Chapter 9: Recommender Systems

9.2 Association Rules

9.2.2 Applying Association Rules

9.2.2.1 Loading the dataset

```
all_txns = []

#open the file
with open('groceries.csv') as f:
    #read each line
    content = f.readlines()
    #Remove white space from the beginning and end of the line
    txns = [x.strip() for x in content]
    # Iterate through each line and create a list of transactions
    for each_txn in txns:
        #Each transaction will contain a list of item in the transaction
        all_txns.append( each_txn.split(',') )
```

```
all_txns[0:5]
```

```
[['citrus fruit', 'semi-finished bread', 'margarine', 'ready soup
s'],
 ['tropical fruit', 'yogurt', 'coffee'],
 ['whole milk'],
 ['pip fruit', 'yogurt', 'cream cheese ', 'meat spreads'],
 ['other vegetables',
  'whole milk',
  'condensed milk',
  'long life bakery product']]
```

9.2.2.2 Encoding the transactions

```
# Import all required libraries
import pandas as pd
import numpy as np
from mlxtend.preprocessing import OnehotTransactions
from mlxtend.frequent_patterns import apriori, association_rules
```

```
/Users/manaranjan/anaconda/lib/python3.5/importlib/_bootstrap.py:22
2: RuntimeWarning: numpy.dtype size changed, may indicate binary inc
ompatibility. Expected 96, got 88
  return f(*args, **kwds)
/Users/manaranjan/anaconda/lib/python3.5/importlib/_bootstrap.py:22
2: RuntimeWarning: numpy.dtype size changed, may indicate binary inc
ompatibility. Expected 96, got 88
  return f(*args, **kwds)
```

```
# Initialize OnehotTransactions
one_hot_encoding = OnehotTransactions()
# Transform the data into one-hot-encoding format
one_hot_txns = one_hot_encoding.fit(all_txns).transform(all_txns)
# Conver the matrix into the dataframe.
one_hot_txns_df = pd.DataFrame(one_hot_txns,
                               columns=one_hot_encoding.columns_)
```

```
one_hot_txns_df.iloc[5:10, 10:20]
```

	berries	beverages	bottled beer	bottled water	brandy	brown bread	butter	butter milk	cake bar	candles
5	0	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

```
one_hot_txns_df.shape
```

```
(9835, 171)
```

9.2.2.3 Generating Rules

```
len(one_hot_txns_df.columns)
```

```
171
```

```
frequent_itemsets = apriori(one_hot_txns_df,
                            min_support=0.02,
                            use_colnames=True)
```

```
frequent_itemsets.sample(10, random_state = 90)
```

	support	itemsets
60	0.020437	[bottled beer, whole milk]
52	0.033859	[sugar]
89	0.035892	[other vegetables, tropical fruit]
105	0.021047	[root vegetables, tropical fruit]
88	0.032740	[other vegetables, soda]
16	0.058058	[coffee]
111	0.024504	[shopping bags, whole milk]
36	0.079817	[newspapers]
119	0.056024	[whole milk, yogurt]
55	0.071683	[whipped/sour cream]

```
rules = association_rules(frequent_itemsets, # itemsets
                          metric="lift",  # lift
                          min_threshold=1)
```

```
rules.sample(5)
```

	antecedants	consequents	support	confidence	lift
101	(tropical fruit)	(pip fruit)	0.104931	0.194767	2.574648
100	(pip fruit)	(tropical fruit)	0.075648	0.270161	2.574648
108	(soda)	(yogurt)	0.174377	0.156851	1.124368
26	(rolls/buns)	(yogurt)	0.183935	0.186844	1.339363
25	(whole milk)	(fruit/vegetable juice)	0.255516	0.104258	1.442160

9.2.1.4 Top 10 Rules

```
rules.sort_values('confidence',
                  ascending = False)[0:10]
```

	antecedants	consequents	support	confidence	lift
29	(other vegetables, yogurt)	(whole milk)	0.043416	0.512881	2.007235
56	(butter)	(whole milk)	0.055414	0.497248	1.946053
52	(curd)	(whole milk)	0.053279	0.490458	1.919481
38	(root vegetables, other vegetables)	(whole milk)	0.047382	0.489270	1.914833
39	(root vegetables, whole milk)	(other vegetables)	0.048907	0.474012	2.449770
48	(domestic eggs)	(whole milk)	0.063447	0.472756	1.850203
85	(whipped/sour cream)	(whole milk)	0.071683	0.449645	1.759754
34	(root vegetables)	(whole milk)	0.108998	0.448694	1.756031
18	(root vegetables)	(other vegetables)	0.108998	0.434701	2.246605
47	(frozen vegetables)	(whole milk)	0.048094	0.424947	1.663094

9.3 Collaborative Filtering

9.3.2 User Based Similarity

9.3.2.1 Loading the dataset

```
rating_df = pd.read_csv( "ml-latest-small/ratings.csv" )
```

```
rating_df.head(5)
```

	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

```
rating_df.drop( 'timestamp', axis = 1, inplace = True )
```

```
len( rating_df.userId.unique() )
```

671

```
len( rating_df.movieId.unique() )
```

9066

```
user_movies_df = rating_df.pivot( index='userId',
                                  columns='movieId',
                                  values = "rating" ).reset_index(drop=True)
user_movies_df.index = rating_df.userId.unique()
```

```
user_movies_df.iloc[0:5, 0:15]
```

movieId	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN
5	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
user_movies_df.fillna( 0, inplace = True )
user_movies_df.iloc[0:5, 0:10]
```

movieId	1	2	3	4	5	6	7	8	9	10
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0
5	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

9.3.2.2 Calculating Cosine Similarity between users

```
from sklearn.metrics import pairwise_distances
from scipy.spatial.distance import cosine, correlation


user_sim = 1 - pairwise_distances( user_movies_df.values, metric="cosine" )
#Store the results in a dataframe
user_sim_df = pd.DataFrame( user_sim )
# set the index and column names to user ids (0 to 671)
user_sim_df.index = rating_df.userId.unique()
user_sim_df.columns = rating_df.userId.unique()
```

```
/Users/manaranjan/anaconda/lib/python3.5/importlib/_bootstrap.py:22
2: RuntimeWarning: numpy.dtype size changed, may indicate binary inc
ompatibility. Expected 96, got 88
  return f(*args, **kwds)
```

```
user_sim_df.iloc[0:5, 0:5]
```

	1	2	3	4	5
1	1.000000	0.000000	0.000000	0.074482	0.016818
2	0.000000	1.000000	0.124295	0.118821	0.103646
3	0.000000	0.124295	1.000000	0.081640	0.151531
4	0.074482	0.118821	0.081640	1.000000	0.130649
5	0.016818	0.103646	0.151531	0.130649	1.000000

```
user_sim_df.shape
```

```
(671, 671)
```

```
np.fill_diagonal( user_sim, 0 )
user_sim_df.iloc[0:5, 0:5]
```

	1	2	3	4	5
1	0.000000	0.000000	0.000000	0.074482	0.016818
2	0.000000	0.000000	0.124295	0.118821	0.103646
3	0.000000	0.124295	0.000000	0.081640	0.151531
4	0.074482	0.118821	0.081640	0.000000	0.130649
5	0.016818	0.103646	0.151531	0.130649	0.000000

9.3.2.3 Filtering Similar User

```
user_sim_df.idxmax(axis=1)[0:5]
```

```
1    325
2    338
3    379
4    518
5    313
dtype: int64
```

```
user_sim_df.iloc[1:2, 330:340]
```

	331	332	333	334	335	336	337	338	339
2	0.030344	0.002368	0.052731	0.047094	0.0	0.053044	0.05287	0.581528	0.093863

9.3.2.4 Loading the movies dataset

```
movies_df = pd.read_csv( "ml-latest-small/movies.csv" )
```

```
movies_df[0:5]
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

```
movies_df.drop( 'genres', axis = 1, inplace = True )
```

9.3.2.5 Finding common movies of similar users

```
def get_user_similar_movies( user1, user2 ):
    # Inner join between movies watched between two users will give the common m
ovies watched.
    common_movies = rating_df[rating_df.userId == user1].merge(
        rating_df[rating_df.userId == user2],
        on = "movieId",
        how = "inner" )
    # join the above result set with movies details
    return common_movies.merge( movies_df, on = 'movieId' )
```

```
common_movies = get_user_similar_movies( 2, 338 )
```

```
common_movies[(common_movies.rating_x >= 4.0) &
              ((common_movies.rating_y >= 4.0))]
```

	userId_x	movieId	rating_x	userId_y	rating_y	title
0	2	17	5.0	338	4.0	Sense and Sensibility (1995)
2	2	47	4.0	338	4.0	Seven (a.k.a. Se7en) (1995)
5	2	150	5.0	338	4.0	Apollo 13 (1995)
28	2	508	4.0	338	4.0	Philadelphia (1993)
29	2	509	4.0	338	4.0	Piano, The (1993)
31	2	527	4.0	338	5.0	Schindler's List (1993)
34	2	589	5.0	338	5.0	Terminator 2: Judgment Day (1991)

```
common_movies = get_user_similar_movies( 2, 332 )
common_movies
```

	userId_x	movieId	rating_x	userId_y	rating_y	title
0	2	552	3.0	332	0.5	Three Musketeers, The (1993)

9.3.3 Item based similarity

9.3.3.1 Calculating Cosine Similarity between movies

```
rating_mat = rating_df.pivot( index='movieId',
                              columns='userId',
                              values = "rating" ).reset_index(drop=True)
# fill all NaNs with 0
rating_mat.fillna( 0, inplace = True )
# Find the correlation between movies
movie_sim = 1 - pairwise_distances( rating_mat.values,
                                    metric="correlation" )
# Fill the diagonal with 0, as it reprereesent the auto-correlation of movies
movie_sim_df = pd.DataFrame( movie_sim )
```

```
movie_sim_df.iloc[0:5, 0:5]
```

	0	1	2	3	4
0	1.000000	0.223742	0.183266	0.071055	0.105076
1	0.223742	1.000000	0.123790	0.125014	0.193144
2	0.183266	0.123790	1.000000	0.147771	0.317911
3	0.071055	0.125014	0.147771	1.000000	0.150562
4	0.105076	0.193144	0.317911	0.150562	1.000000

```
movie_sim_df.shape
```

```
(9066, 9066)
```

9.3.3.2 Finding most similar movies

```
def get_similar_movies( movieid, topN = 5 ):
    movieidx = movies_df[movies_df.movieId == movieid].index[0]
    movies_df['similarity'] = movie_sim_df.iloc[movieidx]
    top_n = movies_df.sort_values( ["similarity"], ascending = False )[0:topN]
    return top_n
```

Finding similar movies to Godfather

```
movies_df[movies_df.movieId == 858]
```

	movieId	title
695	858	Godfather, The (1972)

```
get_similar_movies(858)
```

	movieId	title	similarity
695	858	Godfather, The (1972)	1.000000
977	1221	Godfather: Part II, The (1974)	0.709246
969	1213	Goodfellas (1990)	0.509372
951	1193	One Flew Over the Cuckoo's Nest (1975)	0.430101
1744	2194	Untouchables, The (1987)	0.418966

Finding similar movies to Dumb & Dumber

```
movies_df[movies_df.movieId == 231]
```

	movieId	title	similarity
203	231	Dumb & Dumber (Dumb and Dumber) (1994)	0.054116

```
get_similar_movies(231)
```

	movieId	title	similarity
203	231	Dumb & Dumber (Dumb and Dumber) (1994)	1.000000
309	344	Ace Ventura: Pet Detective (1994)	0.635735
18	19	Ace Ventura: When Nature Calls (1995)	0.509839
447	500	Mrs. Doubtfire (1993)	0.485764
331	367	Mask, The (1994)	0.461103

9.4 Using Surprise Library

```
from surprise import Dataset, Reader, KNNBasic, evaluate, accuracy
```

```
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(rating_df[['userId',
                                       'movieId',
                                       'rating']], reader=reader)
```

9.4.1 Create user based similiarity algorithm

```
item_based_cosine_sim = {'name': 'pearson',
                         'user_based': True}

knn = KNNBasic(k= 20,
               min_k = 5,
               sim_options = item_based_cosine_sim)
```

```
from surprise.model_selection import cross_validate

cv_results = cross_validate(knn,
                            data,
                            measures=['RMSE'],
                            cv=5,
                            verbose=False)
```

```
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Computing the pearson similarity matrix...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
```

```
np.mean( cv_results.get('test_rmse') )
```

```
0.9909383567908019
```

9.4.2 Finding Best Model

```
from surprise.model_selection.search import GridSearchCV
```

```
param_grid = {'k': [10, 20],
              'sim_options': {'name': ['cosine', 'pearson'],
                              'user_based': [True, False]}
              }

grid_cv = GridSearchCV(KNNBasic,
                       param_grid,
                       measures=['rmse'],
                       cv=5,
                       refit=True)

grid_cv.fit(data)
```

```
# best RMSE score
print(grid_cv.best_score['rmse'])

# combination of parameters that gave the best RMSE score
print(grid_cv.best_params['rmse'])
```

```
0.99700876328297
{'k': 20, 'sim_options': {'name': 'cosine', 'user_based': True}}
```

```
results_df = pd.DataFrame.from_dict(grid_cv.cv_results)
results_df[['param_k', 'param_sim_options', 'mean_test_rmse', 'rank_test_rmse']]
```

	param_k	param_sim_options	mean_test_rmse	rank_test_rmse
0	10	{'name': 'cosine', 'user_based': True}	1.009654	4
1	10	{'name': 'cosine', 'user_based': False}	1.043961	8
2	10	{'name': 'pearson', 'user_based': True}	1.013507	6
3	10	{'name': 'pearson', 'user_based': False}	1.031335	7
4	20	{'name': 'cosine', 'user_based': True}	0.997009	1
5	20	{'name': 'cosine', 'user_based': False}	1.012553	5
6	20	{'name': 'pearson', 'user_based': True}	1.002168	2
7	20	{'name': 'pearson', 'user_based': False}	1.004774	3

9.4.3 Making Predictions

```
grid_cv.predict( 1, 2 )
```

```
Prediction(uid=1, iid=2, r_ui=None, est=3.1484553902974177, details=
{'was_impossible': False, 'actual_k': 13})
```

9.5 Matrix Factorization

```
from surprise import SVD

# Use 10 factors for building the model
svd = SVD( n_factors = 5 )
```

```
cv_results = cross_validate(svd,
                            data,
                            measures=['RMSE'],
                            cv=5,
                            verbose=True)
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    St
d
RMSE (testset)    0.8926  0.8956  0.8887  0.8880  0.8870  0.8904  0.
0032
Fit time          2.22    2.35    2.35    2.43    2.37    2.34    0.
07
Test time         0.26    0.23    0.24    0.21    0.21    0.23    0.
02
```