\\imdb_cluster_result.csv'
         imdb_movies = pd.read_csv(imdb_filename, sep=',', encoding='utf-8', quotin
         imdb_movies.head(5)
```

```
Out[54]:          X  certificates_R  certificates_PG  art.direction_1  \
         0    100.0             1.0              0.0         0.322495
         1  10001.0             0.0              1.0         0.027673
         2  10002.0             1.0              0.0         0.212394
         3  10003.0             0.0              1.0         0.019767
         4  10004.0             1.0              0.0         0.104764

            assistant.director_1  casting.director_1  cinematographer_1  \
         0              0.042103            0.010279           0.046254
         1              0.093694            0.319431           0.165250
```

```
2              0.024906             0.006523              0.016308
3              0.024906             0.014133              0.025301
4              0.355406             0.017395              0.049812

   costume.department_1  costume.designer_1  countries_1  \
0              0.307966             0.028662     0.089642
1              0.307966             0.172860     0.014232
2              0.012453             0.005535     0.089642
3              0.126112             0.007116     0.536766
4              0.126112             0.040028     0.536766

            ...             Action  Documentary  Musical  History  Family
0           ...                0.0          0.0      0.0      0.0     0.0
1           ...                0.0          0.0      0.0      1.0     0.0
2           ...                0.0          0.0      0.0      0.0     0.0
3           ...                1.0          0.0      0.0      0.0     0.0
4           ...                0.0          0.0      0.0      0.0     0.0

   Fantasy  Sport  Biography  cluster_response                genres_comb
0      0.0    0.0        0.0               5.0  Horror, Thriller, Action
1      0.0    0.0        0.0               4.0  Romance, Comedy, Fantasy
2      0.0    0.0        0.0               1.0   Horror, Thriller, Drama
3      0.0    0.0        0.0               1.0   Horror, Thriller, Drama
4      1.0    0.0        0.0               1.0   Horror, Thriller, Drama

[5 rows x 58 columns]
```

In [55]:
```
print tmdb_movies.shape
```

```
(12210, 25)
```

In [56]:
```
print imdb_movies.shape
```

```
(10118, 58)
```

In [60]:
```
import numpy as np
tmdb_movies['tmdb_id'] = tmdb_movies['id']
imdb_movies['tmdb_id'] = imdb_movies['X']

tmdb_movies.tmdb_id = tmdb_movies.tmdb_id.astype(np.int64)
imdb_movies.tmdb_id = imdb_movies.tmdb_id.astype(np.int64)
#tmdb_movies[['tmdb_id']] =tmdb_movies[['tmdb_id']].apply(pd.to_integer)
#imdb_movies[['tmdb_id']] =imdb_movies[['tmdb_id']].apply(pd.to_integer)
```

In [61]:
```
training_df = tmdb_movies.join(imdb_movies, how = "inner",on = "tmdb_id",
```

In [42]:
```
training_df.columns
```

```
Out[42]: Index([                      u'adult',                u'backdrop_path',
                    u'belongs_to_collection',                       u'budget',
                                   u'genres',                     u'homepage',
                                       u'id',                      u'imdb_id',
                        u'original_language',               u'original_title',
                                 u'overview',                   u'popularity',
                              u'poster_path',         u'production_companies',
                     u'production_countries',                 u'release_date',
                                  u'revenue',                      u'runtime',
                         u'spoken_languages',                       u'status',
                                  u'tagline',                        u'title',
                                    u'video',                 u'vote_average',
                               u'vote_count',                  u'tmdb_idtmdb',
                                        u'X',               u'certificates_R',
                          u'certificates_PG',             u'art.direction_1',
                     u'assistant.director_1',          u'casting.director_1',
                        u'cinematographer_1',       u'costume.department_1',
                       u'costume.designer_1',                 u'countries_1',
                               u'director_1',              u'distributors_1',
                                 u'editor_1',                 u'languages_1',
                                u'make.up_1', u'miscellaneous.companies_1',
                     u'miscellaneous.crew_1',            u'original.music_1',
                               u'producer_1',    u'production.companies_1',
                     u'production.manager_1',            u'set.decoration_1',
                             u'sound.crew_1',            u'stunt.performer_1',
                       u'visual.effects_1',                     u'writer_1',
                                   u'cast_1',                       u'cast_2',
                                   u'cast_3',                       u'cast_4',
                             u'runtimes_avg',                       u'rating',
                                   u'Sci.Fi',                        u'Crime',
                                  u'Romance',                    u'Animation',
                                    u'Music',                        u'Adult',
                                   u'Comedy',                          u'War',
                                   u'Horror',                    u'Film.Noir',
                                  u'Western',                     u'Thriller',
                                u'Adventure',                      u'Mystery',
                                    u'Short',                        u'Drama',
                                   u'Action',                  u'Documentary',
                                  u'Musical',                      u'History',
                                   u'Family',                      u'Fantasy',
                                    u'Sport',                    u'Biography',
                         u'cluster_response',                  u'genres_comb',
                                  u'tmdb_id'],
      dtype='object')
```

```
In [63]: filename = str(dir_data)+'\\training.csv'
         training_df.to_csv(filename,header =training_df.columns, sep='\t', encodir
```

```
In [ ]:
```