

## 1 데이터분석 단계(Data Analysis Cycle)

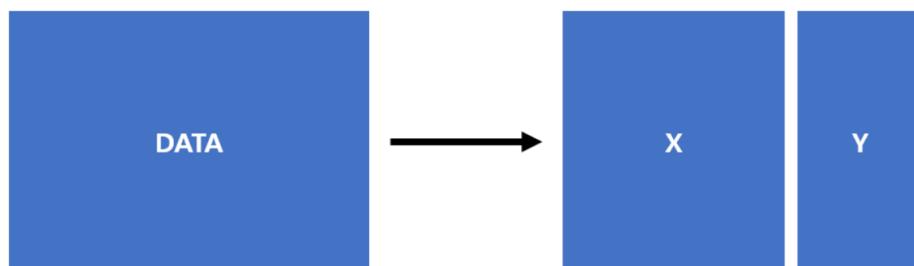
✓ 데이터 전처리: (0) 쓸모 없을 뻔한 Raw를 쓸모 있는 Data로 변환

100	T50	횟수	111	TPU	...
few	Gds	Hvi	Rew	Fa	...
Fre	CT	QTP	D	합	...
'1'	1	23	22	NaN	43
76	NaN	43	32	1	8
'Hi'	NaN	NaN	NaN	NaN	87
23	98	NaN	64	46	NaN
c	90	NaN	'WW'	24	'KK'
t	NaN	2	NaN	NaN	6
64	NaN	90	'IU'	4	76

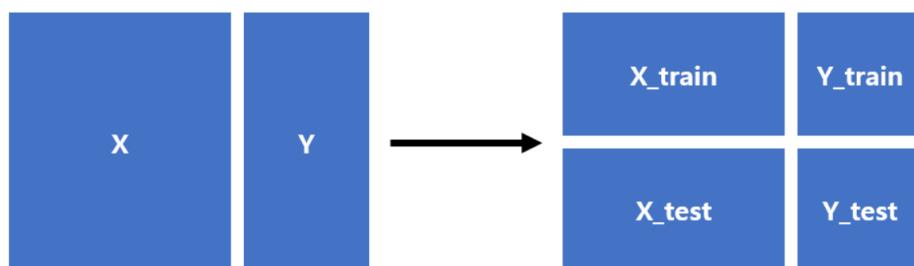
  

번호	시간	총량	기간	누적	...
1	1	23	22	21	43
76	33.3	43	32	1	8
5	33.3	52	35	21	87
23	98	52	64	46	61
90	33.3	2	24	33	4
55	2	52	35	21	6
64	33.3	90	11	4	76

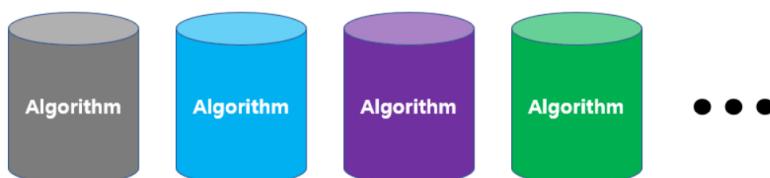
✓ 데이터 분할: (1) 목표/종속변수 Y와 설명/독립변수 X설정



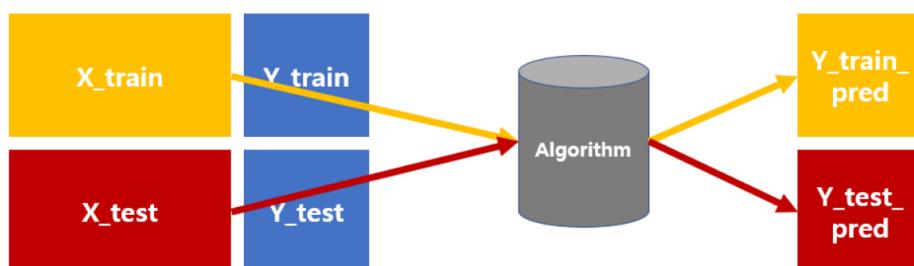
✓ 데이터 분할: (2) 학습데이터 Train과 예측 데이터 Test로 분할



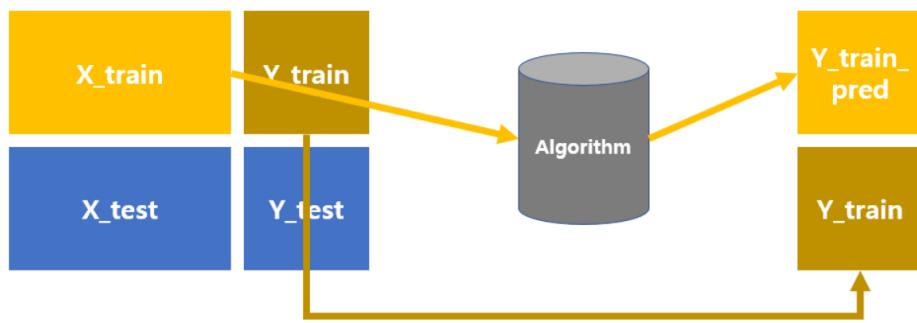
✓ 모델링: (3) 분석 목적에 맞는 알고리즘(Base & Advanced) 후보들 준비



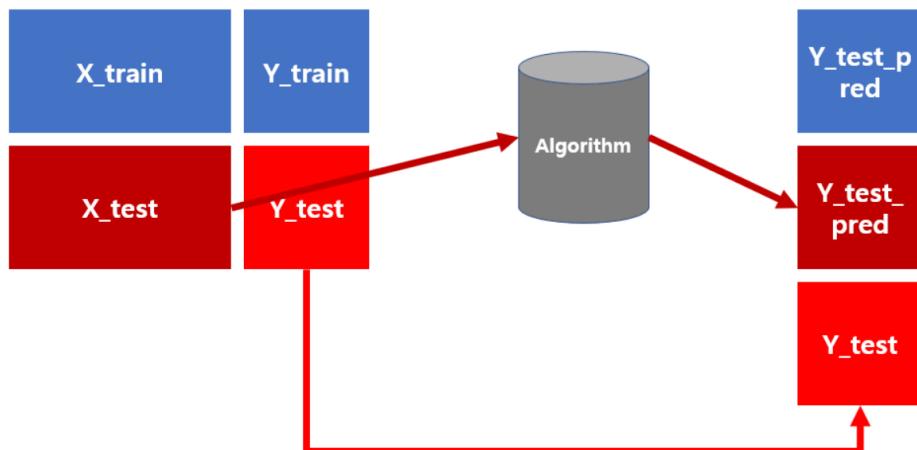
✓ 모델링 & 학습: (4) 알고리즘 평가를 위해 Train/Test의 예측값 추정



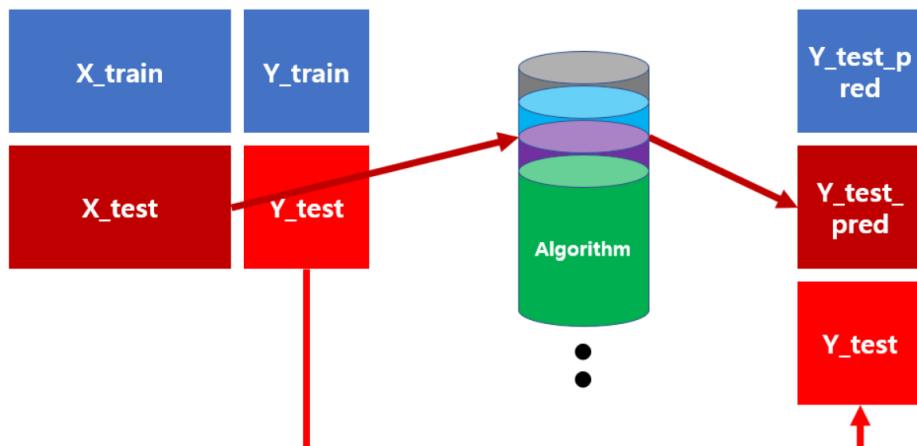
✓ 평가: (5) 학습(Train)이 잘 되었는지 알고리즘 성능검증



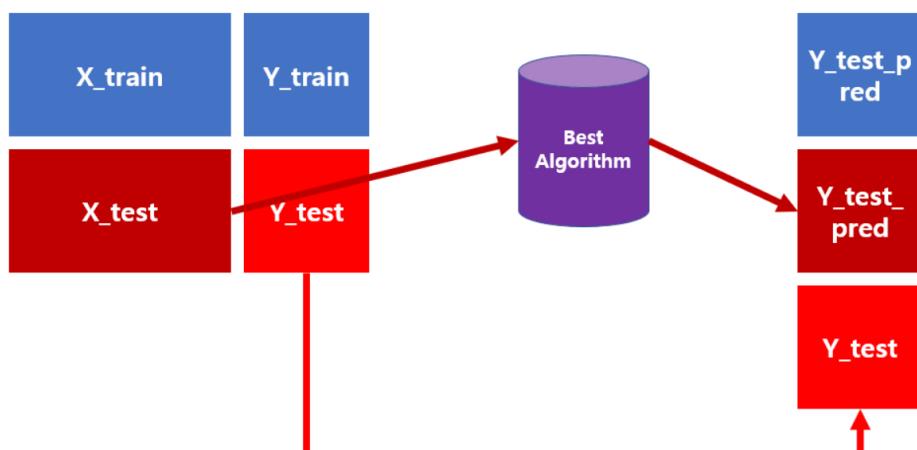
✓ 평가: (6) 예측(Test)이 잘 되었는지 알고리즘 성능검증



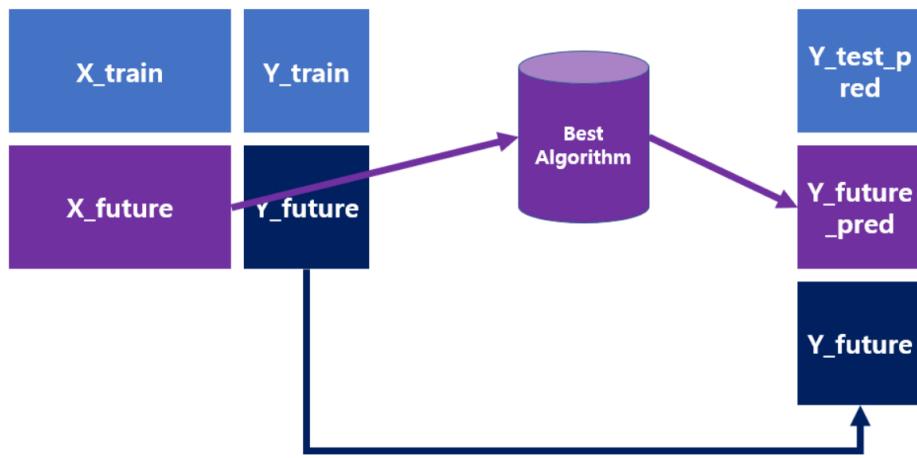
✓ 최적 알고리즘 선택: (7) 알고리즘을 변경하여 위 과정 반복 후



✓ 최적 알고리즘 선택: (7) 최고 성능의 알고리즘 선택



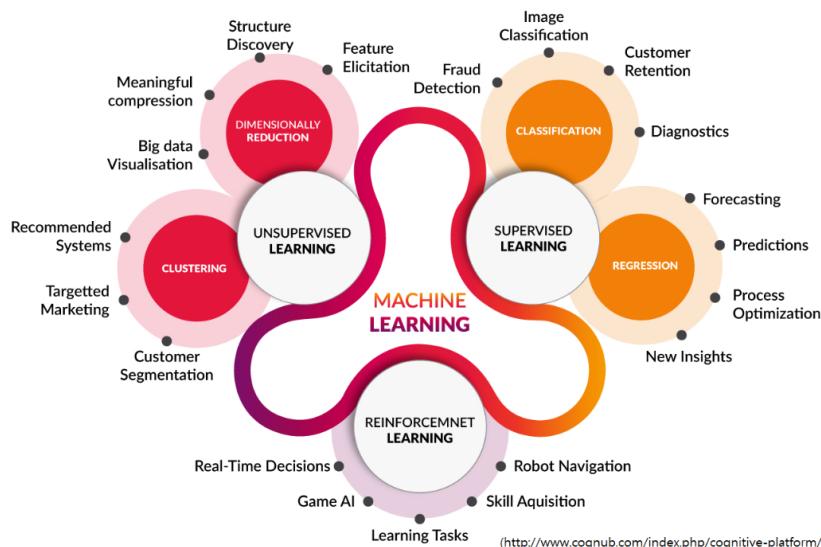
✓ 배포: (8) 실제 비즈니스 서비스 현업 적용 및 매출/수익/개선 정도 평가



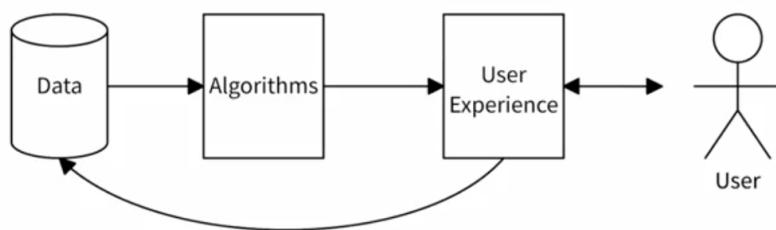
## 2 추천시스템(Recommender System)

### 1) 비지도학습 & 강화학습 알고리즘

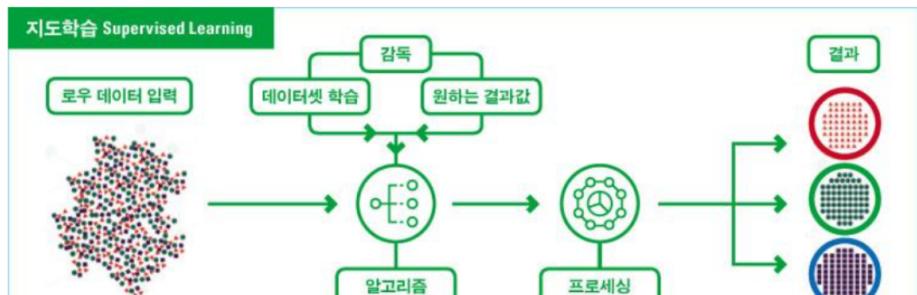
"사용자(User)나 상품(Item)의 패턴을 빅데이터로 학습한 후, 사용자(User)에게 상품(Item)을 제안하는 알고리즘 도구이자 기술로, 추천시스템을 구축하는 것은 이러한 도구를 사용하여 어떤 사용자에게 어떤 상품을 추천할지 이해하는 기계학습의 한 방향" → 비지도학습

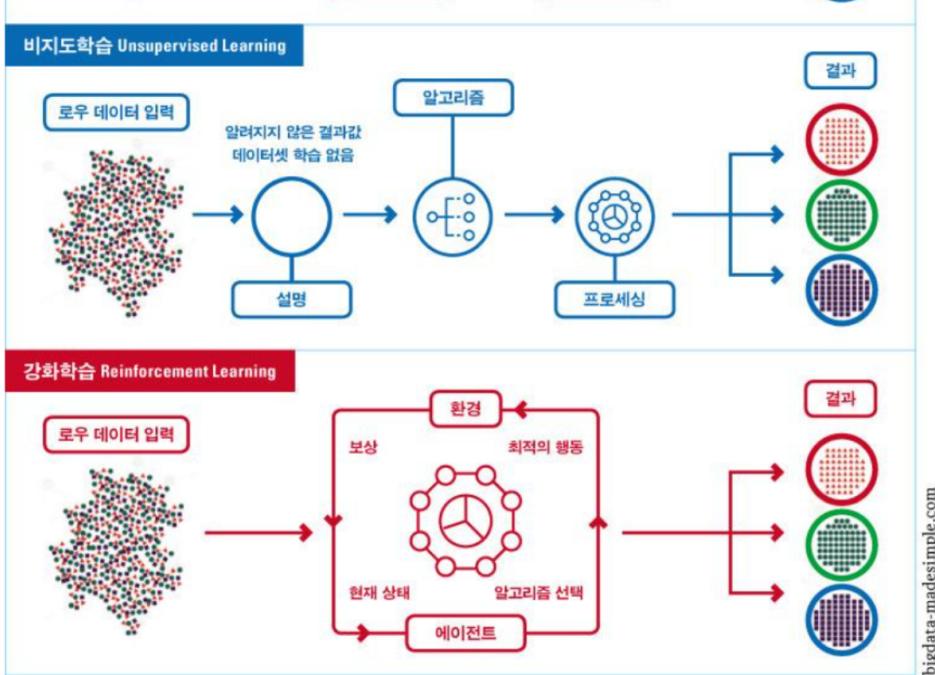


- 어떤 상품을 구매할지, 어떤 음악을 들을지, 어떤 뉴스를 읽을지 등의 의사결정과 연관
- 유튜브(Youtube)나 넷플릭스(Netflix)를 포함하여 모든 상품 추천 비즈니스 및 의사결정에 활용
- 빅데이터 및 인공지능 기술이 많이 발전하면서 Factorization Machine, 강화학습, 딥러닝 방식을 이용한 단순 추천 <<< 실시간 조개인화 추천으로 발전 → 강화학습



(<https://jyoonderv.tistory.com/120>)





bigdata-madesimple.com

- **목적:** 검색을 하지 않고도 직접적으로 컨텐츠를 추천하여 사용자의 시간 수고를 낮추고, 예측 못한 컨텐츠를 접하게하여 사용자에게 컨텐츠의 폭을 넓혀줌

- (1) **Relevance:** 추천된 상품이 사용자에게 관련이 있는가?
- (2) **Novelty:** 추천된 상품이 Top10처럼 진부하지 않고 사용자가 탐색하지 못한 색다른 것인가?
- (3) **Serendipity:** 사용자가 이전에 경험하지 못했던 완전 새로운 것인가?
- (4) **Diversity:** 추천된 Top-? 상품에 다양한 제품들이 포함되는가?
- (5) **Feedback:** 추천 성능(서비스 만족도와 경험 개인화) 확인을 위해 추천 이유를 함께 수집 유용

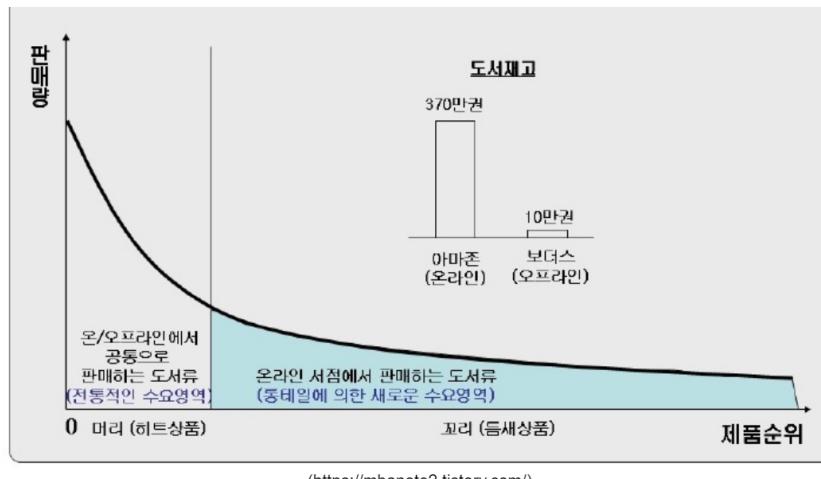
추천방향	방법
비개인적/정형화된	대중적인, 특정 집단이 좋아하는, 개인 선호와 무관한 추천
연관성 기반	특정 물건을 구매하면 다른 물건도 많이 살 것이라는 가정
컨텐츠 기반	개인 선호도를 통해 프로파일링 후 선호도 기반 추천
협업 기반	타인의 선호도 기반 추천

## 2.1 추천 알고리즘의 방향과 역사

### 1) 파레토 법칙 vs 롱테일 법칙:

- 디지털화에 따라 온라인 상에서 롱테일 법칙이 점차 중요해짐
- 글로벌 비즈니스도 파레토 법칙 <<< 롱테일 법칙에 집중하도록 변화
- 아마존닷컴은 98% 비히트상품의 매출이 전체 매출의 25%를 기록하며 패러다임 제시
- 엔터테인먼트와 미디어산업에 집중되고 있지만 경제 사회 전반에 걸쳐 일어날 수 있는 현상
- 패션, 교육, 정치 등의 틈새시장과 오프라인에서도 점차 활용이 증가되고 있음



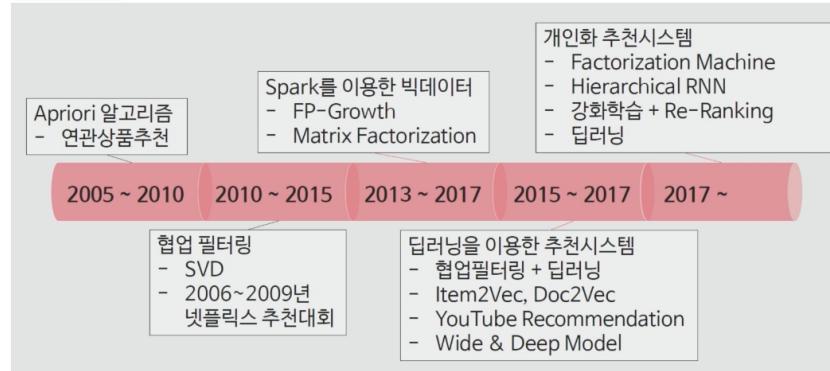


(<https://mbanote2.tistory.com/>)

- 원인1: 컴퓨터라는 도구가 단순한 영역에서 넘어서 모든 것을 통합하여 대중화 됨
- 원인2: 인터넷은 사용자의 접근 비용을 낮춰 유통구조를 대중화시켜 누구나 컨텐츠를 만들수 있는 환경 제공
- 원인3: 구글의 검색서비스, 아이튠즈 음악, 고객이 리뷰와 개인정보를 직접 공유하는 SNS 등 공급자가 제공하는 메인컨텐츠 외에 수요자가 품새컨텐츠를 직접 찾아 모든 상품의 수요와 공급의 연결성이 실시간으로 확대
- [당근마켓, 네이버, 카카오](#)

분야	활용업무
유통업	상품추천, 상품진열, 패키징, 번들링, 방송순서, 카탈로그 배치 등
온라인쇼핑	온라인 쇼핑몰, 인터넷 서점, 온라인 여행사 등
서비스업	백화점, 호텔 등 항후 서비스 제시
금융업	카드, 은행, 보험 등 대출 등의 서비스 받을 고객 식별
의료	특정 증상과 질병 간 연관관계 식별

2) 추천시스템의 역사: 룰레일 법칙을 구축하기 위한 방법론



시점	알고리즘
2000 ~ 2010	연관상품 추천: A Priori
2009 ~ 2009	넷플릭스 추천대회
2010 ~ 2015	협업필터링 추천: Singular Value Decomposition
2013 ~ 2017	Spark 빅데이터 분석: Frequent-Pattern Growth / Matrix Factorization
2015 ~ 2017	딥러닝 추천: Item2Vec / Doc2Vec / Wide & Deep 등
2016 ~ 진행중	개인화 추천: Factorization Machine / Hierarchical RNN / DeepFM 등

(추천시스템의 역사 by T아카데미)

- [Deep User Segment Interest Network Modeling for Click-Through Rate Prediction of Online Advertising](#)

Model	Journal & Year	Affiliation	Paper	Description
FM	ICDM 2010	Osaka Univ.	Factorization Machines	Sparse Feature의 고차원 상호작용 학습 가능
FsNN	ECIR 2016	Univ. College London	Deep Learning over Multi-field Categorical Data: A Case Study on User Response Prediction	FM을 Pre-train하여 DNN을 적용하는 방식 제시
FFM	RecSys 2016	Criteo	Field-aware Factorization Machines for CTR Prediction	고차원 상호작용이 발생하는 Feature Bias 줄임
PNN	ICDM 2016	Carnegie Mellon Univ.	Product-based neural networks for user response prediction	Embedding Layer와 FC Layer와의 Product Layer 제시
<b>Wide &amp; Deep</b>	DLRS 2016	Shanghai Univ. Univ. College London	Wide & Deep Learning for Recommender Systems	고객원과 저자원의 하이브리드 네트워크 구조 제시
<b>Deep &amp; Cross</b>	ADKDD 2017	Google	Deep & Cross Network for Ad Click Predictions	Wide & Deep의 MLP 부분을 Residual Network 변경
AFM	IJCAI 2017	Singapore National Univ.	Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks	FM의 성능을 향상위해 Time-series or Sequence 반영
NFM	SIGIR 2017	Singapore National Univ.	Neural Factorization Machines for Sparse Predictive Analytics	Sparse Feature를 Deep Structure에서 학습 일반화
<b>DeepFM</b>	IJCAI 2017	Harbin ( HIT ) Huawei	DeepFM: A Factorization-Machine based Neural Network for CTR Prediction	추천시스템에 Deep Learning을 결합
FwFM	WWW 2018	Yahoo Alibaba	Field-weighted Factorization Machines for Click-Through Rate Prediction in Display Advertising	Difference Feature 상호작용을 반영하여 작은 피라미티로 높은 성능
xDeepFM	KDD 2018	China (UCTC) Microsoft	xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems	Cross Network을 활용한 Compressed Interaction Network 제시하여 고차 상호작용을 명시적 모델링

DIN	KDD 2018	Alibaba	Deep Interest Network for Click-Through Rate Prediction	시계열 행동 패턴에 따른 관심사 예측 Local Activation Unit 제시
DIEN	AAAI 2019	Alibaba	Deep Interest Evolution Network for Click-Through Rate Prediction	GRU(Chung et al. 2014)를 사용해 Latent Interest 변경을 위한 Attentional Update GRU 제시
DSIN	IJCAI 2019	Zhejiang Univ.	Deep Session Interest Network for Click-Through Rate Prediction	User behavior Sequence의 SessionSegment, Cluster 정보 반영위한 Self-attention + Bi-LSTM 적용
FIBINET	RecSys 2019	Weibo	FiBiNET: Combining Feature Importance and Bilinear feature Interaction for Click-Through Rate Prediction	Sequence-and-Excitation Network(SENET)을 사용하여 Evolving Feature Occupation을 반영

## 2.2 추천 알고리즘 종류

"실시간으로 고성능 알고리즘이 개발 되서 대표 알고리즘 정답이 없고 비공개도 많음"

### (1) 연관성 규칙 알고리즘(Association Rules):

- 가정: 고객의 장바구니에 함께 담긴 상품들은 서로 연관이 있을 것
- 현재는 거의 사용하지 않지만 추천 시스템의 대표적 고전 알고리즘이고 설명력과 이해도가 높음
- 오로지 사람들의 행동 기반으로 분석하여 연관성을 판단하기 위해 지지도(Support), 신뢰도(Confidence), 향상도(Lift)라는 관점으로 제시

### (2) 컨텐츠 기반 필터링(Contents-Based Filtering):

- 고객의 상품 행동(클릭, 구매, 평가 등)을 고려하지 않고, 오로지 상품 특성 만으로 추천하는 알고리즘으로, 상품이 유사한 것들을 추천하는 것
- 사용자가 과거에 경험했던 상품들 중 비슷한 상품을 현재 시점에서 추천

종류	방향
정보 검색	- 방대한 양의 문서들에 대해 질문할 수 있는 시스템 필요 - 도서관의 카탈로그 구축이나 웹페이지의 색인 생성에 활용 - 실시간 조회 기능이 필요했고 빈도수 기반 랭킹 매기는 TF-IDF 방식 활용
정보 필터링	- 정보 검색과 반대로 질문의 답을 찾는 시스템 필요 - 사람들은 모든 기사 <<< 나와 연관있는 + 내가 관심있는 을 원함 - 나의 취향은 잘 변하지 않고(정적) 기사 데이터들은 하루에서 수백번씩 추가되고 변경(동적)되기에 기사보다 사람을 연구

- TF-IDF(Term-Frequency Inverse Document Frequency), Word2Vec, 그리고 문서의 유사성을 찾아주는 Term-Based 기법, Similarity 알고리즘, 문서 다중 분류 등이 활용

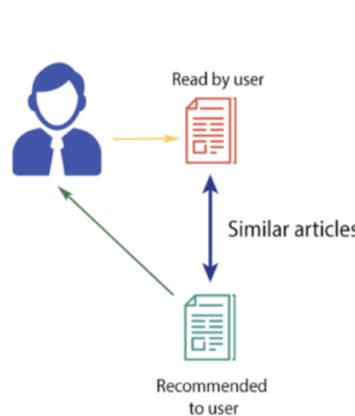


(<https://skyeong.net/265>)

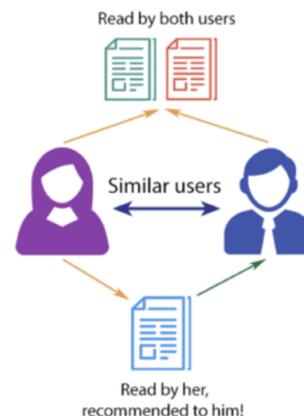
- 컨텐츠 분석: 비정형 데이터로부터 관련 정보를 추출하는 작업으로 Feature Extraction, Vector Representation 등 의 작업 수행
- 유저 프로필 파악: 사용자가 선호하는 상품과 취향을 파악하는 과정으로, 머신러닝 등의 알고리즘으로 사용자 데이터 일반화
- 유사 아이템 선택: 상품 중 가장 적절한 아이템 매핑하는 과정으로, 유사도 등을 이용해 기준 선택과 관련있는 상품 선정

- 추천 상품의 범위가 넓고, 신규 상품이나 비인기 상품 등 의 추천도 가능하며 추천 이유도 설명 가능
- 예측에 도움될 적절한 변수를 찾기 어렵고, 다양한 취향을 반영하기 어려워 반복적으로 동일 상품 추천하기도

### CONTENT-BASED FILTERING



### COLLABORATIVE FILTERING



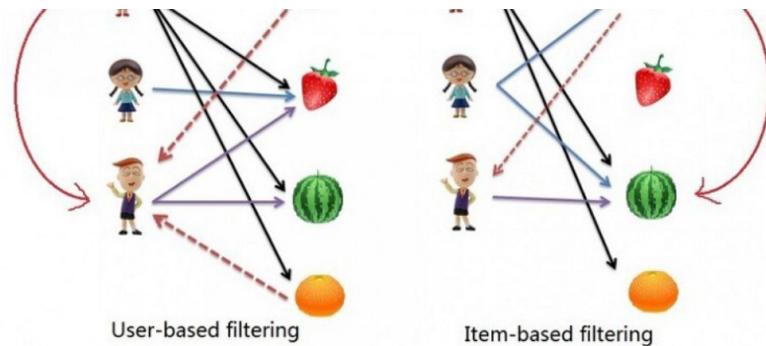
(FUNDAMENTAL 29. 추천시스템 by hwjwjd639)

- 상품 특성기반 추천이기 때문에, 신규 고객 또는 상품의 추천이 불가한 콜드 스타트(Cold Start) 문제 없음

### (3) 협업 필터링(Collaborative Filtering):

- 단순히 키워드 빈도나 연관성 만을 파악하는건 실시간 생성되는 데이터와 변화를 추적하기 어려움
- 사람들의 행동 패턴들을 수집 및 분석하여 나와 가장 유사한 사람 또는 아이템을 추천하는 방식
- 사용자기반 필터링(User-Based Filtering)과 아이템기반 필터링(Item-Based Filtering) 방식





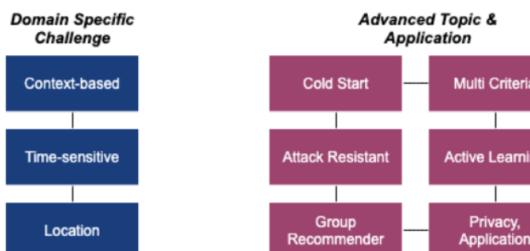
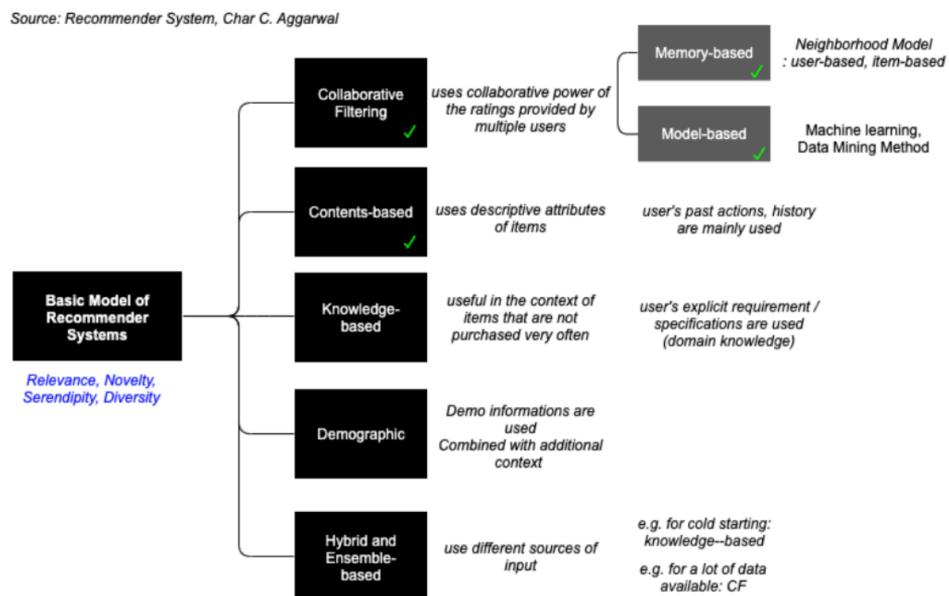
- **User-Based:** 사용자간 유사도가 높을수록 높은 가중치를 부여하기에, 잠재 고객이나 취향이 비슷한 다른 사용자가 선호하는 상품을 추천
- **Item-Based:** A상품과 가장 유사한 Top-N 상품들을 구성하여 특정 유저의 선호도를 예측하여 추천

- 빨뚱맞은 추천도 있을 수 있지만, 코드가 유사한 사람들의 영상이나 상품을 보면서 사람의 행동 패턴을 넓힐
- 내가 봤던 영상들, 시간대, 카테고리 패턴 등을 분석하여 가장 유사한 사람을 찾아낸 후 그 사람들의 아이템에 점수를 산정하여 추천
- 기존 고객의 행동 데이터 기반으로 추천하기에 성능이 좋지만, 신규 고객 또는 상품의 추천이 불가한 콜드 스타트(Cold Start) 문제 존재

#### (4) 하이브리드 방식:

- 강화학습, 딥러닝 등 신기술을 토대로 기존 추천 시스템 알고리즘의 단점을 보완한 새로운 하이브리드 필터링 방법이 개발
- 컨텐츠 기반 필터링 + 협업 필터링을 조합하여 상호보완 방식으로 추천하는 방법
- 콜드 스타트 문제를 해결하기 위해 초반엔 컨텐츠 기반 필터링으로 충분한 데이터가 쌓이면 후반엔 협업 필터링으로 분석
- 고전 알고리즘의 성능이 낮을 수 있지만 고객의 커버리지를 굉장히 넓힐 수 있고, 극단적 컨텐츠를 추천하는 단점을 커버하기 위해 성능이 다소 낮은 협업 필터링으로 선호도 점수를 반영하니, 성능과 커버리지를 모두 성취 가능

### 2.3 알고리즘 비교 및 정리



- **Memory-Based Algorithm:** 사용자 또는 컨텐츠 간 유사도 계산 결과를 기반으로 새로운 사용자를 위한 컨텐츠 추천을 할 때 유사 취향을 가진 다른 사용자나 선호하는 컨텐츠를 추천하는 방법이거나 특정 컨텐츠의 평점을 예측하는 경우 다른 유사한 태깅이 되어 있는 비슷한 컨텐츠의 평점을 과정을 토대로 예측
- **Model-Based Algorithm:** 잠재 요인(Latent Factor) 협업 필터링은 예전이나 현재에서도 많은 기업에서 자주 쓰이는 방법으로 사용자와 아이템 간의 평점 행렬 속에 숨어 있는 잠재 요인 행렬을 분해 및 추출(Matrix Factorization)하여 내적 곱을 통해 사용자가 평가하지 않은 항목들에 대한 평점을 예측
- **Contents-Based Algorithm:** 사용자가 과거에 경험했던 상품들 중 비슷한 상품을 현재 시점에서 추천

- 예시: 사용자가 떡볶이를 시켜 먹었으면, 떡볶이와 관련된 특징(분식?, 매운음식?)을 기반으로 다른 음식 추천

- **Knowledge-Based Algorithm:** 상품의 특성과 명시적 질문으로 사용자의 선호도와 범위 등을 기반으로 추천하기에 주로 사용자들의 구매이력이 적은 경우 사용

• 예시: 음식 배달 서비스를 처음 이용시, 좋아하는 음식 종류, 가격대, 맛 등 의 명시 정보를 기반으로 추천

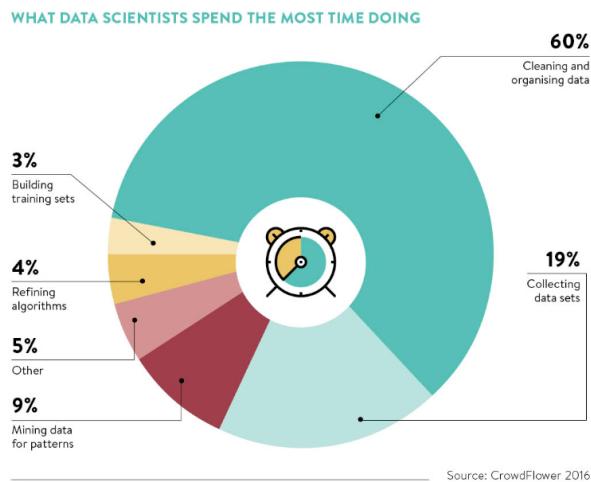
- **Hybrid Algorithm:** 잠재 요인(Latent Factor) 협업 필터링은 예전이나 현재에서도 많은 기업에서 자주 쓰이는 방법으로 사용자와 아이템 간의 평점 행렬 속에 숨어 있는 잠재 요인 행렬을 분해 및 추출(Matrix Factorization) 하여 내적 곱을 통해 사용자가 평가하지 않은 항목들에 대한 평점을 예측

• 예시: 새로운 상품의 평점이 없으면 추천 성능이 떨어지는 협업 필터링과 아이템의 특징을 이용할 수 있는 지식 기반 추천 방법론을 결합

### 3 전처리 방향(Preprocessing)

- 목표:

- 대량으로 수집된 데이터는 그대로 활용 어려움
- 잘못 수집/처리 된 데이터는 엉뚱한 결과를 발생
- 알고리즘이 학습이 가능한 형태로 데이터를 정리



#### 일반적인 전처리 필요항목:

- 데이터 결합
- 결측값 처리
- 이상치 처리
- 자료형 변환
- 데이터 분리
- 데이터 변환
- 스케일 조정

⇒ "알고리즘의 범위와 종류가 다양하여, 각 알고리즘의 입력에 맞게 변환 하는 것이 최선"

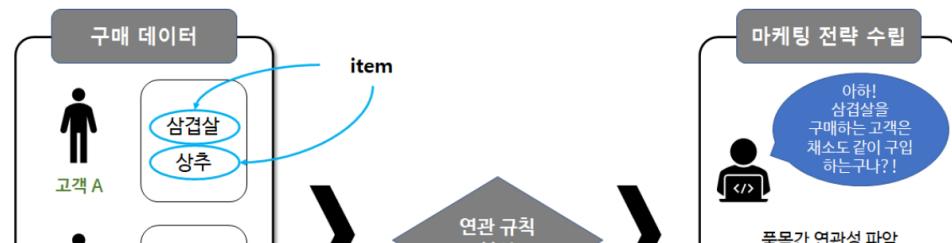
⇒ "횡단면 데이터 및 시계열 데이터가 혼합된 패널데이터의 특성이 많기 때문에 일반 전처리와 시계열 전처리 그리고 데이터 차원증가와 감소 등 다양한 전처리 도구와 능력이 필요하고 다양하고 자유로운 변환이 필요한 고차원 전처리 능력 필요"

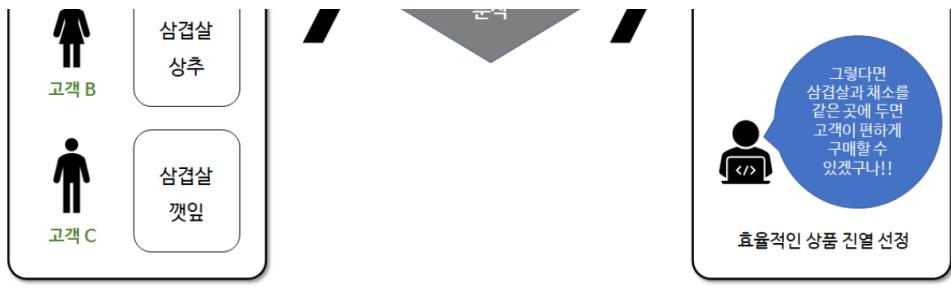
### 4 모델링 방향(Modeling)

#### 4.1 연관규칙분석(A Priori Algorithm, APA)

"경영학에서 장바구니 분석(Market Basket Analysis)로 알려져 있고, 소비자들의 구매이력을 기반으로 A아이템을 구매하는 고객들이 B아이템 구매할 가능성이 높다는 의사결정 알고리즘"

- 어떤 두 아이템 집합이 빈번히 발생하는지 알려주는 규칙을 생성 하는 알고리즘 (Rule-Based Learning)
- 아이템들 간의 if-then 형식의 연관규칙을 찾는 비지도학습의 일종
- 컨텐츠 기반 추천(Contents-Based Recommendation)의 기본이 되는 방법
- Apriori, FP-growth, DHP 등 의 알고리즘이 있으며 Apriori가 비교적 구현이 간단하고 높은 성능





- 데이터 변수들 간의 흥미로운 관계를 탐색하기 위해 고안된 마케팅 및 웹마이닝 등에 주로 사용
- 특정 호텔 서비스 받은 고객이 어떤 서비스를 원하는지 추천하거나 웹페이지 추천 등에 사용
- 각 문장과 문장 내 단어를 거래데이터와 아이템으로 가정하고 문서요약 및 예측에 활용 가능

### 1) 방향:

#### (1) 거래데이터 준비:

ID	Items
1	달걀, 라면, 참치캔
2	라면, 햇반
3	라면, 클라
4	달걀, 라면, 햇반
5	달걀, 클라
6	라면, 클라
7	라면, 햇반
8	달걀, 라면, 클라, 참치캔
9	달걀, 라면, 클라
10	양파

#### (2) 거래별 구매상품 더미변수화:

- 구매이력 상품은 1 그렇지 않은 상품은 0으로 채워진 희소행렬(Sparse Matrix) 생성

ID	달걀	라면	참치캔	햇반	클라	양파
1	1	1	1	0	0	0
2	0	1	0	0	1	0
3	0	1	0	0	0	1
4	1	1	0	0	1	0
5	1	0	0	0	0	1
6	0	1	0	0	0	1
7	0	1	0	0	1	0
8	1	1	1	0	0	1
9	1	1	0	0	0	1
10	0	0	0	0	0	0

#### (3-1) 거래 규칙 파악:

거래 규칙은 굉장히 많지만 모든 규칙이 유용하진 않음

- 달걀 구매자는 라면도 산다, 달걀과 라면 구매자는 참치캔도 산다, ...
- 조건절(If): 달걀 구매자는
- 결과절(Then): 라면도 산다
- 아이템 집합(Item Set): 조건절과 결과절을 구성하는 아이템 집합 (달걀, 라면)으로, 각 절의 아이템은 상호배반(Mutually Exclusive) 형태의 독립이어야 함

#### (3-2) 좋은 규칙 파악:

아이템 집합이 빈번히 발생하는지 파악하기 위해 신뢰도(Confidence), 지지도(Support), 향상도(Lift), 레버리지(Leverage) 사용

- 신뢰도(Confidence): A아이템을 포함하는 거래 중 B아이템이 포함된 비율이 높아야 → 높을수록 A구매 시 B구매율이 높음

$$P(A \rightarrow B) = P(B|A) = \frac{P(A, B)}{P(A)}$$

- 규칙1 신뢰도:  $P(\text{달걀} \rightarrow \text{라면}) = \frac{4/10}{5/10} = \frac{4}{5}$
- 규칙2 신뢰도:  $P(\text{달걀} \rightarrow \text{클라}) = \frac{1/10}{5/10} = \frac{1}{5}$
- 규칙3 신뢰도:  $P(\text{클라} \rightarrow \text{라면}) = \frac{3/10}{3/10} = 1$
- 규칙3 신뢰도 >> 규칙1 신뢰도 >> 규칙2 신뢰도: 신뢰도가 높은 규칙을 더욱 신뢰

- 지지도(Support): A아이템과 B아이템을 동시에 포함하는 비율이 높아야 → 높을수록 자주 발생하는 거래

- 이슈: 달걀+라면 구입은 4건, 클라+라면 3건 임에도 규칙3이 신뢰?
- 규칙1 지지도:  $P(\text{달걀}, \text{라면}) = \frac{4}{10}$
- 규칙2 지지도:  $P(\text{달걀}, \text{클라}) = \frac{1}{10}$
- 규칙3 지지도:  $P(\text{클라}, \text{라면}) = \frac{3}{10}$

- 규칙1 지지도 >> 규칙2 지지도: 지지도가 높은 규칙을 선택

- 향상도(Lift): 신뢰도와 지지도만으로 규칙을 선택하기에 충분한가? → 높을수록 우연(독립)이 아닌 연관성 존재

- 이슈: 규칙1이 좋은건 알겠는데 만약  $P(\text{달걀} \rightarrow \text{라면}) = P(\text{라면}) = \frac{4}{5}$ 라면 달걀은 라면구매에 아무런 도움이 안됨을 의미
- 필요성: 주어진 규칙이 진짜로 의미가 있는지 파악하기 위해  $P(\text{달걀} \rightarrow \text{라면})/P(\text{라면})$  비율 계산

$$\frac{P(A \rightarrow B)}{P(B)}$$

향상도 값	해석
Lift = 1	A아이템은 B아이템 구매에 아무런 영향을 주지 않음
Lift > 1	A아이템은 B아이템 구매에 긍정적 영향
Lift < 1	A아이템은 B아이템 구매에 부정적 영향

- 규칙1 향상도:  $P(\text{달걀} \rightarrow \text{라면})/P(\text{라면}) = \frac{4/5}{8/10} = 1$
- 규칙2 향상도:  $P(\text{달걀} \rightarrow \text{콜라})/P(\text{콜라}) = \frac{1/5}{3/10} = \frac{2}{3}$
- 규칙3 향상도:  $P(\text{콜라} \rightarrow \text{라면})/P(\text{라면}) = \frac{1}{8/10} = \frac{10}{8}$

- 레버리지(Leverage): 향상도가 비율을 이용한다면 레버리지는 차이를 이용

- 필요성: 주어진 규칙이 진짜로 의미가 있다면  $P(\text{달걀}, \text{라면}) \neq P(\text{달걀})P(\text{라면})$

$$P(A, B) - P(A)P(B)$$

레버리지 값	해석
Leverage = 0	A아이템 구매와 B아이템 구매는 서로 독립이며 연관성 없음
Leverage > 0	A아이템은 B아이템 구매에 긍정적 영향
Leverage < 0	A아이템은 B아이템 구매에 부정적 영향

- 규칙1 레버리지:  $P(\text{달걀}, \text{라면}) - P(\text{달걀})P(\text{라면}) = \frac{4}{10} - \frac{5}{10} \cdot \frac{8}{10} = 0$
- 규칙2 레버리지:  $P(\text{달걀}, \text{콜라}) - P(\text{달걀})P(\text{콜라}) = \frac{1}{10} - \frac{5}{10} \cdot \frac{3}{10} = -\frac{5}{100}$
- 규칙3 레버리지:  $P(\text{콜라}, \text{라면}) - P(\text{콜라})P(\text{라면}) = \frac{3}{10} - \frac{3}{10} \cdot \frac{8}{10} = \frac{6}{100}$

- 확신도(Conviction): 어떤 일이 생길것이 아닌 생기지 않을 것에 관심

$$\frac{P(B^c)}{P(A \rightarrow B^c)} = \frac{P(B^c)}{1 - P(A \rightarrow B)}$$

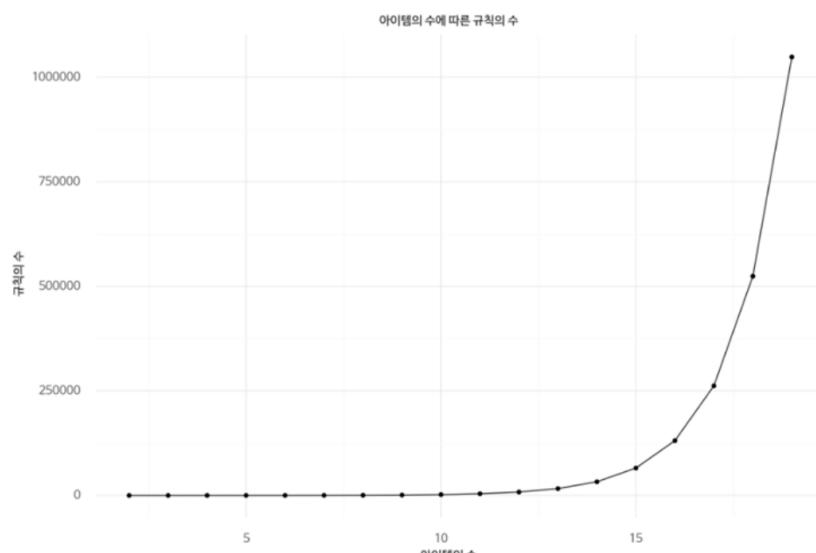
확신도 값	해석
Conviction = 1	A아이템 구매와 B아이템 구매는 서로 독립이며 연관성 없음
Conviction > 1	A아이템은 B아이템 구매에 긍정적 영향
Conviction < 1	A아이템은 B아이템 구매에 부정적 영향

- 규칙1 확신도:  $P(\text{라면}^c)/[1 - P(\text{달걀} \rightarrow \text{라면})] = \frac{2}{10}/[1 - \frac{4}{5}] = 1$
- 규칙2 확신도:  $P(\text{콜라}^c)/[1 - P(\text{달걀} \rightarrow \text{콜라})] = \frac{7}{10}/[1 - \frac{1}{5}] = \frac{7}{8}$
- 규칙3 확신도:  $P(\text{라면}^c)/[1 - P(\text{콜라} \rightarrow \text{라면})] = \frac{2}{10}/[1 - 1] = \infty$

- Others: All-Confidence, Collective Strength, Cosine Similarity 등

## 2) 알고리즘 함수세팅: 최소 지지도와 신뢰도를 갖는 연관규칙 Apriori

- 모든 경우의 수를 탐색하여 지지도, 신뢰도, 향상도가 높은 규칙들을 찾는 것이 가장 이상적
- 아이템의 수가 증가할수록 계산 소요 시간이 기하급수적으로 증가 ( $n$ 개 아이템:  $n(n-1)$  복잡도)



- 빈발 품목 집합(Frequent Item Sets) 만을 고려하여 연관규칙을 생성하는 아이디어가 Apriori

### (1) 빈발 품목 집합(Frequent Item Sets) 생성:

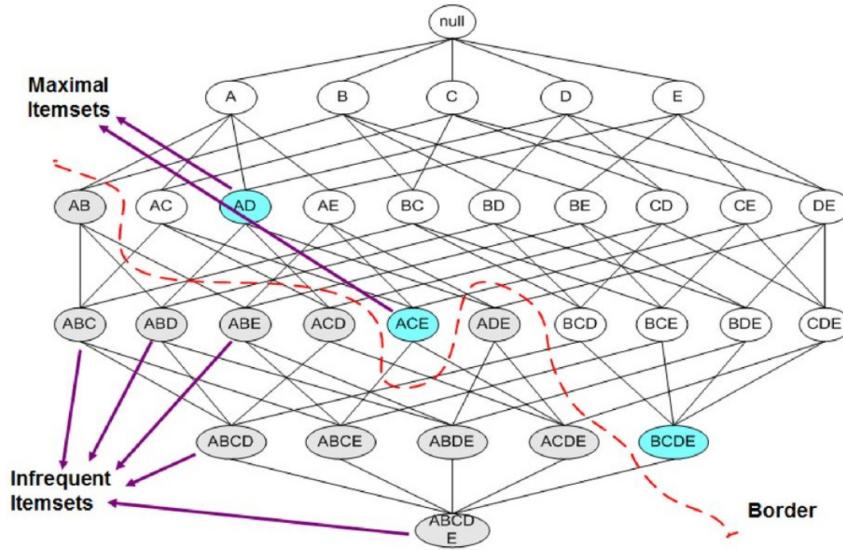
- 빈발 품목 집합(Frequent Item Sets): 각 아이템별로 발생 횟수(비율)가 특정 값 이상인 품목 집합

(1-1) 최소 지지도(최소 발생 비율) 설정: 절대적 방법은 없으며 순전히 분석가의 몫

(1-2) 최소 지지도를 넘는 조합집합의 계산 제외: 아이템 갯수가 늘어나면 엄청난 시간이 소요되니 적절하게 불필요한 계산은 가지치기 (Minimum Support Pruning)

• 초월집합(Superset): 특정 아이템 집합의 지지도가 0.5라고 하면 이를 포함하는 더큰 아이템 집합의 지지도는 0.5를 넘지 못할 것  
이고 이러한 모든 아이템 집합

(1-3) 모든 아이템 집합들에 대해 반복: 최소지지도 이상의 아이템 집합이 더이상 없을때까지 아이템 집합의 크기를 1씩 증가시키며 위 과정을 반복



(https://slidesplayer.org/slide/14686631/)

### (2) 연관 규칙(Associate Rule) 생성:

(2-1) 빈발 품목 집합에서 공집합을 제외한 모든 부분집합 추출

(2-2) 추출한 부분집합 중에서 최소 신뢰도를 넘는 연관 규칙 추출

### 3) 고려 사항:

(1) 유용한 연관규칙 선별: 추정된 연관규칙 또는 연관성이 항상 유용하지는 않을 수 있으며, 너무 빈번한 규칙이나 말이 되지 않는 규칙들도 포함 가능

- ex. 아이폰을 사는 고객은 에어팟을 산다, 식료품 파는 곳에 선풍기가 많이 팔린다

(2) 적절한 품목 그룹핑: 일반적으로 분석 목적에 따라 아이템들을 상위개념으로 묶어서 연관규칙 분석 후, 이를 세분화해서 추가적인 연관규칙 분석을 실행

- ex. 술과 양식의 연관규칙을 찾은 후, 맥주/소주/막걸리 등 또는 피자/햄버거/치킨 등으로 연관규칙 분석 수행

(3) 유용한 연관규칙 해석: 추정된 연관규칙을 의미있도록 해석 되어야 하며 너무 검증 수치가 높은 아이템에만 집중할 필요 없음

- ex. 토요일 방문 고객은 기저귀와 맥주를 산다 → 토요일에 기저귀를 사는 사람은 맥주도 산다
- ex. 아이스크림을 사는 사람은 50대 이상이다 (향상도 < 1) → 아이스크림 판매는 50대 미만에 집중하자

```
In [1]: # 예제 데이터 생성
import numpy as np
import pandas as pd

ts = np.array([['달걀', '라면', '참치캔'],
              ['라면', '콜라'],
              ['라면', '양파'],
              ['달걀', '라면', '콜라'],
              ['달걀', '양파'],
              ['라면', '양파'],
              ['라면', '콜라'],
              ['달걀', '라면', '참치캔', '양파'],
              ['달걀', '라면', '양파'],
              []])
df = pd.DataFrame(ts).reset_index()
df['index'] = df['index'] + 1
df.columns = ['Customer', 'Transaction']
display(df)

# 각 아이템 부록 여부를 one-hot encoding array로 변환
```

```

from mlxtend.preprocessing import TransactionEncoder
df_list = df.Transaction.to_list()
te = TransactionEncoder()
df_prep = te.fit(df_list).transform(df_list)

# one-hot encoding array를 데이터프레임으로 변환
df_prep = pd.DataFrame(df_prep, columns=te.columns_)

# 연관규칙 분석을 위한 A Priori 알고리즘 적용
from mlxtend.frequent_patterns import apriori
item_freq_ap = apriori(df_prep, min_support=0.01, use_colnames=True)

# 연관규칙 복 성능 확인
from mlxtend.frequent_patterns import association_rules
association_rules(item_freq_ap, min_threshold=0.01)

C:\Users\KKK\AppData\Local\Temp\ipykernel_23860\3671193514.py:5: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
    ts = np.array(['[달걀]', '라면', '참치캔'],

```

Customer	Transaction
0	[달걀, 라면, 참치캔]
1	[라면, 풀라]
2	[라면, 양파]
3	[달걀, 라면, 풀라]
4	[달걀, 양파]
5	[라면, 양파]
6	[라면, 풀라]
7	[달걀, 라면, 참치캔, 양파]
8	[달걀, 라면, 양파]
9	[]

Out[1]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(달걀)	(라면)	0.5	0.8	0.4	0.800000	1.000000	0.00	1.000000
1	(라면)	(달걀)	0.8	0.5	0.4	0.500000	1.000000	0.00	1.000000
2	(달걀)	(양파)	0.5	0.5	0.3	0.600000	1.200000	0.05	1.250000
3	(양파)	(달걀)	0.5	0.5	0.3	0.600000	1.200000	0.05	1.250000
4	(달걀)	(참치캔)	0.5	0.2	0.2	0.400000	2.000000	0.10	1.333333
5	(참치캔)	(달걀)	0.2	0.5	0.2	1.000000	2.000000	0.10	inf
6	(달걀)	(풀라)	0.5	0.3	0.1	0.200000	0.666667	-0.05	0.875000
7	(풀라)	(달걀)	0.3	0.5	0.1	0.333333	0.666667	-0.05	0.750000
8	(양파)	(라면)	0.5	0.8	0.4	0.800000	1.000000	0.00	1.000000
9	(라면)	(양파)	0.8	0.5	0.4	0.500000	1.000000	0.00	1.000000
10	(참치캔)	(라면)	0.2	0.8	0.2	1.000000	1.250000	0.04	inf
11	(라면)	(참치캔)	0.8	0.2	0.2	0.250000	1.250000	0.04	1.066667
12	(풀라)	(라면)	0.3	0.8	0.3	1.000000	1.250000	0.06	inf
13	(라면)	(풀라)	0.8	0.3	0.3	0.375000	1.250000	0.06	1.120000
14	(참치캔)	(양파)	0.2	0.5	0.1	0.500000	1.000000	0.00	1.000000
15	(양파)	(참치캔)	0.5	0.2	0.1	0.200000	1.000000	0.00	1.000000
16	(달걀, 양파)	(라면)	0.3	0.8	0.2	0.666667	0.833333	-0.04	0.600000
17	(달걀, 라면)	(양파)	0.4	0.5	0.2	0.500000	1.000000	0.00	1.000000
18	(양파, 라면)	(달걀)	0.4	0.5	0.2	0.500000	1.000000	0.00	1.000000
19	(달걀)	(양파, 라면)	0.5	0.4	0.2	0.400000	1.000000	0.00	1.000000
20	(양파)	(달걀, 라면)	0.5	0.4	0.2	0.400000	1.000000	0.00	1.000000
21	(라면)	(달걀, 양파)	0.8	0.3	0.2	0.250000	0.833333	-0.04	0.933333
22	(달걀, 참치캔)	(라면)	0.2	0.8	0.2	1.000000	1.250000	0.04	inf
23	(달걀, 라면)	(참치캔)	0.4	0.2	0.2	0.500000	2.500000	0.12	1.600000
24	(참치캔, 라면)	(달걀)	0.2	0.5	0.2	1.000000	2.000000	0.10	inf
25	(달걀)	(참치캔, 라면)	0.5	0.2	0.2	0.400000	2.000000	0.10	1.333333
26	(참치캔)	(달걀, 라면)	0.2	0.4	0.2	1.000000	2.500000	0.12	inf
27	(라면)	(달걀, 참치캔)	0.8	0.2	0.2	0.250000	1.250000	0.04	1.066667
28	(달걀, 풀라)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
29	(달걀, 라면)	(풀라)	0.4	0.3	0.1	0.250000	0.833333	-0.02	0.933333
30	(풀라, 라면)	(달걀)	0.3	0.5	0.1	0.333333	0.666667	-0.05	0.750000
31	(달걀)	(풀라, 라면)	0.5	0.3	0.1	0.200000	0.666667	-0.05	0.875000
32	(풀라)	(달걀, 라면)	0.3	0.4	0.1	0.333333	0.833333	-0.02	0.900000
33	(라면)	(달걀, 풀라)	0.8	0.1	0.1	0.125000	1.250000	0.02	1.028571

34	(달걀, 참치캔)	(양파)	0.2	0.5	0.1	0.500000	1.000000	0.00	1.000000
35	(달걀, 양파)	(참치캔)	0.3	0.2	0.1	0.333333	1.666667	0.04	1.200000
36	(참치캔, 양파)	(달걀)	0.1	0.5	0.1	1.000000	2.000000	0.05	inf
37	(달걀)	(참치캔, 양파)	0.5	0.1	0.1	0.200000	2.000000	0.05	1.125000
38	(참치캔)	(달걀, 양파)	0.2	0.3	0.1	0.500000	1.666667	0.04	1.400000
39	(양파)	(달걀, 참치캔)	0.5	0.2	0.1	0.200000	1.000000	0.00	1.000000
40	(참치캔, 양파)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
41	(참치캔, 라면)	(양파)	0.2	0.5	0.1	0.500000	1.000000	0.00	1.000000
42	(양파, 라면)	(참치캔)	0.4	0.2	0.1	0.250000	1.250000	0.02	1.066667
43	(참치캔)	(양파, 라면)	0.2	0.4	0.1	0.500000	1.250000	0.02	1.200000
44	(양파)	(참치캔, 라면)	0.5	0.2	0.1	0.200000	1.000000	0.00	1.000000
45	(라면)	(참치캔, 양파)	0.8	0.1	0.1	0.125000	1.250000	0.02	1.028571
46	(달걀, 참치캔, 양파)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
47	(달걀, 참치캔, 라면)	(양파)	0.2	0.5	0.1	0.500000	1.000000	0.00	1.000000
48	(달걀, 양파, 라면)	(참치캔)	0.2	0.2	0.1	0.500000	2.500000	0.06	1.600000
49	(참치캔, 양파, 라면)	(달걀)	0.1	0.5	0.1	1.000000	2.000000	0.05	inf
50	(달걀, 참치캔)	(양파, 라면)	0.2	0.4	0.1	0.500000	1.250000	0.02	1.200000
51	(달걀, 양파)	(참치캔, 라면)	0.3	0.2	0.1	0.333333	1.666667	0.04	1.200000
52	(달걀, 라면)	(참치캔, 양파)	0.4	0.1	0.1	0.250000	2.500000	0.06	1.200000
53	(참치캔, 양파)	(달걀, 라면)	0.1	0.4	0.1	1.000000	2.500000	0.06	inf
54	(참치캔, 라면)	(달걀, 양파)	0.2	0.3	0.1	0.500000	1.666667	0.04	1.400000
55	(양파, 라면)	(달걀, 참치캔)	0.4	0.2	0.1	0.250000	1.250000	0.02	1.066667
56	(달걀)	(참치캔, 양파, 라면)	0.5	0.1	0.1	0.200000	2.000000	0.05	1.125000
57	(참치캔)	(달걀, 양파, 라면)	0.2	0.2	0.1	0.500000	2.500000	0.06	1.600000
58	(양파)	(달걀, 참치캔, 라면)	0.5	0.2	0.1	0.200000	1.000000	0.00	1.000000
59	(라면)	(달걀, 참치캔, 양파)	0.8	0.1	0.1	0.125000	1.250000	0.02	1.028571

## 4.2 이론예제 실습: A Priori 이해

In [2]: # 예제 데이터 생성

```
import numpy as np
import pandas as pd

ts = np.array([[['달걀', '라면', '참치캔'],
                ['라면', '콜라'],
                ['라면', '양파'],
                ['달걀', '라면', '콜라'],
                ['달걀', '양파'],
                ['라면', '양파'],
                ['라면', '콜라'],
                ['달걀', '라면', '참치캔', '양파'],
                ['달걀', '라면', '양파'],
                []]])
df = pd.DataFrame(ts).reset_index()
df['index'] = df['index'] + 1
df.columns = ['Customer', 'Transaction']
df
```

C:\Users\KKK\AppData\Local\Temp\ipykernel\_23860\197202095.py:5: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.

```
ts = np.array([['달걀', '라면', '참치캔'],
```

Out [2]:

	Customer	Transaction
0	1	[달걀, 라면, 참치캔]
1	2	[라면, 콜라]
2	3	[라면, 양파]
3	4	[달걀, 라면, 콜라]
4	5	[달걀, 양파]
5	6	[라면, 양파]
6	7	[라면, 콜라]
7	8	[달걀, 라면, 참치캔, 양파]
8	9	[달걀, 라면, 양파]
9	10	[]

In [3]: # 각 아이템 포함 여부를 one-hot encoding array로 변환

```
from mlxtend.preprocessing import TransactionEncoder

df_list = df.Transaction.to_list()
te = TransactionEncoder()
df_prep = te.fit(df_list).transform(df_list)
df_prep
```

```
Out[3]: array([[ True,  True, False,  True, False],
 [False,  True, False, False,  True],
 [False,  True,  True, False, False],
 [ True,  True, False, False,  True],
 [ True, False,  True, False, False],
 [False,  True,  True, False, False],
 [False,  True, False, False,  True],
 [ True,  True,  True, False, False],
 [ True,  True,  True, False, False],
 [False, False, False, False, False]])
```

```
In [4]: # 인코더에 저장된 아이템명  
te.columns_
```

```
Out[4]: ['달걀', '라면', '양파', '참치캔', '콜라']
```

```
In [5]: # one-hot encoding array를 데이터프레임으로 변환  
df_prep = pd.DataFrame(df_prep, columns=te.columns_)
```

```
Out[5]:
```

	달걀	라면	양파	참치캔	콜라
0	True	True	False	True	False
1	False	True	False	False	True
2	False	True	True	False	False
3	True	True	False	False	True
4	True	False	True	False	False
5	False	True	True	False	False
6	False	True	False	False	True
7	True	True	True	True	False
8	True	True	True	False	False
9	False	False	False	False	False

```
In [6]: # one-hot encoding array를 데이터프레임으로 변환  
# 1과 0의 값으로 변환도 가능  
df_prep = pd.DataFrame(df_prep.astype('int'), columns=te.columns_)
```

```
Out[6]:
```

	달걀	라면	양파	참치캔	콜라
0	1	1	0	1	0
1	0	1	0	0	1
2	0	1	1	0	0
3	1	1	0	0	1
4	1	0	1	0	0
5	0	1	1	0	0
6	0	1	0	0	1
7	1	1	1	1	0
8	1	1	1	0	0
9	0	0	0	0	0

```
In [7]: # 연관규칙 분석을 위한 A Priori 알고리즘 적용  
from mlxtend.frequent_patterns import apriori
```

```
apriori(df_prep)
```

```
C:\Users\KK\AppData\Roaming\Python\Python39\site-packages\mlxtend\frequent_patterns\fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types result in worse computational performance and their support might be discontinued in the future. Please use a DataFrame with bool type  
warnings.warn(
```

```
Out[7]:
```

	support	itemsets
0	0.5	(0)
1	0.8	(1)
2	0.5	(2)

```
In [8]: # 연관규칙 분석을 위한 A Priori 알고리즘 적용  
# 특정 지지도 이상만 출력하기 위해 min_support 파라미터 활용  
# 아이템명으로 출력하기 위해 use_colnames 파라미터 활용  
from mlxtend.frequent_patterns import apriori
```

```
item_freq_ap = apriori(df_prep, min_support=0.1, use_colnames=True)
```

```
C:\Users\KK\AppData\Roaming\Python\Python39\site-packages\mlxtend\frequent_patterns\fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types result in worse computational performance and their support might be discontinued in the future. Please use a DataFrame with bool type  
warnings.warn(
```

```
Out[8]:
```

	support	itemsets
0	0.5	(달걀)
1	0.8	(라면)
2	0.5	(양파)

3	0.2	(참치캔)
4	0.3	(콜라)
5	0.4	(달걀, 라면)
6	0.3	(달걀, 양파)
7	0.2	(달걀, 참치캔)
8	0.1	(달걀, 콜라)
9	0.4	(양파, 라면)
10	0.2	(참치캔, 라면)
11	0.3	(콜라, 라면)
12	0.1	(참치캔, 양파)
13	0.2	(달걀, 양파, 라면)
14	0.2	(달걀, 참치캔, 라면)
15	0.1	(달걀, 콜라, 라면)
16	0.1	(달걀, 참치캔, 양파)
17	0.1	(참치캔, 양파, 라면)
18	0.1	(달걀, 참치캔, 양파, 라면)

```
In [9]: # 지지도에 따른 아이템 집합 갯수 확인
item_freq_ap.itemsets.apply(lambda x: len(x))
```

```
Out[9]: 0    1
1    1
2    1
3    1
4    1
5    2
6    2
7    2
8    2
9    2
10   2
11   2
12   2
13   3
14   3
15   3
16   3
17   3
18   4
Name: itemsets, dtype: int64
```

```
In [10]: # 특정 갯수 이상의 아이템 집합만 추출
item_freq_ap[item_freq_ap.itemsets.apply(lambda x: len(x) > 2)]
```

```
Out[10]:   support      itemsets
13    0.2    (달걀, 양파, 라면)
14    0.2    (달걀, 참치캔, 라면)
15    0.1    (달걀, 콜라, 라면)
16    0.1    (달걀, 참치캔, 양파)
17    0.1    (참치캔, 양파, 라면)
18    0.1    (달걀, 참치캔, 양파, 라면)
```

```
In [11]: # 특정 아이템 포함 여부 확인
item_freq_ap.itemsets.apply(lambda x: '라면' in list(x))
```

```
Out[11]: 0    False
1     True
2    False
3    False
4    False
5     True
6    False
7    False
8    False
9     True
10   True
11   True
12   False
13   True
14   True
15   True
16   False
17   True
18   True
Name: itemsets, dtype: bool
```

```
In [12]: # 특정 아이템과 함께 구매하는 아이템 집합만 추출
item_freq_ap[item_freq_ap.itemsets.apply(lambda x: '라면' in list(x))]
```

```
Out[12]:   support      itemsets
1    0.8    (라면)
2    0.4    (달걀, 라면)
```

9	0.4	(글글, 라면)
10	0.2	(참치캔, 라면)
11	0.3	(콜라, 라면)
13	0.2	(달걀, 양파, 라면)
14	0.2	(달걀, 참치캔, 라면)
15	0.1	(달걀, 콜라, 라면)
17	0.1	(참치캔, 양파, 라면)
18	0.1	(달걀, 참치캔, 양파, 라면)

```
In [13]: # 연관규칙 별 성능 확인
# metric: 'support', 'confidence', 'lift', 'leverage', 'conviction'
from mlxtend.frequent_patterns import association_rules

item_measures = association_rules(item_freq_ap, min_threshold=0.1)
item_measures
```

Out [13]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(달걀)	(라면)	0.5	0.8	0.4	0.800000	1.000000	0.00	1.000000
1	(라면)	(달걀)	0.8	0.5	0.4	0.500000	1.000000	0.00	1.000000
2	(달걀)	(양파)	0.5	0.5	0.3	0.600000	1.200000	0.05	1.250000
3	(양파)	(달걀)	0.5	0.5	0.3	0.600000	1.200000	0.05	1.250000
4	(달걀)	(참치캔)	0.5	0.2	0.2	0.400000	2.000000	0.10	1.333333
5	(참치캔)	(달걀)	0.2	0.5	0.2	1.000000	2.000000	0.10	inf
6	(달걀)	(콜라)	0.5	0.3	0.1	0.200000	0.666667	-0.05	0.875000
7	(콜라)	(달걀)	0.3	0.5	0.1	0.333333	0.666667	-0.05	0.750000
8	(양파)	(라면)	0.5	0.8	0.4	0.800000	1.000000	0.00	1.000000
9	(라면)	(양파)	0.8	0.5	0.4	0.500000	1.000000	0.00	1.000000
10	(참치캔)	(라면)	0.2	0.8	0.2	1.000000	1.250000	0.04	inf
11	(라면)	(참치캔)	0.8	0.2	0.2	0.250000	1.250000	0.04	1.066667
12	(콜라)	(라면)	0.3	0.8	0.3	1.000000	1.250000	0.06	inf
13	(라면)	(콜라)	0.8	0.3	0.3	0.375000	1.250000	0.06	1.120000
14	(참치캔)	(양파)	0.2	0.5	0.1	0.500000	1.000000	0.00	1.000000
15	(양파)	(참치캔)	0.5	0.2	0.1	0.200000	1.000000	0.00	1.000000
16	(달걀, 양파)	(라면)	0.3	0.8	0.2	0.666667	0.833333	-0.04	0.600000
17	(달걀, 라면)	(양파)	0.4	0.5	0.2	0.500000	1.000000	0.00	1.000000
18	(양파, 라면)	(달걀)	0.4	0.5	0.2	0.500000	1.000000	0.00	1.000000
19	(달걀)	(양파, 라면)	0.5	0.4	0.2	0.400000	1.000000	0.00	1.000000
20	(양파)	(달걀, 라면)	0.5	0.4	0.2	0.400000	1.000000	0.00	1.000000
21	(라면)	(달걀, 양파)	0.8	0.3	0.2	0.250000	0.833333	-0.04	0.933333
22	(달걀, 참치캔)	(라면)	0.2	0.8	0.2	1.000000	1.250000	0.04	inf
23	(달걀, 라면)	(참치캔)	0.4	0.2	0.2	0.500000	2.500000	0.12	1.600000
24	(참치캔, 라면)	(달걀)	0.2	0.5	0.2	1.000000	2.000000	0.10	inf
25	(달걀)	(참치캔, 라면)	0.5	0.2	0.2	0.400000	2.000000	0.10	1.333333
26	(참치캔)	(달걀, 라면)	0.2	0.4	0.2	1.000000	2.500000	0.12	inf
27	(라면)	(달걀, 참치캔)	0.8	0.2	0.2	0.250000	1.250000	0.04	1.066667
28	(달걀, 콜라)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
29	(달걀, 라면)	(콜라)	0.4	0.3	0.1	0.250000	0.833333	-0.02	0.933333
30	(콜라, 라면)	(달걀)	0.3	0.5	0.1	0.333333	0.666667	-0.05	0.750000
31	(달걀)	(콜라, 라면)	0.5	0.3	0.1	0.200000	0.666667	-0.05	0.875000
32	(콜라)	(달걀, 라면)	0.3	0.4	0.1	0.333333	0.833333	-0.02	0.900000
33	(라면)	(달걀, 콜라)	0.8	0.1	0.1	0.125000	1.250000	0.02	1.028571
34	(달걀, 참치캔)	(양파)	0.2	0.5	0.1	0.500000	1.000000	0.00	1.000000
35	(달걀, 양파)	(참치캔)	0.3	0.2	0.1	0.333333	1.666667	0.04	1.200000
36	(참치캔, 양파)	(달걀)	0.1	0.5	0.1	1.000000	2.000000	0.05	inf
37	(달걀)	(참치캔, 양파)	0.5	0.1	0.1	0.200000	2.000000	0.05	1.125000
38	(참치캔)	(달걀, 양파)	0.2	0.3	0.1	0.500000	1.666667	0.04	1.400000
39	(양파)	(달걀, 참치캔)	0.5	0.2	0.1	0.200000	1.000000	0.00	1.000000
40	(참치캔, 양파)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
41	(참치캔, 라면)	(양파)	0.2	0.5	0.1	0.500000	1.000000	0.00	1.000000
42	(양파, 라면)	(참치캔)	0.4	0.2	0.1	0.250000	1.250000	0.02	1.066667
43	(참치캔)	(양파, 라면)	0.2	0.4	0.1	0.500000	1.250000	0.02	1.200000

44	(양파)	(참치캔, 라면)	0.5	0.2	0.1	0.200000	1.000000	0.00	1.000000
45	(라면)	(참치캔, 양파)	0.8	0.1	0.1	0.125000	1.250000	0.02	1.028571
46	(달걀, 참치캔, 양파)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
47	(달걀, 참치캔, 라면)	(양파)	0.2	0.5	0.1	0.500000	1.000000	0.00	1.000000
48	(달걀, 양파, 라면)	(참치캔)	0.2	0.2	0.1	0.500000	2.500000	0.06	1.600000
49	(참치캔, 양파, 라면)	(달걀)	0.1	0.5	0.1	1.000000	2.000000	0.05	inf
50	(달걀, 참치캔)	(양파, 라면)	0.2	0.4	0.1	0.500000	1.250000	0.02	1.200000
51	(달걀, 양파)	(참치캔, 라면)	0.3	0.2	0.1	0.333333	1.666667	0.04	1.200000
52	(달걀, 라면)	(참치캔, 양파)	0.4	0.1	0.1	0.250000	2.500000	0.06	1.200000
53	(참치캔, 양파)	(달걀, 라면)	0.1	0.4	0.1	1.000000	2.500000	0.06	inf
54	(참치캔, 라면)	(달걀, 양파)	0.2	0.3	0.1	0.500000	1.666667	0.04	1.400000
55	(양파, 라면)	(달걀, 참치캔)	0.4	0.2	0.1	0.250000	1.250000	0.02	1.066667
56	(달걀)	(참치캔, 양파, 라면)	0.5	0.1	0.1	0.200000	2.000000	0.05	1.125000
57	(참치캔)	(달걀, 양파, 라면)	0.2	0.2	0.1	0.500000	2.500000	0.06	1.600000
58	(양파)	(달걀, 참치캔, 라면)	0.5	0.2	0.1	0.200000	1.000000	0.00	1.000000
59	(라면)	(달걀, 참치캔, 양파)	0.8	0.1	0.1	0.125000	1.250000	0.02	1.028571

In [14]: # 특정 아이템과 함께 구매하는 아이템 집합만 추출  
item\_measures[item\_measures.consequents.apply(lambda x: '라면' in list(x))]

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(달걀)	(라면)	0.5	0.8	0.4	0.800000	1.000000	0.00	1.000000
8	(양파)	(라면)	0.5	0.8	0.4	0.800000	1.000000	0.00	1.000000
10	(참치캔)	(라면)	0.2	0.8	0.2	1.000000	1.250000	0.04	inf
12	(콜라)	(라면)	0.3	0.8	0.3	1.000000	1.250000	0.06	inf
16	(달걀, 양파)	(라면)	0.3	0.8	0.2	0.666667	0.833333	-0.04	0.600000
19	(달걀)	(양파, 라면)	0.5	0.4	0.2	0.400000	1.000000	0.00	1.000000
20	(양파)	(달걀, 라면)	0.5	0.4	0.2	0.400000	1.000000	0.00	1.000000
22	(달걀, 참치캔)	(라면)	0.2	0.8	0.2	1.000000	1.250000	0.04	inf
25	(달걀)	(참치캔, 라면)	0.5	0.2	0.2	0.400000	2.000000	0.10	1.333333
26	(참치캔)	(달걀, 라면)	0.2	0.4	0.2	1.000000	2.500000	0.12	inf
28	(달걀, 콜라)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
31	(달걀)	(콜라, 라면)	0.5	0.3	0.1	0.200000	0.666667	-0.05	0.875000
32	(콜라)	(달걀, 라면)	0.3	0.4	0.1	0.333333	0.833333	-0.02	0.900000
40	(참치캔, 양파)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
43	(참치캔)	(양파, 라면)	0.2	0.4	0.1	0.500000	1.250000	0.02	1.200000
44	(양파)	(참치캔, 라면)	0.5	0.2	0.1	0.200000	1.000000	0.00	1.000000
46	(달걀, 참치캔, 양파)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
50	(달걀, 참치캔)	(양파, 라면)	0.2	0.4	0.1	0.500000	1.250000	0.02	1.200000
51	(달걀, 양파)	(참치캔, 라면)	0.3	0.2	0.1	0.333333	1.666667	0.04	1.200000
53	(참치캔, 양파)	(달걀, 라면)	0.1	0.4	0.1	1.000000	2.500000	0.06	inf
56	(달걀)	(참치캔, 양파, 라면)	0.5	0.1	0.1	0.200000	2.000000	0.05	1.125000
57	(참치캔)	(달걀, 양파, 라면)	0.2	0.2	0.1	0.500000	2.500000	0.06	1.600000
58	(양파)	(달걀, 참치캔, 라면)	0.5	0.2	0.1	0.200000	1.000000	0.00	1.000000

In [15]: # 가장 끌리 진을해도 괜찮은 아이템  
item\_measures[item\_measures.consequents == set(['라면'])].sort\_values(by='lift')

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
16	(달걀, 양파)	(라면)	0.3	0.8	0.2	0.666667	0.833333	-0.04	0.6
0	(달걀)	(라면)	0.5	0.8	0.4	0.800000	1.000000	0.00	1.0
8	(양파)	(라면)	0.5	0.8	0.4	0.800000	1.000000	0.00	1.0
10	(참치캔)	(라면)	0.2	0.8	0.2	1.000000	1.250000	0.04	inf
12	(콜라)	(라면)	0.3	0.8	0.3	1.000000	1.250000	0.06	inf
22	(달걀, 참치캔)	(라면)	0.2	0.8	0.2	1.000000	1.250000	0.04	inf
28	(달걀, 콜라)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
40	(참치캔, 양파)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf
46	(달걀, 참치캔, 양파)	(라면)	0.1	0.8	0.1	1.000000	1.250000	0.02	inf

- 사용법 정리: 확실한 정답은 없고 어떻게 활용 하냐에 따라 비즈니스 성패

(1) 최소 Support + 최소 Confidence로 의미없는 규칙은 제외

• 전체 거래 중에서 너무 빈도가 적거나 조건 부록률이 낮은 규칙 제외

## (2) Lift를 내림차순 정렬하여 의미있는 규칙을 평가

- 향상도가 큰 규칙이 조건절이 결과절에 연관성이 높음을 의미

### 4.3 이론예제 실습: FP-Growth 이해

1) 알고리즘 함수세팅: - FP-Tree 구조를 사용하여 A Priori 알고리즘의 속도측면의 단점을 개선한 알고리즘

(1) 모든 거래를 확인하여 각 아이템마다 지지도(support) 계산 → Apriori 와 동일

(2) 최소 지지도 이상의 아이템만 선택 → Apriori 와 동일

(3) 모든 거래에서 빈도가 높은 아이템 순서대로 정렬

(4) 부모노드를 중심으로 거래를 자식노드로 추가 해주면서 Tree 생성

(5) 새로운 아이템이 나올 경우에는 부모노드부터 시작하고, 그렇지 않으면 기존 노드에서 확장

(6) 위 과정을 모든 거래에 반복하여 FP-Tree 생성 후 최소지지도 이상의 패턴만 추출

#### A Priori

#### FP-Growth

장점	<ul style="list-style-type: none"> <li>- 원리가 간단하여 사용자가 쉽게 이해 및 의미 파악 가능</li> <li>- 유의한 연관성을 갖는 구매패턴 추출</li> </ul>	<ul style="list-style-type: none"> <li>- A Priori 알고리즘보다 빠르고 2번 탐색으로 종료</li> <li>- 후보 아이템 집합 생성 미필요</li> </ul>
단점	<ul style="list-style-type: none"> <li>- 데이터가 클 경우 속도가 느리고 연산량 많음</li> <li>- 지난치게 많은 연관상품이 나타나기에 상관관계는 있더라도 인과관계(연관성) 파악 어렵게</li> </ul>	<ul style="list-style-type: none"> <li>- 대용량 데이터에서 메모리 사용이 비효율적</li> <li>- A Priori 알고리즘 대비 설계 어려움</li> <li>- 지지도 계산을 위해선 FP-Tree를 만들어야 여전히 연관성 파악 어렵게</li> </ul>

```
In [16]: # 예제 데이터 생성
import numpy as np
import pandas as pd

ts = np.array([
    ['달걀'], ['라면'], ['참치캔'],
    ['라면'], ['콜라'],
    ['라면'], ['양파'],
    ['달걀'], ['라면'], ['콜라'],
    ['달걀'], ['양파'],
    ['라면'], ['양파'],
    ['라면'], ['콜라'],
    ['달걀'], ['라면'], ['참치캔'], ['양파'],
    ['달걀'], ['라면'], ['양파'],
    []
])
df = pd.DataFrame(ts).reset_index()
df['index'] = df['index'] + 1
df.columns = ['Customer', 'Transaction']
display(df)

# 각 아이템 포함 여부를 one-hot encoding array로 변환
from mlxtend.preprocessing import TransactionEncoder
df_list = df.Transaction.to_list()
te = TransactionEncoder()
df_prep = te.fit(df_list).transform(df_list)

# one-hot encoding array를 데이터프레임으로 변환
df_prep = pd.DataFrame(df_prep, columns=te.columns_)

# 연관규칙 분석을 위한 A Priori 알고리즘 적용
from mlxtend.frequent_patterns import apriori
item_freq_ap = apriori(df_prep, min_support=0.01, use_colnames=True)
item_freq_ap.sort_values(by='support', ascending=False)
```

C:\Users\KKK\AppData\Local\Temp\ipykernel\_23860\1150898423.py:5: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.

ts = np.array([['달걀'], ['라면'], ['참치캔'],

Customer	Transaction
0	[달걀, 라면, 참치캔]
1	[라면, 콜라]
2	[라면, 양파]
3	[달걀, 라면, 콜라]
4	[달걀, 양파]
5	[라면, 양파]
6	[라면, 콜라]
7	[달걀, 라면, 참치캔, 양파]
8	[달걀, 라면, 양파]
9	[]

```
Out[16]:
```

	support	itemsets
1	0.8	(라면)
0	0.5	(달걀)
2	0.5	(양파)

9	0.4	(멸글, 디귿)
4	0.3	(콜라)
6	0.3	(달걀, 양파)
11	0.3	(콜라, 라면)
7	0.2	(달걀, 참치캔)
3	0.2	(참치캔)
10	0.2	(참치캔, 라면)
13	0.2	(달걀, 양파, 라면)
14	0.2	(달걀, 참치캔, 라면)
8	0.1	(달걀, 콜라)
12	0.1	(참치캔, 양파)
15	0.1	(달걀, 콜라, 라면)
16	0.1	(달걀, 참치캔, 양파)
17	0.1	(참치캔, 양파, 라면)
18	0.1	(달걀, 참치캔, 양파, 라면)

```
In [17]: # 연관규칙 분석을 위한 FP-Growth 알고리즘 적용
from mlxtend.frequent_patterns import fpgrowth

item_freq_fpg = fpgrowth(df_prep, min_support=0.1, use_colnames=True)
item_freq_fpg.sort_values(by='support', ascending=False)
```

	support	itemsets
0	0.8	(라면)
4	0.5	(양파)
1	0.5	(달걀)
5	0.4	(달걀, 라면)
18	0.4	(양파, 라면)
3	0.3	(콜라)
6	0.3	(달걀, 양파)
15	0.3	(콜라, 라면)
7	0.2	(달걀, 양파, 라면)
8	0.2	(달걀, 참치캔)
2	0.2	(참치캔)
11	0.2	(달걀, 참치캔, 라면)
9	0.2	(참치캔, 라면)
10	0.1	(참치캔, 양파)
12	0.1	(달걀, 참치캔, 양파)
13	0.1	(참치캔, 양파, 라면)
14	0.1	(달걀, 참치캔, 양파, 라면)
16	0.1	(달걀, 콜라)
17	0.1	(달걀, 콜라, 라면)

```
In [18]: # A Priori vs FP-Growth
item_freq_compare = pd.concat([item_freq_ap.sort_values(by='support', ascending=False).reset_index().iloc[:,1],
                               item_freq_fpg.sort_values(by='support', ascending=False).reset_index().iloc[:,1],
                               item_freq_ap.sort_values(by='support', ascending=False).reset_index().iloc[:,2],
                               item_freq_fpg.sort_values(by='support', ascending=False).reset_index().iloc[:,2]], axis=1)
item_freq_compare.columns = ['support_ap', 'support_fpg', 'itemsets_ap', 'itemsets_fpg']
```

	support_ap	support_fpg	itemsets_ap	itemsets_fpg
0	0.8	0.8	(라면)	(라면)
1	0.5	0.5	(달걀)	(양파)
2	0.5	0.5	(양파)	(달걀)
3	0.4	0.4	(달걀, 라면)	(달걀, 라면)
4	0.4	0.4	(양파, 라면)	(양파, 라면)
5	0.3	0.3	(콜라)	(콜라)
6	0.3	0.3	(달걀, 양파)	(달걀, 양파)
7	0.3	0.3	(콜라, 라면)	(콜라, 라면)
8	0.2	0.2	(달걀, 참치캔)	(달걀, 양파, 라면)
9	0.2	0.2	(참치캔)	(달걀, 참치캔)
10	0.2	0.2	(참치캔, 라면)	(참치캔)
11	0.2	0.2	(달걀, 양파, 라면)	(달걀, 참치캔, 라면)
12	0.2	0.2	(달걀, 참치캔, 라면)	(참치캔, 라면)

			(달걀, 콜라)	(참치캔, 양파)
13	0.1	0.1	(참치캔, 양파)	(달걀, 참치캔, 양파)
14	0.1	0.1	(달걀, 콜라, 라면)	(참치캔, 양파, 라면)
15	0.1	0.1	(달걀, 참치캔, 양파)	(참치캔, 양파, 라면)
16	0.1	0.1	(달걀, 참치캔, 양파, 라면)	(달걀, 참치캔, 양파, 라면)
17	0.1	0.1	(참치캔, 양파, 라면)	(달걀, 콜라)
18	0.1	0.1	(달걀, 참치캔, 양파, 라면)	(달걀, 콜라, 라면)

## 4.4 타이타닉 생존자 연관규칙 분석

In [19]:

```
# 불러오기
import pandas as pd
import numpy as np
import seaborn as sns

C:\Users\KK\anaconda3\lib\site-packages\seaborn\rcmod.py:82: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
    if LooseVersion(mpl.__version__) >= "3.0":
C:\Users\KK\anaconda3\lib\site-packages\setuptools\distutils\version.py:351: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
    other = LooseVersion(other)
```

In [20]:

```
# 타이타닉 데이터 일부 추출
df = sns.load_dataset('titanic')
df = df[['class', 'sex', 'age', 'alive']]
df = df.dropna(axis=0)
df
```

Out [20]:

	class	sex	age	alive
0	Third	male	22.0	no
1	First	female	38.0	yes
2	Third	female	26.0	yes
3	First	female	35.0	yes
4	Third	male	35.0	no
...	...	...	...	...
885	Third	female	39.0	no
886	Second	male	27.0	no
887	First	female	19.0	yes
889	First	male	26.0	yes
890	Third	male	32.0	no

714 rows × 4 columns

In [21]:

```
# 나이 범주화 및 라벨링
child_idx = df.age < 20
adult_idx = (df.age >= 20) & (df.age < 60)
old_idx = df.age >= 60

df.loc[child_idx, "age"] = "child"
df.loc[adult_idx, "age"] = "adult"
df.loc[old_idx, "age"] = "old"
df
```

Out [21]:

	class	sex	age	alive
0	Third	male	adult	no
1	First	female	adult	yes
2	Third	female	adult	yes
3	First	female	adult	yes
4	Third	male	adult	no
...	...	...	...	...
885	Third	female	adult	no
886	Second	male	adult	no
887	First	female	child	yes
889	First	male	adult	yes
890	Third	male	adult	no

714 rows × 4 columns

In [22]:

```
# 각 아이템 포함 여부를 one-hot encoding array로 변환
items = np.unique(df.values.ravel())

encoded_vals = []
for index, row in df.iterrows():
    labels = {}
    uncommons = list(set(items) - set(row))
    commons = list(set(items).intersection(row))
    for uc in uncommons:
        labels[uc] = 0
    for com in commons:
        labels[com] = 1
```

```

    encoded_vals.append(labels)
df_prep = pd.DataFrame(encoded_vals)
df_prep

```

Out [22] :

	yes	old	First	child	Second	female	Third	no	male	adult
0	0	0	0	0	0	0	1	1	1	1
1	1	0	1	0	0	1	0	0	0	1
2	1	0	0	0	0	1	1	0	0	1
3	1	0	1	0	0	1	0	0	0	1
4	0	0	0	0	0	0	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...
709	0	0	0	0	0	1	1	1	0	1
710	0	0	0	0	1	0	0	1	1	1
711	1	0	1	1	0	1	0	0	0	0
712	1	0	1	0	0	0	0	0	1	1
713	0	0	0	0	0	0	1	1	1	1

714 rows × 10 columns

In [23] :

```
# 연관규칙 분석을 위한 A Priori 알고리즘 적용
from mlxtend.frequent_patterns import apriori

item_freq_ap = apriori(df_prep, min_support=0.1, use_colnames=True)
```

C:\Users\KKK\AppData\Roaming\Python\Python39\site-packages\mlxtend\frequent\_patterns\fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types result in worse computational performance and their support might be discontinued in the future. Please use a DataFrame with bool type
warnings.warn(

Out [23] :

	support	itemsets
0	0.406162	(yes)
1	0.260504	(First)
2	0.229692	(child)
3	0.242297	(Second)
4	0.365546	(female)
5	0.497199	(Third)
6	0.593838	(no)
7	0.634454	(male)
8	0.733894	(adult)
9	0.170868	(yes, First)
10	0.110644	(yes, child)
11	0.116246	(yes, Second)
12	0.275910	(yes, female)
13	0.119048	(Third, yes)
14	0.130252	(yes, male)
15	0.285714	(yes, adult)
16	0.119048	(First, female)
17	0.141457	(First, male)
18	0.207283	(First, adult)
19	0.105042	(child, female)
20	0.151261	(Third, child)
21	0.119048	(no, child)
22	0.124650	(child, male)
23	0.103641	(female, Second)
24	0.126050	(no, Second)
25	0.138655	(male, Second)
26	0.187675	(Second, adult)
27	0.142857	(Third, female)
28	0.254902	(female, adult)
29	0.378151	(Third, no)
30	0.354342	(Third, male)
31	0.338936	(Third, adult)
32	0.504202	(no, male)
33	0.448179	(no, adult)
34	0.478992	(male, adult)
35	0.114846	(yes, First, female)
36	0.140056	(yes, First, adult)
37	0.196078	(yes, female, adult)
38	0.112045	(First, male, adult)
39	0.100810	(Third, child, no)

```

39 0.100040 (male, child, no)
40 0.117647 (no, male, Second)
41 0.109244 (Second, no, adult)
42 0.106443 (Second, male, adult)
43 0.301120 (Third, no, male)
44 0.271709 (Third, no, adult)
45 0.260504 (Third, male, adult)
46 0.389356 (no, male, adult)
47 0.100840 (Second, no, male, adult)
48 0.224090 (Third, no, male, adult)

```

```
In [24]: # 연관규칙 별 성능 확인
# metric: 'support', 'confidence', 'lift', 'leverage', 'conviction'
from mlxtend.frequent_patterns import association_rules

item_measures = association_rules(item_freq_ap, min_threshold=0.1)
item_measures
```

Out [24]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(yes)	(First)	0.406162	0.260504	0.170868	0.420690	1.614905	0.065061	1.276511
1	(First)	(yes)	0.260504	0.406162	0.170868	0.655914	1.614905	0.065061	1.725840
2	(yes)	(child)	0.406162	0.229692	0.110644	0.272414	1.185997	0.017352	1.058717
3	(child)	(yes)	0.229692	0.406162	0.110644	0.481707	1.185997	0.017352	1.145757
4	(yes)	(Second)	0.406162	0.242297	0.116246	0.286207	1.181224	0.017835	1.061516
...	...	...	...	...	...	...	...	...	...
147	(male, adult)	(Third, no)	0.478992	0.378151	0.224090	0.467836	1.237167	0.042958	1.168529
148	(Third)	(no, male, adult)	0.497199	0.389356	0.224090	0.450704	1.157564	0.030502	1.111686
149	(no)	(Third, male, adult)	0.593838	0.260504	0.224090	0.377358	1.448570	0.069392	1.187675
150	(male)	(Third, no, adult)	0.634454	0.271709	0.224090	0.353201	1.299925	0.051703	1.125993
151	(adult)	(Third, no, male)	0.733894	0.301120	0.224090	0.305344	1.014024	0.003099	1.006079

152 rows × 9 columns

```
In [25]: # 성종여부 연관규칙 추출
item_survive = pd.concat([item_measures[item_measures.consequents == frozenset({'yes'})],
                           item_measures[item_measures.consequents == frozenset({'no'})]], axis=0)
item_survive.sort_values(by='lift', ascending=False)
```

Out [25]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
54	(First, female)	(yes)	0.119048	0.406162	0.114846	0.964706	2.375172	0.066493	16.825397
66	(female, adult)	(yes)	0.254902	0.406162	0.196078	0.769231	1.893899	0.092547	2.573296
7	(female)	(yes)	0.365546	0.406162	0.275910	0.754789	1.858343	0.127439	2.421744
60	(First, adult)	(yes)	0.207283	0.406162	0.140056	0.675676	1.663560	0.055865	1.830999
1	(First)	(yes)	0.260504	0.406162	0.170868	0.655914	1.614905	0.065061	1.725840
126	(adult, male, Second)	(no)	0.106443	0.593838	0.100840	0.947368	1.595333	0.037631	7.717087
140	(Third, male, adult)	(no)	0.260504	0.593838	0.224090	0.860215	1.448570	0.069392	2.905624
101	(Third, male)	(no)	0.354342	0.593838	0.301120	0.849802	1.431035	0.090699	2.704187
84	(male, Second)	(no)	0.138655	0.593838	0.117647	0.848485	1.428816	0.035308	2.680672
120	(male, adult)	(no)	0.478992	0.593838	0.389356	0.812865	1.368835	0.104913	2.170431
107	(Third, adult)	(no)	0.338936	0.593838	0.271709	0.801653	1.349953	0.070436	2.047736
47	(male)	(no)	0.634454	0.593838	0.504202	0.794702	1.338248	0.127439	1.978404
40	(Third)	(no)	0.497199	0.593838	0.378151	0.760563	1.280760	0.082896	1.696326
3	(child)	(yes)	0.229692	0.406162	0.110644	0.481707	1.185997	0.017352	1.145757
5	(Second)	(yes)	0.242297	0.406162	0.116246	0.479769	1.181224	0.017835	1.141488
76	(Third, child)	(no)	0.151261	0.593838	0.100840	0.666667	1.122642	0.011016	1.218487
49	(adult)	(no)	0.733894	0.593838	0.448179	0.610687	1.028374	0.012366	1.043280
89	(adult, Second)	(no)	0.187675	0.593838	0.109244	0.582090	0.980217	-0.002205	0.971889
13	(adult)	(yes)	0.733894	0.406162	0.285714	0.389313	0.958515	-0.012366	0.972409
31	(Second)	(no)	0.242297	0.593838	0.126050	0.520231	0.876050	-0.017835	0.846580
25	(child)	(no)	0.229692	0.593838	0.119048	0.518293	0.872785	-0.017352	0.843173
8	(Third)	(yes)	0.497199	0.406162	0.119048	0.239437	0.589509	-0.082896	0.780786
11	(male)	(yes)	0.634454	0.406162	0.130252	0.205298	0.505458	-0.127439	0.747246

In [ ]: