 # 머하웃 프로젝트 위키

https://cwiki.apache.org/confluence/display/MAHOUT/Algorithms

> 머하웃 집중 영역 : 추천엔진(협업필터링), 군집, 분류

- 추천엔진 :
  - 현재 사용되는 기계학습 분야에서 가장 이해하기 쉬운 영역
  - 사용자의 취향과 선호를 추론해서 관심을 가질만한 책이나 영화, 뉴수 가서 등을 추천하는 서비스

- 군집 :
  - 많은 수의 사물을 여러 개의 유사성이 높은 클러스터 그룹으로 나눔.
  - 양이 많아서 이해하기 어려운 데이터 셋의 계층을 발견하고, 흥미로운 패턴을 도출하거나 데이터셋을 이해하기 쉽게 만들 수 있음.
  - 파악하기 힘든 대규모 사물의 구조나 계층 체계를 파악하기 쉬워짐.

    기업 환경에서는 군집 기술을 활용해서 숨겨진 사용자 그룹을 발견하거나 많은 수의 문서를 이해하기 쉽게 구조화하거나 사이트 로그를 분석해 공통적인 사용 패턴을 찾기도 함.

- 분류 :
  - 어떤 사물이 특정 카테고리에 종속되는지 또는 특정 속성을 포함하는지 결정할 수 있음.

    알고리즘은 좀 더 복잡

> 특징

- 규모 확장성을 가장 높은 우선순위

2.2.1 입력 데이터 만들기

일단 추천기에 입력되고 처리될 데이터가 필요하다.
이러한 데이터를 머하웃에서는 선호(preference)라고 한다.
사용자에게 아이템을 추천해주는 추천엔진이 우리에게 가장 친숙하므로, 사용자와 아이템과의 관계를 선호라고 부르는 것이 편리할 것이다.

물론 앞서 말한 대로 사용자와 아이템은 어떤 것이든 될 수 있다.
이런 선호 정보는 주로 사용자 ID와 아이템 ID 그리고 사용자의 아이템 선호 수준을 표현하는 숫자로 구성된다.

머하웃에서 ID는 언제나 숫자인 정수를 사용한다.
큰 값이 선호도가 높다는 규칙만 지킨다면 선호값(preference value)은 어떤 수든 상관 없다.
예를 들어 '1'에서 '5'의 5점 척도를 사용할 경우는 '1'은 사용자가 싫어하는 것을 나타내고 '5'는 좋아하는 것을 나타낸다.

3.1 선호 데이터 표현

추천엔진에 입력하는 데이터는 누가, 무엇을, 얼마나 좋아하는지 표현한 선호 데이터다.
이 선호 데이터는 사용자 ID, 아이템 ID, 선호값 튜플tuple로 구성되며, 대용량으로 머하웃 추천기에 입력된다.
어떤 경우에는 선호값조차 생략되는 경우도 있다.

##############################

- UserSimilarity : 사용자간의 유사도 개념을 캡슐화한 클래스
- UserNeighborhood : 가장 유사한 사용자 그룹의 개념을 캡슐화한 클래스

 # 추천기용 컴포넌트

* DataModel : 데이터 연산을 위해 모든 선호, 사용자, 그리고 아이템 정보를 저장하거나 접근하도록 한다.
* UserSimilarity : 하나 혹은 여러 개의 가능한 측정 혹은 연산을 통해 얼마나 두 사용자가 유사한지를 구한다.
* UserNeighborhood : 주어진 사용자와 가장 유사한 사용자 그룹을 알려준다.
* Recommender : 모든 컴포넌트를 함께 활용하여 사용자에게 아이템을 추천한다.

 # 고정 크기 이웃

> 클래스 : NearestNUserNeighborhood([이웃의 수], similarity, model)

* 가장 유사한 사용자로 구성된 n명의 이웃에서 추천할 대상을 구성
* 100명에서 10명으로 상대적으로 적은 수의 유사한 사용자를 기반으로 추천하게 되면, 유사성이 비교적 덜한 사용자는 분석 대상에서 빠지게 됨.
* 사용자들끼리 거리가 멀수록 덜 유사함
* 향상도(피어슨) : 10=0.91 < 100=0.83 , 평가값이 더 큰 경우가 좋지 않다는 의미

 # 임계치 기반 이웃

> 클래스 : ThresholdUserNeighborhood([임계치], similarity, model)

* 임계치는 -1에서 1사이로 설정해야 함
  이유는 모든 유사도 측정이 이 범위의 유사도값을 반환하기 때문.

* 향상도(피어슨) : 0.9=0.85 < 0.7=0.79 < -1.2=0.75 < 0.5=0.74
  유클리드 유사도 : 0.7=0.78 < 0.5=0.66

#############
 # 추천기 평가
#############

 # 학습 데이터와 점수

실제 데이터에서 일부 데이터(학습 데이터)를 제거한 후 남은 데이터(테스트 데이터)로 미래 선호를 시뮬레이션해볼 수 있다.

학습 데이터와 테스트 데이터를 명확히 분리해야 올바른 평가가 가능하기 때문에 이런 테스트 선호는 추천기에 입력된 학습 데이터에는 나타나면 안 된다.
추천기가 테스트 선호를 올바르게 도출하려면 학습 데이터가 제외된 테스트 데이터만 필요하기 때문이다.

대신에 추천기는 제거한 테스트 데이터의 선호를 추정하고 실제값과 비교한다.
이렇게 하면 추천기를 간단하게 평가해볼 수 있다.

예를 들어 추정 선호와 실제 선호 간의 평균 차이를 계산할 수 있다.
이런 점수 방식에서는 점수가 낮을수록 좋은 것인데, 추정 선호값이 실제 선호값과 차이가 적음을 의미하기 때문이다.
점수가 0.0이면 실제값과 예측값에 전혀 차이가 없는 완벽한 추정인 것이다.

종종 차이값 비교로 제곱평균제곱근이 사용된다.
이것은 실제 선호와 추정 선호값의 차이를 제곱한 것의 평균값에 제곱근을 적용한 것이다.
값은 낮을수록 좋다.

평균 차이와 제곱평균제곱근 계산의 예

구분	아이템 1	아이템 2	아이템 3
실제값	3	5.0	4.0
예측값	3.5	2.0	5.0
차이값	0.5	3.0	1.0
평균 차이값	$=(0.5 + 3.0 + 1.0) / 3 = 1.5$		
제곱평균제곱근	$=\sqrt{((0.5^2 + 3.0^2 + 1.0^2) / 3)} = 1.8484$		

아이템 2처럼 누군가 원할 수도 있지만 예측이 잘못된 아이템에 대해 더 심하게 왜곡할 수 있다.
예를 들어 별점 2개 차이가 별점 1개 차이보다 2배 이상 나쁘게 추정될 수 있다.

#############

유사도 측정법

#############

사용자 기반 추천기에서 또 하나의 중요한 부분은 UserSimilarity 구현일 것이다.

사용자 기반 추천기는 이 컴포넌트에 전적으로 영향을 받는다.

따라서 사용자 간의 신뢰할 만한 유사성 여부를 효과적으로 구할 수 없다면 사용자 기반 추천에서는 형편없는 결과만 나올 것이다.

사용자 기반 추천기의 조카뻘 되는 아이템 기반 추천기도 역시 이러한 유사도에 의존한다.

사용자	아이템	선호값
1	101	5
1	102	3
1	103	2.5
2	101	2
2	102	2.5
2	103	5
2	104	2
3	101	2.5
3	104	4
3	105	4.5
3	107	5
4	101	5
4	103	3
4	104	4.5
4	106	4
5	101	4
5	102	3
5	103	2
5	104	4
5	105	3.5
5	106	4

비교대상	비교제외

피어슨 상관관계 기반의 유사도

> 클래스 : PearsonCorrelationSimilarity(model)

* 피어슨 상관관계는 -1과 1사이의 값으로, 두 개의 연속적인 숫자열의 일대일 비교를 통해 경향성을 측정한다.

한 숫자열의 각 숫자가 다른 숫자열의 대응되는 값보다 얼마나 상대적으로 큰지 측정한다는 뜻.

즉, 두 숫자열 간의 대략적인 선형관계를 이루는지 숫자열내의 값들과 다른 숫자열값의 공통적인 방향성을 측정해서 확인해보는 것.

경향성이 크면 상관계수는 1에 가까워진다.

관계가 적어지거나 거의 없을 경우에는 값이 0에 가까워진다.

숫자열 내의 숫자가 높고 다른 숫자열 내의 값은 작아지는 서로 대립하는 상관성을 가질 경우에는 값이 -1에 가까워진다.

* 피어슨 상관관계로 하나의 아이템에 대해 두 사용자의 선호값의 이동 경향성이 상대적으로 높은지 아니면 낮은지 측정.

사용자별 선호를 가진 아이템에 대한 피어슨 상관관계

사용자	아이템 101	아이템 102	아이템 103	사용자 1과의 상관관계
사용자 1	5.0	3.0	2.5	1.000
사용자 2	2.0	2.5	5.0	-0.764
사용자 3	2.5	-	-	-
사용자 4	5.0	-	3.0	1.000
사용자 5	4.0	3.0	2.0	0.945

> 문제점

1) 두 사용자의 선호가 겹쳐지는 아이템의 숫자를 고려하지 않기 때문에 추천엔진을 사용할 때 약점이 될 수 있다.

예) 두 사용자가 같은 영화 200편을 본 경우에는 같은 평점을 주지 않았더라도 영화 2편만 공통으로 본 두 사용자보다는 유사도가 높아지는 경향이 있다.

2) 만약 두 사용자의 아이템 선호 중 단지 하나만 겹친다면 계산 방식을 어떻게 정의할지 모르기 때문에 상관관계를 계산할 수 없다.

3) 선호값의 열이 모두 일치하는 경우에도 정의하기 어렵다.

예) 사용자 5가 3개의 아이템 모두에 3.0의 선호를 표현했다면, 사용자 1이 3.0 외의 다른 선호를 가지고 있어도 사용자 피어슨 상관관계를 정의할 수 없다.

따라서 사용자 1과 사용자 5 사이의 유사도를 계산할 수 없다.

가중치 적용

위의 PearsonCorrelationSimilarity에서 언급한 이슈를 완화하도록 표준 계산 방식을 확장한 가중치 처리 기능을 제공한다.

> 두번째 인자 : PearsonCorrelationSimilarity(model, Weighting.WEIGHTED)

가중치(Weighting.WEIGHTED)를 적용하면 결과적으로는 상관관계 값을 계산할때, 얼마나 많은 데이터 포인트를 사용하느냐에 따라 상관계수가 1.0 혹은 -1.0에 근접한다.

평가 프레임워크를 빠르게 실행해보면 이 경우에는 0.77로 점수가 약간 좋아진다.

유클리드 거리 기반의 유사도 정의

> 클래스 : EuclideanDistanceSimilarity(model)

이 구현 방법은 사용자 사이의 거리에 기반한다.

이 아이디어는 선호값으로 좌표체계를 구성한 많은 다차원 공간(가지고 있는 아이템 수만큼 존재할 수 있다)에 사용자를 하나의 위치로 봄

이 유사도 측정법은 두 사용자 위치 사이의 유클리드 거리 d를 계산한다.

즉, 유클리드 거리가 큰 값을 가지면 사용자 간의 거리가 멀다는 의미이기 때문에 사용자 간 유사성이 떨어져서 값 자체로는 유의미한 유사도 측정으로 보이진 않는다.

사용자끼리 유사할 수록 값이 작아야 한다. 그래서 1/(1+d)의 값을 반환하도록 구현했다.

거리가 0일 때는(사용자의 선호가 완전히 동일할 때) 결과값이 1이고, 거리 d값이 증가함에 따라 값이 작아지는 것을 확인할 수 있다.

유사도 측정값으로 음수를 반환하지 않지만 여전히 값이 클수록 유사도가 커진다.

* 피어슨 상관관계에서 사용자 1과 사용자 3 사용자 간의 계산이 가능해졌다.

* 여기서는 사용자 1과 사용자 4가 더 높은 유사도를 가진다.

사용자 1과 다른 사용자 간의 유클리드 거리와 유사도 점수 결과

사용자	아이템 101	아이템 102	아이템 103	거리	사용자 1과의 유사도
사용자 1	5.0	3.0	2.5	0.000	1.000
사용자 2	2.0	2.5	5.0	3.937	0.203
사용자 3	2.5	-	-	2.500	0.286
사용자 4	5.0	-	3.0	0.500	0.667
사용자 5	4.0	3.0	2.0	1.118	0.472

스피어만 상관관계의 관련 순위로 유사도 정의

> 클래스 : SpearmanCorrelationSimilarity(model)

스피어만 상관관계는 피어슨 상관관계의 흥미로운 변형이라고 볼수 있다.

원래의 선호값으로 상관관계를 계산하기보다는 상대적인 선호값 순위에 기반해서 상관관계를 계산하기 때문이다.

각 사용자의 선호가 가장 없는 아이템의 선호값을 1로 고쳐 쓰는 경우를 생각해보자.

그다음으로 선호가 없는 아이템의 선호값을 2로 바꾼다.

영화 순위를 매길 때 가장 선호가 낮은 영화에 별점 1을 주고 그다음 영화에는 별점 2를 주는 식이다.

그러면 피어슨 상관관계를 이렇게 변환된 값으로 계산할 수 있게 된다. 이것이 스피어만 상관관계이다.

하지만 처리 과정 중에 몇몇 정보를 잃기도 한다.

즉 선호값의 핵심인 순위를 보존하는 대신에 사용자가 각 아이템을 얼마나 선호하는지를 나타내는 값을 제거한 것이다.

이것은 좋은 아이디어일수도, 좋지 않은 아이디어일 수도 있다.

선호값을 보존하거나 아니면 전체를 잃어버리는 두 가지 가능한 경우를 이전에 살펴보았지만, 아마 그 중간 정도이지 싶다.

이미지에서 스피어만 상관관계의 결과를 확인할 수 있다.

이미 간단한 데이터 셋을 다시 간략화하기 때문에 일부 결과값이 극단적이다.

사실 모든 상관관계가 여기에서는 1 또는 -1인데, 사용자 1의 선호값을 수용할지 아닐지에 의해 결정된다.

피어슨 상관에서는 사용자 1과 사용자 3간의 값을 계산할 수 없었다.

선호값을 순위로 바꾼 것과 사용자 1과 다른 사용자 간의 스피어만 상관관계를 계산한 결과

사용자	아이템 101	아이템 102	아이템 103	사용자 1과의 상관관계
사용자 1	3.0	2.0	1.0	1.0
사용자 2	1.0	2.0	3.0	-1.0
사용자 3	1.0	-	-	-
사용자 4	2.0	-	1.0	1.0
사용자 5	3.0	2.0	1.0	1.0

 # 캐시 랩핑

> 클래스 : CachingUserSimilarity

UserSimilarity의 결과를 캐시 처리하는 또 하나의 구현

CachingUserSimilarity에서는 계산 처리는 다른 곳에서 하고 그 결과는 내부에 기억한다.

나중에 사용자-사용자 간의 유사도값이 필요할 때 즉시 값을 변환할 수 있도록 미리 계산하고 다시 계산하지 않도록 구현한 것이다.

이런 방법으로 어떤 유사도 구현에도 캐시 기능을 추가할 수 있다.

계산 처리 비용이 상대적으로 높다면 캐시를 적용할 만한 가치가 있다.

캐시를 사용할 때 발생하는 비용은 메모리 소모를 말한다.

```
UserSimilarity similarity =
        new CachingUserSimilarity(new SpearmanCorrelationSimilarity(model), model);
```

########## 주요 학습 #########

1) 평가 모델 범위및 평가 방식 학습

2) 고정 크기 이웃 & 유사도 임계치 기반 이웃

3) -1 에서 1의 경계및 사용자 기준 이해

평가 매트릭스

Precision : 정확률
recall : 재현율

정확률과 재현율은 많은 가능성 있는 결과 중에서 최선의 결과를 선택하고 반환하는 검색엔진의 평가에서 전형적으로 사용된다.

검색엔진은 연관성이 없는 결과를 최상위 결과로 반환하면 안 되지만, 가능한 많은 연관성 있는 결과를 반영하려는 노력도 필요하다.

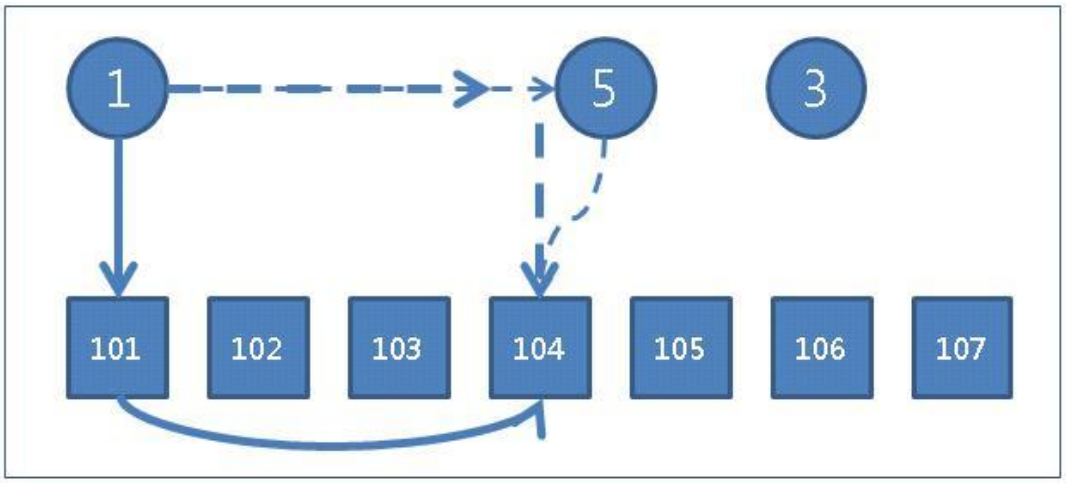
정확률은 연관성과 관련된 개념으로서 상위 결과가 어느 정도 연관성이 있는지의 비율이다.
정확률 10은 상위 결과 10개를 가지고 평가한 것이다.
재현율은 상위 검색 결과를 포함해서 모든 연관성 있는 결과와의 비율이다.

아이템 기반 추천

아이템 기반의 추천은 사용자와 사용자가 아니라 아이템과 아이템이 얼마나 유사한지로 도출한다.
이는 머하웃에서 UserSimilarity 대신에 ItemSimilarity에 기반한다는 것을 의미한다.

[그림 4.4]에서 사용자 기반 추천기와 아이템 기반 추천기의 근본적인 차이를 확인할 수 있다.
유사한 사용자와 유사한 아이템이라는 상대적으로 다른 경로를 통해 추천 아이템을 찾는다.

[그림 4.4] 사용자 기반과 아이템 기반 추천기의 차이를 설명한 기본 개념도 사용자 기반 추천(점선)은 유사한 사용자를 찾아서 그들이 좋아하는 것을 찾는다.
반면에 아이템 기반 추천(실선)은 사용자가 좋아하는 아이템을 찾고 그것과 유사한 아이템을 찾는다.



> 알고리즘

```
사용자 u가 선호를 아직 가지고 있지 않는 모든 아이템 i에 대해
        사용자 u가 선호를 가지고 있는 모든 아이템 j에 대하여
                아이템 i와 아이템 j 간의 유사도 s를 계산한다. (아이템과 아이템 간의 유사도)
                사용자 u의 아이템 j에 대한 선호를 더하고 유사도 s의 가중치를 적용해 평균을 계산하여
```

```
평균 가중치로 순위를 매겨 상위 아이템을 반환한다.
```

세 번째 라인은 예전처럼 사용자-사용자 간의 유사도가 아니라 아이템과 아이템 간의 유사도를 어떻게 반영하는지를 보여준다.

알고리즘이 서로 유사해 보이지만 전체적으로 대칭이지는 않다. 현저하게 다른 점도 있다.

예를 들어 사용자 기반 추천기의 동작 시간은 사용자가 늘어남에 따라 늘어나지만 아이템 기반의 추천기의 동작 시간은 아이템의 수에 따라 늘어난다.

아이템-아이템 유사도는 고정적이며 미리 계산해놓기가 유리하다.
미리 유사도를 계산해야 하지만 추천기의 실행 속도는 훨씬 빨라진다.

머하웃에서는 GenericItemSimilarity 클래스로 어떤 ItemSimilarity의 결과도 미리 계산해서 저장할 수 있도록 한다.

 # 기본 아이템 기반 추천기의 핵심

```
public Recommender buildRecommender(DataModel model) throws TasteException{
    ItemSimilarity similarity = // 아이템 기반 추천
        new PearsonCorrelationSimilarity(model); // 피어슨 상관관계 유사도 측정

        return new GenericItemBasedRecommender(model, similarity); // 아이템 기반 추천기
}
```

UserSimilarity 인터페이스와 전체가 동일한 ItemSimilarity 인터페이스를 구현했기 때문에, PearsonCorrelationSimilarity는 여전히 여기서 사용할 수 있다.
아이템 기반 추천기는 사용자가 아닌 아이템 간의 피어슨 상관관계를 기반해서 같은 개념의 유사도를 구현했다.

즉, **많은 아이템에 대한 한 사용자가 아닌, 한 개의 아이템에 대한 많은 사용자의 선호 목록을 비교**한다는 말이다.

5.22 아이템 기반 추천기

아이템 기반 추천기에서 아이템 유사도 측정은 한 가지만 선택할 수 있다.
각 유사도 측정을 간단하게 적용해보고, 어떤 것이 가장 적합한지 찾아본다.[표 5.3]에 이러한 유사도 측정의 결과가 요약되어 있다.
여기에선 특히 점수가 나빠졌다.
평균 에러 또는 예측과 실제 선호값의 차이가 두 배나 그 이상이기 때문이다.
이 데이터에서는 아이템 기반 접근 방식이 효과적이지 않은 몇 가지 이유가 있다. 왜 그럴까?

먼저 사용자 기반의 접근으로 사용자 간의 유사도를 계산하는 알고리즘은 사용자가 다른 사용자의 프로파일에 평점을 어떻게 주는가에 달려 있기 때문이다.
이제 사용자 프로파일 간의 유사도 계산은 사용자가 그 프로파일을 어떻게 평가하는지에 달려있다.
어쩌면 이건 별로 의미가 없을지도 모른다. 아마 평점이 평가한 프로파일보다는 평가에 대해 더 많은 의미를 담고 있을 것이다.
어떤 설명을 해도, 결과를 봐서는 아이템 기반 추천이 최고의 선택이 아니라는 점이 명확해 보인다.

[표 5.3] 여러 개의 다른 유사도 측정기법을 사용하는 아이템 기반 추천기를 평가할 때의 예측 선호와 실제 선호 간의 절대차이평균(average absolute difference)

유사도	점수
유클리드	2.36
피어슨	2.32
로그 우도	2.38
타니모토	2.40

5.3.3 IDRescorer로 추천 개선하기

Recommender.recommend()의 옵션인 IDRescorer 타입의 마지막 인자가, recommend(long userID, int howMany)를 호출하는 대신에 recommend(long userID, int howMany, IDRescorer)를 호출할 수 있다는 것을 눈치 챘는가?

이런 유형의 인터페이스의 구현은 머하웃 추천기 API의 여러 곳에서 볼 수 있다.

이런 구현을 사용하면 로직으로 추천엔진에서 사용하는 값을 다른 값으로 변환하거나 프로세스로 특정 엔터티를 고려 대상에서 제외할 수 있다.

예를 들어 추천기가 예측한 아이템 선호값을 임의로 변경하는 데 IDRescorer를 사용할 수 있다.
아니면 특정 아이템을 고려 대상에서 완전히 제거할 수도 있다.

전자상거래 사이트에서 사용자에게 책을 추천하는 경우를 가정해보자.
사용자는 지금 미스터리 소설을 살펴보고 있다.
그래서 지금으로서는 이 사용자에게 책을 추천한다면 모든 미스터리 소설의 예측 선호값에 가중치를 부여하고 싶을 수도 있다.
또한 재고가 없는 책은 추천하지 않기를 원할 수도 있다.
IDRescorer를 사용해서 이런 문제를 해결할 수 있다.

다음 리스트에서 가상 서점의 장르와 같은 클래스 로직을 캡슐화하도록 구현한 IDRescorer를 확인할 수 있다.

[리스트 5.3] 재고가 없는 책은 제외하고 장르는 가중 처리하는 IDRescorer 사용 예제

```
import org.apache.mahout.cf.taste.recommender.IDRescorer;

public class GenreRescorer implements IDRescorer {

  private final Genre currentGenre;

  public GenreRescorer(Genre currentGenre) {
    this.currentGenre = currentGenre;
  }

  @Override
  public double rescore(long itemID, double originalScore) {
      // BookManager가 있다고 가정한다.
    Book book = BookManager.lookupBook(itemID);
    if (book.getGenre().equals(currentGenre)) {
      return originalScore * 1.2; // 예측치의 20%를 가중 처리
    }
    return originalScore; // 변경 없음
  }

  @Override
  public boolean isFiltered(long itemID) {
    Book book = BookManager.lookupBook(itemID);
    return book.isOutOfStock(); // 재고가 없는 책 필터링
  }
}
```

미스터리 소설의 예측 선호값을 rescore() 메소드로 가중 처리한다.

재고가 없는 책을 추천 대상에서 제외하는 IDRescorer의 다른 사용 사례를 isFiltered() 메소드로 확인할 수 있다.

Evaluated with user 2 in 16ms => 사용자 2에 대한 평가

Precision/recall/fall-out/nDCG/reach: 1.0 / 1.0 / 0.0 / 1.0 / 1.0

사용자 1의 정확률과 재현율 평가

정보: Evaluated with user 1 in 15ms

정보: Precision/recall/fall-out/nDCG/reach: NaN / 0.0 / 0.0 / NaN / 0.0

정보: Processed 5 users

정보: Evaluated with user 2 in 0ms

정보: Precision/recall/fall-out/nDCG/reach: NaN / 0.0 / 0.0 / NaN / 0.0

정보: Processed 5 users

정보: Evaluated with user 4 in 0ms

정보: Precision/recall/fall-out/nDCG/reach: NaN / 0.0 / 0.0 / NaN / 0.0

NaN

0.0

사용자 2의 정확률과 재현율 평가

정보: Processed 5 users

정보: Evaluated with user 2 in 16ms

정보: Precision/recall/fall-out/nDCG/reach: 1.0 / 1.0 / 0.0 / 1.0 / 1.0

정보: Processed 5 users

정보: Evaluated with user 4 in 0ms

정보: Precision/recall/fall-out/nDCG/reach: 0.75 / 1.0 / 0.08333333333333333 / 1.0 / 1.0

0.75

1.0

사용자 3의 정확률과 재현율 평가

정보: Processed 5 users

정보: Processed 5 users

정보: Processed 5 users

NaN

NaN