추천시스템 분석 경진대회 참여하기

"브런치 사용자를 위한 글 추천대회"

한양대학교

김현우

Contents

0. 추천시스템 이해

- 추천시스템 개요
- 기업에서의 추천시스템
- 과거의 추천시스템

1. 컨텐츠기반추천

- 컨텐츠 기반 모델 이론
- TF-IDF 모델
- 유사도 함수

2. 협업필터링

- KNN
- SGD
- ALS

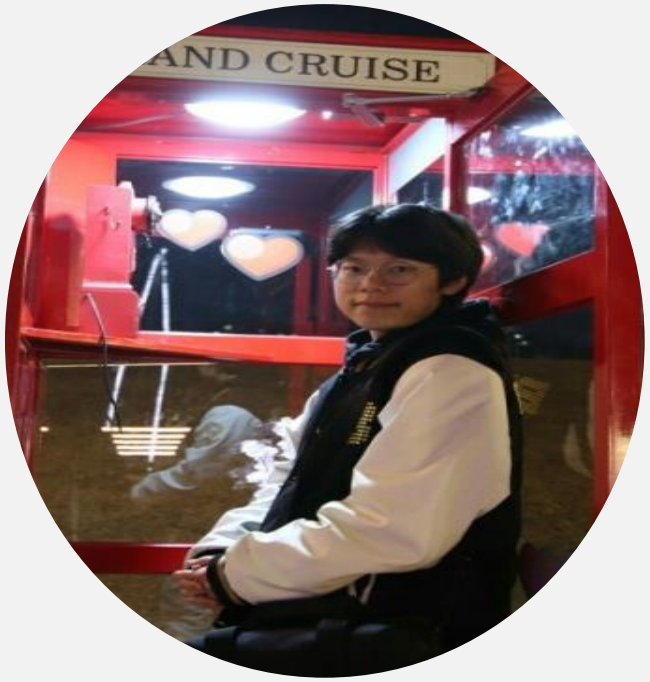
3. 분석 실습

브런치 사용자를 위한 글 추천대회
- Data Exploratory Analysis
- Baseline Recommendation
- Contents Based Recommendation
- Collaborative filtering Recommendation

4. 그 외

- 도전 분야

0 본인 소개



김 현 우

한양대학교, 산업공학과

TEAM-EDA 블로그

Recommender System KR 페이스북 그룹 운영진

Recommender System KR

Recommender System KR 페이스북 그룹 운영진

가입자 : (2020.08.04 기준)

https://www.facebook.com/groups/2611614312273351/about

00 추천시스템 이해

0 추천시스템의 개요

정의

추천시스템은 **사용자(user)**에게 **상품(item)**을 **제안**하는 소프트웨어 도구 이자 기술입니다. 이러한 제안은 어떤 상품을 구매할 지, 어떤 음악을 들을지 또는 어떤 온라인 뉴스를 읽을지와 같은 다양한 의사결정과 연관있습니다.

목표

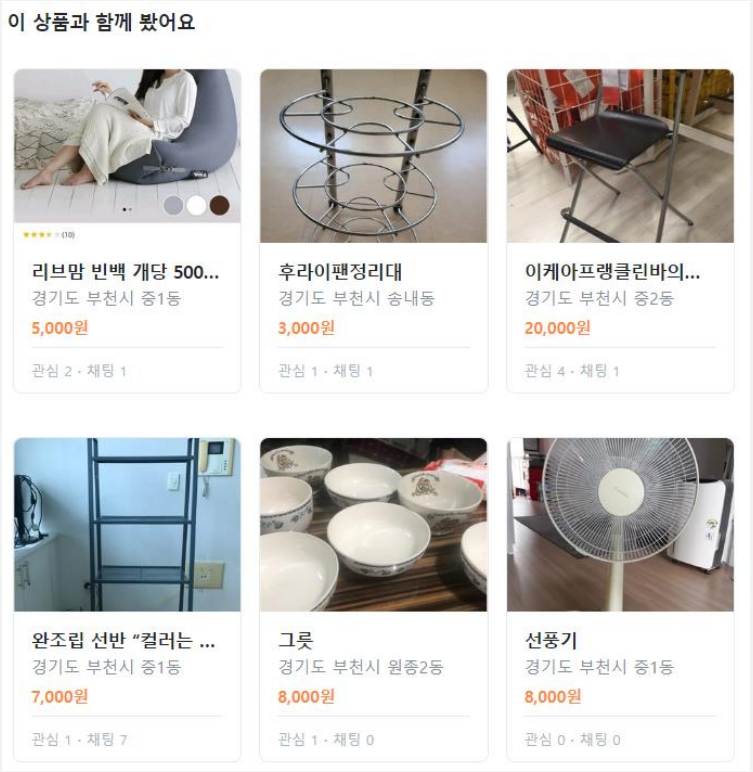
어떤 사용자에게 **어떤 상품**을 **어떻게 추천**할지에 대해 이해

출처 : Recommender Systems Handbook, Francesco Ricci

0 기업에서의 추천시스템

당근마켓

당근마켓에서의 추천 사례
- 다른 사람들이 같이 본 상품 추천

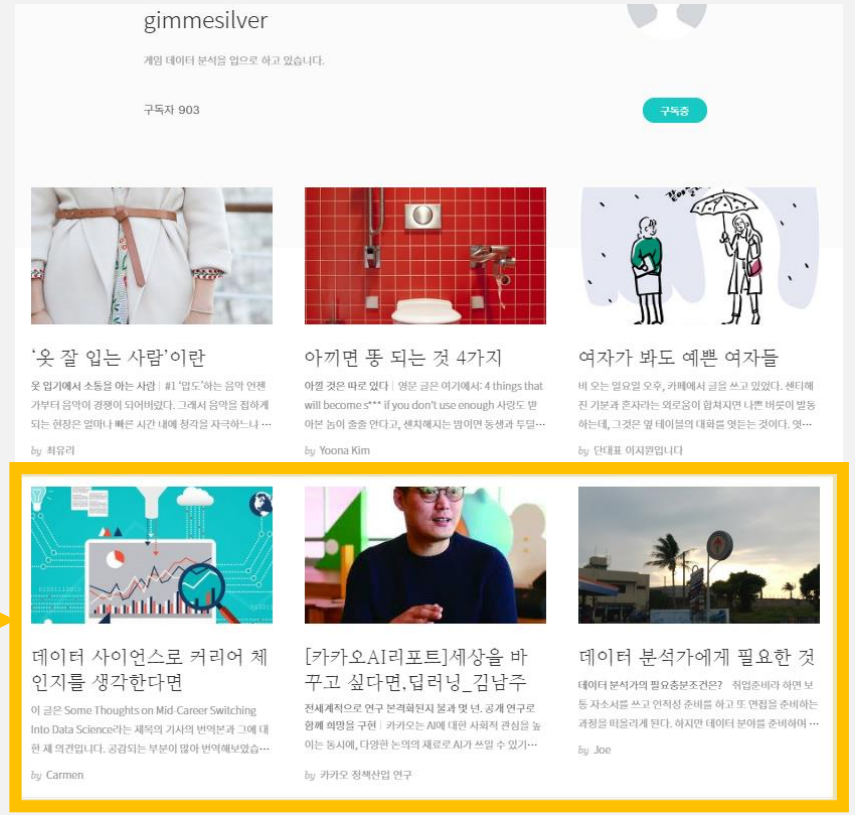


0 기업에서의 추천시스템

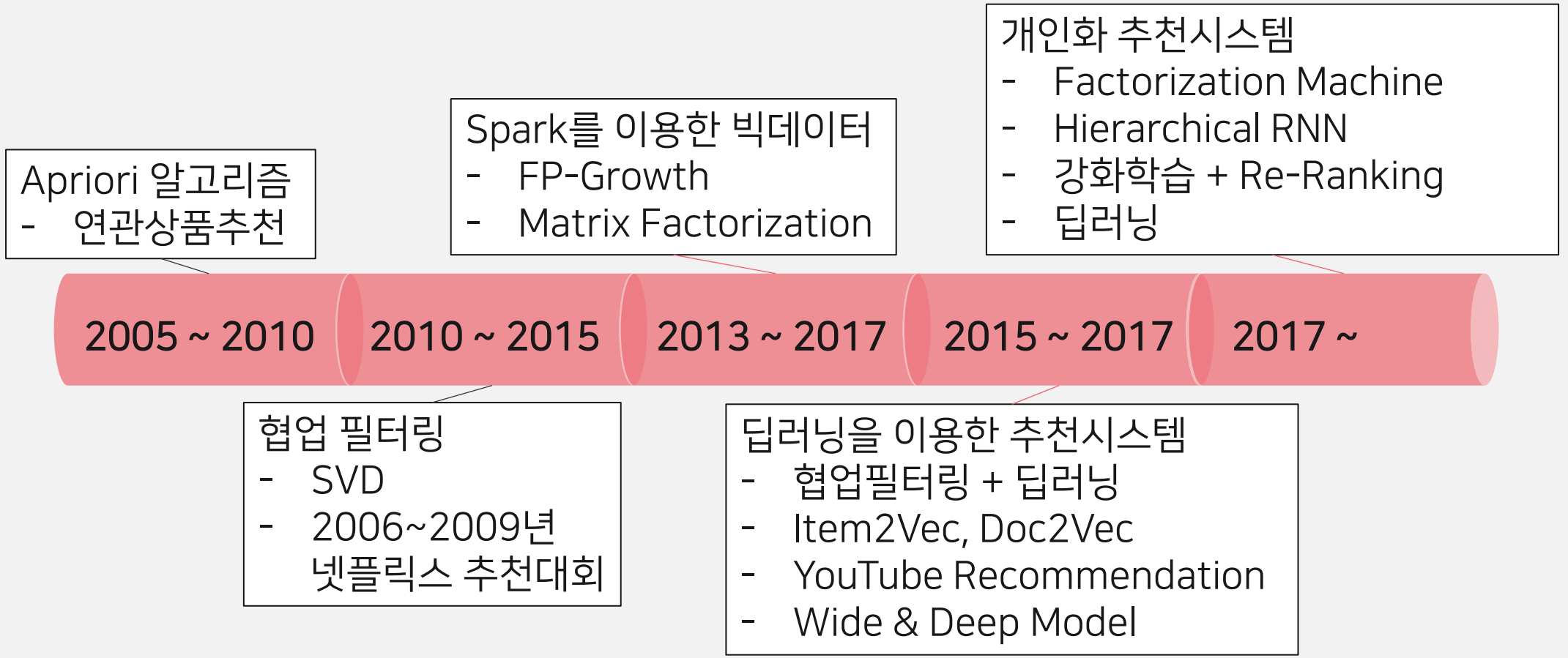
카카오 (브런치)

카카오에서의 추천 사례
- 해당 글과 유사한 글을 추천

실전 이탈 예측 모델링을 위한 세 가지 고려 사항 #1



0 추천시스템의 역사



HoonDongKim, 추천 시스템 – 알고리즘 Trend 정리

0 연관분석(Association Analysis)

정의

룰기반의 모델로서 상품과 상품사이에 어떤 **연관**이 있는지 찾아내는 알고리즘입니다. 이러한 연관은 2가지 형태로 존재합니다.

- 첫번째, **얼마나(frequent)** 같이 구매가 되는가?
- 두번째, A아이템을 구매하는 사람이 B아이템을 구매하는가?

라는 규칙을 찾아내는 형태입니다. 어떤 상품들이 한 장바구니 안에 담기는 지 살피는 모습과 비슷하기 때매 장바구니 분석이라고 표현하기도 합니다.

예시

가장 유명한 일화로는 월마트에서 맥주를 구매할 때 기저귀를 같이 구매하는 경향이 크다는 것을 밝혀서 둘을 함께 진열하는 전략을 세우기도 했습니다.

0 연관분석(Association Analysis) - 규칙평가지표

support (지지도)

$$For \quad the \quad rule \quad A \rightarrow B,$$
$$support(A) = P(A)$$

lift (향상도)

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

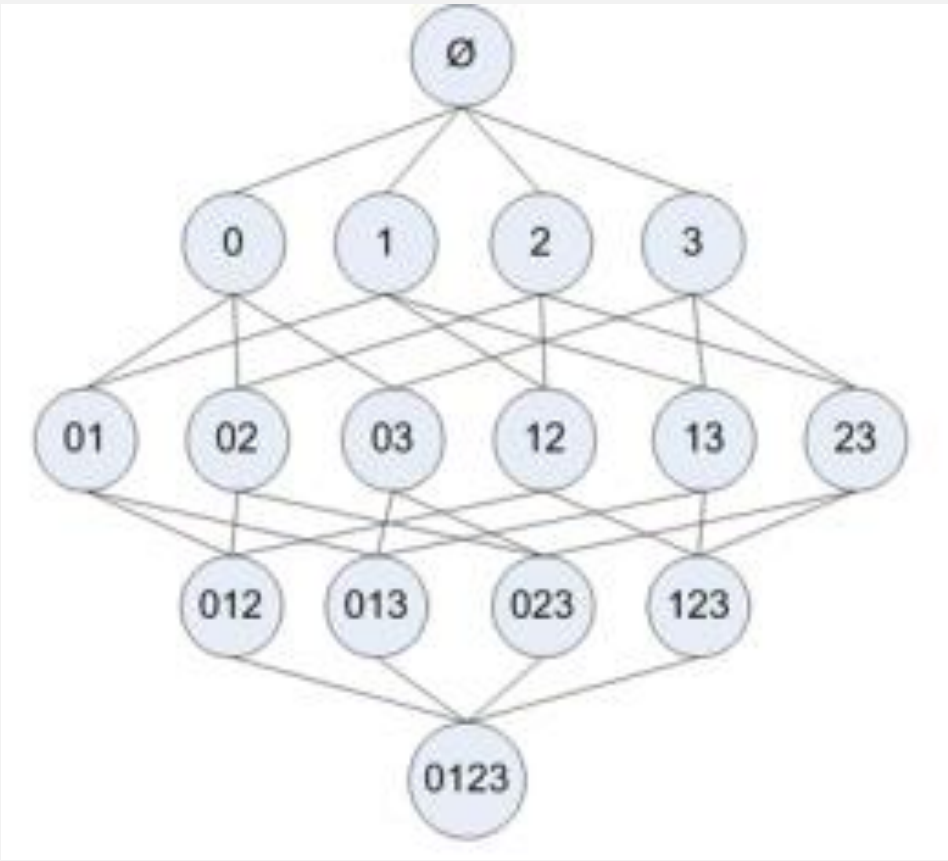
confidence (신뢰도)

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

ratsgo's blog, 연관규칙분석(A Priori Algorithm)

0 연관분석(Association Analysis) – 규칙 생성

규칙 생성



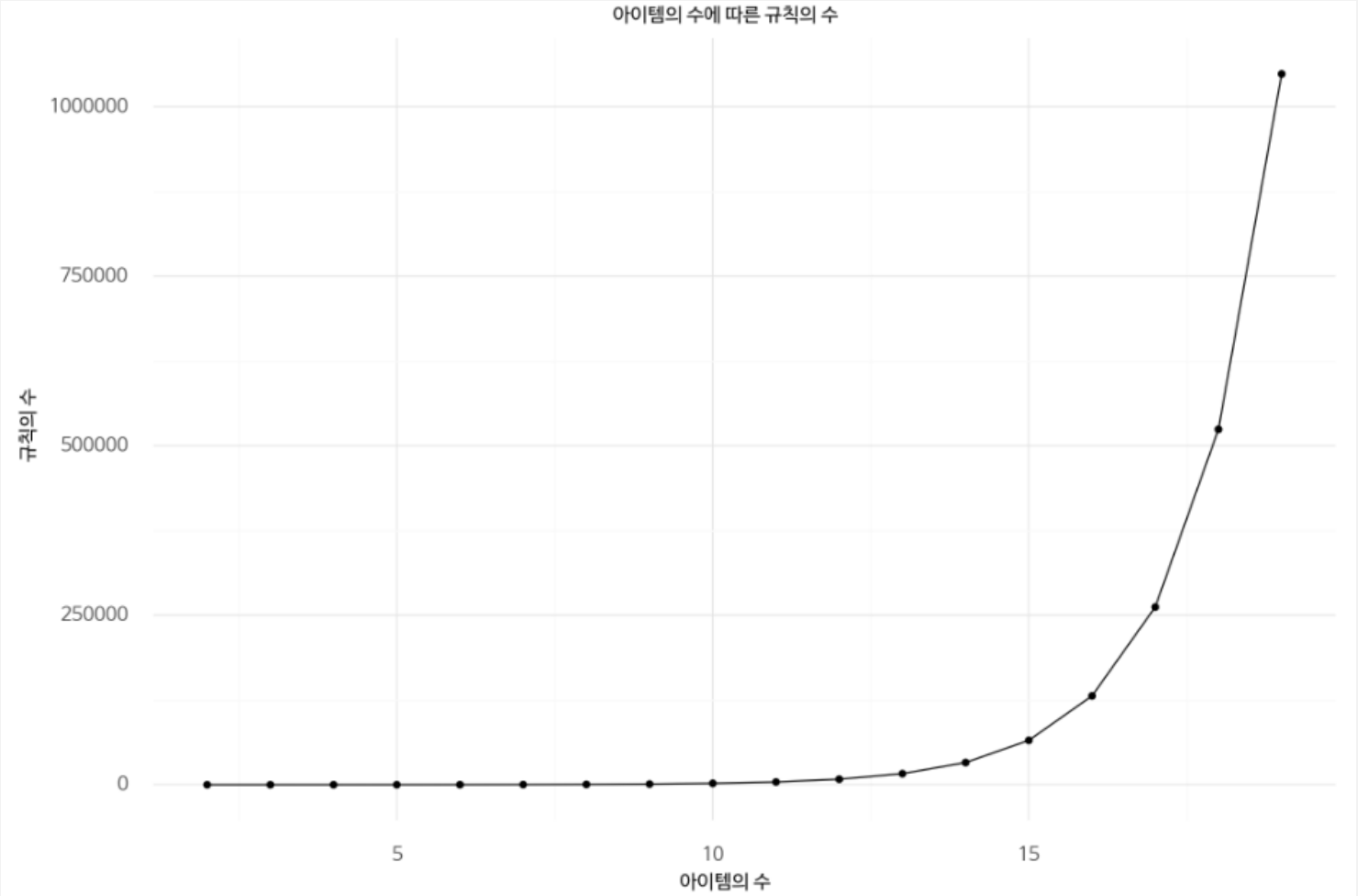
가능한 모든 경우의 수를 탐색해서 지지도, 신뢰도, 향상도가 높은 규칙들을 찾아내는 방식

상품이 4개일 때, 전체 경우의 수
- 4C1 : 4
- 4C2 : 6
- 4C3 : 4
- 4C4 : 1

전체 경우의 수 : 4 + 6 + 4 + 1 = 15

0 연관분석(Association Analysis) – 문제점

문제점



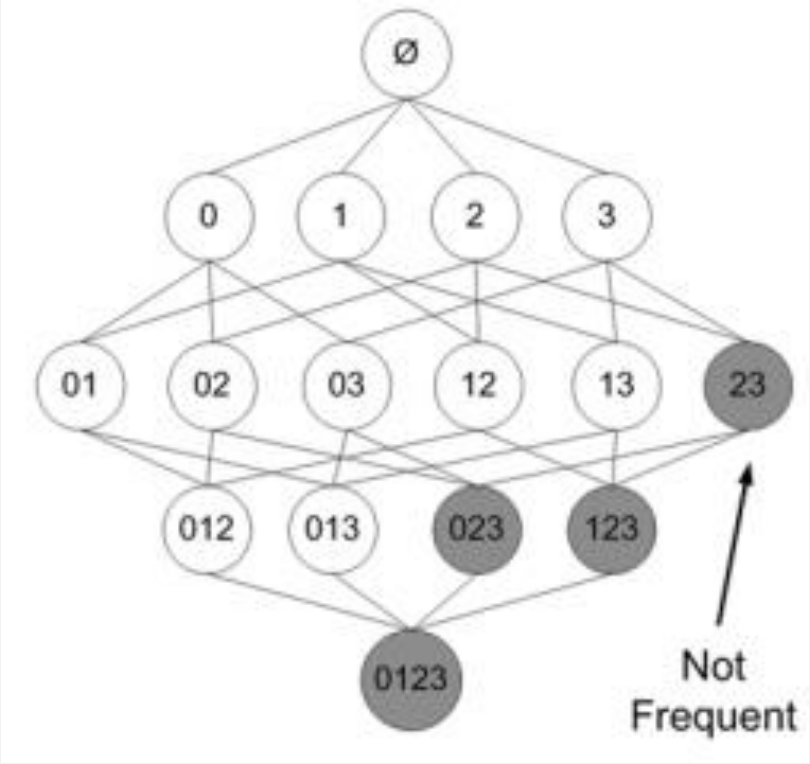
아이템의 증가에 따른 규칙의 수의 증가가 기하급수적으로 증가

아이템이 100개인 경우에는 규칙의 수가 1.26 * 10^30

0 Apriori 알고리즘

Apriori

A priori 원리는 아이템셋의 증가를 줄이기 위한 방법입니다. 기본적인 아이디어는 “빈번한 아이템셋은 하위 아이템셋 또한 빈번할 것이다 " 입니다. 즉, “빈번하지 않은 아이템셋은 하위 아이템셋 또한 빈번하지 않다 " 를 이용해서 아이템셋의 증가를 줄이는 방법입니다.



아이디어

{2, 3}의 지지도 > {0, 2, 3}, {1, 2, 3}의 지지도

- P(item 2, item 3) > P(item 0, item 2, item 3)
- P(item 2, item 3) > P(item 1, item 2, item 3)

0 Apriori 알고리즘

Apriori

1. k개의 item을 가지고 단일항목집단 생성 (one-item frequent set)
2. 단일항목집단에서 최소 지지도(support) 이상의 항목만 선택
3. 2에서 선택된 항목만을 대상으로 2개항목집단 생성
4. 2개항목집단에서 최소 지지도 혹은 신뢰도 이상의 항목만 선택
5. 위의 과정을 k개의 k-item frequent set을 생성할 때까지 반복

0 Apriori 알고리즘 - 데이터

데이터 (Implicit Feedback)

거래 번호	상품 목록
0	우유, 기저귀, 쥬스
1	양상추, 기저귀, 맥주
2	우유, 양상추, 기저귀, 맥주
3	양상추, 맥주

거래번호	우유	양상추	기저귀	쥬스	맥주
0	1	0	1	1	0
1	0	1	1	0	1
2	1	1	1	0	1
3	0	1	0	0	1

0 Apriori 알고리즘 - 예시

예시

거래번호	우유	양상추	기저귀	쥬스	맥주
0	1	0	1	1	0
1	0	1	1	0	1
2	1	1	1	0	1
3	0	1	0	0	1

1. 5개의 item을 가지고 단일항목집단 생성 (one-item frequent set) : 우유, 양상추, 기저귀, 맥주, 쥬스
2. 단일항목집단에서 최소 지지도(support) 이상의 항목만 선택 (예: 최소지지도 0.5)

P(우유) : 0.5

P(양상추) : 0.75

P(기저귀) : 0.75

~~P(쥬스) : 0.25~~

P(맥주) : 0.75

0 Apriori 알고리즘 - 예시

거래번호	우유	양상추	기저귀	쥬스	맥주
0	1	0	1	1	0
1	0	1	1	0	1
2	1	1	1	0	1
3	0	1	0	0	1

예시

3. 2에서 선택된 항목만을 대상으로 2개항목집단 생성 {우유, 양상추, 기저귀, 맥주}

{우유, 양상추}, {우유, 기저귀}, {우유, 맥주}, {양상추, 기저귀}, {양상추, 맥주}, {기저귀, 맥주}

4. 2개항목집단에서 최소 지지도 이상의 항목만 선택

~~{우유, 양상추} : 0.25~~ {우유, 기저귀} : 0.5 ~~{우유, 맥주} : 0.25~~

{양상추, 기저귀} : 0.5 {양상추, 맥주} : 0.75 {기저귀, 맥주} : 0.5

0 Apriori 알고리즘 - 예시

예시

거래번호	우유	양상추	기저귀	쥬스	맥주
0	1	0	1	1	0
1	0	1	1	0	1
2	1	1	1	0	1
3	0	1	0	0	1

5. 위의 과정을 k개의 k-item frequent set을 생성할 때까지 반복

- {우유}, {양상추}, {기저귀}, {맥주}
- {우유, 기저귀}, {양상추, 기저귀}, {양상추, 맥주}, {기저귀, 맥주}
- {양상추, 기저귀, 맥주}

0 Apriori 알고리즘 - 예시

예시

위의 예시는 Support를 바탕으로
진행했지만, Confidence와 Lift를
이용해서도 진행 가능하고 함께 사용가능

0 Apriori 알고리즘 - 장단점

장점

- 원리가 간단하여 사용자가 쉽게 이해할 수 있고 의미를 파악할 수 있음
- 유의한 연관성을 갖는 구매패턴을 찾아줌

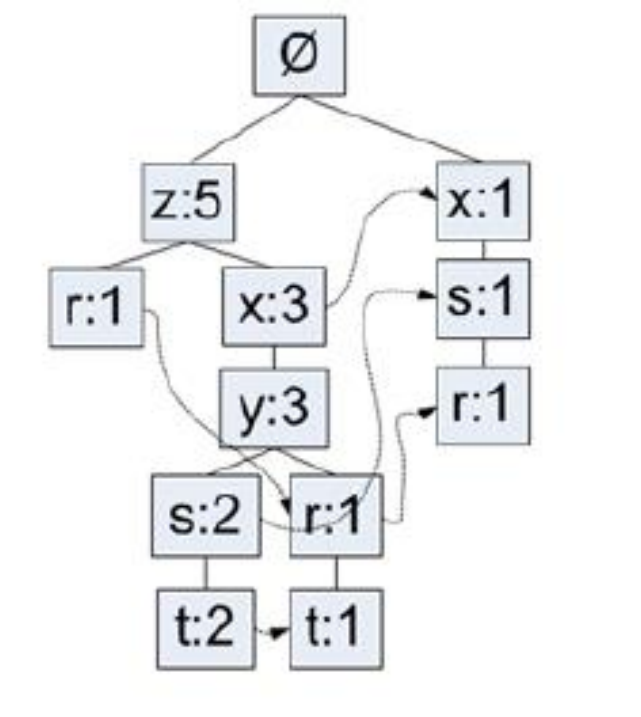
단점

- 데이터가 클 경우 (item이 많은 경우)에 속도가 느리고 연산량이 많음
- 실제 사용시에 많은 연관상품들이 나타나는 단점이 있음

0 FP-Growth 알고리즘

FP-Growth

FP Growth는 이전에 언급한 A Priori의 속도측면의 단점을 개선한 알고리즘입니다. Apriori와 비슷한 성능을 내지만 FP Tree라는 구조를 사용해서 따른 속도를 가진다는게 장점입니다. 하지만, 동일하게 발생하는 아이템셋(frequent itemsets)을 찾는데는 좋지만 **아이템간의 연관성**을 찾는 것은 어렵다는 단점이 있습니다.



0 FP-Growth 알고리즘

원리

1. 모든 거래를 확인하여, 각 아이템마다의 지지도(support)를 계산하고 최소 지지도이상의 아이템만 선택
2. 모든 거래에서 빈도가 높은 아이템 순서대로 순서를 정렬
3. 부모 노드를 중심으로 거래를 자식노드로 추가해주면서 tree를 생성
4. 새로운 아이템이 나올 경우에는 부모노드부터 시작하고, 그렇지 않으면 기존의 노드에서 확장
5. 위의 과정을 모든 거래에 대해 반복하여 FP TREE를 만들고 최소 지지도 이상의 패턴만을 추출

0 FP-Growth 알고리즘

거래번호	우유	양상추	기저귀	쥬스	맥주
0	1	0	1	1	0
1	0	1	1	0	1
2	1	1	1	0	1
3	0	1	0	0	1

예시

1. 모든 거래를 확인하여, 각 아이템마다의 지지도(support)를 계산하고 최소 지지도이상의 아이템만 선택
2. 모든 거래에서 빈도가 높은 아이템 순서대로 순서를 정렬

거래번호	아이템
0	우유, 기저귀, ~~쥬스~~
1	양상추, 기저귀, 맥주
2	우유, 양상추, 기저귀, 맥주
3	양상추, 맥주

거래번호	정렬된 아이템
0	기저귀, 우유
1	양상추, 기저귀, 맥주
2	양상추, 기저귀, 맥주, 우유
3	양상추, 맥주

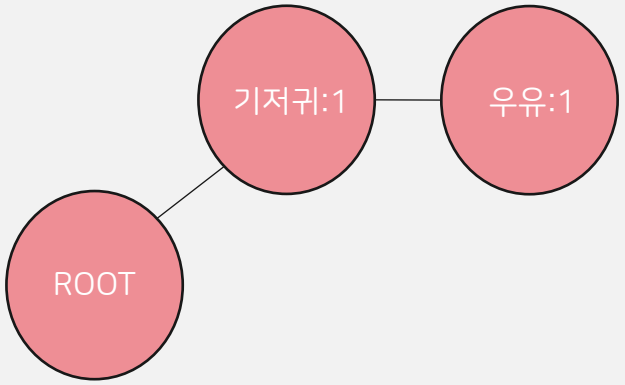
쥬스가 삭제되고, 빈도가 높은 {양상추, 기저귀, 맥주} -> {우유} 순서대로 정렬

0 FP-Growth 알고리즘

예시

거래번호	정렬된 아이템
0	기저귀, 우유
1	양상추, 기저귀, 맥주
2	양상추, 기저귀, 맥주, 우유
3	양상추, 맥주

3. 부모 노드를 중심으로 거래를 자식노드로 추가해주면서 tree를 생성

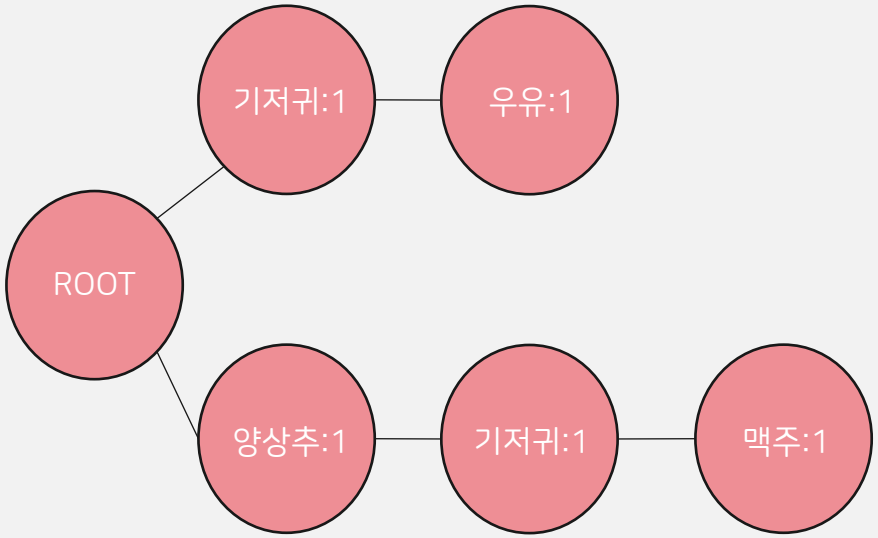


0 FP-Growth 알고리즘

예시

거래번호	정렬된 아이템
0	기저귀, 우유
1	양상추, 기저귀, 맥주
2	양상추, 기저귀, 맥주, 우유
3	양상추, 맥주

4. 새로운 아이템이 나올 경우에는 부모노드부터 시작하고, 그렇지 않으면 기존의 노드에서 확장

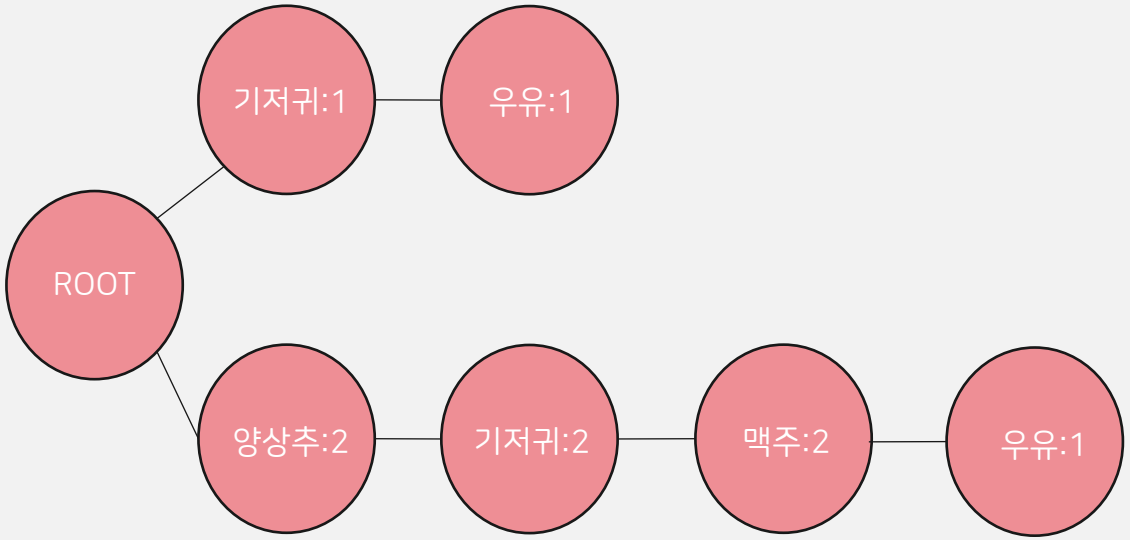


0 FP-Growth 알고리즘

예시

거래번호	정렬된 아이템
0	기저귀, 우유
1	**양상추, 기저귀, 맥주**
2	**양상추, 기저귀, 맥주**, 우유
3	양상추, 맥주

4. 새로운 아이템이 나올 경우에는 부모노드부터 시작하고, 그렇지 않으면 기존의 노드에서 확장

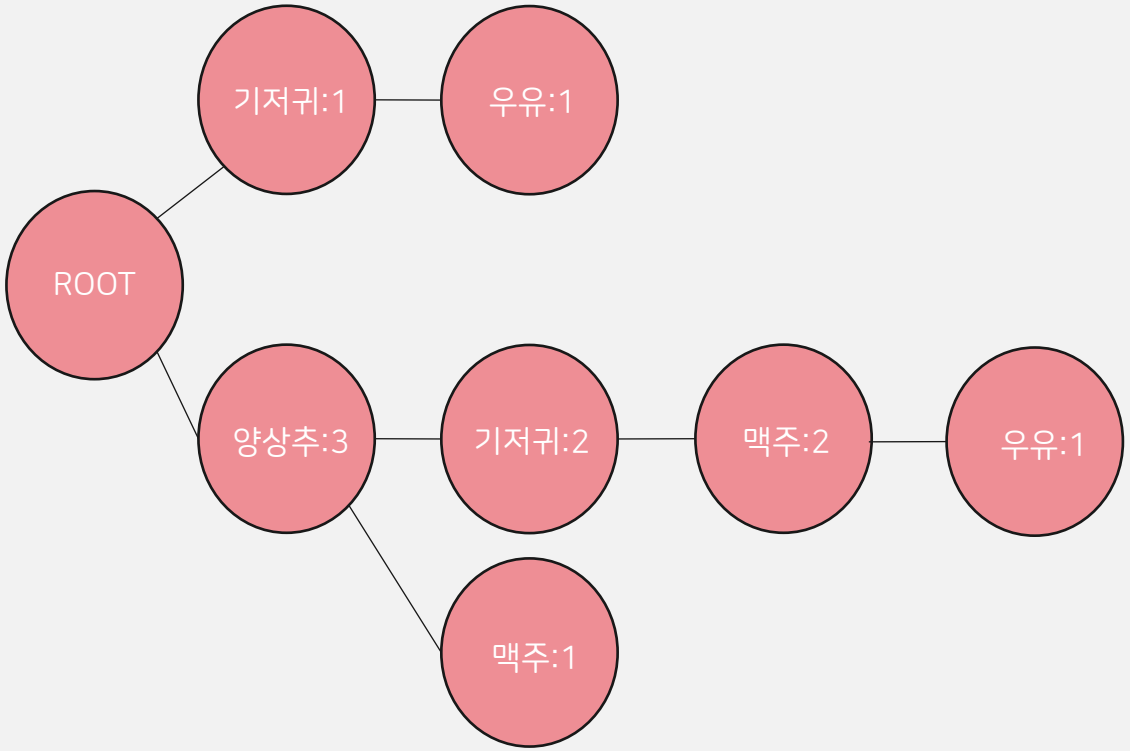


0 FP-Growth 알고리즘

예시

거래번호	정렬된 아이템
0	기저귀, 우유
1	**양상추**, 기저귀, 맥주
2	**양상추**, 기저귀, 맥주, 우유
3	**양상추**, 맥주

4. 새로운 아이템이 나올 경우에는 부모노드부터 시작하고, 그렇지 않으면 기존의 노드에서 확장



0 FP-Growth 알고리즘 - 장단점

장점

- Apriori 알고리즘보다 빠르고 2번의 탐색만 필요로 함
- 후보 Itemsets 을 생성할 필요없이 진행 가능

단점

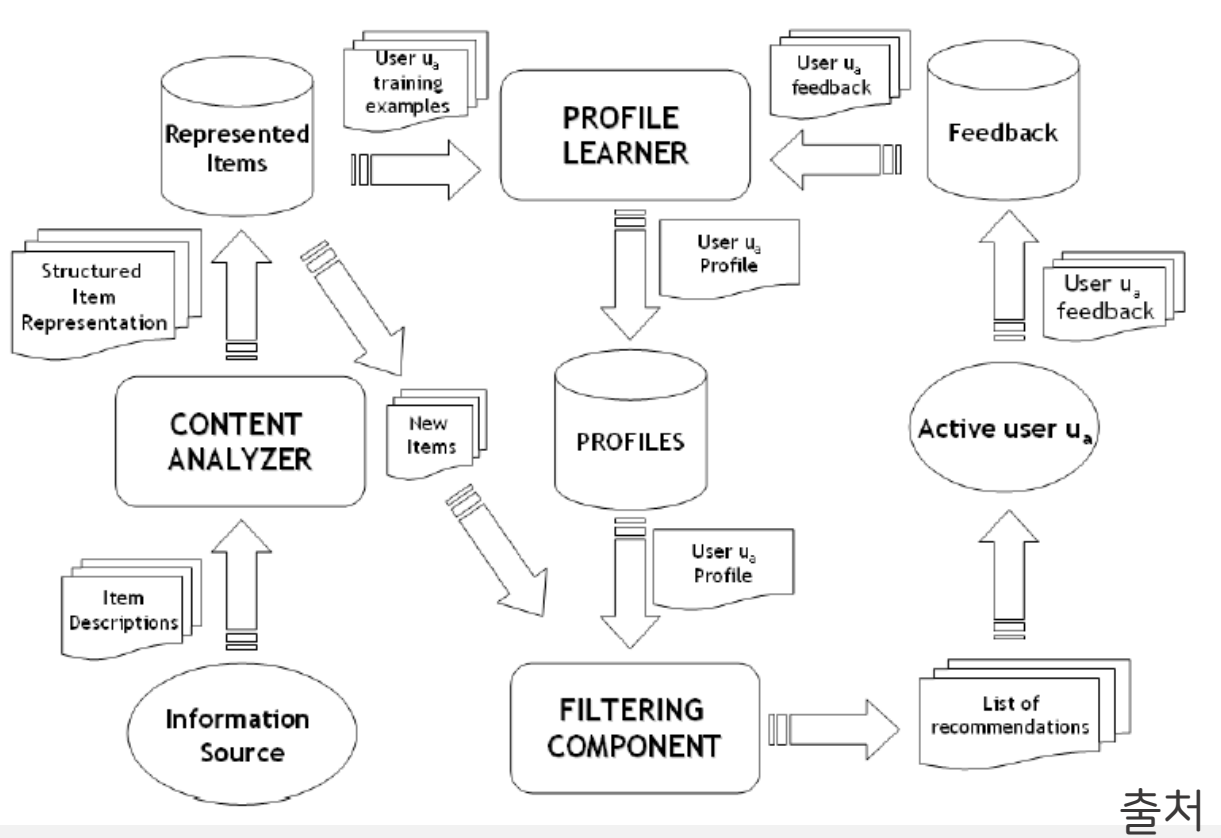
- 대용량의 데이터셋에서 메모리를 효율적으로 사용하지 않음
- Apriori 알고리즘에 비해서 설계하기 어려움
- 지지도의 계산이 FP-Tree가 만들어지고 나서야 가능함

01 컨텐츠 기반 추천

1 컨텐츠 기반 모델

정의

컨텐츠 기반 추천시스템은 사용자가 이전에 구매한 상품중에서 좋아하는 상품들과 **유사한 상품들**을 추천하는 방법입니다.



출처 : Recommender Systems Handbook, Francesco Ricci

1 컨텐츠 기반 모델

Represented Items

Items을 벡터 형태로 표현. 도메인에 따라 다른 방법이 적용

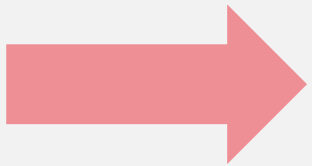
Text

이번 글에서는 기존의 RNN Basics에 이어서 PyTorch로 RNN를 구현하는 것에 대해서 심화적으로 배워보도록 하겠습니다. 이번 글은 EDWITH에서 진행하는 파이토치로 시작하는 딥러닝 기초를 토대로 하였고 같이 스터디하는 팀원분들의 자료를 바탕으로 작성하였습니다. Cross entropy loss에 대한 이론적인 설명은 hyuwk님의 블로그와 ratsgo님의 블로그를 참고하였습니다.

목차

- 'Hihello' proble
- Data Setting
  - one hot enc
- Cross entropy l
- Code run
- 'longseq' exam
- RNN timeseries

Image



벡터

1 TF-IDF

정의

TF-IDF는 특정 문서 내에 특정 단어가 얼마나 자주 등장하는 지를 의미하는 **단어 빈도(TF)**와 전체 문서에서 특정 단어가 얼마나 자주 등장하는지를 의미하는 역문서 빈도(DF)를 통해서 “다른 문서에서는 등장하지 않지만 특정 문서에서만 자주 등장하는 단어 " 를 찾아서 문서 내 단어의 가중치를 계산하는 방법입니다.

용도로는 문서의 핵심어를 추출, 문서들 사이의 유사도를 계산, 검색 결과의 중요도를 정하는 작업등에 활용할 수 있습니다.

1 TF-IDF

- TF(d, t)

  특정 문서 d에서의 특정 단어 t의 등장 횟수

- DF(t)

  특정 단어 t가 등장한 문서의 수

- IDF(d, t)

  DF(t)에 반비례하는 수

$$idf(d,t) = log(\frac{n}{1+df(t)})$$

TF(d, t) * IDF(d, t) = TF-IDF(d, t)

1 TF-IDF

데이터

문서	내용
0	먹고 싶은 사과
1	먹고 싶은 바나나
2	길고 노란 바나나 바나나
3	저는 과일이 좋아요

1 TF-IDF

문서	내용
0	먹고 싶은 사과
1	먹고 싶은 바나나
2	길고 노란 바나나 바나나
3	저는 과일이 좋아요

알고리즘

- 문서내 단어의 TF 값 계산

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

- 단어의 DF 값 계산

총합	1	1	1	2	3	1	2	1	1

1 TF-IDF

문서	내용
0	먹고 싶은 사과
1	먹고 싶은 바나나
2	길고 노란 바나나 바나나
3	저는 과일이 좋아요

알고리즘

- 문서내 단어의 IDF 값 계산

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
총합	1	1	1	2	3	1	2	1	1

단어	IDF(역 문서 빈도)
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(2+1)) = 0.287682
사과	ln(4/(1+1)) = 0.693147
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

$$idf(d,t) = log(\frac{n}{1+df(t)})$$

1 TF-IDF

문서	내용
0	먹고 싶은 사과
1	먹고 싶은 바나나
2	길고 노란 바나나 바나나
3	저는 과일이 좋아요

알고리즘

- 문서내 단어의 TF * IDF 값 계산

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

1 TF-IDF

문서	내용
0	먹고 싶은 사과
1	먹고 싶은 바나나
2	길고 노란 바나나 바나나
3	저는 과일이 좋아요

알고리즘

- 문서간의 유사도를 계산

	문서1	문서2	문서3	문서4
문서1	1	0.413308	0	0
문서2	0.413308	1	0.292253	0
문서3	0	0.292253	1	0
문서4	0	0	0	1

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

1 TF-IDF

코드

```
docs = [
  '먹고 싶은 사과',
  '먹고 싶은 바나나',
  '길고 노란 바나나 바나나',
  '저는 과일이 좋아요'
]
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidv = TfidfVectorizer(use_idf=True, smooth_idf=False, norm=None).fit(docs)
tfidv_df = pd.DataFrame(tfidv.transform(docs).toarray())
tfidv_df
```

	0	1	2	3	4	5	6	7	8
0	0.000000	0.000000	0.000000	1.693147	0.000000	2.386294	1.693147	0.000000	0.000000
1	0.000000	0.000000	0.000000	1.693147	1.693147	0.000000	1.693147	0.000000	0.000000
2	0.000000	2.386294	2.386294	0.000000	3.386294	0.000000	0.000000	0.000000	0.000000
3	2.386294	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.386294	2.386294

```
print(sorted(tfidv.vocabulary_))
```

```
['과일이', '길고', '노란', '먹고', '바나나', '사과', '싶은', '저는', '좋아요']
```

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity(tfidv_df, tfidv_df)
```

```
array([[1.        , 0.57833696, 0.        , 0.        ],
       [0.57833696, 1.        , 0.40894599, 0.        ],
       [0.        , 0.40894599, 1.        , 0.        ],
       [0.        , 0.        , 0.        , 1.        ]])
```

```
The formula that is used to compute the tf-idf for a term t of a document d
in a document set is tf-idf(t, d) = tf(t, d) * idf(t), and the idf is
computed as idf(t) = log [ n / df(t) ] + 1 (if ``smooth_idf=False``), where
n is the total number of documents in the document set and df(t) is the
document frequency of t; the document frequency is the number of documents
in the document set that contain the term t. The effect of adding "1" to
the idf in the equation above is that terms with zero idf, i.e., terms
that occur in all documents in a training set, will not be entirely
ignored.
(Note that the idf formula above differs from the standard textbook
notation that defines the idf as
idf(t) = log [ n / (df(t) + 1) ]).
```

1 TF-IDF

장점

- 직관적인 해석이 가능함

단점

- 대규모 말뭉치를 다룰 때 메모리상의 문제가 발생
  - 높은 차원을 가짐
  - 매우 sparse한 형태의 데이터임

1 유사도 함수

코사인 유사도

- 문서간의 유사도를 계산

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

```
1 from sklearn.metrics.pairwise import cosine_similarity
2 cosine_similarity(countvect_df, countvect_df)
```

```
array([[1.        , 0.66666667, 0.        , 0.        ],
       [0.66666667, 1.        , 0.47140452, 0.        ],
       [0.        , 0.47140452, 1.        , 0.        ],
       [0.        , 0.        , 0.        , 1.        ]])
```

$$\frac{1+1}{\sqrt{1^2+1^2+1^2}\cdot\sqrt{1^2+1^2+1^2}} = 0.6667$$

1 유사도 함수

피어슨 유사도

- 문서간의 유사도를 계산

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

$$r_{XY} = \frac{\sum_i^n \left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right)}{\sqrt{\sum_i^n \left(X_i - \overline{X}\right)^2}\sqrt{\sum_i^n \left(Y_i - \overline{Y}\right)^2}}$$

```
import pandas as pd
pearson_sim = countvect_df.T.corr()
np.array(pearson_sim)
```

```
array([[ 1.        ,  0.5       , -0.45883147, -0.5       ],
       [ 0.5       ,  1.        ,  0.22941573, -0.5       ],
       [-0.45883147,  0.22941573,  1.        , -0.45883147],
       [-0.5       , -0.5       , -0.45883147,  1.        ]])
```

1 유사도 함수

자카드 유사도

- 문서간의 유사도를 계산

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

```
1 from sklearn.metrics.pairwise import pairwise_distances
2 jac_sim = 1 - pairwise_distances(np.array(countvect_df), metric = "jaccard")
3 jac_sim
```

```
array([[1. , 0.5, 0. , 0. ],
       [0.5, 1. , 0.2, 0. ],
       [0. , 0.2, 1. , 0. ],
       [0. , 0. , 0. , 1. ]])
```

$$\frac{2}{4} = 0.5$$

1 유사도 함수

그 외

- Euclidean 유사도
- Sorensen 유사도
- Soergel 유사도
- Lorentzian 유사도
- Canberra 유사도
- WaveHedges 유사도
- Motyka 유사도
- Kulczynski 유사도
- Ruzicka 유사도
- Dice 유사도
- Cosine 유사도
- Jaccard 유사도
- hellinger 유사도
- Sq-chord 유사도
- Sq-kai 유사도
- Divergence 유사도
- Clark 유사도
- Topsoe 유사도
- Jensen-Diff 유사도

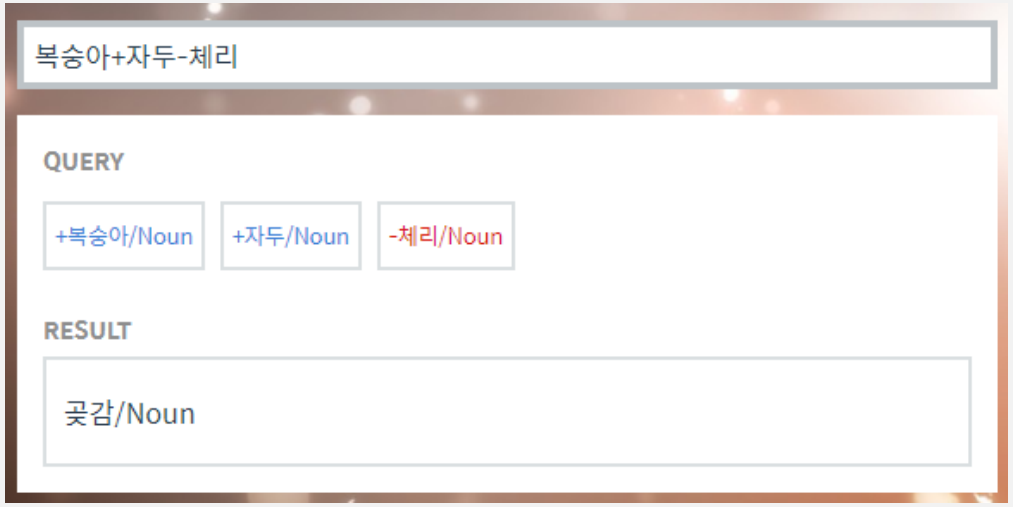
- From scikit-learn: ['cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan']. These metrics support sparse matrix inputs. ['nan_euclidean'] but it does not yet support sparse matrices.
- From scipy.spatial.distance: ['braycurtis', 'canberra', 'chebyshev', 'correlation', 'dice', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'minkowski', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule'] See the documentation for scipy.spatial.distance for details on these metrics. These metrics do not support sparse matrix inputs.

1 Word2Vec

정의

Word2Vec은 단어간 유사도를 반영하여 단어를 벡터로 바꿔주는 임베딩 방법론입니다. 원-핫벡터 형태의 sparse matrix 이 가지는 단점을 해소하고자 저차원의 공간에 벡터로 매핑하는 것이 특징입니다. Word2Vec은 “비슷한 위치에 등장하는 단어들은 비슷한 의미를 가진다“ 라는 가정을 통해서 학습을 진행합니다. 저차원에 학습된 단어의 의미를 분산하여 표현하기에 단어 간 유사도를 계산할 수 있습니다.

추천시스템에서는 단어를 구매 상품으로 바꿔서 구매한 패턴에 Word2Vec을 적용해서 비슷한 상품을 찾을 수 있습니다.



1 Word2Vec - CBOW

알고리즘

CBOW는 주변에 있는 단어들을 가지고, 중간에 있는 단어들을 예측하는 방법입니다. 반대로, Skip-Gram은 중간에 있는 단어로 주변 단어들을 예측하는 방법입니다.

예) you say goodbye and I say hello.

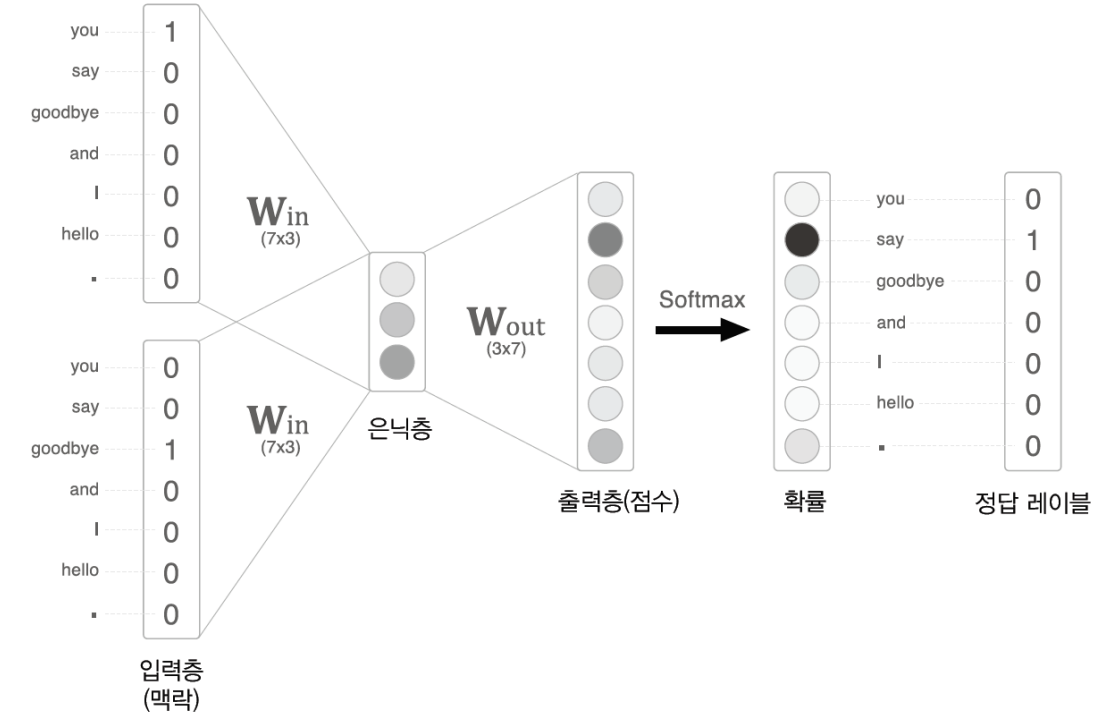
주변 단어 : 주변에 있는 단어 (you, goodbye)
중심 단어 : 중간에 있는 단어 (say)
윈도우 크기 : 주변을 몇 칸까지 볼 지에 대한 크기 (1)

1 Word2Vec - CBOW

알고리즘

주변단어를 통해서 중심단어가 되는 단어의 오차를 학습해서 역전파

그림 3-12 CBOW 모델의 구체적인 예(노드 값의 크기를 흑백의 진하기로 나타냄)



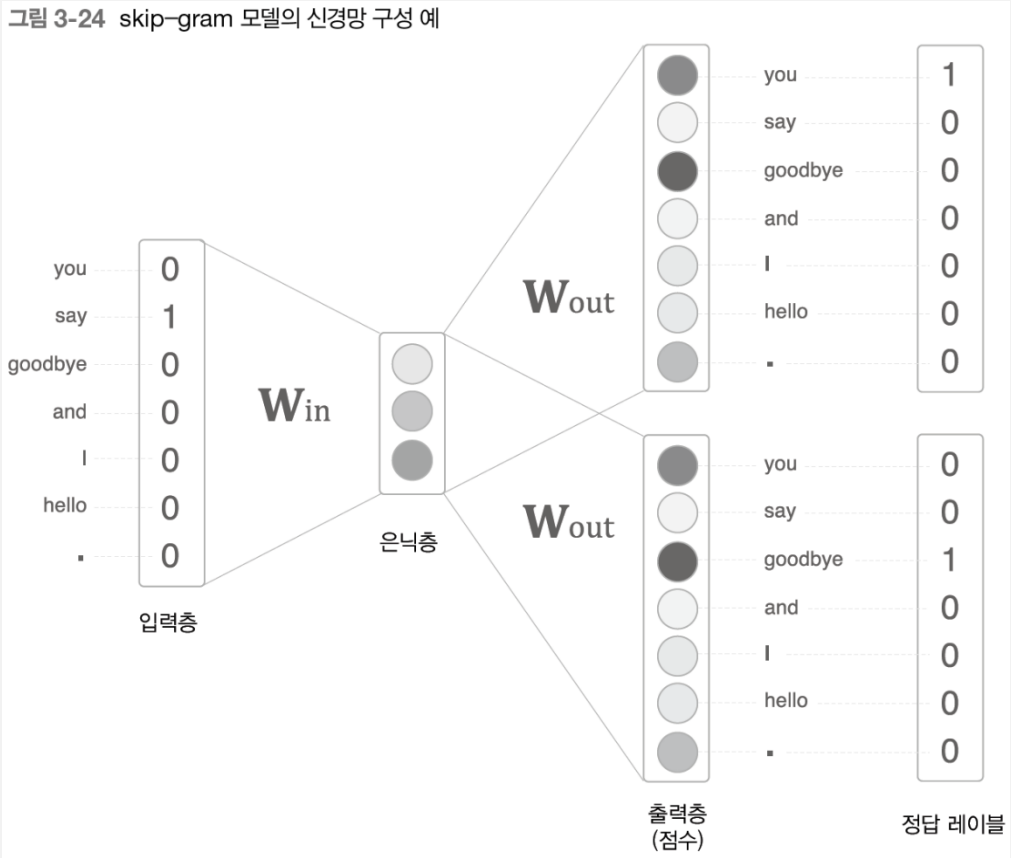
딥러닝을 이용한 자연어 처리 입문, Word2Vec
밑바닥부터 시작하는 딥러닝2

1 Word2Vec – Skip-gram

알고리즘

중심단어를 통해서 주변단어가 되는 단어들의 오차를 학습해서 역전파

그림 3-24 skip-gram 모델의 신경망 구성 예



1 Word2Vec - Skip-gram

알고리즘

1. 윈도우의 크기에 따라 데이터 셋 생성

#1	you	say	goodbye	and	I	say	hello	.
#2	you	say	goodbye	and	I	say	hello	.
#3	you	say	goodbye	and	I	say	hello	.
#4	you	say	goodbye	and	I	say	hello	.
#5	you	say	goodbye	and	I	say	hello	.
#6	you	say	goodbye	and	I	say	hello	.
#7	you	say	goodbye	and	I	say	hello	.
#8	you	say	goodbye	and	I	say	hello	.

원래는 #1부터 진행해야하나
이해의 편의상 #2부터 진행

딥러닝을 이용한 자연어 처리 입문, Word2Vec
밑바닥부터 시작하는 딥러닝2

1 Word2Vec - Skip-gram

알고리즘

2. Forward
- w_in과 w_out의 가중치 초기화

W_in

1.3315865	0.71527897	-1.54540029
-0.00838385	0.62133597	-0.72008556
0.26551159	0.10854853	0.00429143
-0.17460021	0.43302619	1.20303737
-0.96506567	1.02827408	0.22863013
0.44513761	-1.13660221	0.13513688
1.484537	-1.07980489	-1.97772828

W_out

-1.7433723	0.26607016	2.38496733	1.12369125	1.67262221	0.09914922	1.39799638
-0.2712479	0.61320418	-0.2673171	-0.5493090	0.1327083	-0.4761420	1.30847308
0.19501328	0.40020999	-0.3376323	1.25647226	-0.7319695	0.66023155	-0.3508718

1 Word2Vec - Skip-gram

알고리즘

2. Forward
- hidden layer의 값 계산

Input

you	0
say	1
goodbye	0
and	0
I	0
hello	0
.	0

np.dot

W_in

1.3315865	0.71527897	-1.54540029
-0.00838385	0.62133597	-0.72008556
0.26551159	0.10854853	0.00429143
-0.17460021	0.43302619	1.20303737
-0.96506567	1.02827408	0.22863013
0.44513761	-1.13660221	0.13513688
1.484537	-1.07980489	-1.97772828

=

Hidden Layer - h

-0.00838385
0.62133597
-0.72008556

1 Word2Vec - Skip-gram

알고리즘

2. Forward
- Output layer 계산

Hidden Layer - h

-0.00838385
0.62133597
-0.72008556

np.dot

W_out

-1.7433723	0.26607016	2.38496733	1.12369125	1.67262221	0.09914922	1.39799638
-0.27124799	0.61320418	-0.2673171	-0.5493090	0.1327083	-0.4761420	1.30847308
0.19501328	0.40020999	-0.3376323	1.25647226	-0.7319695	0.66023155	-0.3508718

=

Output Layer

-0.29434621
0.09058969
0.05703518
-1.25549384
0.59551409
-0.77209862
1.05393859

1 Word2Vec - Skip-gram

알고리즘

2. Forward
- softmax 계산후에 Error 계산

Output Layer
-0.29434621
0.09058969
0.05703518
-1.25549384
0.59551409
-0.77209862
1.05393859

Softmax - y_pred
0.08945517
0.13145618
0.12711841
0.03421246
0.21780452
0.05547793
0.34447532

Softmax		Token	Diff
0.08945517	1	you	-0.91054483
0.13145618	0	say	0.13145618
0.12711841	0	goodbye	0.12711841
0.03421246	0	and	0.03421246
0.21780452	0	I	0.21780452
0.05547793	0	hello	0.05547793
0.34447532	0	.	0.34447532

Softmax		Token	Diff
0.08945517	0	you	0.08945517
0.13145618	0	say	0.13145618
0.12711841	1	goodbye	-0.87288159
0.03421246	0	and	0.03421246
0.21780452	0	I	0.21780452
0.05547793	0	hello	0.05547793
0.34447532	0	.	0.34447532

Sum of Diff
-0.82108965
0.26291236
-0.74576317
0.06842493
0.43560904
0.11095586
0.68895064

딥러닝을 이용한 자연어 처리 입문, Word2Vec
밑바닥부터 시작하는 딥러닝2

1 Word2Vec - Skip-gram

알고리즘

3. Backward

Hidden Layer

-0.00838385
0.62133597
-0.72008556

np.outer

Sum of Diff

-0.82108965
0.26291236
-0.74576317
0.06842493
0.43560904
0.11095586
0.68895064

=

Delta for W_out

6.8838900e-03	-2.2042200e-03	6.2523700e-03	-5.7366000e-04	-3.6520800e-03	-9.3024000e-04	-5.7760600e-03
-5.1017254e-01	1.6335691e-01	-4.6336949e-01	4.2514870e-02	2.7065957e-01	6.8940860e-02	4.2806982e-01
5.9125480e-01	-1.8931939e-01	5.3701329e-01	-4.9271800e-02	-3.1367578e-01	-7.9897710e-02	-4.9610341e-01

1 Word2Vec - Skip-gram

알고리즘

3. Backward

W_out

-1.7433723	0.26607016	2.38496733	1.12369125	1.67262221	0.09914922	1.39799638
-0.2712479	0.61320418	-0.2673179	-0.5493090	0.1327083	-0.4761420	1.30847308
0.19501328	0.40020999	-0.3376323	1.25647226	-0.7319695	0.66023155	-0.3508718

np.dot

Sum of Diff

-0.8210896
0.26291236
-0.7457631
0.06842493
0.43560904
0.11095586
0.68895064

=

np.dot(W_out, Sum of Diff)

1.50244682
1.45215832
-0.20446484

1 Word2Vec - Skip-gram

알고리즘

3. Backward

Input

0
1
0
0
0
0
0

np.dot(W_out, Sum of Diff)

1.50244682
1.45215832
-0.20446484

np.outer =

Delta for W_input

0	0	0
1.50244682	1.45215832	-0.20446484
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

1 Word2Vec - Skip-gram

알고리즘

3. Backward
- Update

W_in

1.3315865	0.71527897	-1.54540029
-0.00838385	0.62133597	-0.72008556
0.26551159	0.10854853	0.00429143
-0.17460021	0.43302619	1.20303737
-0.96506567	1.02827408	0.22863013
0.44513761	-1.13660221	0.13513688
1.484537	-1.07980489	-1.97772828

— 학습률 *

Delta for W_input

0	0	0
1.50244682	1.45215832	-0.20446484
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

= Updated W_in

1 Word2Vec - Skip-gram

알고리즘

3. Backward
- Update

W_output

-1.7433	0.2660	2.3849	1.1236	1.6726	0.0991	1.3979
-0.2712	0.6132	-0.2673	-0.5493	0.1327	-0.4761	1.3084
0.1950	0.4002	-0.3376	1.2564	-0.7319	0.6602	-0.3508

— 학습률 *

Delta for W_output

6.8838900e-03	-2.2042200e-03	6.2523700e-03	-5.7366000e-04	-3.6520800e-03	-9.3024000e-04	-5.7760600e-03
-5.1017254e-01	1.6335691e-01	-4.6336949e-01	4.2514870e-02	2.7065957e-01	6.8940860e-02	4.2806982e-01
5.9125480e-01	-1.8931939e-01	5.3701329e-01	-4.9271800e-02	-3.1367578e-01	-7.9897710e-02	-4.9610341e-01

= Updated W_output

딥러닝을 이용한 자연어 처리 입문, Word2Vec
밑바닥부터 시작하는 딥러닝2

1 Word2Vec - Skip-gram

알고리즘

4. 남은 '맥락 ' 들에 대해서 3, 4의 과정을 반복

#1	you	say	goodbye	and	I	say	hello	.
#2	you	say	goodbye	and	I	say	hello	.
#3	you	say	goodbye	and	I	say	hello	.
#4	you	say	goodbye	and	I	say	hello	.
#5	you	say	goodbye	and	I	say	hello	.
#6	you	say	goodbye	and	I	say	hello	.
#7	you	say	goodbye	and	I	say	hello	.
#8	you	say	goodbye	and	I	say	hello	.

1 Word2Vec - Skip-gram

코드

gensim 패키지의 Word2Vec을 이용

class gensim.models.word2vec.Word2Vec(*sentences=None, corpus_file=None, size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001, seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5, ns_exponent=0.75, cbow_mean=1, hashfxn=<built-in function hash>, iter=5, null_word=0, trim_rule=None, sorted_vocab=1, batch_words=10000, compute_loss=False, callbacks=(), max_final_vocab=None*) ¶

```python
from gensim.models import Word2Vec

docs = [
  'you say goodbye and I say hello .'
]

sentences = [list(sentence.split(' ')) for sentence in docs]
model = Word2Vec(size=3, window=1, min_count=1, sg=1)
model.build_vocab(sentences)
model.wv.most_similar("say")
```

```
[('and', 0.8423022627830505),
 ('hello', 0.6423846483230591),
 ('goodbye', 0.45526981353759766),
 ('I', 0.2640129029750824),
 ('.', -0.09787103533744812),
 ('you', -0.9029324054718018)]
```

https://radimrehurek.com/gensim/models/word2vec.html

1 컨텐츠 기반 모델

장점

- 협업필터링은 다른 사용자들의 평점이 필요한 반면에, 자신의 평점만을 가지고 추천시스템을 만들 수 있음
- item의 feature를 통해서 추천을 하기에 추천이 된 이유를 설명하기 용이함
- 사용자가 평점을 매기지 않은 새로운 item이 들어올 경우에도 추천이 가능함

단점

- item의 feature을 추출해야 하고 이를 통해서 추천하기때문에 제대로 feature을 추출하지 못하면 정확도가 낮음. 그렇기에 Domain Knowledge가 분석시에 필요할 수도 있음
- 기존의 item과 유사한 item 위주로만 추천하기에 새로운 장르의 item을 추천하기 어려움
- 새로운 사용자에 대해서 충분한 평점이 쌓이기 전까지는 추천하기 힘듬

02 협업 필터링

2 협업필터링 개요

정의

협업필터링은 사용자의 구매 패턴이나 평점을 가지고 다른 사람들의 구매 패턴, 평점을 통해서 추천을 하는 방법입니다. 추가적인 사용자의 개인정보나 아이템의 정보가 없이도 추천할 수 있는게 큰 장점이며 2006부터 2009년동안 열린 Netflix Prize Competition에서 우승한 알고리즘으로 유명새를 떨쳤습니다.

종류

1. 최근접 이웃기반
2. 잠재 요인기반

2 Neighborhood based method

정의

Neighborhood based Collaborative Filtering은 메모리 기반 알고리즘으로 협업 필터링을 위해 개발된 초기 알고리즘입니다.

알고리즘

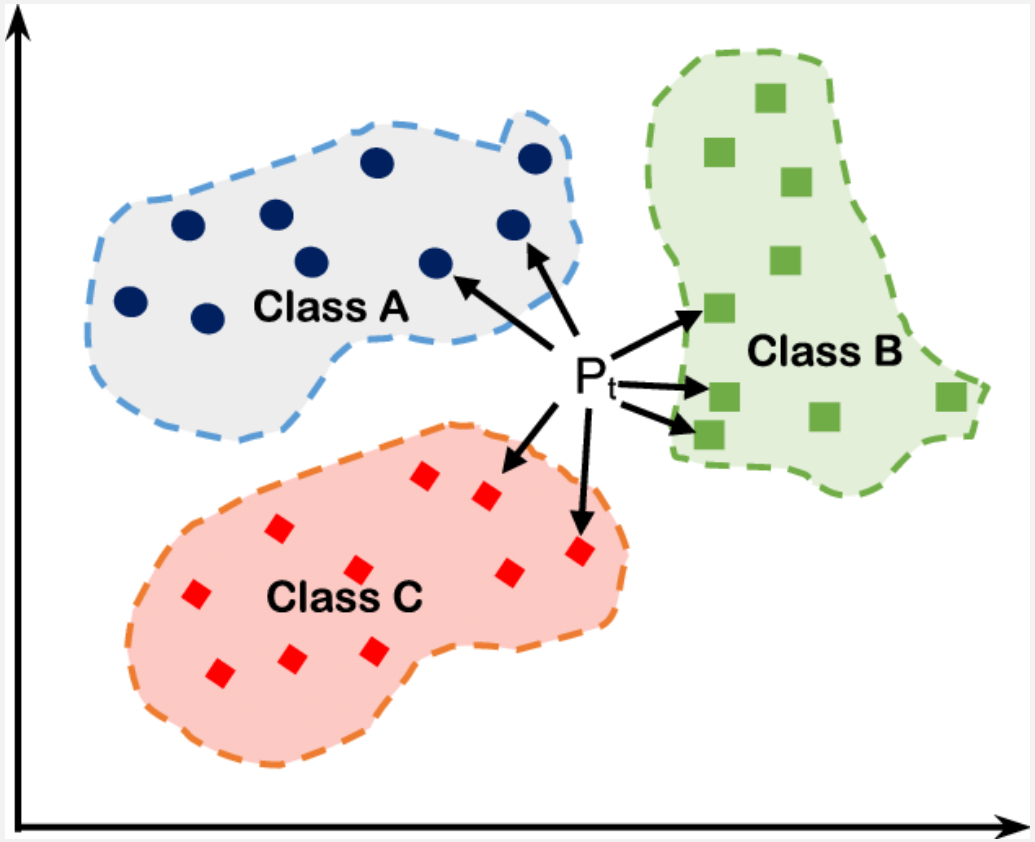
1. User-based collaborative filtering
사용자의 구매 패턴(평점)과 유사한 사용자를 찾아서 추천 리스트 생성

2. Item-based collaborative filtering
특정 사용자가 준 점수간의 유사한 상품을 찾아서 추천 리스트 생성

2 Neighborhood based method – KNN

K Nearest Neighbors

가장 근접한 K 명의 Neighbors를 통해서 예측하는 방법



https://www.researchgate.net/figure/Example-on-KNN-classifier_fig1_331424423

2 Neighborhood based method – KNN

데이터 (Explicit Feedback)

유저가 자신의 선호도를 집적적으로 표현한 데이터

	아이템1	아이템2	아이템3	아이템4	아이템5	아이템6
사용자1	7	6	7	4	5	4
사용자2	6	7	?	4	3	4
사용자3	?	3	3	1	1	?
사용자4	1	2	2	3	3	4
사용자5	1	?	1	2	3	3

2 Neighborhood based method – KNN

User Based Collaborative Filtering

유저가 자신의 선호도를 집적적으로 표현한 데이터

	아이템1	아이템2	아이템3	아이템4	아이템5	아이템6	평균	Cosine(i, 3)	Pearson(i, 3)
사용자1	7	6	7	4	5	4	5.5	0.956	0.894
사용자2	6	7	?	4	3	4	4.8	0.981	0.939
사용자3	?	3	3	1	1	?	2	1.0	1.0
사용자4	1	2	2	3	3	4	2.5	0.789	-1.0
사용자5	1	?	1	2	3	3	2	0.645	-0.817

2 Neighborhood based method – KNN

User Based Collaborative Filtering

유저가 자신의 선호도를 집적적으로 표현한 데이터

	평균	Cosine(i, 3)	Pearson(i, 3)
사용자1	5.5	0.956	0.894
사용자2	4.8	0.981	0.939
사용자3	2	1.0	1.0
사용자4	2.5	0.789	-1.0
사용자5	2	0.645	-0.817

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u,v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u,v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u,v)|}$$

$$\text{Cosine}(1,3) = \frac{6*3+7*3+4*1+5*1}{\sqrt{6^2+7^2+4^2+5^2} \cdot \sqrt{3^2+3^2+1^2+1^2}} = 0.956$$

$$\text{Pearson}(1,3) =$$

$$= \frac{(6-5.5)*(3-2)+(7-5.5)*(3-2)+(4-5.5)*(1-2)+(5-5.5)*(1-2)}{\sqrt{1.5^2+1.5^2+(-1.5)^2+(-0.5)^2} \cdot \sqrt{1^2+1^2+(-1)^2+(-1)^2}}$$

$$= 0.894$$

2 Neighborhood based method – KNN

User Based Collaborative Filtering

유저가 자신의 선호도를 집적적으로 표현한 데이터

	아이템1	아이템6	평균	Pearson(i, 3)
사용자1	7	4	5.5	0.894
사용자2	6	4	4.8	0.939
사용자3	3.35	0.86	2	1.0
사용자4	1	4	2.5	-1.0
사용자5	1	3	2	-0.817

$$\hat{r}_{31} = \frac{7 * 0.894 + 6 * 0.939}{0.894 + 0.939} \approx 6.49$$

$$\hat{r}_{36} = \frac{4 * 0.894 + 4 * 0.939}{0.894 + 0.939} = 4$$

사용자1의 아이템1의 평점 – 사용자1의 평균 평점

$$\hat{r}_{31} = 2 + \frac{1.5 * 0.894 + 1.2 * 0.939}{0.894 + 0.939} \approx 3.35$$

$$\hat{r}_{36} = 2 + \frac{-1.5 * 0.894 - 0.8 * 0.939}{0.894 + 0.939} \approx 0.86$$

사용자3의 평균 평점

2 Neighborhood based method – KNN

Item Based Collaborative Filtering

유저가 자신의 선호도를 집적적으로 표현한 데이터

	아이템1	아이템2	아이템3	아이템4	아이템5	아이템6	평균
사용자1	7	6	7	4	5	4	5.5
사용자2	6	7	?	4	3	4	4.8
사용자3	?	3	3	1	1	?	2
사용자4	1	2	2	3	3	4	2.5
사용자5	1	?	1	2	3	3	2
Cosine(1, j)	1	0.735	0.912	-0.848	-0.813	-0.990	
Cosine(6, j)	-0.990	-0.622	-0.912	0.829	0.730	1	

2 Neighborhood based method – KNN

Item Based Collaborative Filtering

유저가 자신의 선호도를 집적적으로 표현한 데이터

$$\text{AdjustedCosine}(1,3) = \frac{1.5 * 1.5 + (-1.5) * (-0.5) + (-1) * (-1)}{\sqrt{1.5^2 + (-1.5)^2 + (-1)^2} \cdot \sqrt{1.5^2 + (-0.5)^2 + (-1)^2}} = 0.912$$

	아이템1	아이템2	아이템3	아이템4	아이템5	아이템6	평균
사용자1	1.5	0.5	1.5	-1.5	-0.5	-1.5	5.5
사용자2	1.2	2.2	?	-0.8	-1.8	-0.8	4.8
사용자3	?	1	1	-1	-1	?	2
사용자4	-1.5	-0.5	-0.5	0.5	0.5	1.5	2.5
사용자5	-1	?	-1	0	1	1	2
Cosine(1, j)	1	0.735	0.912	-0.848	-0.813	-0.990	
Cosine(6, j)	-0.990	-0.622	-0.912	0.829	0.730	1	

2 Neighborhood based method – KNN

Item Based Collaborative Filtering

	아이템1	아이템2	아이템3	아이템4	아이템5	아이템6
사용자1	7	6	7	4	5	4
사용자2	6	7	?	4	3	4
사용자3	3	3	3	1	1	1
사용자4	1	2	2	3	3	4
사용자5	1	?	1	2	3	3
Cosine(1, j)	1	0.735	0.912	-0.848	-0.813	-0.990
Cosine(6, j)	-0.990	-0.622	-0.912	0.829	0.730	1

$$\hat{r}_{31} = \frac{3 * 0.735 + 3 * 0.912}{0.735 + 0.912} = 3$$

$$\hat{r}_{36} = \frac{1 * 0.829 + 1 * 0.730}{0.829 + 0.730} = 1$$

2 Neighborhood based method

장점

- 간단하고 직관적인 접근 방식 때문에 구현 및 디버그가 쉬움
- 특정 Item을 추천하는 이유를 정당화하기 쉽고 Item 기반 방법의 해석 가능성이 두드러짐
- 추천 리스트에 새로운 item과 user가 추가되어도 상대적으로 안정적

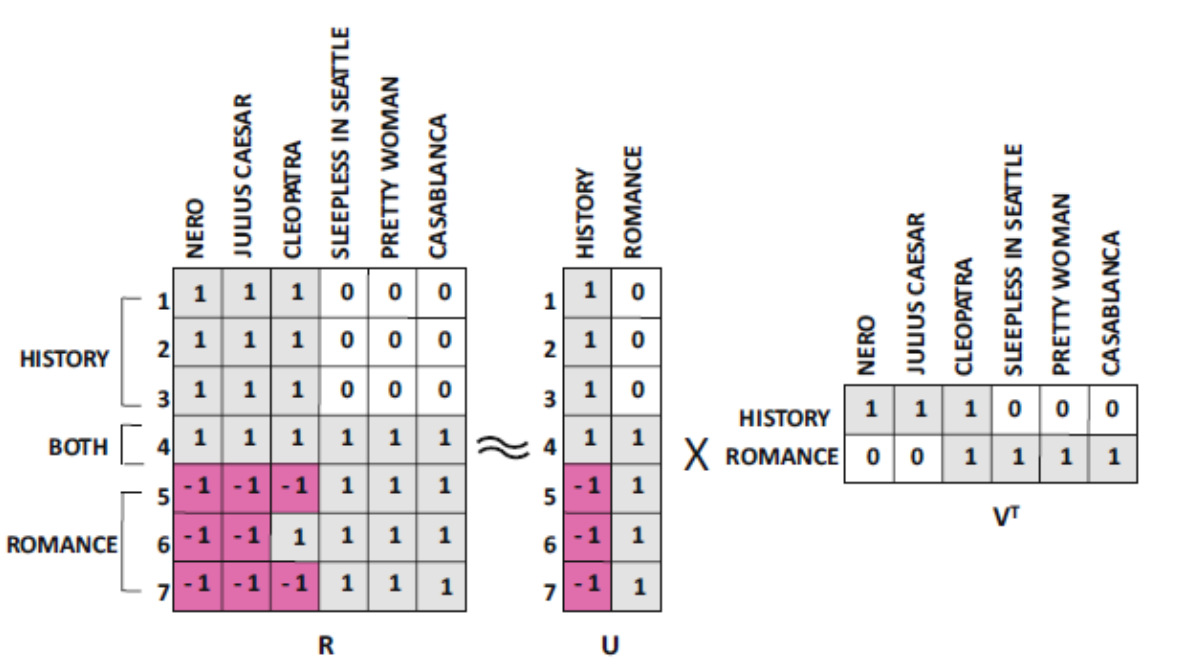
단점

- User 기반 방법의 시간, 속도, 메모리가 많이 필요
- 희소성 때문에 제한된 범위가 있음
  - John의 Top-K 에만 관심이 있음
  - John과 비슷한 이웃중에서 아무도 해리포터를 평가하지 않으면, John의 해리포터에 대한 등급 예측을 제공할 수가 없음

2 Latent Factor Collaborative Filtering

정의

잠재 요인 협업 필터링은 Rating Matrix에서 빈 공간을 채우기 위해서 사용자와 상품을 잘 표현하는 차원(Latent Factor)을 찾는 방법입니다. 잘 알려진 행렬 분해는 추천 시스템에서 사용되는 협업 필터링 알고리즘의 한 종류입니다. 행렬 분해 알고리즘은 사용자-아이템 상호 작용 행렬을 두 개의 저 차원 직사각형 행렬의 곱으로 분해하여 작동합니다. 이 방법은 Netflix 챌린지에서 널리 알려지게되었습니다



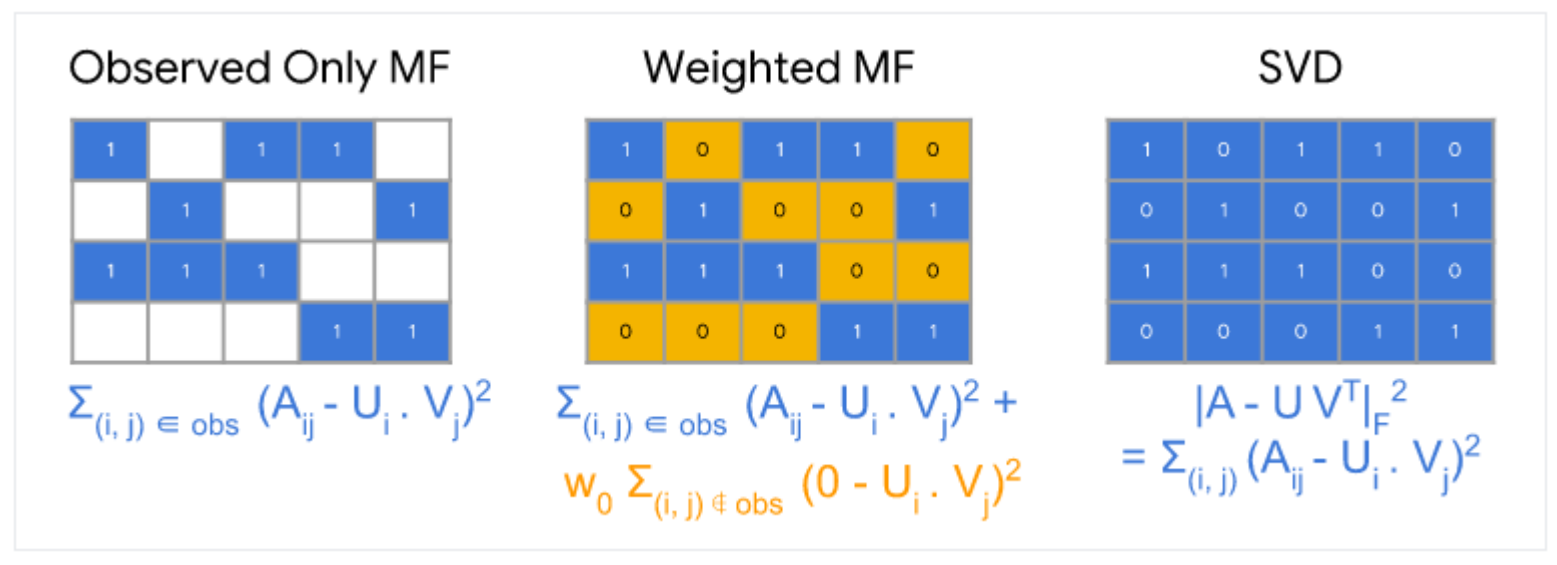
(a) Example of rank-2 matrix factorization

2 Matrix Factorization Principles

원리

사용자의 잠재요인과 아이템의 잠재요인을 내적해서 평점 매트릭스를 계산

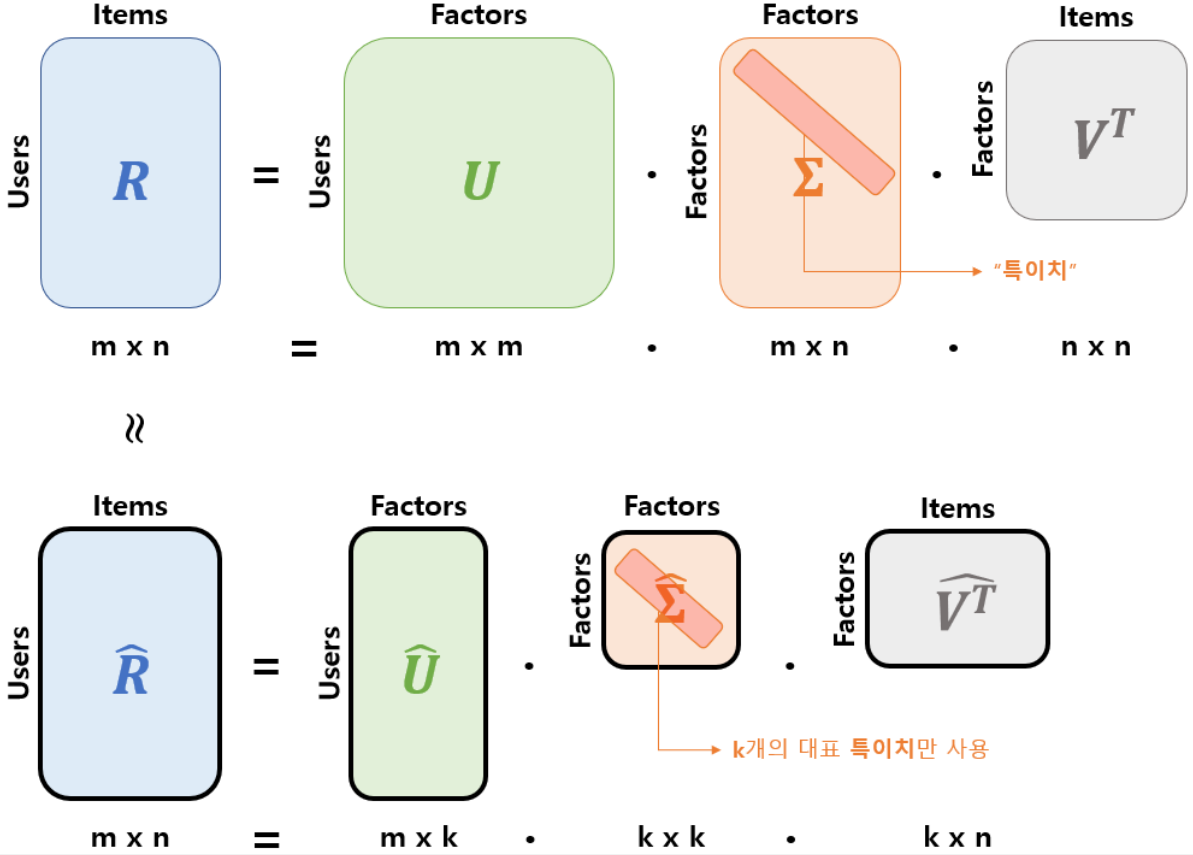
$$R \approx UV^T$$



2 SVD

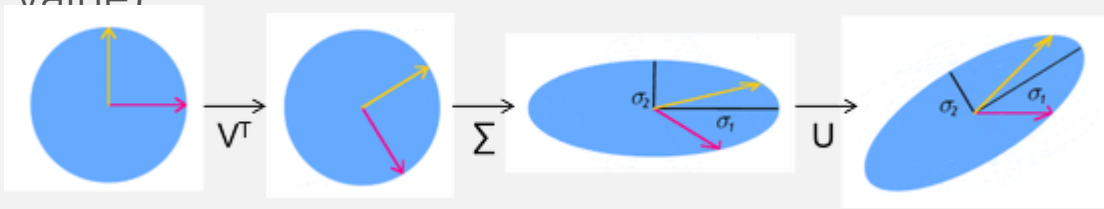
정의

고유값 분해(eigen value Decomposition)와 같은 행렬을 대각화 하는 방법



$$R = U\Sigma V^T$$

U: (m, m), R의 left singular 벡터로 구성된 직교행렬
V: (n, n), R의 right singular 벡터로 구성된 직교행렬
Σ: (m, n), 주 대각성분이 √λi인 직사각 대각행렬(Singular value)



https://seing.tistory.com/67

2 SVD

한계

1. 데이터에 결측치가 없어야 함
2. 대부분의 현업 데이터는 Sparse한 데이터

https://seing.tistory.com/67

2 SGD

정의

고유값 분해(eigen value Decomposition)와 같은 행렬을 대각화 하는 방법

$$\text{Minimize } J = \frac{1}{2}||R - UV^T||^2$$
$$\text{subject to:}$$
$$\text{No constraints on } U \text{ and } V$$
$$S = \{(i,j) : r_{ij} \text{ is observed}\}$$

$$\text{Minimize } J = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2 = \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2$$
$$\text{subject to:}$$
$$\text{No constraints on } U \text{ and } V$$

Gradient Descent에 의해 편미분 된 값

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-v_{jq}) \quad \forall i \in \{1\dots m\}, q \in \{1\dots k\}$$
$$= \sum_{j:(i,j)\in S}(e_{ij})(-v_{jq}) \quad \forall i \in \{1\dots m\}, q \in \{1\dots k\}$$
$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-u_{iq}) \quad \forall j \in \{1\dots n\}, q \in \{1\dots k\}$$
$$= \sum_{i:(i,j)\in S}(e_{ij})(-u_{iq}) \quad \forall j \in \{1\dots n\}, q \in \{1\dots k\}$$

2 SGD

Regularization

고유값 분해(eigen value Decomposition)와 같은 행렬을 대각화 하는 방법

$$\text{Minimize } J = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\sum_{s=1}^{k} v_{js}^2$$

$$= \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\sum_{s=1}^{k} v_{js}^2$$

Gradient Descent에 의해 편미분 된 값

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1\ldots m\}, q \in \{1\ldots k\}$$

$$= \sum_{j:(i,j)\in S}(e_{ij})(-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1\ldots m\}, q \in \{1\ldots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1\ldots n\}, q \in \{1\ldots k\}$$

$$= \sum_{i:(i,j)\in S}(e_{ij})(-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1\ldots n\}, q \in \{1\ldots k\}$$

$$u_{iq} \Leftarrow u_{iq} + \alpha\left(\sum_{j:(i,j)\in S} e_{ij}\cdot v_{jq} - \lambda\cdot u_{iq}\right) \quad \forall q \in \{1\ldots k\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha\left(\sum_{i:(i,j)\in S} e_{ij}\cdot u_{iq} - \lambda\cdot v_{jq}\right) \quad \forall q \in \{1\ldots k\}$$

2 SGD

예시

Explict Feedback 된 형태의 4명의 유저에 대한 3개의 아이템에 대한 평점 Matrix

Rating Matrix

?	3	2
5	1	2
4	2	1
2	?	4

2 SGD

예시

1. User Latent 와 Item Latent의 임의로 초기화

User Latent (U)

0.5756	1.4534
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

np.dot

Item Latent (V)의 Transpose

0.3668	-1.1078	1.4593
-0.3392	0.8972	0.4528

=

-0.2819	0.6663	1.4981
0.3403	-0.8728	-0.8421
0.8384	-2.5933	4.2008
0.8354	-2.2043	-1.1928

2 SGD

예시

2. Gradient Descent 진행

?	3	2
5	1	2
4	2	1
2	?	4

-0.2819	0.6663	1.4981
0.3403	-0.8728	-0.8421
0.8384	-2.5933	4.2008
0.8354	-2.2043	-1.1928

0.5756	1.4534
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

0.3668	-1.1078	1.4593
-0.3392	0.8972	0.4528

Error : 3 – 0.6663 = 2.3337
dUser = -2.3337*[-1.1078 , 0.8972] + 0.01 *[0.5756, 1.4534]
dItem = -2.3337*[0.5756, 1.4534] + 0.01 *[-1.1078 , 0.8972]

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \dots m\}, q \in \{1 \dots k\}$$

$$= \sum_{j:(i,j)\in S} (e_{ij})(-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \dots m\}, q \in \{1 \dots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1 \dots n\}, q \in \{1 \dots k\}$$

$$= \sum_{i:(i,j)\in S} (e_{ij})(-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1 \dots n\}, q \in \{1 \dots k\}$$

2 SGD

0.5756	1.4534
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

0.3668	-1.1078	1.4593
-0.3392	0.8972	0.4528

예시

2. Gradient Descent 진행

Error : 3 – 0.6663 = 2.3337
dUser = -2.3337*[-1.1078 , 0.8972] + 0.01 *[0.5756, 1.4534]
dItem = -2.3337*[0.5756, 1.4534] + 0.01 *[-1.1078 , 0.8972]

Updated User Latent = [0.5756, 1.4534] - Learning rate * dUser
[0.446, 1.5574] = [0.5756, 1.4534] – 0.05 * [2.591, -2.0793]

Updated User Latent = [-1.1078, 0.8972] - Learning rate * dItem
[-1.0401, 1.0663] = [-1.1078 , 0.8972] – 0.05 * [-1.3544, -3.3828]

0.446	1.5574
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

0.3668	-1.0401	1.4593
-0.3392	1.0663	0.4528

-0.3647	1.1967	1.3560
0.3403	-0.8728	-0.8421
0.8384	-2.5933	4.2008
0.8354	-2.2043	-1.1928

2 SGD

예시

3. 모든 평점에 대해서 반복 (epoch : 1)

- ?를 제외한 모든 평점에 대해서 진행

?	3	2
5	1	2
4	2	1
2	?	4

-0.3522	1.1628	1.5157
0.3403	-1.0924	-0.9057
0.8384	-2.3273	4.2620
0.8354	-2.6308	-1.3184

0.4928	1.5711
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

0.3668	-1.0401	1.4729
-0.3392	1.0663	0.5027

2 SGD

예시

3. 모든 평점에 대해서 반복 (epoch : 1)

- ?를 제외한 모든 평점에 대해서 진행

?	3	2
5	1	2
4	2	1
2	?	4

-0.8209	1.1628	1.5157
0.7715	-1.2650	-0.8190
0.5752	-2.3273	4.2619
1.5482	-2.6308	-1.3184

0.4928	1.5711
-0.113	-1.296
2.7297	0.48
-0.039	-2.506

0.3203	-1.0401	1.4729
-0.6230	1.0663	0.5027

2 SGD

예시

3. 모든 평점에 대해서 반복 (epoch : 1)

- ?를 제외한 모든 평점에 대해서 진행

?	3	2
5	1	2
4	2	1
2	?	4

-0.5947	1.3736	-0.0028
0.6763	-1.0994	0.3561
1.4991	-0.4721	2.3222
1.7260	-2.5722	1.0869

0.4927	1.5711
-0.015	-1.101
2.3345	0.5363
0.2363	-2.464

0.7858	-0.4137	1.0724
-0.6250	1.0040	-0.3381

2 SGD

예시

4. 2~3의 과정을 10번 반복 (epoch : 10)

1.6462	1.0993	
1.6740	-1.072	
2.0550	0.6342	
0.3135	-2.663	

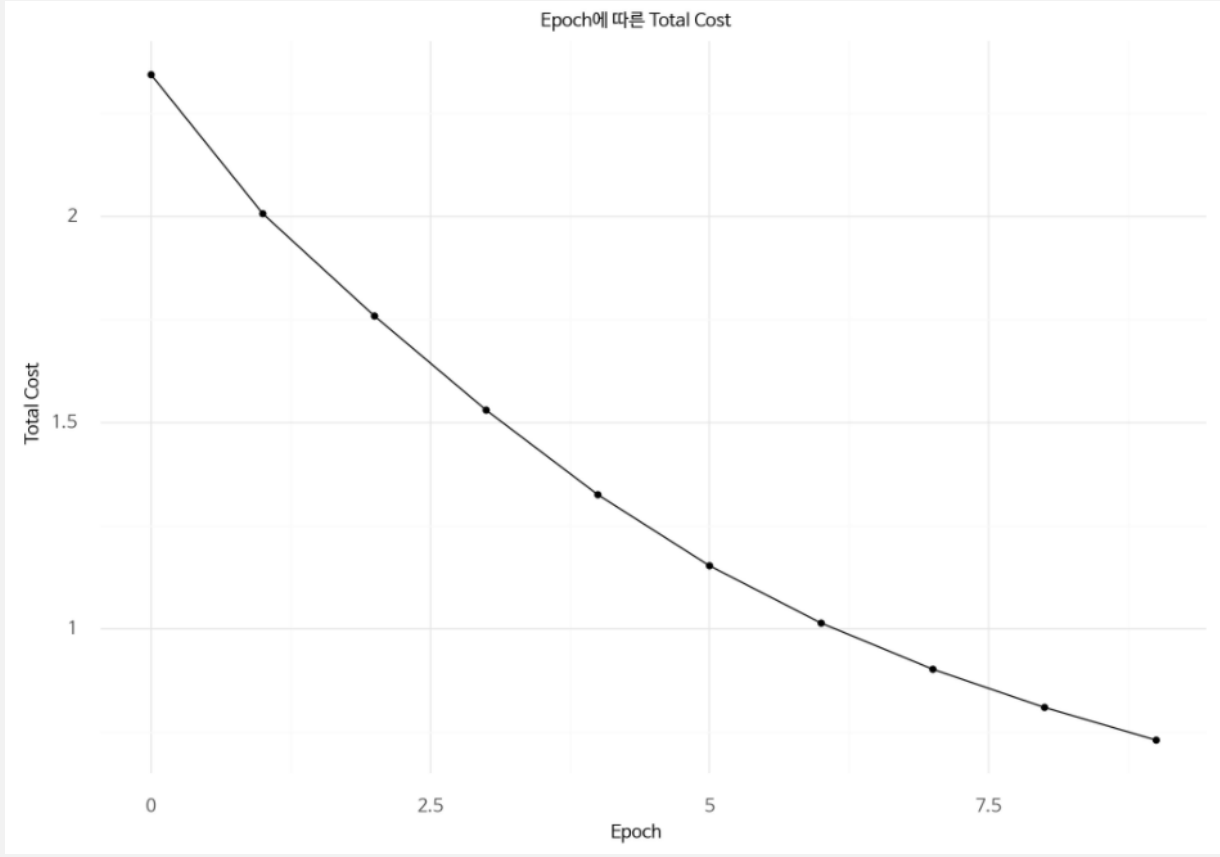
2.1118	0.8951	0.9768
-0.6646	0.8560	-1.1103

?	3	2
5	1	2
4	2	1
2	?	4

평점이 높은 1번 상품은 1번 유저에게 추천 x

2.7458	2.4147	0.3873
4.2476	0.5806	2.8256
3.9181	2.3825	1.3030
2.4323	-1.9994	3.2637

평점이 낮은 2번 상품은 4번 유저에게 추천 x



2 ALS

정의

기존의 SGD가 두개의 행렬(User Latent, Item Latent)을 동시에 최적화하는 방법이라면, ALS는 두 행렬 중 하나를 고정시키고 다른 하나의 행렬을 순차적으로 반복하면서 최적화 하는 방법입니다. 이렇게 하면, 기존의 최적화 문제가 convex 형태로 바뀌기에 수렴된 행렬을 찾을 수 있는 장점이 있습니다.

알고리즘

1. 초기 아이템, 사용자 행렬을 초기화
2. 아이템 행렬을 고정하고 사용자 행렬을 최적화
3. 사용자 행렬을 고정하고 아이템 행렬을 최적화
4. 위의 2, 3 과정을 반복

2 ALS

수렴

기존의 SGD가 두개의 행렬(User Latent, Item Latent)을 동시에 최적화하는 방법이라면, ALS는 두 행렬 중 하나를 고정시키고 다른 하나의 행렬을 순차적으로 반복하면서 최적화 하는 방법입니다. 이렇게 하면, 기존의 최적화 문제가 convex 형태로 바뀌기에 수렴된 행렬을 찾을 수 있는 장점이 있습니다.

$$||y - X\beta||_2$$

$$\beta = (X^T X)^{-1} X^T y$$

$$\forall u_i : J(u_i) = ||R_{i.} - u_i \times V^T||_2 + \lambda \cdot ||u_i||_2$$

$$\forall v_j : J(v_j) = ||R_{i.} - U \times v_j{}^T||_2 + \lambda \cdot ||v_j||_2$$

$$u_i = (V^T \times V + \lambda I)^{-1} \times V^T \times R_{i.}$$

$$v_j = (U^T \times U + \lambda I)^{-1} \times U^T \times R_{.j}$$

0	3	2
5	1	2
4	2	1
2	0	4

?의 경우는 모두 0으로 바꿔줍니다.

출처 : Codelog (통계쟁이 엔지니어), Matrix Factorization에 대해 이해, Alternating Least Square (ALS) 이해

2 ALS

알고리즘

1. 초기 아이템, 사용자 행렬을 초기화

User Latent

0.5756	1.4534
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

Item Latent 의 Transpose

0.3668	-1.1078	1.4593
-0.3392	0.8972	0.4528

2 ALS

알고리즘

2. 아이템 행렬을 고정하고 사용자 행렬을 최적화 $u_i = (V^T \times V + \lambda I)^{-1} \times V^T \times R_i.$

0.3151	3.2962
1.1118	0.5423
0.3225	0.9144
2.1187	1.8521

모든 Row에 대해서 진행

$u_1 = (V^T \times V + \lambda I)^{-1} \times V^T \times R_1. = [0.3151, 3.2962]$

2 ALS

알고리즘

3. 사용자 행렬을 고정하고 아이템 행렬을 최적화 $v_j = (U^T \times U + \lambda I)^{-1} \times U^T \times R_{.j}$

1.9557	-0.090
-0.525	0.9939
1.5017	0.4663

모든 Row에 대해서 진행

$v_1 = (U^T \times U + \lambda I)^{-1} \times U^T \times R_{.1} = [1.9557, -0.0900]$

2 ALS

알고리즘

4. 2와 3의 과정을 설정한 Iterations 만큼 반복

2 ALS

코드

implicit 패키지의 ALS를 사용

```
from implicit.evaluation import  *
from implicit.als import AlternatingLeastSquares as ALS
from implicit.bpr import BayesianPersonalizedRanking as BPR
import scipy
```

```
R = scipy.sparse.csr_matrix(np.array([[0, 3, 2],
                                      [5, 1, 2],
                                      [4, 2, 1],
                                      [2, 0, 4]]))
```

```
als_model = ALS(factors=2, regularization=0.01, iterations = 10)
als_model.fit(R.T)
```

```
100% 10/10 [00:44<00:00, 4.45s/it]
```

```
np.dot(als_model.user_factors, als_model.item_factors.T)
```

```
array([[0.16003628, 1.0713842 , 0.85777044],
       [0.97194386, 0.78128916, 1.143041  ],
       [0.96414506, 0.91053617, 1.2301167 ],
       [1.1111488 , 0.3173616 , 0.897782  ]], dtype=float32)
```

```
print("user factors")
print(als_model.user_factors)

print("₩nItem factors")
print(als_model.item_factors.T)
```

```
user factors
[[0.5168162 2.7060897]
 [2.3859909 3.6030152]
 [2.384663  3.877818 ]
 [2.652012  2.8285315]]

Item factors
[[ 0.44694868 -0.38000512  0.00056835]
 [-0.02622014  0.46849036  0.3168693 ]]
```

2 협업필터링

장점

- 도메인 지식이 필요하지 않음
- 사용자의 새로운 흥미를 발견하기 좋음
- 시작단계의 모델로 선택하기 좋음 (추가적인 문맥정보등의 필요가 없음)

단점

- 새로운 아이템에 대해서 다루기가 힘듬
- side features(고객의 개인정보, 아이템의 추가정보)를 포함시키기 어려움

2 협업필터링 - SGD

장점

- 매우 유연한 모델로 다른 Loss function을 사용할 수 있음
- parallelized가 가능함

단점

- 수렴까지 속도가 매우 느림

2 협업필터링 - ALS

장점

- SGD보다 수렴속도가 빠름
- parallelized가 가능함

단점

- 오직 Loss Squares만 사용가능

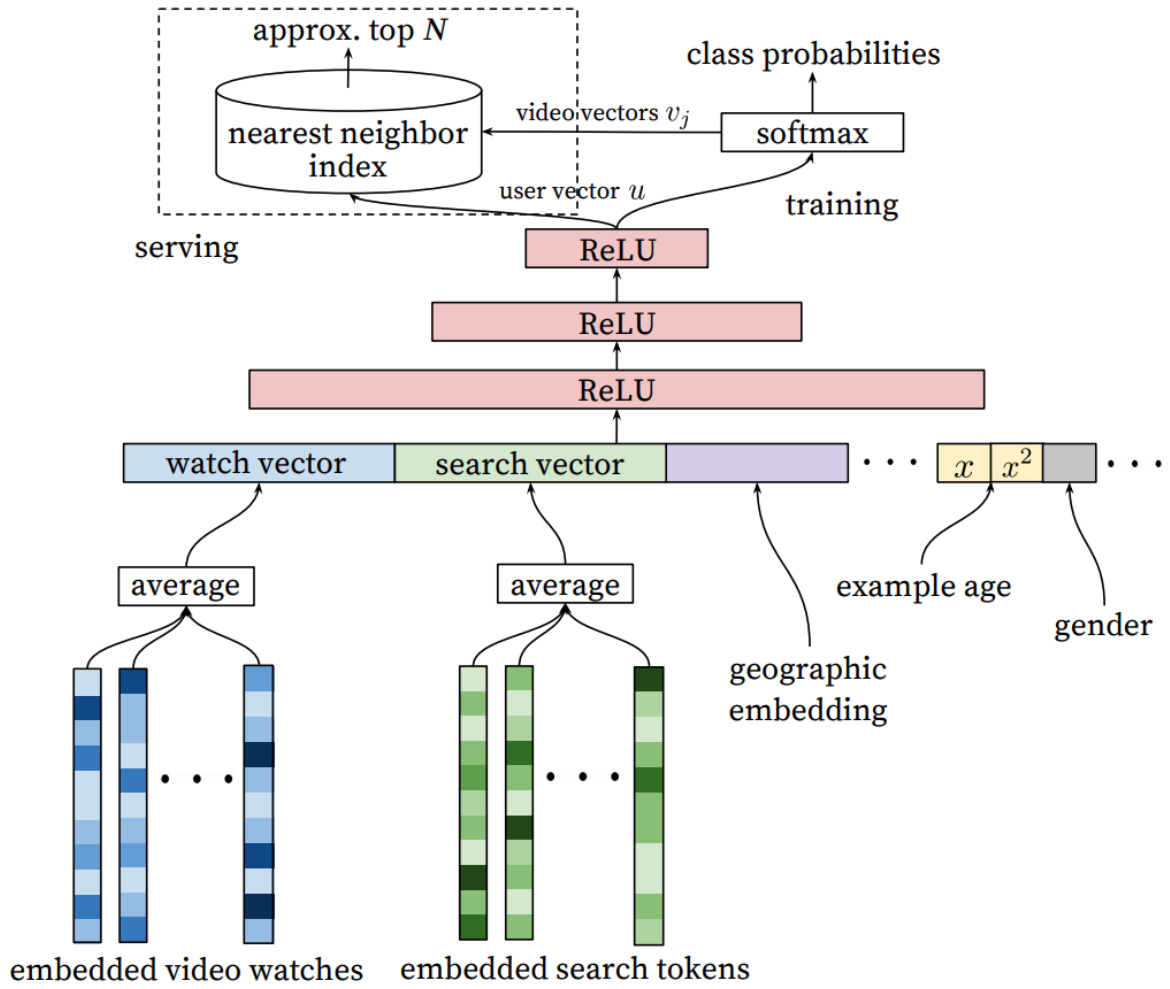
03 실습

3 실습 환경

링크 : https://arena.kakao.com/c/6

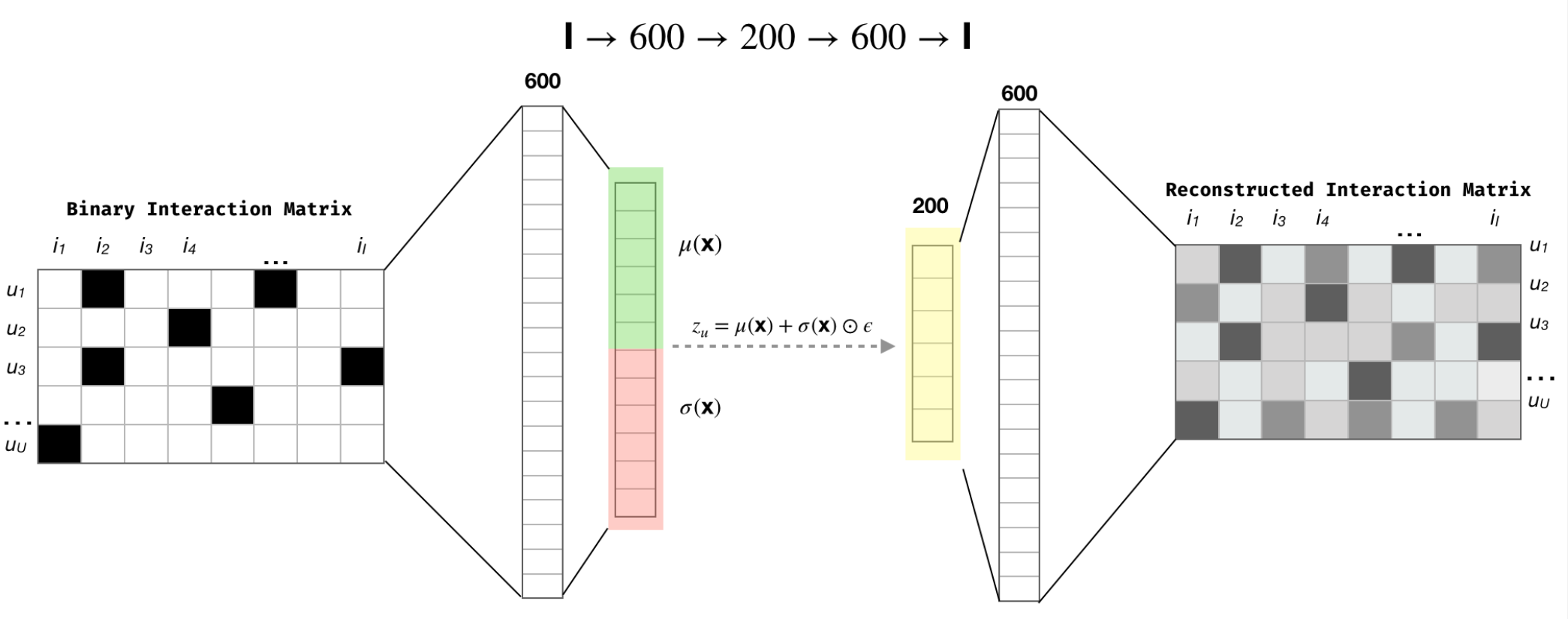
04 도전 과제

4 딥러닝



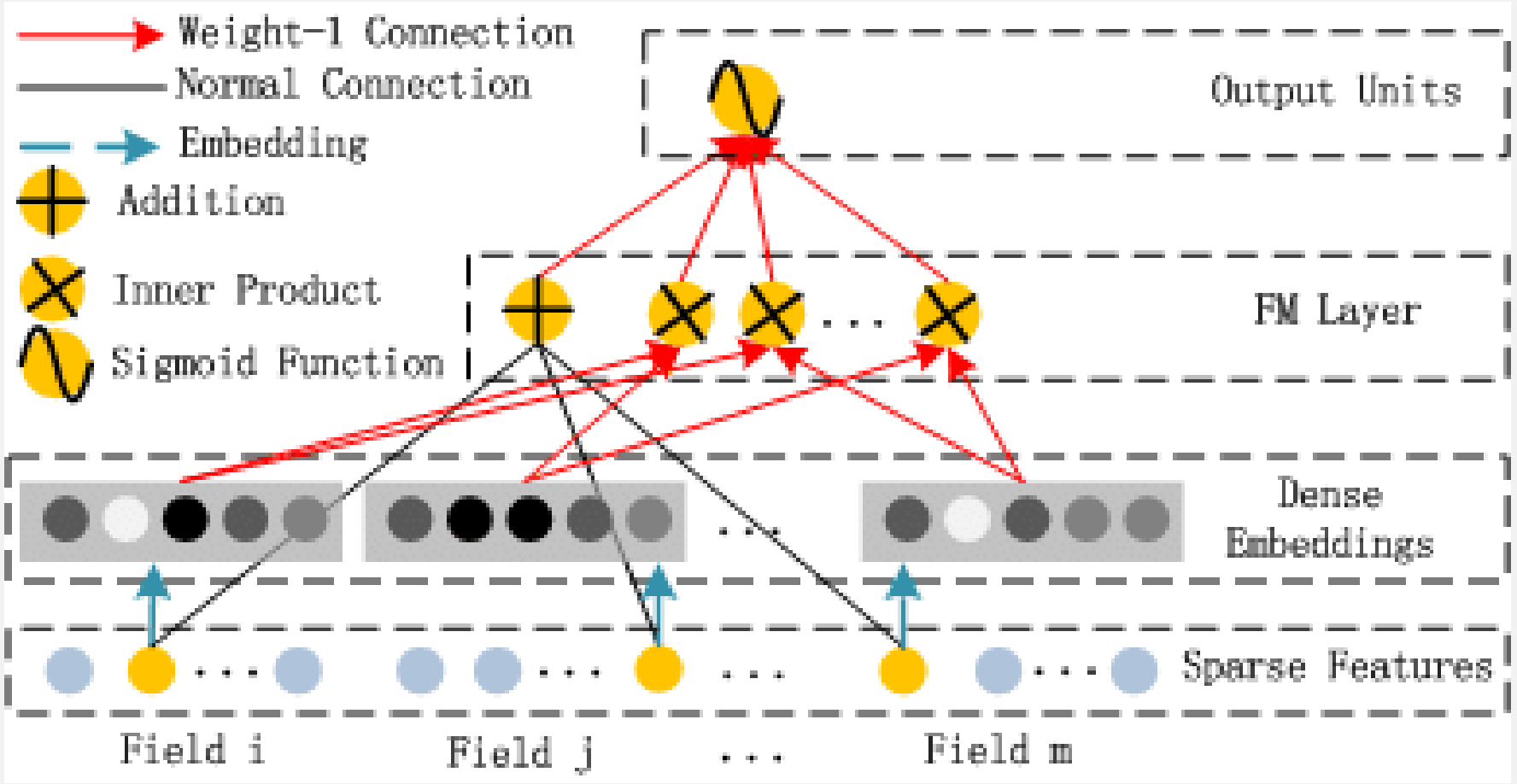
출처 : Deep Neural Networks for YouTube Recommendations (2016)

4 Variational Autoencoders for Collaborative Filtering



출처 : Variational Autoencoders for Collaborative Filtering (2018)

4 Factorization Machine



출처 : DeepFM: A Factorization-Machine based Neural Network for CTR Prediction (2017)

4 강화학습

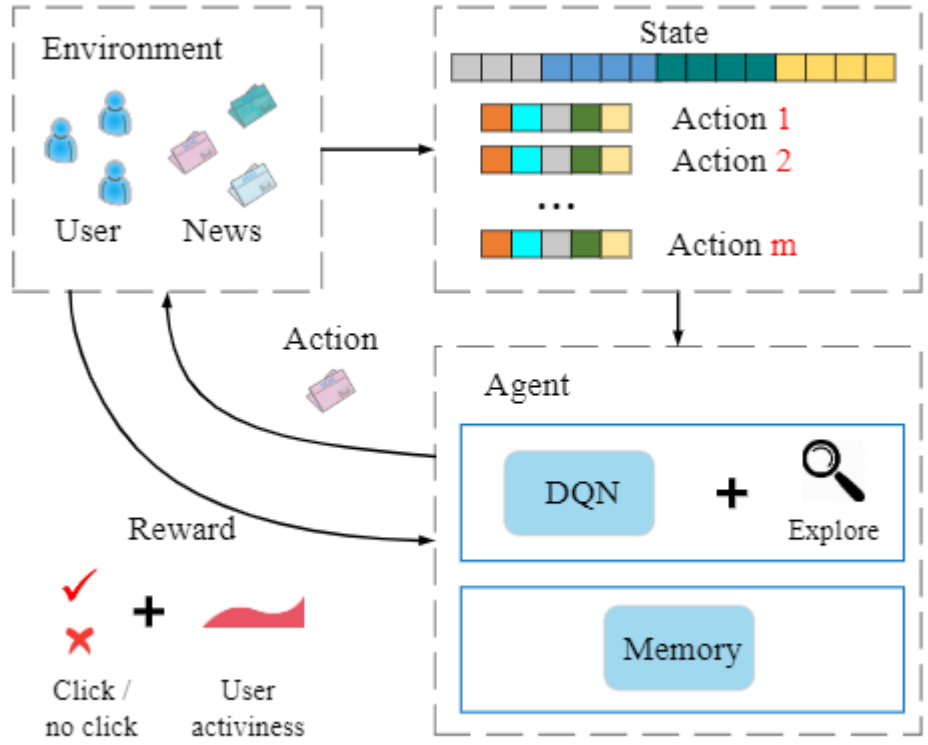
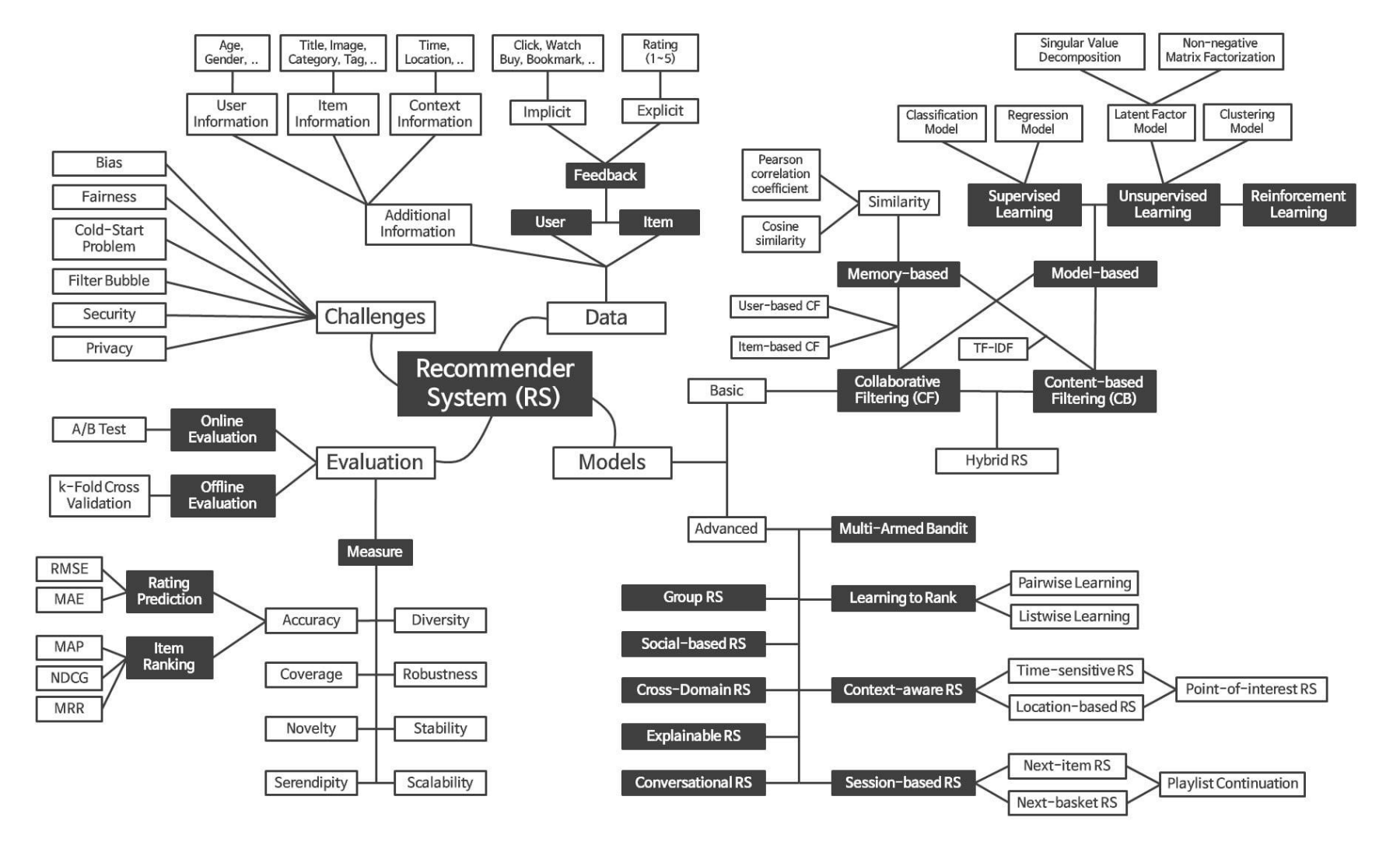


Figure 2: Deep Reinforcement Recommendation System

출처 : DRN: A Deep Reinforcement Learning Framework for News Recommendation

4 그 외



https://www.facebook.com/groups/2611614312273351/abo

5 참고자료

1. Word2vec

밑바닥 부터 시작하는 딥러닝
딥러닝을 이용한 자연어 처리 입문
ratsgo, Word2vec
An implementation guide to Word2Vec using NumPy and Google Sheets, Drerek chai

2. Matrix Factorization

굿, 추천 알고리즘 - SVD (Singular Value Decomposition)
Codelog (통계쟁이 엔지니어), Matrix Factorization에 대해 이해, Alternating Least Square (ALS) 이해
Developer and Analyst, YW & YY. Matrix Factorization 설명 및 논분 리뷰
JuHyung Son, 추천 시스템 들어가기 (Collaborative Filtering, Singular Value Decomposition)

감사합니다