In [1]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from ast import literal_eval
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate

import warnings; warnings.simplefilter('ignore')
```

In [2]:
```
md = pd.read_csv('Data-Asset/movies_metadata.csv')
md.head(2)
```

Out[2]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overview	...	release_date
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody, Andy's toys live happily in his ...	...	1995-10-30
1	False		NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an encha...	...	1995-12-15

2 rows × 24 columns

In [3]:
```
md['genres'] = md['genres'].fillna('[]').apply(literal_eval).apply(lambda x: [i['name'] for i in x] if isinstance(x, list) else
```

In [4]:
```
md['year'] = pd.to_datetime(md['release_date'], errors='coerce').apply(lambda x: str(x).split('-')[0] if x != np.nan else np.nan)
```

In [5]:
```
links_small = pd.read_csv('Data-Asset/links_small.csv')
links_small = links_small[links_small['tmdbId'].notnull()]['tmdbId'].astype('int')
```

In [6]:
```
md = md.drop([19730, 29503, 35587])
```

In [7]:
```
md['id'] = md['id'].astype('int')
smd = md[md['id'].isin(links_small)]
print(smd.shape)
```

```
(9099, 25)
```

In [8]:
```
smd['tagline'] = smd['tagline'].fillna('')
smd['description'] = smd['overview'] + smd['tagline']
smd['description'] = smd['description'].fillna('')
```

In [9]:
```
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(smd['description'])
```

In [10]:
```
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [11]:
```
smd = smd.reset_index()
titles = smd['title']
indices = pd.Series(smd.index, index=smd['title'])
```

In [12]:
```
credits = pd.read_csv('Data-Asset/credits.csv')
keywords = pd.read_csv('Data-Asset/keywords.csv')
```

In [13]:
```
keywords['id'] = keywords['id'].astype('int')
credits['id'] = credits['id'].astype('int')
md['id'] = md['id'].astype('int')
```

In [14]:
```
md.shape
```

Out[14]:
```
(45463, 25)
```

In [15]:
```
md = md.merge(credits, on='id')
md = md.merge(keywords, on='id')
```

In [16]:
```
smd = md[md['id'].isin(links_small)]
smd.shape
```

Out[16]:
```
(9219, 28)
```

In [17]:
```
smd['cast'] = smd['cast'].apply(literal_eval)
smd['crew'] = smd['crew'].apply(literal_eval)
smd['keywords'] = smd['keywords'].apply(literal_eval)
smd['cast_size'] = smd['cast'].apply(lambda x: len(x))
smd['crew_size'] = smd['crew'].apply(lambda x: len(x))
```

In [18]:
```
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return np.nan
```

In [19]:
```
smd['director'] = smd['crew'].apply(get_director)
```

In [20]:
```
smd['cast'] = smd['cast'].apply(lambda x: [i['name'] for i in x] if isinstance(x, list) else [])
smd['cast'] = smd['cast'].apply(lambda x: x[:3] if len(x) >=3 else x)
```

In [21]:
```
smd['keywords'] = smd['keywords'].apply(lambda x: [i['name'] for i in x] if isinstance(x, list) else [])
```

In [22]:
```
smd['cast'] = smd['cast'].apply(lambda x: [str.lower(i.replace(" ", "")) for i in x])
```

In [23]:
```
smd['director'] = smd['director'].astype('str').apply(lambda x: str.lower(x.replace(" ", "")))
smd['director'] = smd['director'].apply(lambda x: [x,x, x])
```

In [24]:
```
s = smd.apply(lambda x: pd.Series(x['keywords']),axis=1).stack().reset_index(level=1, drop=True)
s.name = 'keyword'
```

In [25]:
```
s = s.value_counts()
s[:5]
```

Out[25]:
```
independent film        610
woman director          550
murder                  399
duringcreditsstinger    327
based on novel          318
Name: keyword, dtype: int64
```

In [26]:
```
s = s[s > 1]
```

In [27]:
```
stemmer = SnowballStemmer('english')
stemmer.stem('dogs')
```

Out[27]:
```
'dog'
```

In [28]:
```
def filter_keywords(x):
    words = []
    for i in x:
        if i in s:
            words.append(i)
    return words
```

In [29]:
```
smd['keywords'] = smd['keywords'].apply(filter_keywords)
smd['keywords'] = smd['keywords'].apply(lambda x: [stemmer.stem(i) for i in x])
smd['keywords'] = smd['keywords'].apply(lambda x: [str.lower(i.replace(" ", "")) for i in x])
```

In [30]:
```
smd['soup'] = smd['keywords'] + smd['cast'] + smd['director'] + smd['genres']
smd['soup'] = smd['soup'].apply(lambda x: ' '.join(x))
```

In [31]:
```
count = CountVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
count_matrix = count.fit_transform(smd['soup'])
```

In [32]:
```
cosine_sim = cosine_similarity(count_matrix, count_matrix)
```

In [33]:
```
smd = smd.reset_index()
titles = smd['title']
indices = pd.Series(smd.index, index=smd['title'])
```

In [34]:
```
reader = Reader()
```

In [35]:
```
ratings = pd.read_csv('Data-Asset/archive/ratings_small.csv')
ratings.head()
```

Out[35]:

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

In [36]:
```
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

In [37]:
```
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8979  0.8931  0.8936  0.9037  0.8929  0.8962  0.0042
MAE (testset)     0.6896  0.6869  0.6915  0.6941  0.6872  0.6898  0.0027
Fit time          6.13    5.42    5.07    4.97    5.31    5.38    0.41
Test time         0.23    0.40    0.12    0.14    0.12    0.20    0.11
```

Out[37]:
```
{'test_rmse': array([0.89787345, 0.89312045, 0.89360814, 0.90373571, 0.89287223]),
 'test_mae': array([0.68955409, 0.68687974, 0.69149659, 0.69407371, 0.68722668]),
 'fit_time': (6.1342010498046875,
  5.4237635135650635,
  5.068285942077637,
  4.974091529846191,
  5.308546543121338),
 'test_time': (0.22762417793273926,
  0.40192365646362305,
  0.11768531799316406,
  0.14062762260437012,
  0.1176764965057373)}
```

In [38]:
```
trainset = data.build_full_trainset()
svd.fit(trainset)
```

Out[38]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x24285051df0>
```

In [39]:
```
ratings[ratings['userId'] == 1]
```

Out[39]:

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

5	1	1263	2.0	1260759151
6	1	1287	2.0	1260759187
7	1	1293	2.0	1260759148
8	1	1339	3.5	1260759125
9	1	1343	2.0	1260759131
10	1	1371	2.5	1260759135
11	1	1405	1.0	1260759203
12	1	1953	4.0	1260759191
13	1	2105	4.0	1260759139
14	1	2150	3.0	1260759194
15	1	2193	2.0	1260759198
16	1	2294	2.0	1260759108
17	1	2455	2.5	1260759113
18	1	2968	1.0	1260759200
19	1	3671	3.0	1260759117

In [41]:
```
svd.predict(1, 302, 3)
```

Out[41]:
```
Prediction(uid=1, iid=302, r_ui=3, est=2.6327782372018618, details={'was_impossible': False})
```

In [42]:
```
def convert_int(x):
    try:
        return int(x)
    except:
        return np.nan
```

In [43]:
```
id_map = pd.read_csv('Data-Asset/links_small.csv')[['movieId','tmdbId']]
id_map['tmdbId'] = id_map['tmdbId'].apply(convert_int)
id_map.columns = ['movieId', 'id']
id_map = id_map.merge(smd[['title','id']], on='id').set_index('title')
```

In [44]:
```
indices_map = id_map.set_index('id')
```

In [45]:
```
def hybrid(userId, title):
    idx = indices[title]
    tmdbId = id_map.loc[title]['id']
    #print(idx)
    movie_id = id_map.loc[title]['movieId']

    sim_scores = list(enumerate(cosine_sim[int(idx)]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:26]
    movie_indices = [i[0] for i in sim_scores]

    movies = smd.iloc[movie_indices][['title', 'vote_count', 'vote_average', 'year', 'id']]
    movies['est'] = movies['id'].apply(lambda x: svd.predict(userId, indices_map.loc[x]['movieId']).est)
    movies = movies.sort_values('est', ascending=False)
    return movies.head(10)
```

In [46]:
```
hybrid(1, 'Avatar')
```

Out[46]:

	title	vote_count	vote_average	year	id	est
1011	The Terminator	4208.0	7.4	1984	218	3.042874
974	Aliens	3282.0	7.7	1986	679	3.033396
2014	Fantastic Planet	140.0	7.6	1973	16306	2.958148
8658	X-Men: Days of Future Past	6155.0	7.5	2014	127585	2.910004
522	Terminator 2: Judgment Day	4274.0	7.7	1991	280	2.902521
1376	Titanic	7770.0	7.5	1997	597	2.855535
8401	Star Trek Into Darkness	4479.0	7.4	2013	54138	2.838015
1621	Darby O'Gill and the Little People	35.0	6.7	1959	18887	2.813024
831	Escape to Witch Mountain	60.0	6.5	1975	14821	2.759581
922	The Abyss	822.0	7.1	1989	2756	2.676701

In [47]:
```
hybrid(500, 'Avatar')
```

Out[47]:

	title	vote_count	vote_average	year	id	est
1011	The Terminator	4208.0	7.4	1984	218	3.368131
2014	Fantastic Planet	140.0	7.6	1973	16306	3.235502
8401	Star Trek Into Darkness	4479.0	7.4	2013	54138	3.223711
1621	Darby O'Gill and the Little People	35.0	6.7	1959	18887	3.200418
7265	Dragonball Evolution	475.0	2.9	2009	14164	3.020219
7088	Star Wars: The Clone Wars	434.0	5.8	2008	12180	2.953747
974	Aliens	3282.0	7.7	1986	679	2.952261
1668	Return from Witch Mountain	38.0	5.6	1978	14822	2.949510
1376	Titanic	7770.0	7.5	1997	597	2.914402
8724	Jupiter Ascending	2816.0	5.2	2015	76757	2.910885

In [ ]:
```

```