6. Clustering

March 30, 2022

[1]:
```
import numpy as np
import pandas as pd
import nltk
import re
import os
import codecs
from sklearn import feature_extraction
```

[2]:
```
#import three lists: titles, links and wikipedia synopses
titles = open('title_list.txt').read().split('\n')
# ensures that only the first 100 are read
titles = titles[:100]
titles
```

[2]:
```
['The Godfather',
 'The Shawshank Redemption',
 "Schindler's List",
 'Raging Bull',
 'Casablanca',
 "One Flew Over the Cuckoo's Nest",
 'Gone with the Wind',
 'Citizen Kane',
 'The Wizard of Oz',
 'Titanic',
 'Lawrence of Arabia',
 'The Godfather: Part II',
 'Psycho',
 'Sunset Blvd.',
 'Vertigo',
 'On the Waterfront',
 'Forrest Gump',
 'The Sound of Music',
 'West Side Story',
 'Star Wars',
 'E.T. the Extra-Terrestrial',
 '2001: A Space Odyssey',
 'The Silence of the Lambs',
```

```
'Chinatown',
'The Bridge on the River Kwai',
"Singin' in the Rain",
"It's a Wonderful Life",
'Some Like It Hot',
'12 Angry Men',
'Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb',
'Amadeus',
'Apocalypse Now',
'Gandhi',
'The Lord of the Rings: The Return of the King',
'Gladiator',
'From Here to Eternity',
'Saving Private Ryan',
'Unforgiven',
'Raiders of the Lost Ark',
'Rocky',
'A Streetcar Named Desire',
'The Philadelphia Story',
'To Kill a Mockingbird',
'An American in Paris',
'The Best Years of Our Lives',
'My Fair Lady',
'Ben-Hur',
'Doctor Zhivago',
'Patton',
'Jaws',
'Braveheart',
'The Good, the Bad and the Ugly',
'Butch Cassidy and the Sundance Kid',
'The Treasure of the Sierra Madre',
'The Apartment',
'Platoon',
'High Noon',
'Dances with Wolves',
'The Pianist',
'Goodfellas',
'The Exorcist',
'The Deer Hunter',
'All Quiet on the Western Front',
'The French Connection',
'City Lights',
"The King's Speech",
'It Happened One Night',
'A Place in the Sun',
'Midnight Cowboy',
'Mr. Smith Goes to Washington',
```

```
'Rain Man',
'Annie Hall',
'Out of Africa',
'Good Will Hunting',
'Terms of Endearment',
'Tootsie',
'Fargo',
'Giant',
'The Grapes of Wrath',
'Shane',
'The Green Mile',
'Close Encounters of the Third Kind',
'Network',
'Nashville',
'The Graduate',
'American Graffiti',
'Pulp Fiction',
'The African Queen',
'Stagecoach',
'Mutiny on the Bounty',
'The Maltese Falcon',
'A Clockwork Orange',
'Taxi Driver',
'Wuthering Heights',
'Double Indemnity',
'Rebel Without a Cause',
'Rear Window',
'The Third Man',
'North by Northwest',
'Yankee Doodle Dandy']
```

[3]:
```
synopsis = open('synopsis_list.txt').read().split('\n BREAKS HERE')
synopsis = synopsis[:100]
```

[4]:
```
# load nltk's English stopwords as variable called 'stopwords'
stopwords = nltk.corpus.stopwords.words('english')

# load nltk's SnowballStemmer as variabled 'stemmer'
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer("english")
```

[5]:
```
def tokenize_and_stem(text):
    # first tokenize by sentence, then by word
    # to ensure that punctuation is caught as it's own token
    tokens = [word for sent in nltk.sent_tokenize(text) \
              for word in nltk.word_tokenize(sent)]
    filtered_tokens = []
```

```
    # filter out any tokens not containing letters
    # (e.g., numeric tokens, raw punctuation)
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    stems = [stemmer.stem(t) for t in filtered_tokens]
    return stems


def tokenize_only(text):
    # first tokenize by sentence then by word
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) \
              for word in nltk.word_tokenize(sent)]
    filtered_tokens = []
    # filter out any tokens not containing letters
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    return filtered_tokens

totalvocab_stemmed = []
totalvocab_tokenized = []
for i in synopsis:
    # for each item in 'synopses', tokenize/stem
    allwords_stemmed = tokenize_and_stem(i)
    # extend the 'totalvocab_stemmed' list
    totalvocab_stemmed.extend(allwords_stemmed)

    allwords_tokenized = tokenize_only(i)
    totalvocab_tokenized.extend(allwords_tokenized)
len(totalvocab_tokenized)
```

[5]: 147611

[6]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

#define vectorizer parameters
tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=200000, min_df=0.2,
                   use_idf=True, tokenizer=tokenize_and_stem, ngram_range=(1,3))

#fit the vectorizer to synopses
%time tfidf_matrix = tfidf_vectorizer.fit_transform(synopsis)

print(tfidf_matrix.shape)

#terms = tfidf_vectorizer.get_feature_names()
#print(terms)
```

```
Wall time: 4.87 s
(100, 809)
```

[7]:
```
from sklearn.cluster import KMeans

num_clusters = 5

km = KMeans(n_clusters=num_clusters)

%time km.fit(tfidf_matrix)

clusters = km.labels_.tolist()
print(clusters)
```

```
Wall time: 101 ms
[3, 2, 3, 0, 1, 3, 1, 1, 1, 1, 3, 3, 1, 1, 1, 2, 1, 1, 1, 3, 3, 3, 1, 1, 3, 1,
2, 1, 0, 3, 4, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 2, 3, 3, 3, 3, 3, 3,
3, 1, 3, 0, 3, 3, 2, 1, 3, 0, 3, 0, 3, 1, 4, 4, 3, 2, 1, 1, 2, 0, 1, 1, 0, 0, 0,
2, 3, 0, 1, 1, 1, 3, 0, 0, 0, 1, 2, 1, 0, 1, 0, 1, 1, 1, 0]
```

[ ]:
```

```