출처가 명시되지 않은 모든 자료(이미지 등)는 조성현 강사님 블로그 및 강의 자료 기반.

<< 머신러닝 - 연관 분석 >>

[Apriori]

1. 연관분석

데이터 집합 간 숨겨진 규칙을 찾아내기 위한 분석 방법이다. 구매성향 분석 등에 자주 사용된다.

예컨대 다음과 같은 장바구니 데이터가 있다고 하자.

[표-1] 장비구니 트랜잭션 예시

TID	Items
1	Bread, Milk
2	Bread, Diapers, Beer, Eggs
3	Milk, Diapers, Beer, Cola
4	Bread, Milk, Diapers, Beer
5	Bread, Milk, Diapers, Cola

* 예시 출처 : Introduction to Data Mining (데이터 마이닝), p 320 ~, (by 용환승, 나연묵, 박종수 등)

위의 장바구니 데이터로부터 의미 있는 상품 간 관계를 찾아내고 싶다. 이를 위해 장바구니 데이터를 이진 행렬로 변환한다. 이진행렬에서 행은 트랜잭션 ID, 열은 상품 목록이 된다. 수량, 가격 등은 무시하고, 구매했는지의 여부만을 `1`과 `0`으로 나타내자.

[표–1] 장비구니 트랜잭션 예시

TID	Items
1	Bread, Milk
2	Bread, Diapers, Beer, Eggs
3	Milk, Diapers, Beer, Cola
4	Bread, Milk, Diapers, Beer
5	Bread, Milk, Diapers, Cola

변환 →

[표–2] 이진 목록 행렬 (Item Matrx in sparse format)

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

이렇게 변환된 이진 행렬은 `Sparse Matrix`이다. 일단은 대부분이 0이고, 가끔씩 1이 아닌 값이 등장하는 행렬이라고 이해하고 넘어가도록 하자.

이제 각 상품 데이터 간 연관 규칙을 찾아내기 위한 **척도**를 알아 보자.

1. 지지도(Support)

특정 (상품) 데이터를 포함하는 트랜잭션의 개수를 **지지도 카운트**라고 한다. 이 때, **지지도**는 전체 트랜잭션의 개수 대비 지지도 카운트로 정의된다.

지지도 (Support) : $s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$ → 이 규칙이 주어진 데이터 집합에서 얼마나 자주 발생하는지를 측정함.

예컨대, 위의 예에서 Milk와 Diapers를 사는 사람이 Cola도 같이 사는 경향이 있다는 규칙을 찾아내려 한다고 하자.

- `X = {Milk, Diapers}`
- `Y = {Cola}`
- 찾아내고자 하는 규칙: `s(X →Y)` = `X`를 구매하는 사람이 `Y`도 구매하는가?

이 때 X의 지지도 카운트는 2, 전체 트랜잭션의 개수 `N`은 5가 되기 때문에, `s(X →Y)`는 0.4가 된다.

지지도는 한 연관규칙이 주어진 데이터 집합에서 얼마나 자주 나타나는지를 결정한다. 지지도가 낮을수록 *우연하게* 같이 등장했을 확률이 높기 때문에, **유용하지 않은 규칙**을 제거하는 데에 사용된다.

2. 신뢰도(confidence)

지지도는 두 상품의 합집합이 같을 때 그 수치가 모두 같게 나타난다. 따라서 `X` 조합 이후 `Y`가 오는지, `Y` 조합 이후 `X`가 오는지를 판단하기에는 적절하지 않은 지표이다. `X ∪ Y` 가 같다면 지지도는 모두 동일하기 때문이다.

따라서 신뢰도라는 규칙을 도입한다. `X`를 포함하는 항목 중 `Y`와 함께 나타나는 항목이 얼마나 되는지, 그 조건부 확률을 측정한다.

신뢰도 (Confidence) : $c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$ → X에 속한 항목들 중에 Y가 얼마나 빈번히 발생하는지에 대한 척도

Milk, Diapers, Beer의 예시에서 신뢰도를 계산하면 `P(X)= 3`, `p(X ∪ Y)=2`가 되기 때문에

이 때 X의 지지도 카운트는 2, 전체 트랜잭션의 개수 `N`은 5가 되기 때문에, `s(X →Y)`는 0.4가 된다.

지지도는 한 연관규칙이 주어진 데이터 집합에서 얼마나 자주 나타나는지를 결정한다. 지지도가 낮을수록 *우연하게* 같이 등장했을 확률이 높기 때문에, **유용하지 않은 규칙**을 제거하는 데에 사용된다.

3. 향상도(lift)

신뢰도에도 문제가 없는 것은 아니다. 만약 어떠한 데이터에 `Y`를 포함하고 있는 데이터가 매우 많다면, `X`에 어떠한 데이터를 포함하더라도 신뢰도는 높게 측정된다. 따라서 `Y`에 대한 영향이 무시되는 측면이 있다.

$$Lift(X \rightarrow Y) = \frac{c(X \rightarrow Y)}{s(Y)} = \frac{s(X \cup Y)}{s(X) \cdot s(Y)}$$

교재 오타 수정

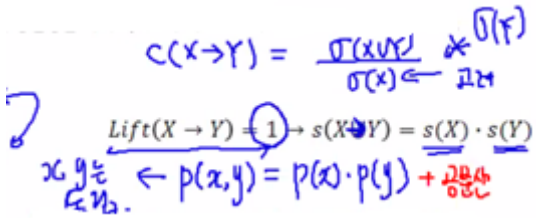
이러한 문제점을 보완하기 위해 **향상도**를 사용한다. 향상도는 신뢰도를 `Y` 발생 비율로 나눈 척도이다. `Y`가 많이 발생할수록 향상도 척도가 낮아진다.

향상도 척도는 두 데이터 집합이 어떠한 상관성을 갖는지를 판단하는 데 주로 활용된다.

만약 향상도가 1이라고 하자. 그렇다면 다음의 식이 성립한다.

$$s(X \cup Y) = s(X) \cdot s(Y)$$

X와 Y의 지지도가 X와 Y의 지지도의 곱과 같아진다. X와 Y를 함께 구매하는 게 X를 구매하고 Y를 구매하는 각각의 건수의 곱과 같다면, 두 사건은 독립이란 말이 된다.

> *강사님의 가르침*
>
> 
>
>
> 두 사건이 동시에 발생할 때의 확률과 같다. 원래 두 사건이 독립이면 확률은 각각의 곱과 같지만, 독립이 아니라면 뒤에 공분산이 붙는다.

같은 원리로, 향상도가 1보다 크면 두 사건은 양의 상관성을, 1보다 작으면 음의 상관성을 가진다.

양의 상관성을 가질수록 의미가 있기 때문에 향상도가 1보다 클수록 규칙의 의미가 커진다.

[참고] 인과관계 여부 주의

위의 연관규칙에 의해 만들어진 규칙은 인과관계에 해당하지는 않는다. 다만, 둘 간의 관계가 동시에 발생하는 것이라는 점을 나타낸다.

앞에서 말한 $X \rightarrow Y$ 의 규칙을 나타내고자 할 때, X를 antecedent, Y를 consequent라고 한다. 연관규칙은 antecedent와 consequent가 동시에 발생하고 있음을 나타낼 뿐이다.

2. Apriori 알고리즘

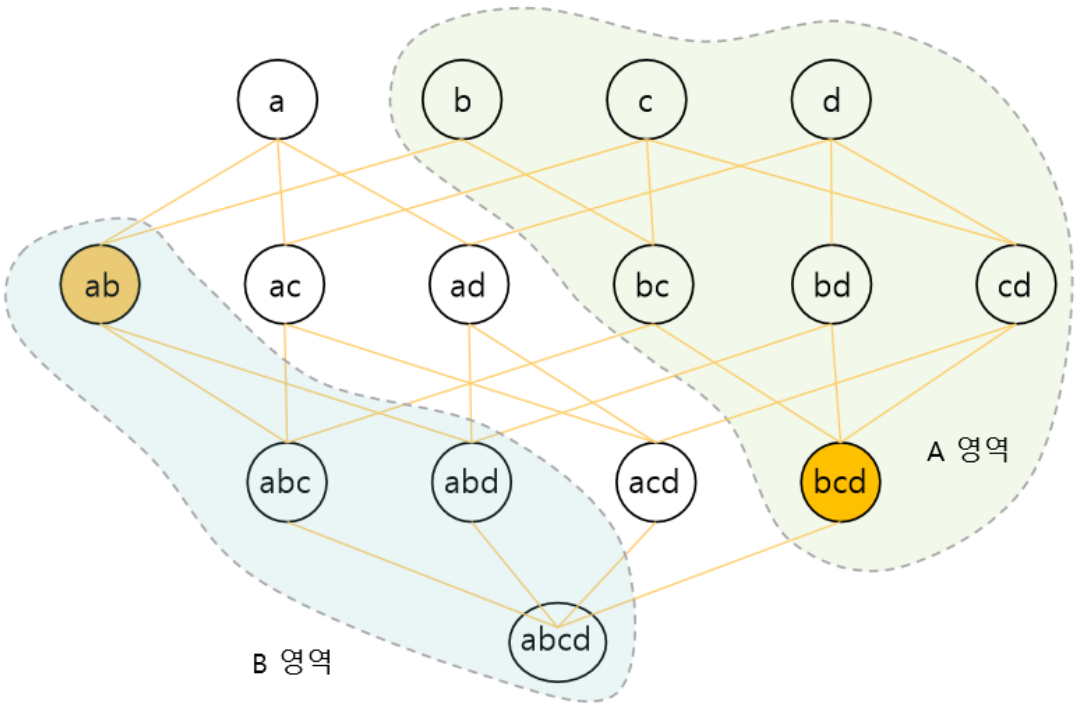
위에서 살펴 본 세 가지 규칙은 저마다 다른 효용성을 지니기 때문에, 세 지표 모두가 좋아야 효과적인 규칙이라고 할 수 있다.

데이터에서 연관 규칙을 찾고자 하는 알고리즘이다.

연관 규칙을 찾고자 할 때, 가장 쉬운 방법은 Brute Force 방식으로 모든 데이터 조합에 대해 지지도, 신뢰도 등 모든 규칙을 계산하는 것이다. 그러나 이와 같은 방식으로는 수많은 조합에 대해 수많은 연관 규칙을 모두 계산할 수 없다. ~~물론 할 수는 있지만, 그 비용이 어마어마하다.~~

무수히 많은 조합, 그리고 그 규칙을 줄이기 위해 Apriori 알고리즘이 사용된다. 다음과 같은 방식을 따른다.

- 지지도 기준의 임계치를 설정한다. **조합을 줄인다**.



BCD를 포함하는 집합이 빈발하면, 그것의 부분집합도 모두 빈발하다. 반면, AB를 포함하는 집합이 빈발하지 않다면, 그것을 포함하는 모든 조합도 모두 빈발하지 않다.

  - 지지도는 $X \cup Y$에만 의존하므로 그 합집합만 같으면 모두 동일하다.

    - 지지도가 낮다는 말은 결국 해당 데이터 조합이 빈번하지 않다는 의미이므로, 지지도가 낮은 조합에 대해 *굳이* 신뢰도를 계산할 필요가 없다.
    - **가지치기**의 단계라고도 볼 수 있다.(`Support based Pruning`)
- 조합의 개수를 줄인 후, 연관규칙 점수를 계산한다.

실습 1. mlxtend 패키지

- `mlxtend.preprocessing.TransactionEncoder` : 트랜잭션 데이터를 이진 행렬로 변환해 준다.

```
# 변환 전 데이터
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

# TransactionEncoder 변환
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)

# 변환 후 데이터
print(te_ary)
[[False False False  True False  True  True  True  True False  True]
 [False False  True  True False  True False  True  True False  True]
 [ True False False  True False  True  True False False False False]
 [False  True False False False  True  True False False  True  True]
 [False  True False  True  True  True False False  True False False]]
```

- confidence 최소 임계치 설정 : `min_support`

실습 2. 장바구니 데이터

위의 실습에서와 달리 데이터 크지가 커지면서, `min_support`를 더 낮췄다.

```
                0                 1   ...       18          19
0          shrimp           almonds   ...  spinach  olive oil
1         burgers         meatballs   ...      NaN         NaN
2         chutney               NaN   ...      NaN         NaN
3          turkey           avocado   ...      NaN         NaN
4     mineral water              milk   ...      NaN         NaN
             ...               ...  ...      ...         ...
7496       butter        light mayo   ...      NaN         NaN
7497      burgers  frozen vegetables   ...      NaN         NaN
7498      chicken               NaN   ...      NaN         NaN
7499     escalope        green tea   ...      NaN         NaN
7500         eggs  frozen smoothie   ...      NaN         NaN
```

원래 데이터 생김새는 위와 같다.

- `df.iterrow()` : pandas DataFrame의 row를 iterate하면서 row의 index와 data를 반환한다. ~~(약간 `enumerate`의 row 버전 느낌인가?)~~

```
# 데이터 구조
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7501 entries, 0 to 7500
Data columns (total 20 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       7501 non-null   object
 1   1       5747 non-null   object
 2   2       4389 non-null   object
...
 19  19      1 non-null      object
dtypes: object(20)
memory usage: 1.1+ MB

# iterrow() 메소드 수행
for i, row in df.iterrows():
    print(i, row)

    ...
4107 0        chocolate
1     french fries
2              NaN
3              NaN
4              NaN
5              NaN
6              NaN
7              NaN
8              NaN
9              NaN
10             NaN
11             NaN
12             NaN
13             NaN
14             NaN
15             NaN
16             NaN
```

```
17              NaN
18              NaN
19              NaN
Name: 4107, dtype: object
4108 0           burgers
1             shrimp
2          spaghetti
3          olive oil
4             salmon
5       cooking oil
6          escalope
7      tomato juice
8               NaN
9               NaN
10              NaN
11              NaN
12              NaN
13              NaN
14              NaN
15              NaN
16              NaN
17              NaN
18              NaN
19              NaN
Name: 4108, dtype: object
      ...
```

- 결측값 제거 : `str`으로 변환 후 `nan`이면 제거했다.

```
dataset = []
for i, row in df.iterrows():
    dataset.append([x for x in list(row) if str(x) != 'nan'])
```

왜 `dropna()`하지 않고 `for문` 사용해서 문자열 비교 후 제거했을까?

- `applymap` : DataFrame 객체에 함수 매핑 및 적용.

```
rules[cols] = rules[cols].applymap(lambda x : tuple(x))
```

- 결과

tuple	frozenset