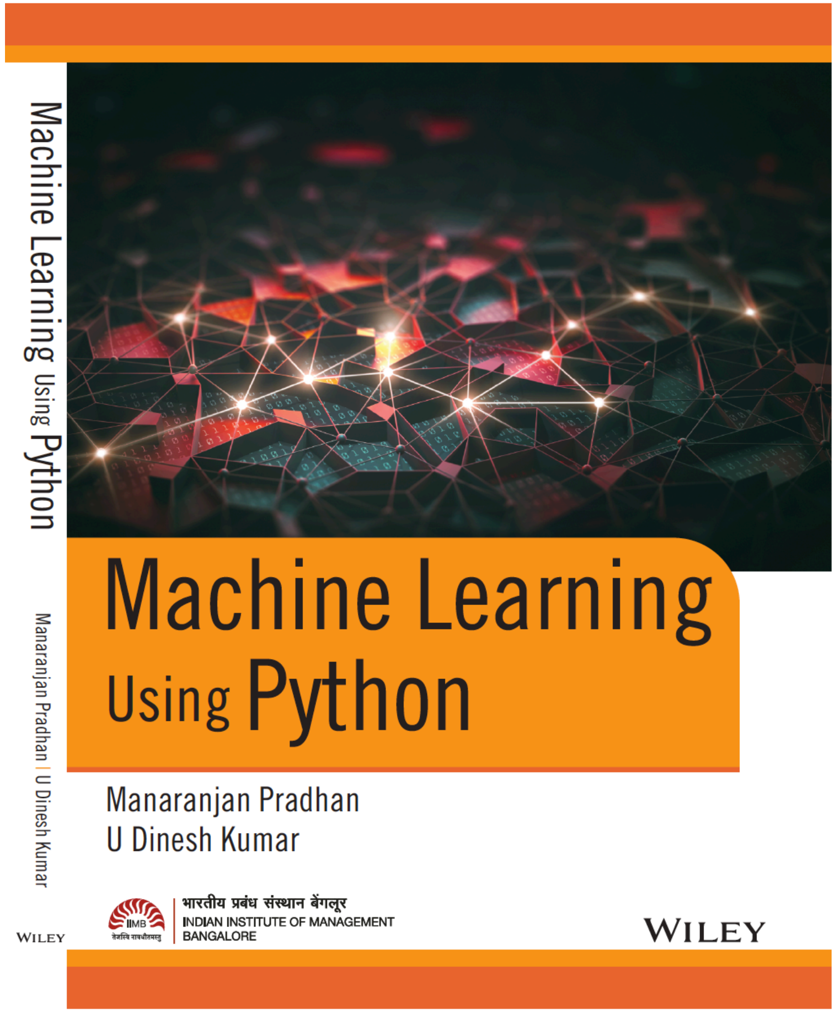



# Chapter 09: Recommender Systems

# Learning Objectives

- Understand recommender systems and their business applications.
- Learn about the datasets and algorithm required for building recommendation systems.
- Learn recommender system development techniques such as association rules and collaborative filtering.
- Learn how to build and evaluate recommendation systems using Python libraries.

# OVERVIEW

Three algorithms that are widely used for building recommendation systems:

- **Association Rules**
- **Collaborative Filtering**
- **Matrix Factorization**

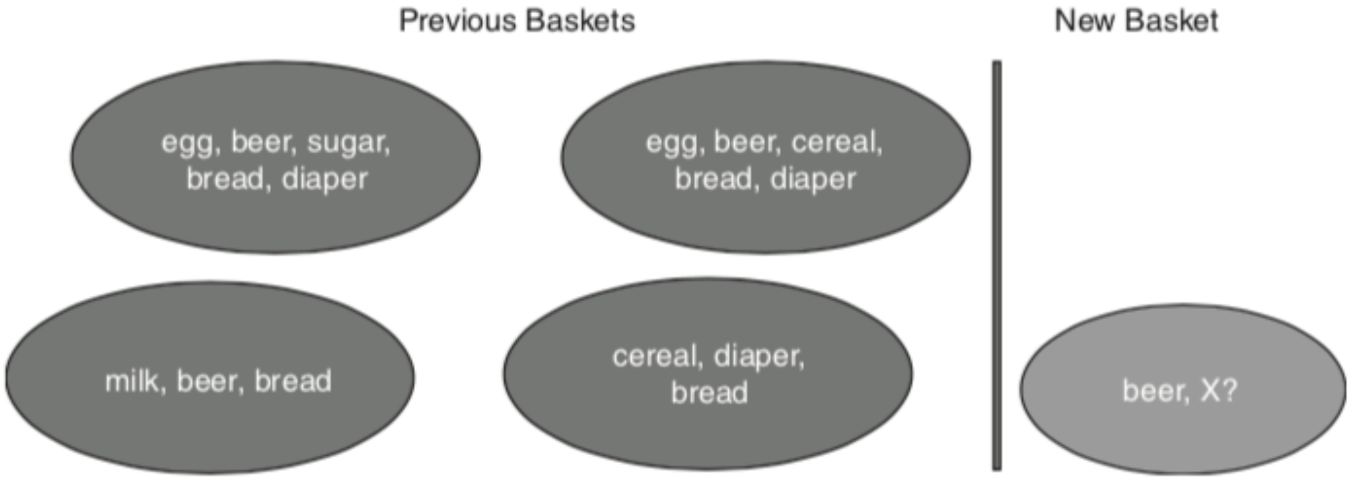
Datasets:

- groceries.csv: contains transactions of a grocery store and can be downloaded from http://www.sci.csueastbay.edu/~esuess/classes/Statistics_6620/Presentations/ml13/groceries.csv.
- Movie Lens: contains ratings and tag applications movies. The dataset can be downloaded from the link https://grouplens.org/datasets/movielens/.

# ASSOCIATION RULES (ASSOCIATION RULE MINING)

1. Basket 1: egg, beer, sugar, bread, diaper
2. Basket 2: egg, beer, cereal, bread, diaper
3. Basket 3: milk, beer, bread
4. Basket 4: cereal, diaper, bread

$\{\text{diapers}\} \rightarrow \{\text{beer}\}$



# METRICS - SUPPORT

Assume that $X$ and $Y$ are items being considered. Let

1. $N$ be the total number of baskets.
2. $N_{XY}$ represent the number of baskets in which $X$ and $Y$ appear together.
3. $N_X$ represent the number of baskets in which $X$ appears.
4. $N_Y$ represent the number of baskets in which $Y$ appears.

Then the support between $X$ and $Y$, Support($X$, $Y$), is given by

$$\text{Support}(X,\ Y) = \frac{N_{XY}}{N}$$

# METRICS - CONFIDENCE

Confidence measures the proportion of the transactions that contain $X$, which also contain $Y$. $X$ is called antecedent and $Y$ is called consequent. Confidence can be calculated using the following formula:

$$\text{Confidence}(X \rightarrow Y) = P(Y \mid X) = \frac{N_{XY}}{N_X}$$

where $P(Y|X)$ is the conditional probability of $Y$ given $X$.

# METRICS - LIFT

Lift is calculated using the following formula:

$$\text{Lift} = \frac{\text{Support}(X,Y)}{\text{Support}(X) \times \text{Support}(Y)} = \frac{N_{XY}}{N_X N_Y}$$

Lift can be interpreted as the degree of association between two items.

- Lift value 1 – The items are independent (no association)
- Lift value of less than 1 – The products are substitution (purchase one product will decrease the probability of purchase of the other product)
- Lift value of greater than 1 - purchase of Product *X* will increase the probability of purchase of Product *Y*

# APPLYING ASSOCIATION RULES

- The code opens the file *groceries.csv*
- Reads all the lines from the file
- Removes leading or trailing white spaces from each line
- Splits each line by a comma to extract items
- Stores the items in each line in a list

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

The output is shown below:

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

# ENCODING THE TRANSACTIONS

```
import pandas as pd import numpy as np
from mlxtend.preprocessing import OnehotTransactions
from mlxtend.frequent_patterns import apriori, association_rules

# 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)

# Convert the matrix into the dataframe. one_hot_txns_df =
pd.DataFrame(one_hot_txns, columns=one_hot_encoding.columns_)
```

| | 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 |

# GENERATING ASSOCIATION RULES

If we have 171 items across all transactions, for itemset containing 2 items in each set, the total number of item sets will be $^{171}C_2$ **14535**

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

| | 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] |

# ASSOCIATION RULES

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

| | Antecedants | Consequents | Support | Confidence | Lift |
|---|---|---|---|---|---|
| 7 | (soda) | (rolls/buns) | 0.174377 | 0.219825 | 1.195124 |
| 55 | (yogurt) | (bottled water) | 0.139502 | 0.164723 | 1.490387 |
| 74 | (soda) | (yogurt) | 0.174377 | 0.156851 | 1.124368 |
| 89 | (root vegetables) | (whole milk) | 0.108998 | 0.448694 | 1.756031 |
| 59 | (citrus fruit) | (yogurt) | 0.082766 | 0.261671 | 1.875752 |

# TOP 10 RULES – SORT BY CONFIDENCE

| | Antecedants | Consequents | Support | Confidence | Lift |
|---|---|---|---|---|---|
| 42 | (yogurt, other vegetables) | (whole milk) | 0.043416 | 0.512881 | 2.007235 |
| 48 | (butter) | (whole milk) | 0.055414 | 0.497248 | 1.946053 |
| 120 | (curd) | (whole milk) | 0.053279 | 0.490458 | 1.919481 |
| 80 | (other vegetables, root vegetables) | (whole milk) | 0.047382 | 0.489270 | 1.914833 |
| 78 | (root vegetables, whole milk) | (other vegetables) | 0.048907 | 0.474012 | 2.449770 |
| 27 | (domestic eggs) | (whole milk) | 0.063447 | 0.472756 | 1.850203 |
| 0 | (whipped/sour cream) | (whole milk) | 0.071683 | 0.449645 | 1.759754 |
| 89 | (root vegetables) | (whole milk) | 0.108998 | 0.448694 | 1.756031 |
| 92 | (root vegetables) | (other vegetables) | 0.108998 | 0.434701 | 2.246605 |
| 24 | (frozen vegetables) | (whole milk) | 0.048094 | 0.424947 | 1.663094 |

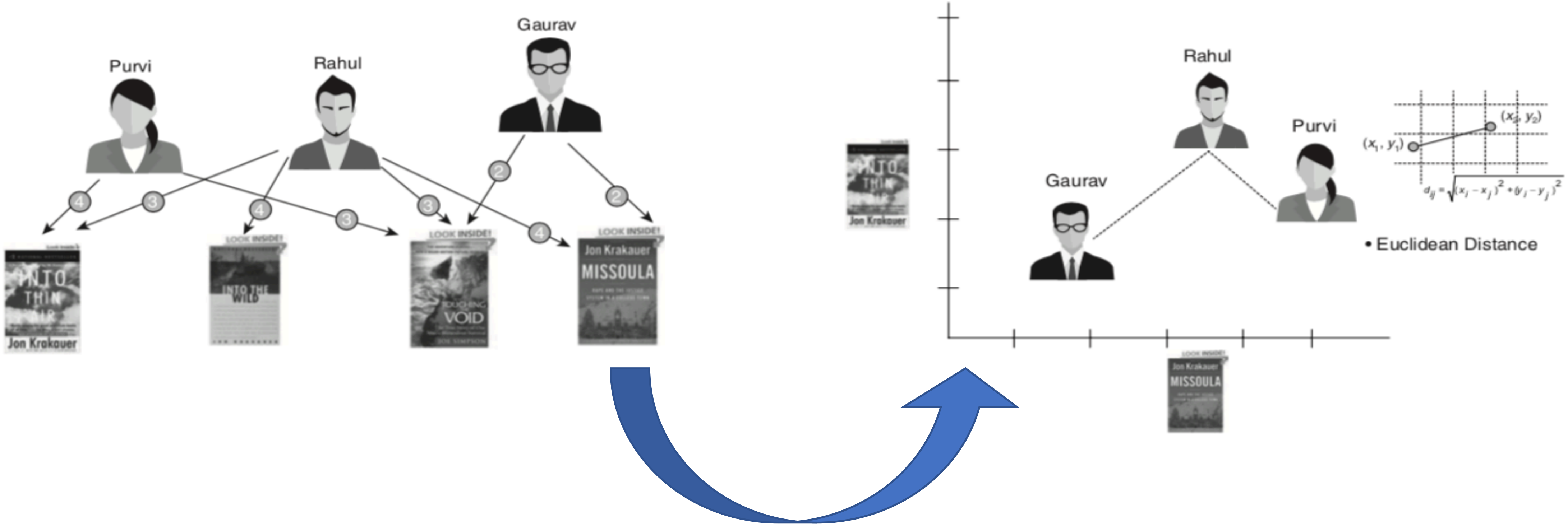
# PROS AND CONS OF ASSOCIATION RULE MINING

- PROS:
  - Transactions data, which is used for generating rules, is always available and mostly clean
  - The rules generated are simple and can be interpreted

- CONS:
  - Association rules do not take the preference or ratings given by customers into account

# COLLABORATIVE FILTERING

## How to Find Similarity Between Users?



# USER-BASED SIMILARITY

**Dataset**: https://grouplens.org/datasets/movielens/

Features in Dataset:

- userId
- movieId
- Rating
- timestamp

| | 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 |

# USER-BASED SIMILARITY (Contd.)

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

user_movies_df.index=rating_df.userId.unique()
```

| movieId | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | 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 | NaN |
| 3 | NaN | 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 | NaN |
| 5 | NaN | NaN | 4.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

```
user_movies_df.fillna( 0, inplace = True )
```

# 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” )
user_sim_df = pd.DataFrame( user_sim )

user_sim_df.index = rating_df.userId.unique() user_sim_df.columns =
rating_df.userId.unique()
```

| | 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 |

| | 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 |

# FILTERING SIMILAR USERS

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

```
1 325
2 338
3 379
4 518
5 313
```

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

| | 331 | 332 | 333 | 334 | 335 | 336 | 337 | 338 | 339 | 340 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.030344 | 0.002368 | 0.052731 | 0.047094 | 0.0 | 0.053044 | 0.05287 | 0.581528 | 0.093863 | 0.081814 |

# LOADING THE MOVIES DATASET

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

| | 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 |

# COMMON MOVIES OF SIMILAR USERS

```
def get_user_similar_movies( user1, user2 ):
# Inner join between movies watched between two users will give
# the common movies 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’ )
```

| | 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) |

# USERS WITH DISSIMILAR BEHAVIOR

```
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) |

<u>Challenges with User-Based Similarity</u>

- ***cold start*** problem in recommender systems

# ITEM BASED SIMILARITY

## 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 reprerersents the auto-
correlation # of movies movie_sim_df = pd.DataFrame( movie_sim )
```

| | 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 |

# MOST SIMILAR MOVIES

```
def get_similar_movies( movieid, topN = 5 ):
# Get the index of the movie record in movies_df
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
```

| | movieId | title | similarity |
|---|---|---|---|
| 695 | 858 | Godfather, The (1972) | 1.0 |

| | 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 |

# 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)
```

# USER-BASED SIMILARITY ALGORITHM

## Set coefficient and similarity parameters for building model 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**)

```
np.mean(cv_results.get(‘test_rmse’))
```

```
0.9909387452695102
```

# FINDING THE 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)

print(grid_cv.best_params['rmse'])
```

```
0.9963783863084851
```

| | param_k | param_sim_options | mean_test_rmse | rank_test_rmse |
|---|---|---|---|---|
| 0 | 10 | {'name': 'cosine', 'user_based': True} | 1.009724 | 4 |
| 1 | 20 | {'name': 'cosine', 'user_based': True} | 0.996378 | 1 |
| 2 | 10 | {'name': 'cosine', 'user_based': False} | 1.048802 | 8 |
| 3 | 20 | {'name': 'cosine', 'user_based': False} | 1.015225 | 6 |
| 4 | 10 | {'name': 'pearson', 'user_based': True} | 1.012283 | 5 |
| 5 | 20 | {'name': 'pearson', 'user_based': True} | 1.000766 | 2 |
| 6 | 10 | {'name': 'pearson', 'user_based': False} | 1.030900 | 7 |
| 7 | 20 | {'name': 'pearson', 'user_based': False} | 1.004205 | 3 |

# MATRIX FACTORIZATION

Users–Movies Rating Matrix

| | | Movies | | | | |
|---|---|---|---|---|---|---|
| | | M1 | M2 | M3 | M4 | M5 |
| Users | U1 | 3 | 4 | 2 | 5 | 1 |
| | U2 | 2 | 4 | 1 | 2 | 4 |
| | U3 | 3 | 3 | 5 | 2 | 2 |

Users–Factors Matrix

| | | Factors | | |
|---|---|---|---|---|
| | | F1 | F2 | F3 |
| Users | U1 | 0.73 | 3.22 | 0 |
| | U2 | 0 | 1.57 | 2.53 |
| | U3 | 1.62 | 0 | 1.44 |

Factors–Movies Matrix

| | | Movies | | | | |
|---|---|---|---|---|---|---|
| | | M1 | M2 | M3 | M4 | M5 |
| Factors | F1 | 1.47 | 1 | 2.73 | 1.73 | 0 |
| | F2 | 0.6 | 1.01 | 0 | 1.27 | 0.31 |
| | F3 | 0.42 | 0.95 | 0.39 | 0 | 1.39 |

# MATRIX FACTORIZATION (Contd.)

```
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    Std
RMSE (testset)    0.8904  0.8917  0.8983  0.8799  0.8926  0.8906  0.0060
Fit time          1.91    1.99    1.96    1.87    1.84    1.91    0.06
Test time         0.21    0.18    0.18    0.17    0.18    0.18    0.01
```

# Thank You!