Data Collection, Cleaning, Feature Extraction and Sampling

In [ ]:

```
# PART 1:  STANDALONE TO GRAB ALL MOVIES AND KEYWORDS

import csv
import time
import requests


############################################################
'''
BASE STUFF THAT IS ALSO DEFINED ON TOP
'''
def requestResults(url):
    r = requests.get(BASE_URL + url + "&api_key=" + API_KEY)
    return r.json()

# Constants
BASE_URL = "https://api.themoviedb.org/3/"
API_KEY = "9767d17413ec9d9729c2cca238df02da"
GENRE_MAP = {}
for g in requestResults("genre/movie/list?x=1")[u'genres']:
    GENRE_MAP[g['id']] = g['name']

############################################################


def _getKeywordsStringById(movie_id):

    keywords_dict = requestResults("movie/" + str(movie_id) + "/keywords?language=e
    if u'keywords' not in keywords_dict:
        return ''
    keywords_dict = keywords_dict[u'keywords']
    kstring = ''
    for k in keywords_dict:
        kstring += k[u'name'] + ','
    return str(kstring.encode('utf-8').strip())[:-1]

def _tidyRow(m, keywords):
    # Makes sure the row of movie is well-formatted
    output = {}
    for k in m:
        typem = type(m[k])
        k = str(k)
        if typem == str or typem == unicode:
            output[k] = m[k].encode('utf-8').strip()
        else:
            output[k] = m[k]
    output['keywords'] = keywords
    return output

def downloadMoviesToCSV(start_page, increment, filename):
    genre_count = {}

    with open(filename, 'w') as csvfile:
        fieldnames = ['id', 'genre_ids', 'poster_path', 'title', 'overview', 'releas
                      'popularity', 'original_title', 'backdrop_path', 'keywords',
                     'vote_count', 'video', 'adult', 'vote_average', 'original_langu
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

```
        # Get keywords for movies
        writer.writeheader()

        # TMDB limits 4 requests per second
        hit = 3 # Once hit reaches 0, call timer and reset hit to 3

        for p in range(start_page,start_page+increment):
            results_p = requestResults("discover/movie?sort_by=popularity.desc&page=
            hit -= 1
            if hit <= 0:
                hit = 3
                time.sleep(1)

            # Write to CSV
            for m in results_p:
                mid = m[u'id']
                keywords = _getKeywordsStringById(mid)
                hit -= 1
                if hit <= 0:
                    hit = 3
                    time.sleep(1)

                row = _tidyRow(m, keywords)
                writer.writerow(row)
            print('%d pages done' % p)
```

In [ ]:

```
### Run Part 1: REMEMBER TO CHANGE start_page to your start page, don't have to cha
downloadMoviesToCSV(start_page=1, increment=400, filename='tmdb-movies-1-to-400.csv
```

In [ ]:

```
# PART 2: STANDALONE THAT TAKES IN .CSV FILE AND GETS ALL IMDB IDS in a separate fi

import pandas as pd
import csv
import time
import requests


############################################################
'''
BASE STUFF THAT IS ALSO DEFINED ON TOP
'''
def requestResults(url):
    r = requests.get(BASE_URL + url + "&api_key=" + API_KEY)
    return r.json()

# Constants
BASE_URL = "https://api.themoviedb.org/3/"
API_KEY = "9767d17413ec9d9729c2cca238df02da"
GENRE_MAP = {}
for g in requestResults("genre/movie/list?x=1")[u'genres']:
    GENRE_MAP[g['id']] = g['name']

############################################################

def downloadIMDBIds(input_filename, output_filename):
    df = pd.read_csv(input_filename)

    with open(output_filename, 'w') as csvfile:
        fieldnames = ['id', 'imdb_id']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()

        # TMDB limits 4 requests per second
        hit = 3 # Once hit reaches 0, call timer and reset hit to 3

        count = 0
        for tmid in df['id']:
            count += 1
            results = requestResults('movie/' + str(tmid) + '?x=1')
            if u'imdb_id' not in results or results[u'imdb_id'] is None:
                continue
            imid = results[u'imdb_id'].strip('tt')
            row = {'id': tmid, 'imdb_id': imid}
            writer.writerow(row)
            hit -= 1
            if hit <= 0:
                hit = 3
                time.sleep(1)
            if count % 200 == 0:
                print 'done with %d movies' % count
```

In [ ]:

```
### Run Part 2: Get imdb ids from tmdb ids input csv file
downloadIMDBIds(input_filename='tmdb-movies-1-to-400.csv', output_filename='imdb-ids
```

In [ ]:

```
# PART 3: STANDALONE THAT TAKES IN IMDB IDs and gets IMDB features
'''
Make sure you have IMDB installed.
- Go to: http://imdbpy.sourceforge.net/
- Download and unzip, then cd into it and make sure there is a setup.py file
- Run python setup.py install
- You're done! It's globally installed.
'''
import imdb
import pandas as pd
import csv
import requests
import numpy as np

def getIMDBFeatures(input_filename, output_filename, start, increment):

    # Note: This cannot be terminated via the stop button (interrupt the kernel),
    # got to restart the kernel (use rewind button) :(

    ia = imdb.IMDb()
    df = pd.read_csv(input_filename)
    # Download increment movies at a time
    df = df[start:start+increment]

    imids = np.array(df['imdb_id'])

    with open(output_filename + '-' + str(start), 'w') as csvfile:
        # Grab these features from IMDB
        fieldnames = ['imdb_id', 'director', 'imdb_votes', 'certificate', 'num_stun
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()

        count = 0
        for imid in imids:
            count += 1
            # Tries twice because sometimes it fails
            for i in range(2):
                try:
                    movie = ia.get_movie(str(int(imid)))
                    director = movie['director'][0]
                    imdb_votes = movie['votes']
                    certificate = movie['certificates'][-2].split(':')[1]
                    num_stunts = len(movie['stunt performer'])
                    num_fx = len(movie['special effects department'])
                    row = {'imdb_id': imid, 'director': director, 'imdb_votes': imdb
                           'num_stunts': num_stunts, 'num_fx': num_fx}
                    writer.writerow(row)
                    break
                except:
                    pass
            if count % 100 == 0:
                print 'Done with %d movies' % count
    print 'Done with page %d' % ((start%increment) + 1)
```

In [ ]:

```
### Run Part 3: Get imdb features from imdb ids

# NOTE: This downloads 500 movies at a time and stores each in a different file.

import pandas as pd
df = pd.read_csv('imdb-ids-1001-to-1200.csv')
N = df.shape[0]
increment = 500 # Work on 500 movies at a time
end_page = N/increment

################################################################
# NOTE: If you are done with page 2 (1000 movies), then change this to 2 the next t
start_page = 0
################################################################

starts = []
for i in range(start_page,end_page): # default starts: [500,1000,1500,2000,2500,..,
    starts.append((i+1)*increment)

for start in starts:
    getIMDBFeatures(input_filename='imdb-ids-1-to-400.csv', output_filename='imdb-fe
                    start=start, increment=increment)
```

In [ ]:

```
### PART 4: Merge all and output CSV file

import pandas as pd

# NOTE: Change to your filepath and start and end movie
prefix_filepath = ''
start = 1
end = 1000

# Merge all imdb features into one
imdb_features = pd.read_csv(prefix_filepath + 'imdb-features-'+str(start)+'-to-'+str
for p in range(2,8):
    imdb_features_ = pd.read_csv(prefix_filepath + 'imdb-features-'+str(start)+'-to-
    imdb_features = imdb_features.append(imdb_features_)


# Merge imdb ids with imdb features
imdb_ids = pd.read_csv(prefix_filepath + 'imdb-ids-'+str(start)+'-to-'+str(end)+'.cs
imdb_ids = imdb_ids.rename(index=str, columns={"id": "tmdb_id"})
imdb_merged = imdb_ids.merge(imdb_features, how='outer', left_on='imdb_id', right_on
imdb_merged = imdb_merged.dropna()

# Merge tmdb with imdb_merge
tmdb_movies = pd.read_csv(prefix_filepath + 'tmdb-movies-'+str(start)+'-to-'+str(end
tmdb_movies = tmdb_movies.rename(index=str, columns={"id": "tmdb_id"})
full_movies = tmdb_movies.merge(imdb_merged, how='outer', left_on='tmdb_id', right_o
full_movies = full_movies.dropna()

# Output this to CSV of full movies
full_movies.to_csv('full-movies-'+str(start)+'-to-'+str(end)+'.csv', index=False)
```

In [ ]:

```
### PART 5: Clean up columns to choose the right ones for Milestone 3
import pandas as pd
import numpy as np
import requests

#############################################################
'''
BASE STUFF THAT IS ALSO DEFINED ON TOP
'''
def requestResults(url):
    r = requests.get(BASE_URL + url + "&api_key=" + API_KEY)
    return r.json()

# Constants
BASE_URL = "https://api.themoviedb.org/3/"
API_KEY = "9767d17413ec9d9729c2cca238df02da"
GENRE_MAP = {}
for g in requestResults("genre/movie/list?x=1")[u'genres']:
    GENRE_MAP[g['id']] = g['name']

#############################################################

# Merge the few tmdb-movies files together
df1 = pd.read_csv('full-movies-1-to-400.csv')
df2 = pd.read_csv('full-movies-401-to-800.csv')
df3 = pd.read_csv('full-movies-801-to-1000.csv')
df = (df1.append(df2)).append(df3)

# Choose only columns we need
cols = ['genre_ids', 'title', 'poster_path', 'tmdb_id', 'release_date', 'popularity
       'vote_average', 'director', 'imdb_votes', 'certificate', 'num_stunts', 'num_
df = df[cols]

# Break down release date into month and year
datesplit = df['release_date'].str.split('-')
years = [int(d[0]) for d in datesplit]
months = [int(d[1]) for d in datesplit]
df['year'] = years
df['month'] = months
del df['release_date']

# Split year into decades
base = 1910
mod_var = 10
years = np.array(df['year'])
decades = (years - base) / mod_var
df['decade'] = decades
df.head()
del df['year']

# Tidy up the certificate and one-hot encoding

# Look at the certs
cert_ratings = np.unique(df['certificate'], return_counts=True)[0]
cert_counts = np.unique(df['certificate'], return_counts=True)[1]
df_cert = pd.DataFrame(columns=['name', 'count'])
df_cert['name'] = cert_ratings
```

```
df_cert['count'] = cert_counts

df_cert.sort_values('count', ascending=False)

# Relabel certs
df = df[~df['certificate'].isin(['(Banned)','10'])]
certs_formatted = []
certs = df['certificate']
for c in certs:
    if c in ['Tous publics', 'U', 'T', 'M/6', 'L', 'G', 'Btl', 'All', 'AL', 'AA', '
        certs_formatted.append('U')
    elif c in ['Unrated', 'Not Rated', 'H', 'B', ]: # Unrated
        certs_formatted.append('Unrated')
    elif c in ['X', 'TV-MA', 'R21', 'R(A)', 'IIB', '(Banned)', 'R']: # R
        certs_formatted.append('R')
    elif c in ['VM14', 'TV-14', 'R16', 'R-16', 'R-15', 'NC16',
               'NC-17', 'NC-16', 'MA15+', 'M/16', 'M/14', 'M', 'K-15', 'B-15', '16'
              '14', '14A']: # 15
        certs_formatted.append('15')
    elif c in ['TV-Y7', 'TV-G', 'TV-PG', 'Passed', 'PG', 'M/PG', 'K-7', 'IIA', 'GP'
              '7', '8', '9', '6', '10']: # PG
        certs_formatted.append('PG')
    elif c in ['R18+', 'R18', 'R-18', 'M18', 'M/18', '18']: # 18
        certs_formatted.append('18')
    elif c in ['R13', 'R-13', 'R-12', 'PG12', 'PG13', 'PG-12', 'PG-13',
               'P13', 'M/12', 'K-13', 'K-12', 'K-11', '13+', '13', '12A', '12+', '1
        certs_formatted.append('12')
    else: # If we miss out, which is unlikely, just mark Unrated
        print c
certs_formatted = np.array(certs_formatted)
df['certificate'] = certs_formatted

df.to_csv('full-movies-merged.csv', index=False)
```

In [ ]:

```
### PART 6: Based on the genre correlation heatmap from Milestone 1
#to create 7 genre groups with correponding genre ids

combine_genre = {'group1': ["War","History"],
                 'group2': ["Crime","Mystery","Thriller","Drama","Horror"],
                 'group3': ["Fantasy"],
                 'group4': ["Family","Animation"],
                 'group5': ["Romance","Music"],
                 'group6': ["Science Fiction","Action","Adventure"],
                 'group7': ["Comedy"]}
# no western, tv movie, and documentary due to the insignificant number of movies f

combine_genre_ids = {'group1': ['10752', '36'],
                 'group2': ['80', '9648', '53', '18','27'],
                 'group3': ['14'],
                 'group4': ['10751','16'],
                 'group5': ['10749', '10402'],
                 'group6': ['878', '28', '12'],
                 'group7': ['35']}
```

In [ ]:

```
### PART 7: One hot encoding of genre groups

full_df = pd.read_csv("full-movies-merged.csv")

def one_hot_encoding_genre_group(df):
    num_row = len(df)
    group_df = pd.DataFrame()
    for i in range(7):
        col_name = combine_genre_ids.keys()[i]
        print(col_name)
        col_genre = combine_genre_ids.values()[i]
        #print(col_genre)
        group_df[col_name] = [0]*num_row
        for row in range(num_row):
            genres_id = df['genre_ids'][row][1:-1].split(",")
            genres_list = []
            for i in genres_id:
                genres_list.append(str(i.strip()))
            #print(genres_list)
            overlap = bool(set(genres_list) & set(col_genre))
            if overlap == True:
                #print("there is a match")
                group_df[col_name][row] = 1
        print(col_name + " done")
    return(group_df)

group_df = one_hot_encoding_genre_group(full_df)

# merge_with_group dataframe includes all the attributes in the original dataset,
# along with 7 additional binary-value columns for genre groups

merge_with_group = pd.concat([full_df, group_df], axis=1)
```

In [ ]:

```
### Part 8: Split full dataset to 60% - training set(inbalanced) & 40% - testing se

import pandas as pd
import numpy as np
import random

df = merge_with_group
counts = []
train_df = pd.DataFrame()
test_df = pd.DataFrame()

for i in range(1,8):
    sub_df = df[df['group' + str(i)] == 1]
    count = sub_df.shape[0]
    rows = random.sample(sub_df.index, int(count*0.6))
    sub_train = sub_df.ix[rows]
    sub_test = sub_df.drop(rows)

    train_df = pd.concat([train_df, sub_train], axis=0)
    test_df = pd.concat([test_df, sub_test], axis=0)

    counts.append(count)
```