DB Mining and Recommendation Using Python

406.426B 데이터관리와 분석

2024.10.30

강명오

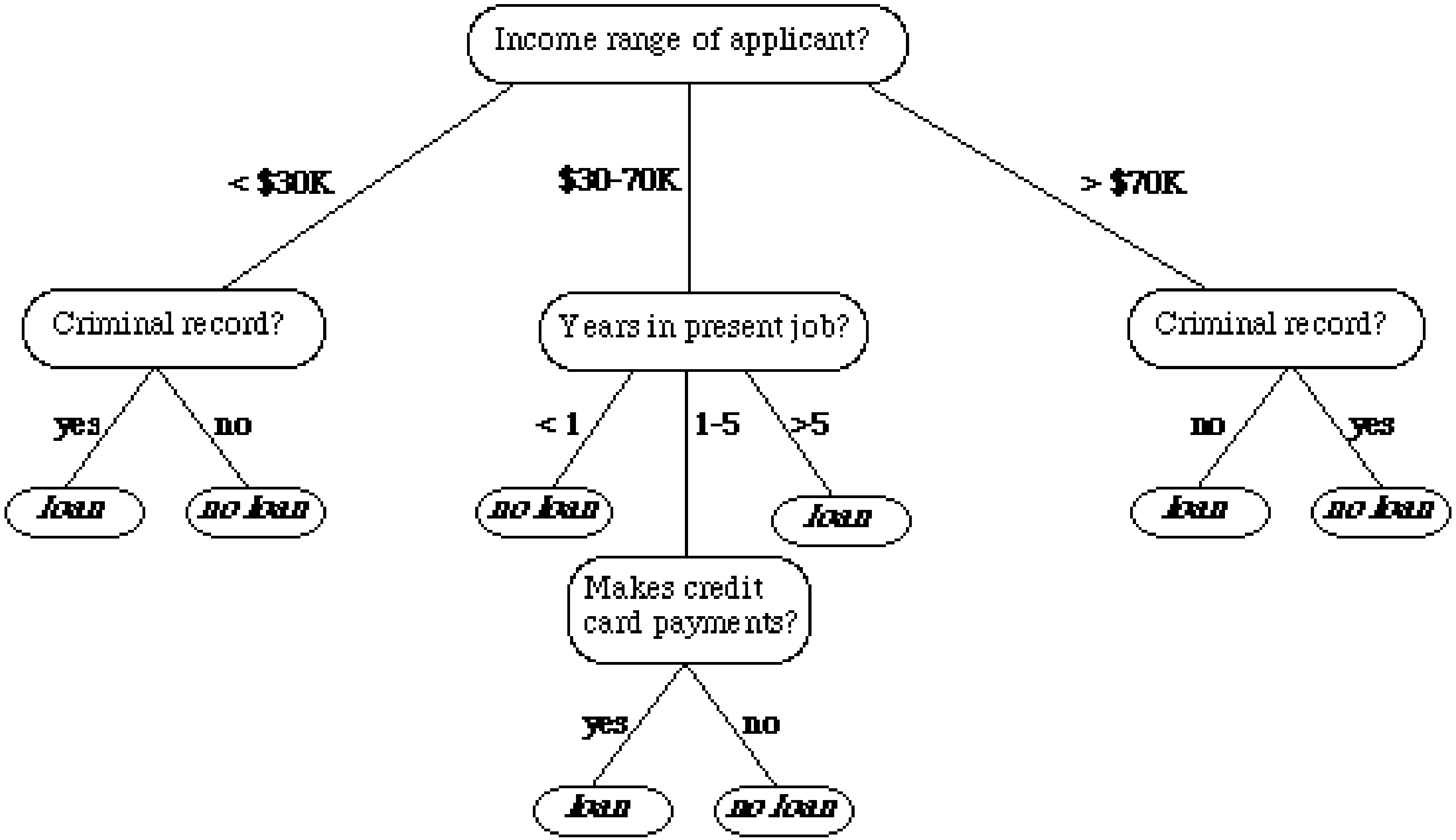
mo970610@snu.ac.kr

서울대학교 산업공학과

DB Mining and Recommendation Using Python

PART 1. DB MINING; DECISION TREE

Decision Tree



Scikit-Learn

- Scikit Learn
  - classification, clustering 등 다양한 분석 모델을 만드는 라이브러리
  - https://scikit-learn.org
  - 다양한 기능, 함수에 대한 documentation이 잘 정리되어 있음

- 설치 및 사용
  - pip install scikit-learn 또는 conda install scikit-learn
  - import sklearn
    - 혹은 사용하고자 하는 특정 모듈만 import
    - from sklearn import tree

Decision Tree with Scikit-Learn

- DecisionTreeClassifier 생성자의 parameters
  - https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
  - criterion: 의사결정 나무를 그릴 때 사용할 척도, 'gini' 혹은 'entropy'
  - 그 외에 의사결정 나무를 어디까지 그릴 지에 대한 다양한 parameters
    - max_depth 등
  - 실행 시 random state 고정 추천 (결과 재현을 위함)

- fit()
  - DecisionTreeClassifier를 생성한 후, 데이터로 의사결정 나무를 학습함
  - X: 의사결정 나무의 node(features)
  - y: 의사결정 나무의 target
    - 분류 문제를 학습하는 것이기에 2개 이상의 class를 입력해야 함 (mult-class 역시 가능)

Decision Tree with Scikit-Learn

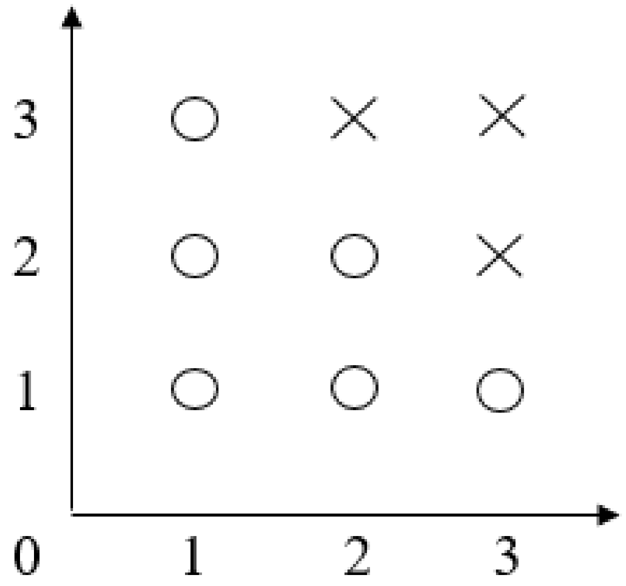
```python
from sklearn import tree
classes = []
features = []

example= [ [1,  [1,1]],
           [1,  [1,2]],
           [1,  [1,3]],
           [1,  [2,1]],
           [1,  [2,2]],
           [1,  [3,1]],
           [-1,  [2,3]],
           [-1,  [3,2]],
           [-1,  [3,3]] ]

for dot in example:
    classes.append(dot[0])
    features.append(dot[1])

print(classes)
print(features)
```

```
[1, 1, 1, 1, 1, 1, -1, -1, -1]
[[1, 1], [1, 2], [1, 3], [2, 1], [2, 2], [3, 1], [2, 3], [3, 2], [3, 3]]
```



Decision Tree with Scikit-Learn

- feature_importances_
  - DT에서 각 feature의 importances
  - Normalized total reduction of criteria by feature

```
DT = tree.DecisionTreeClassifier(criterion='gini')
DT.fit(X=features, y=classes)
print(DT.get_params())
```

```
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_featu
res': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': Non
e, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_
state': None, 'splitter': 'best'}
```

```
print(DT.feature_importances_)
```

```
[0.625 0.375]
```

- predict
  - Predict class for X

```
DT.predict([[1,2], [3,3], [2,2]])
```

```
array([ 1, -1,  1])
```

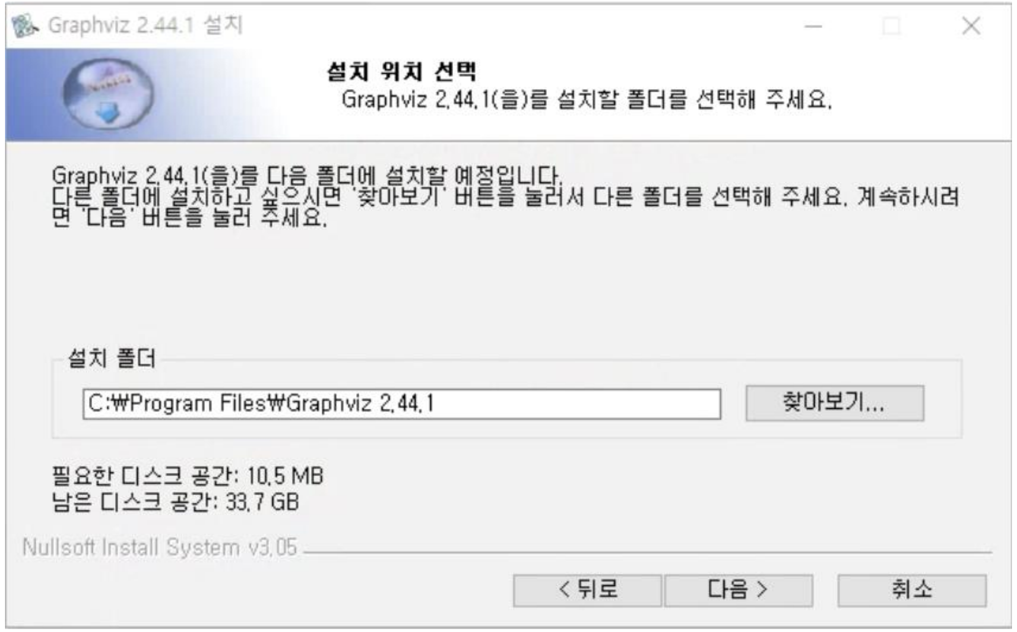
- predict-proba
  - Predict class probabilities of X
  - fraction of samples of the same class in leaf

```
DT.predict_proba([[1,2], [3,3], [2,2]])
```

```
array([[0., 1.],
       [1., 0.],
       [0., 1.]])
```

Visualize Decision Tree

- Download Graphviz
  - Graph Visualization Software
  - https://graphviz.org/download/#windows
  - Windows는 Stable Windows install packages 설치
  - 설치 도중 오른쪽과 같은 화면이 나옴 [**설치된 경로 꼭 기록 !**]
  - macOS의 경우 brew 이용하여 설치
    - brew install graphviz



- library 설치
  - pip install graphviz
  - 또는 conda install graphviz, conda install python-graphviz, conda install pydot

- 이후 import graphviz에 문제가 생길 경우
  - https://inflearn.com/questions/61605 참고

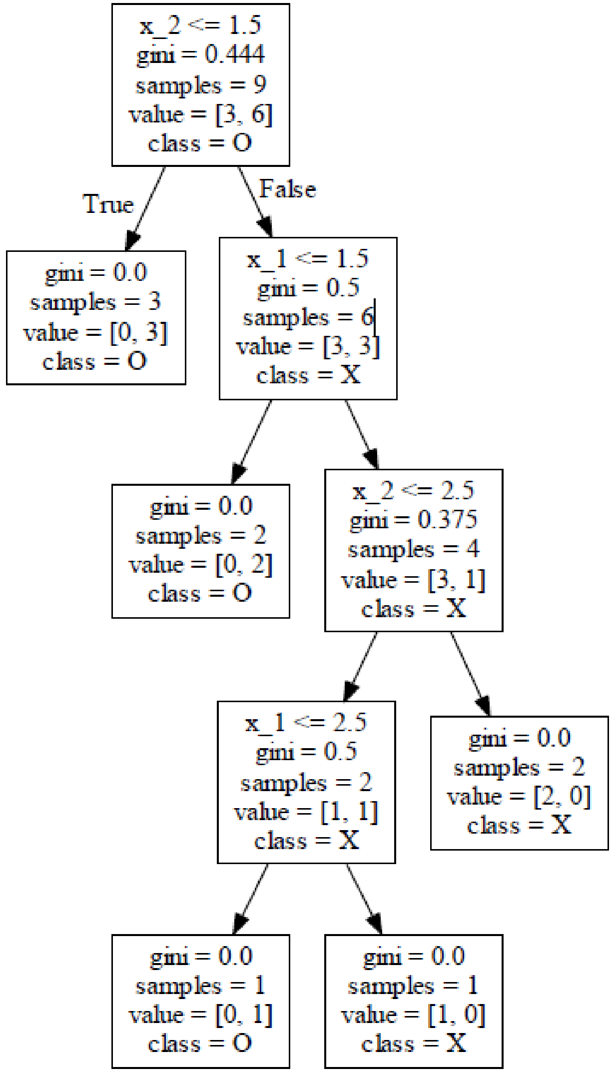
Visualize Decision Tree

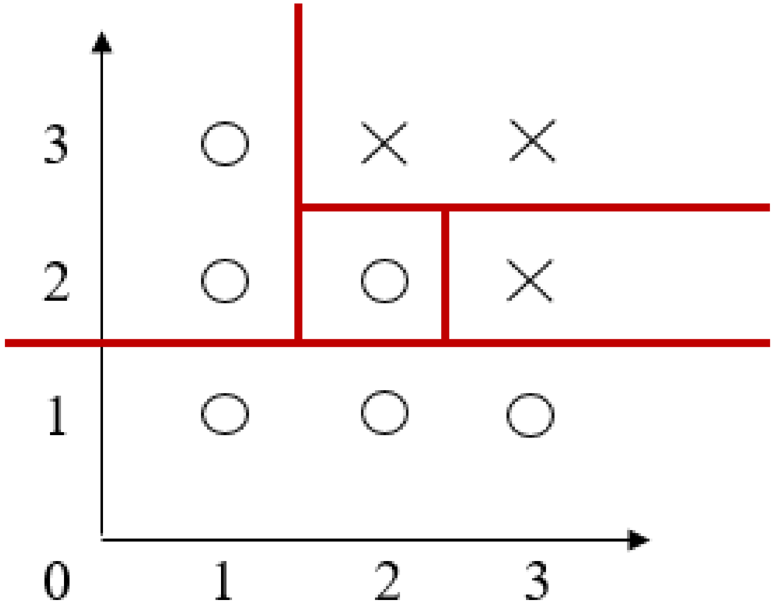
```
import graphviz
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz 2.47.1/bin/'

graph = tree.export_graphviz(DT, out_file=None, feature_names=['x_1', 'x_2'], class_names=['X', 'O'])
graph =graphviz.Source(graph)
graph.render('example_gini', view=True)
```

- import graphviz, import os
- os library를 사용해서 환경변수 설정
  - graphviz 설치 경로(windows의 경우 C:/Program Files/Graphviz 2.47.1/ , mac의 경우 brew로 설치했을 때 /usr/local/Cellar/graphviz/2.47.1/) + 'bin/' 을 환경변수에 추가
- export_graphviz
  - feature_names, class_names 입력(out_file=None 고정)
  - class_names는 작은 숫자로 설정한 class부터 적어야 함
  - Feature_names에 한글 포함 시 글꼴 깨짐이 발생할 수 있음. 이 때 fontname='Malgun Gothic' 등으로 폰트를 설정해주면 해결됨
- 코드 실행 시 example_gini.pdf 생성됨
  - render에서 view=True는 실행 이후 바로 pdf를 실행하라는 의미

Visualize Decision Tree





Data with pandas

- Data를 Decision tree에 input으로 사용하기 위하여 pandas library 활용
  - scikit-learn은 numpy, pandas와 높은 호환성을 가짐
  - sql문을 통해 얻은 view도 pandas의 Dataframe 함수를 통해 쉽게 input으로 활용 가능
    - pd.read_sql(sql_query, con=connect객체) 또는 pd.DataFrame(cursor.fetchall())

- example: iris data

```python
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
clf_iris = tree.DecisionTreeClassifier(random_state=0)
clf_iris.fit(X=iris.data, y=iris.target)

iris_graph = tree.export_graphviz(clf_iris, out_file=None, feature_names=iris.feature_names, class_names=iris.target_names)
iris_graph = graphviz.Source(iris_graph)
iris_graph.render('example_iris', view=True)
```

```
'example_iris.pdf'
```

```python
df = pd.DataFrame(iris.data, columns=iris.feature_names)
sy = pd.Series(iris.target, dtype='category')
df['target'] = sy

clf_iris_pd = tree.DecisionTreeClassifier(random_state=0)
clf_iris_pd.fit(X=df.iloc[:,:-1], y=df.iloc[:,-1])

iris_pd_graph = tree.export_graphviz(clf_iris_pd, out_file=None, feature_names=iris.feature_names, class_names=iris.target_names)
iris_pd_graph = graphviz.Source(iris_pd_graph)
iris_pd_graph.render('example_iris_pd', view=True)
```

```
'example_iris_pd.pdf'
```

DB Mining and Recommendation Using Python

PART 2. DB MINING; ASSOCIATION ANALYSIS

Association Analysis

Market Basket Example

- Where should detergents be placed in the Store to maximize their sales?
- Are window cleaning products purchased when detergents and orange juice are bought together?
- Is soda typically purchased with bananas? Does the brand of soda make a difference?
- How are the demographics of the neighborhood affecting what customers are buying?

Transaction Data

- Vertical Data
  - data per row
  - column redundancy 최소화

CustomerId	BasketId	Day	ProductId	Quantity	StoreId
1,364	26,984,896,261	1	842,930	1	31,742
1,364	26,984,896,261	1	897,044	1	31,742
1,364	26,984,896,261	1	920,955	1	31,742
1,364	26,984,896,261	1	937,406	1	31,742
1,364	26,984,896,261	1	981,760	1	31,742
1,130	26,984,905,972	1	833,715	2	31,642
1,130	26,984,905,972	1	866,950	2	31,642

- Horizontal data
  - transaction per row
  - 이해하기 쉬움

BasketId	Category1	Category2	Category3	Category4	Category5
26,984,896,261	0	0	0	0	0
26,984,905,972	0	0	0	0	0
26,984,951,769	0	0	0	0	1
26,985,025,264	0	0	0	0	0
26,985,040,735	0	0	0	0	1
26,985,205,886	0	0	1	0	0
26,985,360,571	0	0	0	1	0
26,992,197,681	0	0	0	0	0

Creating Horizontal Data

- example(Project 1 restaurant)

```sql
SELECT restaurant_id, restaurant_name,
if(category = 0, 1, 0) as 'category=아메리칸음식',
if(category = 1, 1, 0) as 'category=스시오마카세',
if(category = 2, 1, 0) as 'category=퓨전음식',
if(category = 3, 1, 0) as 'category=바베큐'
from restaurant
```

restaurant_id	restaurant_name	category=아메리칸음식	category=스시오마카세	category=퓨전음식	category=바베큐
-2taImqJHwGm5T6M8q8YGw	데일리픽스	1	0	0	0
1MKfjWTEV8KE2UPrGvlKdQ	혜화동버거	1	0	0	0
2ezAr-VGBKm0v9iNAfAsbQ	미식맥주	1	0	0	0
42pI7dkFqixGI55KmtZpTA	프랭클린스 델리	1	0	0	0
5X2_00mIDCI9AC4xpjyr0g	군몽	1	0	0	0
__2wghHUrj6Mh3ifIl9STw	함루 서울역점	0	0	0	0
__ecNH4GY-V1V5yQ3J88pg	호빈초밥	0	1	0	0
__LdzuMhSLULvvPkHrq6SQ	울산은하수	0	0	0	0
__SqT3QKV8KX5l0o99htLQ	어거스트힐	0	0	0	0
__VxLjL-dxcIDL7OshNL1g	카밀로 한남	0	0	0	0
_-hP-1O_bz8CWlKL9eePUw	키친마이야르	0	0	1	0

MySQL to pandas.DataFrame

- example (project 1 DB)

```
hor_view = pd.DataFrame(cursor.fetchall())
hor_view.columns = cursor.column_names
hor_view = hor_view.set_index('seller_id')
print(hor_view)
conn.close()
```

- cursor.fetchall(): 결과를 pandas의 DataFrame으로 변환
- Dataframe의 column 명을 select 문의 column 명으로 변환
- Dataframe의 특정 column을 index로 사용

Association Analysis with MLxtend

- 연관 분석 라이브러리
  - Mlxtend
  - Orange3

- Mlxtend(Machine Learning Extensions)
  - pip install mlxtend
  - from mlxtend.frequent_patterns import apriori, association_rules
  - pandas의 DataFrame과 함께 사용하기에 편리함

Association Analysis with MLxtend

```
import numpy as np
np.random.seed(0)

N = 100
X = np.random.random((N,100)) > 0.9

df = pd.DataFrame.from_records(X)
df
```

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
0	False	False	False	False	False	False	False	False	True	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	True	False	False	False	False	False	True	...	False	False	False	True	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True	True
3	True	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	True	False	False	False	False
4	False	True	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	True	False
96	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	True
97	False	False	False	False	True	False	False	False	False	False	...	False	False	False	False	False	True	False	False	False	False
98	False	False	False	False	False	False	False	True	False	False	...	False	False	True	False	False	False	False	True	False	False
99	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

Association Analysis with MLxtend

- frequent itemsets: apriori 함수 사용
  - min_support가 넘는 itemset 생성
  - use_colnames: DataFrame의 column명을 item 이름으로 활용할 것인지

```
from mlxtend.frequent_patterns import association_rules, apriori

frequent_itemsets = apriori(df, min_support=0.05, use_colnames=True)
print(frequent_itemsets)
```

- association rules: association_rules 함수 사용
  - metric이 min_threshold를 넘는 rule들 생성

```
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)
print(rules.to_string())
```

- 특정 조건을 만족하는 규칙 추출 (pandas의 boolean indexing 기능 활용)
  - ex. rules[rules['confidence'] 〉 0.75]
  - ex. rules[(rules['confidence'] 〉 0.75) & (rules['lift'] 〉 2)]

Association Analysis with MLxtend

```python
from mlxtend.frequent_patterns import association_rules, apriori

frequent_itemsets = apriori(df, min_support=0.05, use_colnames=True)
print(frequent_itemsets)
```

```
     support   itemsets
0       0.08        (0)
1       0.13        (1)
2       0.05        (2)
3       0.08        (3)
4       0.12        (4)
..       ...        ...
111     0.05   (58, 63)
112     0.05   (72, 80)
113     0.05   (72, 97)
114     0.05   (98, 75)
115     0.05   (97, 77)

[116 rows x 2 columns]
```

```python
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)
print(rules.to_string())
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(48)	(1)	0.12	0.13	0.05	0.416667	3.205128	0.0344	1.491429
1	(1)	(48)	0.13	0.12	0.05	0.384615	3.205128	0.0344	1.430000
2	(20)	(54)	0.15	0.16	0.05	0.333333	2.083333	0.0260	1.260000
3	(54)	(20)	0.16	0.15	0.05	0.312500	2.083333	0.0260	1.236364
4	(72)	(21)	0.16	0.14	0.05	0.312500	2.232143	0.0276	1.250909
5	(21)	(72)	0.14	0.16	0.05	0.357143	2.232143	0.0276	1.306667
6	(24)	(27)	0.13	0.14	0.05	0.384615	2.747253	0.0318	1.397500
7	(27)	(24)	0.14	0.13	0.05	0.357143	2.747253	0.0318	1.353333
8	(25)	(36)	0.14	0.09	0.05	0.357143	3.968254	0.0374	1.415556
9	(36)	(25)	0.09	0.14	0.05	0.555556	3.968254	0.0374	1.935000
10	(25)	(54)	0.14	0.16	0.06	0.428571	2.678571	0.0376	1.470000
11	(54)	(25)	0.16	0.14	0.06	0.375000	2.678571	0.0376	1.376000
12	(25)	(58)	0.14	0.13	0.05	0.357143	2.747253	0.0318	1.353333
13	(58)	(25)	0.13	0.14	0.05	0.384615	2.747253	0.0318	1.397500
14	(32)	(30)	0.12	0.09	0.05	0.416667	4.629630	0.0392	1.560000
15	(30)	(32)	0.09	0.12	0.05	0.555556	4.629630	0.0392	1.980000
16	(49)	(77)	0.09	0.13	0.05	0.555556	4.273504	0.0383	1.957500
17	(77)	(49)	0.13	0.09	0.05	0.384615	4.273504	0.0383	1.478750
18	(51)	(71)	0.14	0.11	0.05	0.357143	3.246753	0.0346	1.384444
19	(71)	(51)	0.11	0.14	0.05	0.454545	3.246753	0.0346	1.576667
20	(51)	(75)	0.14	0.10	0.05	0.357143	3.571429	0.0360	1.400000
21	(75)	(51)	0.10	0.14	0.05	0.500000	3.571429	0.0360	1.720000
22	(58)	(54)	0.13	0.16	0.05	0.384615	2.403846	0.0292	1.365000
23	(54)	(58)	0.16	0.13	0.05	0.312500	2.403846	0.0292	1.265455
24	(60)	(54)	0.13	0.16	0.05	0.384615	2.403846	0.0292	1.365000
25	(54)	(60)	0.16	0.13	0.05	0.312500	2.403846	0.0292	1.265455
26	(58)	(63)	0.13	0.10	0.05	0.384615	3.846154	0.0370	1.462500
27	(63)	(58)	0.10	0.13	0.05	0.500000	3.846154	0.0370	1.740000
28	(72)	(80)	0.16	0.10	0.05	0.312500	3.125000	0.0340	1.309091
29	(80)	(72)	0.10	0.16	0.05	0.500000	3.125000	0.0340	1.680000
30	(72)	(97)	0.16	0.12	0.05	0.312500	2.604167	0.0308	1.280000
31	(97)	(72)	0.12	0.16	0.05	0.416667	2.604167	0.0308	1.440000
32	(98)	(75)	0.11	0.10	0.05	0.454545	4.545455	0.0390	1.650000
33	(75)	(98)	0.10	0.11	0.05	0.500000	4.545455	0.0390	1.780000
34	(97)	(77)	0.12	0.13	0.05	0.416667	3.205128	0.0344	1.491429
35	(77)	(97)	0.13	0.12	0.05	0.384615	3.205128	0.0344	1.430000

DB Mining and Recommendation Using Python

PART 3. RECOMMENDATION SYSTEM

Surprise

- Surprise
  - 추천 시스템을 위한 python scikit
  - simple python recommendation system engine

- 설치 및 실행
  - cmd창에 conda install –c conda-forge scikit-surprise
  - proceed ([y]/n)? 메세지 뜨면 y+엔터 눌러서 설치 진행

  - import surprise

Surprise 사용 예제 – Movie-Lens data

- Loading data

```
import surprise

data = surprise.Dataset.load_builtin('ml-100k')
df = pd.DataFrame(data.raw_ratings, columns=['user', 'item', 'rate', 'id'])
del df['id']
df.head(10)
```

	user	item	rate
0	196	242	3.0
1	186	302	3.0
2	22	377	1.0
3	244	51	2.0
4	166	346	1.0
5	298	474	4.0
6	115	265	2.0
7	253	465	5.0
8	305	451	3.0
9	6	86	3.0

학습을 위한 data 형태

: (user id, item id, rating)

Surprise 사용 예제 – Movie-Lens data

- Loading data
  - user-item matrix 형태로 확인

```
df_table = df.set_index(['user', 'item']).unstack()
df_table.fillna("").iloc[212:222, 808:817]
```

	rate								
item	211	212	213	214	215	216	217	218	219
user									
290	3					4		2	
291		4		4	4			4	4
292				3					
293	4		3		4	4	3	2	
294									
295			5		5	5	4	5	
296	4								
297	4		3		2	4		3	
298	5		3		5				
299	4	4	5			5			

Surprise 사용 예제 – Movie-Lens data

- Algorithm 지정 후 k-fold training

```
from surprise.model_selection import KFold

bsl_options = { 'method': 'als',
               'n_epochs':5,
               'reg_u':12,
               'reg_i': 5
               }

algo = surprise.BaselineOnly(bsl_options=bsl_options)

np.random.seed(0)
kf = KFold(n_splits=3)
acc=[]
for i, (trainset, testset) in enumerate(kf.split(data)):
    algo.fit(trainset)
    predictions = algo.test(testset)
    acc.append(surprise.accuracy.rmse(predictions, verbose=True))
np.mean(acc)
```

```
Estimating biases using als...
RMSE: 0.9453
Estimating biases using als...
RMSE: 0.9377
Estimating biases using als...
RMSE: 0.9500

0.9443304984013942
```

Surprise 사용 예제 – Movie-Lens data

- Predict

```
uid = str(293)
iid = str(214)
algo.predict(uid, iid)
```

```
Prediction(uid='293', iid='214', r_ui=None, est=2.8359640039096501
```

```
uid = str(295)
iid = str(216)
algo.predict(uid, iid)
```

```
Prediction(uid='295', iid='216', r_ui=None, est=4.5665294215198156.
```

	rate								
item	211	212	213	214	215	216	217	218	219
user									
290	3					4		2	
291		4		4	4			4	4
292				3					
293	4		3		4	4	3	2	
294									
295			5		5	5	4	5	
296	4								
297	4		3		2	4		3	
298	5		3		5				
299	4	4	5			5			

Surprise 사용 예제 – Movie-Lens data

- Evaluation – cross validation

```
surprise.model_selection.cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=3, verbose=True)
```

```
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Evaluating RMSE, MAE of algorithm BaselineOnly on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
MAE (testset)     0.7508  0.7474  0.7481  0.7488  0.0014
RMSE (testset)    0.9436  0.9452  0.9453  0.9447  0.0008
Fit time          0.18    0.10    0.10    0.13    0.04
Test time         0.62    3.57    0.22    1.47    1.49

{'fit_time': (0.18211984634399414, 0.10307121276855469, 0.09708261489868164),
 'test_mae': array([ 0.75079592,  0.74744677,  0.74810824]),
 'test_rmse': array([ 0.94355622,  0.94523088,  0.94530769]),
 'test_time': (0.6195931434631348, 3.5672948360443115, 0.21670198440551758)}
```

Surprise Algorithms

- Baseline algorithm (surprise.BaselineOnly)

$$\widehat{r_{ui}} = b_{ui} = \mu + b_u + b_i$$

- Options
  - Method: ALS(Alternating Least Squares)(=default), SGD(Stochastic Gradient Descent)
  - n_epochs: ALS 혹은 SGD algorithm 수행 횟수
    (ALS는 기본값 10, SGD는 기본값 20)
  - ALS
    - reg_i: item의 regularization parameter. 기본값은 10
    - reg_u: user의 regularization parameter. 기본값은 15
  - SGD
    - reg: regularization parameter. 기본값은 0.02
    - learning_rate: learning rate. 기본값은 0.005

Surprise Similarity Measures

- Similarity measure configuration
  - bsl_option처럼 dictionary 형태로 지정

```
sim_options = {'name': 'cosine', 'user_based': False}
algo = surprise.KNNBasic(sim_options=sim_options)
```

- Options
  - name: cosine, msd(mean squared difference), pearson, pearson_baseline
  - user_based: True이면 user based, False이면 item based. 기본값은 True
  - min_support: 취급할 user간 공통 item(혹은 item간 공통 user)의 최소값. 즉 $|I_{uv}| < \min_support$이면 $sim(u,v) = 0$
  - shrinkage: pearson_baseline일 때 shrinkage parameter. 기본값은 100

Surprise Similarity Measures

- Pearson similarity

$$\text{pearson\_sim}(u, v) = \frac{\sum\limits_{i \in I_{uv}} (r_{ui} - \mu_u) \cdot (r_{vi} - \mu_v)}{\sqrt{\sum\limits_{i \in I_{uv}} (r_{ui} - \mu_u)^2} \cdot \sqrt{\sum\limits_{i \in I_{uv}} (r_{vi} - \mu_v)^2}}$$

- Pearson_baseline similarity

$$\text{pearson\_baseline\_sim}(u, v) = \hat{\rho}_{uv} = \frac{\sum\limits_{i \in I_{uv}} (r_{ui} - b_{ui}) \cdot (r_{vi} - b_{vi})}{\sqrt{\sum\limits_{i \in I_{uv}} (r_{ui} - b_{ui})^2} \cdot \sqrt{\sum\limits_{i \in I_{uv}} (r_{vi} - b_{vi})^2}}$$

  - Shrinkage parameter

$$\text{pearson\_baseline\_shrunk\_sim}(u, v) = \frac{|I_{uv}| - 1}{|I_{uv}| - 1 + \text{shrinkage}} \cdot \hat{\rho}_{uv}$$

Surprise Algorithms: k-NN based

- Basic k-NN algorithm (surprise.KNNBasic)

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v) \cdot r_{vi}}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v)}$$

(user based)

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in N_u^k(i)} \text{sim}(i, j) \cdot r_{uj}}{\sum\limits_{j \in N_u^k(j)} \text{sim}(i, j)}$$

(item based)

- Parameters
  - k: k-NN에서 nearest neighbors의 (최대) 수. 기본값은 40
  - min_k: neighbors 수의 최소값. 기본값은 1
  - sim_options: similarity measure 정보 dictionary
  - verbose: bias 추정값을 출력할지 결정. 기본값은 True

Surprise Algorithms: k-NN based

- Mean centered k-NN algorithm (surprise.KNNWithMeans)

$$\hat{r}_{ui} = \underline{\mu_u} + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} \underline{- \mu_v})}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)}$$

(user based)

- Z-score k-NN algorithm (surprise.KNNWithZScore)

$$\hat{r}_{ui} = \underline{\mu_u + \sigma_u} \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} \underline{- \mu_v) / \sigma_v}}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)}$$

(user based)

Surprise Algorithms: k-NN based

- Baseline centered k-NN algorithm (surprise.KNNBaseline)

$$\hat{r}_{ui} = \underline{b_{ui}} + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} \underline{- b_{vi}})}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)}$$

(user based)

- Parameters
  - k: k-NN에서 nearest neighbors의 (최대) 수. 기본값은 40
  - min_k: neighbors 수의 최소값. 기본값은 1
  - sim_options: similarity measure 정보 dictionary
  - bsl_options: baseline option 정보 dictionary
  - verbose: bias 추정값을 출력할지 결정. 기본값은 True

Surprise Algorithms: MF based

- SVD(Singular Vector Decomposition) (surprise.SVD)

$$R = UV^T$$

  - $\widehat{r_{ui}} = {q_i}^T p_u$ (unbiased)
  - $\widehat{r_{ui}} = \mu + b_u + b_i + {q_i}^T p_u$ (biased)

- Parameters
  - n_factors: latent factor 수. 기본값은 100
  - n_epochs: SGD 수행 횟수. 기본값은 20
  - biased: True면 biased, False면 unbiased
  - lr_(all/bu/bi/pu/qi): learning rate. All의 기본값은 0.005
  - reg_(all/bu/bi/pu/qi): regularization term. All의 기본값은 0.02

http://surprise.readthedocs.io/en/stable/matrix_factorization.html

Surprise Algorithms: MF based

- SVD++ (surprise.SVDpp)

$$R = (U + \underline{FY})V^T$$

(F : implicit feedback, Y : implicit item−factor)

  - $\widehat{r_{ui}} = {q_i}^T(p_u + \sum_{j \in I_u} \frac{y_j}{\sqrt{|I_u|}})$ (unbiased)
  - $\widehat{r_{ui}} = \mu + b_u + b_i + {q_i}^T(p_u + \sum_{j \in I_u} \frac{y_j}{\sqrt{|I_u|}})$ (biased)

- NMF(surprise.NMF)
  - Non−negative matrices

http://surprise.readthedocs.io/en/stable/matrix_factorization.html

참고문헌

- https://surprise.readthedocs.io
- https://datascienceschool.net/view-notebook/fcd3550f11ac4537acec8d18136f2066/
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/

End of the Document