In [2]:
```
import pandas as pd
import scipy as sp
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import numpy as np
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
from sklearn.decomposition import TruncatedSVD

#Load Datasets
df_movies = pd.read_csv('movies.csv', usecols=['movieId', 'title'],
    dtype={'movieId': 'int32', 'title': 'str'})
df_ratings = pd.read_csv('ratings.csv', usecols=['userId', 'movieId', 'rating'],
    dtype={'userId': 'int32', 'movieId': 'int32', 'rating': 'float32'})
```

In [3]:
```
#Install surprise
```

In [74]:
```
#Show dataframe 1
df_movies

#Movie dictionary
movies_dict = dict(zip(df_movies.movieId, df_movies.title))
```

In [75]:
```
#Show dataframe 2
df_ratings.head()
```

Out[75]:

	userId	movieId	rating
0	1	1	4.0
1	1	3	4.0
2	1	6	4.0
3	1	47	5.0
4	1	50	5.0

In [76]:
```
#Merge movie and rating
rating_with_title = pd.merge(df_ratings, df_movies, on='movieId')
rating_with_title.head()
```

Out[76]:

	userId	movieId	rating	title
0	1	1	4.0	Toy Story (1995)
1	5	1	4.0	Toy Story (1995)
2	7	1	4.5	Toy Story (1995)
3	15	1	2.5	Toy Story (1995)
4	17	1	4.5	Toy Story (1995)

In [7]:
```
#Check sparsity of new matrix
numratings = len(rating_with_title['rating'])
numusers = len(rating_with_title['userId'].unique())
```

```
numitems = len(rating_with_title['movieId'].unique())

sparse = 1 - (numratings / (numusers*numitems))
sparse
#The dataset is very sparse with sparsity of 0.9830003169443864
```

Out[7]:
```
0.9830003169443864
```

In [77]:
```
#Filter dataframe

#Filter out users who have rated less than 200 movies
rating_with_title = rating_with_title.groupby('userId').filter(lambda x: len(x)>200)
#Filter out movies who have rated less than 10 movies
rating_with_title = rating_with_title.groupby('movieId').filter(lambda x: len(x)>10)
rating_with_title.head()
```

Out[77]:

	userId	movieId	rating	title
0	1	1	4.0	Toy Story (1995)
5	18	1	3.5	Toy Story (1995)
6	19	1	4.0	Toy Story (1995)
7	21	1	3.5	Toy Story (1995)
15	45	1	4.0	Toy Story (1995)

In [9]:
```
#Check sparsity again
numratings = len(rating_with_title['rating'])
numusers = len(rating_with_title['userId'].unique())
numitems = len(rating_with_title['movieId'].unique())

sparse = 1 - (numratings / (numusers*numitems))
sparse
#The sparsity of dataset is a lot better with only 0.738161578367764
```

Out[9]:
```
0.7931695867508024
```

In [14]:
```
#Benchmark, getting all Rmse values from different algorithms for comparison
from surprise import SVD, SVDpp, BaselineOnly
from surprise.prediction_algorithms import KNNWithMeans, KNNBasic, KNNBaseline
from surprise import Dataset, Reader
from surprise import accuracy
from surprise.model_selection import cross_validate, train_test_split, GridSearchCV
#Benchmark
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(rating_with_title[['userId', 'movieId', 'rating']], reader)
benchmark = []
# Iterate over all algorithms
for algorithm in [SVD(), SVDpp(), KNNBaseline(), KNNBasic(), KNNWithMeans(), BaselineOnly
    results = cross_validate(algorithm, data, measures=['RMSE'], cv=3, verbose=False)
    tmp = pd.DataFrame.from_dict(results).mean(axis=0)
    tmp = tmp.append(pd.Series([str(algorithm).split(' ')[0].split('.')[-1]], index=['Algo
    benchmark.append(tmp)
```

```
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
```

```
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
```

In [15]:

```
#Listing out all benchmark
pd.DataFrame(benchmark)
#SVDpp has only 0.809325 of rmse, but with incredibly long fit_time and test_time
```

Out[15]:

	test_rmse	fit_time	test_time	Algorithm
0	0.827064	1.467875	0.092519	SVD
1	0.810085	118.197996	3.348141	SVDpp
2	0.815769	0.084170	0.798312	KNNBaseline
3	0.884730	0.038343	0.551294	KNNBasic
4	0.823896	0.046329	0.630625	KNNWithMeans
5	0.823489	0.050513	0.069136	BaselineOnly

In [68]:

```
#SVD
# get the list of the movie ids
combined_movies_data = rating_with_title[['userId','movieId', 'rating']]
unique_ids = combined_movies_data['movieId'].unique()

#Apply grid search to do a brute-force search for the hyper-parameters for the SVD algori
def find_score_svd(data):
    from surprise.model_selection import RandomizedSearchCV
    param_grid = {'n_epochs': [5, 10, 20, 30], 'lr_all': [.0025, .005, .001, .01]}
    grid_search = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)
    grid_search.fit(data)
    algo = grid_search.best_estimator['rmse']
    print(grid_search.best_score['rmse'])
    print(grid_search.best_params['rmse'])
    cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

data = Dataset.load_from_df(rating_with_title[['userId', 'movieId', 'rating']], reader)
find_score_svd(data)
#{'n_epochs': 10, 'lr_all': 0.01} These are the hyper-parameters that works well
```

```
0.8247529409205835
{'n_epochs': 10, 'lr_all': 0.01}
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8106  0.8323  0.8080  0.8246  0.8152  0.8181  0.0091
```

```
MAE (testset)     0.6221  0.6371  0.6224  0.6310  0.6258  0.6277  0.0057
Fit time          0.90    0.94    0.94    0.90    0.91    0.92    0.02
Test time         0.10    0.05    0.05    0.05    0.05    0.06    0.02
```

In [81]:

```
#Apply the hyper-parameters into the function

def svd(userID):
    reader = Reader(rating_scale=(1, 5))
    data = Dataset.load_from_df(rating_with_title[['userId', 'movieId', 'rating']], reader
    trainset, testset = train_test_split(data, test_size=.25)
    algo = SVD(n_epochs=10, lr_all= 0.01)
    algo.fit(trainset)
    predictions = algo.test(testset)

    performance = cross_validate(algo, data, measures=['RMSE', 'MAE'],cv = 5, verbose=Fals
    recommandation_list = []

    # get the list of the ids that the userid 1001 has rated
    id_user = combined_movies_data.loc[combined_movies_data['userId'] == userID, 'movieId'
    # remove the rated movies for the recommendations
    movies_to_predict = np.setdiff1d(unique_ids, id_user)

    recommandation_list = []
    for iid in movies_to_predict:
        recommandation_list.append((iid, algo.predict(uid = userID, iid = iid).est))

    df_final = pd.DataFrame(recommandation_list, columns=['movieId', 'predictions']).sort_
    df_final['title'] = df_final['movieId'].map(movies_dict)
    return df_final, predictions, performance

def askUser():
    user = int(input("Enter user id: "))
    df_final, predictions, performance = svd(user)
    display(df_final)
    print('---------------SVD RMSE result-------------')
    accuracy.rmse(predictions)

askUser()
```

```
Enter user id: 1
```

	movieId	predictions	title
191	904	5.0	Rear Window (1954)
78	318	5.0	Shawshank Redemption, The (1994)
1029	7022	5.0	Battle Royale (Batoru rowaiaru) (2000)
184	858	5.0	Godfather, The (1972)
1347	79132	5.0	Inception (2010)

```
---------------SVD RMSE result-------------
RMSE: 0.8137
```

In [ ]: