


Collaborate Filtering for Implicit Feedback Datasets

☰ Property	
📧 Property 1	
☰ abstract	<p>공통적인 추천 시스템 Task는 prior implicit feedback에 기반해서 개인화된 추천을 통해 고객 경험을 향상시키는 것이다. 이러한 시스템들은 패시브하게 서로 다른 유저 행동들을 트래킹한다. 여기서의 서로 다른 종류의 유저 행동은 유저의 선호 모델링을 하기 위한 구매 히스토리, 시청 습관 그리고 브라우징 활동을 말한다. 훨씬 더 연구된 explicit feedback과는 다르게, 우리는 선호도에 대한 직접적인 사용자로부터의 입력이 어떤 것도 가지고 있지 않다. 특히, 우리는 고객이 선호하지 않는 상품들에 대한 충분한 증거가 부족하다. 이 연구에서 우리는 implicit feedback 데이터셋에 대한 고유한 속성을 인식을 확인한다. 우리는 데이터를 긍정적이고 부정적인 선호도를 매우 다양한 confidence level(신뢰 수준)과 연관해서 다루는 것을 제시한다. 이것이 implicit feedback에 대한 추천에 맞게 조정된 Factor 모델로 이어진다. 여기서는 추가로 확장 가능한 최적화 과정을 제안한다. 여기서 확장 가능한 최적화 과정은 데이터 사이즈에 맞게 선형적으로 모델의 사이즈를 조정하는 것이다. TV쇼를 위한 추천 시스템에서 알고리즘은 성공적으로 사용되었다. 알고리즘은 최적화가 잘된 다른 알려진 방법과도 수월하게 비교가 되고 추가로,</p>

	논문에서는 factor 모델로 주어진 추천에 대한 설명을 위한 획기적인 방법도 제공한다.
기타	Yahoo
논문 링크	http://yifanhu.net/PUB/cf.pdf
발제자	강석우
발표자	 석우 강
키워드	Implicit FeedBack, Confidence Interval, Factor Model, Collarborate Filtering
한글 링크	

1. Introduction

E-커머스에 대한 인기가 높아지면서, 중요한 챌린지는 제공되는 수 많은 제품들을 분류해서 그들이 가장 좋아할만한 것을 쉽게 찾도록 돕는 것이다. 이러한 도전을 다루는 하나의 도구 중 하나인 추천 시스템은 최근에 많은 주목을 받는 시스템이다. 이러한 시스템들은 그들의 독특한 취향과 요구에 잘 맞도록 제품들이나 서비스들에 대해 개인화된 추천을 제공한다. 이러한 시스템들의 기술은 유저, 제품들을 프로파일링하거나 어떻게 그것들을 어떻게 연결할 것인지에 대해 기반한다.

광범위 하게 말하면, 추천 시스템들은 두 다른 전략(혹은 이들의 조합)에 기반한다. Content Based Approach는 각각의 유저나 제품들의 특성에 맞춰 프로파일을 생성한다. 예를 들어 영화 프로파일은 장르, 출연한 배우들, 박스 오피스 흥행도 등의 항목을 포함한다. 유저 프로파일은 데모그래피 정보나 적합한 설문지에 대한 답변을 포함한다. 이런 프로파일들은 프로그램으로 유저와 제품을 연관할 수 있도록 한다. 하지만 Content Base 전략은 쉽게 모으기 힘들거나 쉽게 사용할 수 없는 외부 정보를 모으는 것이 필요하다.

대안 전략으로, Explicit Profile을 생성하지 않고 오직 과거 유저 행동에 의존하는데 초점을 맞출 수 있다. 이 접근 방법이 Collarborate Filtering (CF)다. CF는 새로운 유저-아이템 연결 관계를 식별하기 위해서 유저와 상품들에 대한 상호 의존성을 분석한다. 예를 들어, 어떤 CF 시스템들은 유사한 점수를 매긴 히스토리나 구매 히스토리를 사용해서 유사한 평가를 하거나 유사한 마인드를 가지는 사용자로 아이템 쌍을 식별해서 사용자들과 아이템들 사이의 잘 알려지지 않은 관계를 추론한다. 필요한 정보는 오직 유저의 과거 행동이다. 여기서 말하는 과거 행동은 그들의 과거 거래나 제품들에 점수를 매긴 방식이다. CF의 주요 강점은 도

메인에 구매 받지 않는다는 점이다. 그러나 종종 파악하기 애매한 데이터 측면을 다룰 수도 있고 Content 기반 기술을 사용해서 프로파일링 하기 어려울 수도 있다. 일반적으로 Content 기반 기술은 보다는 정확하지만, CF는 cold start 문제는 해결하기 어려울 수도 있다. Content 기반 접근 방법에서는 적합한 시스템에 새롭게 나오는 제품들을 처리 할 수 없기 때문이다.

추천 시스템은 다른 타입들의 입력에 의존한다. 가장 편한 것은 높은 퀄리티의 explicit feedback이다. explicit feedback은 사용자들에 의한 제품에 대한 관심의 명시적인 입력이다. 예를 들어, 넷플릭스는 영화에 대한 별 점을 수집한다. TiVo 유저들은 TV 쇼들에 좋아요와 싫어요 버튼으로 그들의 선호를 나타낸다. 그러나 explicit feedback은 항상 사용 가능하지는 않다. 그래서 추천 시스템은 간접적인 implicit feedback으로 부터, 그들의 선호들을 추론할 수 있다. implicit feedback은 유저 행동을 관찰해서 간접적으로 의견을 대변하는 것을 말한다. implicit feedback은 구매 히스토리, 브라우징 히스토리, 검색 패턴 심지어는 마우스의 움직임 까지도 포함한다. 예를들어 사용자가 같은 작가의 책을 많이 구매 했다면 그 작가를 좋아하는 것이다.

이 분야의 많은 문헌들은 explicit feedback을 처리하는데 초점을 맞춘다. 아마도 이러한 종류의 pure한 정보를 사용하기에 편하기 때문이다. 하지만 많은 실제 상황에서 추천 시스템은 implicit feedback에 대해 중점을 뒤야할 할 필요가 있다. 이것이 사용자가 제품을 평가하는 것을 꺼리는 것을 반영하거나 explicit feedback을 수집할 수 없는 한계를 반영할수도 있다. implicit model에서 사용자가 사용 데이터를 수집하기로 한다면, 사용자 파트에서는 더이상 추가적인 explicit feedback가 필요하지 않는다.

여기에서 implicit feedback을 처리하기에 적합한 알고리즘에 대한 탐색을 한다. TV 쇼 추천 엔진을 만들면서 달성한 중요한 교훈과 발전을 반영한다. 우리의 셋업은 사용자로부터의 explicit feedback을 적극적으로 수집하지 못하므로 그래서 시스템은 implicit feedback을 기반으로 한다. 익명화된 사용자의 시청 습관을 분석해서

explicit feedback을 염두해 두고 설계한 알고리즘을 직접적으로 사용하는 것을 방지 했기 때문에 implicit feedback의 독특한 특징을 식별하는 것은 중요하다. 다음과 같이 리스트한 주요한 특징은

1. Negative Feedback은 없다. 유저의 행동을 관찰해서, 어떤 아이템들을 그들이 아마도 좋아할지 그리고 소비할 지를 추론할 수 있다. 하지만, 어떤 아이템을 유저가 싫어할지 확실하게 추론하는 것은 어렵다. 예를 들어, 유저가 특정 쇼를 보지 않았다면, 그건 그녀가 쇼를 싫어하거나 그저 쇼에 대해서 몰랐거나 볼 상황이 안되었기 때문이다. explicit feedback에는 사용자가 그들이 어떤 것을 좋아하거나 싫어하는 것을 표현하기 때문에 이런 근원적인 비대칭이 존재하지 않는다. 몇 가지 의미가 있다. 예를 들어, explicit 추천 시스템은 정보를 모으는데 초점을 맞추는 경향이 있는데 - 사용자-아이템 쌍(그들이 점수를 어떻게 매긴지 아는) - 유저 선호도에 대해 균형된 그림을 제공한다. 그래서, 일반적으로 데이터의 대부분을 구성하는 나머지 사용자-아이템의 관계, 는 "누락 된 데이터"로 처리되고 분석에서 제외가 된다. 이것은 implicit feedback에서는 불가능하다.

수집된 피드백에만 집중하면 긍정적인 피드백을 남기고 전체 사용자 프로필을 잘못 표현 할 수도 있다. 그래서 누락 데이터도 다루는 것이 중요하다. 대부분의 부정적인 피드백이 찾아질 것으로 예상되는 곳이라서

2. Implicit feedback은 본질적으로 노이즈하다. 유저 행동을 트래킹을 패시브하게 할 때, 우리는 그들의 선호도와 진짜 모티브를 추측할 수 있다. 예를 들어, 우리는 개인에 대한 구매 행동을 볼 수 있다. 그러나 구매 행동이 제품에 대한 긍정적인 시점을 표현하는 것은 아닐 수도 있다. 아이템이 선물의 형태로 구매 될 수도 있고, 또한 아마도 제품에 대해 유저가 실망했을 수도 있다. 특정 시간에서나 특정 채널에 대해 티비를 보는 것이 아니라 시청자가 잠들었을 수도 있다.
3. explicit feedback의 수치 값은 선호도를 나타낸다. implicit feedback에서의 수치 값은 confidence를 나타낸다. explicit feedback에서 출발한 시스템은 사용자를 그들의 선호 정도를 레벨로 표현하게 한다. 별점으로 치면 1 - 5로. 그러나 implicit feedback의 수치 값은 행동의 빈도를 설명한다. 얼마나 특정 쇼를 보았는가. 얼마나 특정 아이템을 유저가 구매했는가 등으로. 더 큰 값이 더 높은 선호도를 나타내지는 않는다. 예를 들어 가장 사랑 받는 쇼는 유저가 딱 한번만 본 영화일 수도 있다. 그런데 유저가 시리즈는 어느 정도는 좋아하면 매주 본다. 그러나 feedback의 수치 값은 특정 관찰에 대한 신뢰도를 우리에게 알려주기 때문에 확실히 유용하다. 유저 선호도에 대해 영향을 미치지 않는 일회성의 이벤트는 다양한 이유로 진행 될 수 있다. 하지만 반복되는 이벤트는 사용자 의견을 대부분 반영한다.
4. implicit-feedback 추천 시스템은 적절한 성능 검증 방법이 요구된다. 전통적인 세팅에서는 사용자가 점수를 구체화하고 정해져있는 예측 성공을 측정하기 위한 mean squared error 같은 깔끔한 메트릭이 있다. 그러나 implicit 모델은 아이템의 가용성, 다른 아이템들과의 아이템의 경쟁 그리고 피드백의 반복을 고려해야 한다. 예를 들어, 만약 우리가 텔레비전 시청에 대한 데이터를 모은다면, 확실하지 않다. 한번 이상 시청한 쇼와, 두 쇼가 같은 시간에 있다면 사용자들에 동시에 시청되지 않으므로 비교할 방법, 검증할 방법이 확실하지 않다.

2. Preliminaries (들어가기에 앞서)

사용자를 아이템에서 분리하기 위해 특별한 인덱싱 문자들을 예약한다. 사용자에게는 u, v 아이템에게는 i, j . 유저들과 아이템들이 연관되는 입력 데이터는 r_{ui} 로 관찰 데이터다. explicit feedback 데이터셋에서는, 유저 u 와 아이템 i 에 의해 점수를 통해 선호도를 나타내는 값들이다. 높은 값을 가질 수록 더 강한 선호도를 의미한다. implicit feedback 데이터셋에서는 유저 행동을 관측한 값을 나타낸다. 예를 들어, r_{ui} 는 아이템 i 를 유저가 구매한 횟수를 나타내거나 사용자 u 가 웹페이지 i 에 사용한 시간을 나타낸다. TV 추천의 경우에는 r_{ui} 는 사용자 u 가 쇼 i 를 끝까지 다 본 횟수를 나타낸다. 예를 들어 $r_{ui} = 0.7$ 는 사용자 u 가 쇼의 70%를 본 것을 나타내고 사용자가 쇼를 두번 보면 $r_{ui} = 2$ 가 된다.

일반적으로 대부분의 사용자-아이템 쌍에 대해 Explicit rating을 알 수 없다. 그러므로 상용화가 가능한 알고리즘은 상대적으로 적은 수의 알려진 rating으로 동작하고 누락된 값들을 무시한다. 그러나 implicit feedback을 사용하면 모든 r_{ui} 값에 값을 자연스럽게 할당할 수 있게 된다. 만약 관측된 액션으로 r_{ui} 가 0으로 세팅된다면, 예제에서는 레코드 상으로는 시청 시간이 0으로 구매가 0이다.

3. Previous work

3.1 Neighborhood models

CF의 가장 공통적인 접근 방법은 Neighborhood 모델로 시작된다. 사실 상 모든 초기 CF 시스템에서 공유된 원래의 형태는 user-oriented이다. 좋은 분석을 위해 [9]번 논문을 보면 된다. 다음과 같은 user-oriented 메소드는 알려지지 않은 rating들을 예측하는데 같은 생각을 가지는 유저들의 rating 기록들로 시작된다. 후에는, 비슷한 item-oriented 접근 방법이 유명해졌다. 이러한 메소드들에서는 유사한 아이템에 대해 동일한 사용자가 만든 rating을 사용해서 rating이 추정했다. 더 좋은 확장성과 향상된 정확도가 item-oriented 접근 방법을 많은 케이스에서 더 잘 동작하도록 만들었다. 추가로 item-oriented 메소드들은 예측에 숨겨진 원리를 설명하는데 더 적합하다. 이유는 사용자들이 이전에 그들에 의해 선호 했던 아이템은 익숙하지만 평소에는 비슷한 마인드를 가지는 사용자들이 좋아한다고 알려진 아이템들을 모르기 때문이다.

item-oriented 접근방법들은 아이템 간의 유사도를 측정한다. s_{ij} 는 i 와 j 의 유사도이다. 피어슨 상관 계수를 기반으로 하는 경우가 많다. 우리의 목표는 r_{ui} (관측되지 않은 사용자 u 와 아이템 i 에 대한)를 예측이다. 유사도 측정을 사용해서, 아이템 i 와 가장 유사한 사용자 u 가 평가한 k 개 아이템들을 식별한다. k 이웃들의 집합을 $S^k(i; u)$ 라고 한다. 예측된 값인 r_{ui} 는 이웃하는 아이템들에 대해 rating의 가중 평균을 취한다.

$$\hat{r}_{ui} = \frac{\sum_{j \in S^k(i; u)} s_{ij} r_{uj}}{\sum_{j \in S^k(i; u)} s_{ij}} \quad (1)$$

이 스키마에 대해 몇가지 개선하면 explicit feedback과도 잘 된다. 사용자와 아이템들에 따라 다른 다양한 평균 rating에 대해 biases를 개선하는 것과 같다. 이러한 변경들이 implicit feedback 데이터 셋들에 덜 중요하지만, ratings들을 동일한 scale에 놓는 것 대신에 같은 사용자가 소비한 아이템들의 빈도를 사용한다. 어플리케이션에 따라서 서로 다른 사용자들이 발생시키는 빈도의 스케일이 굉장히 달라서, 유사도를 계산하는 방식을 확정할

수 없다. 어떻게 item-oriented 방식을 implicit feedback에서 사용할 것인가는 Deshpande와 Karypis가 했었다.

모든 item-oriented 모델들은 implicit feedback에 대해서 disadvantage를 가지는데 - 사용자 선호와 우리가 그런 선호에 대해 가지는 신뢰도를 구별하게 하는 유연성을 가지지는 않는다.

3.2 Latent factor models

latent factor 모델 들은 관측한 rating을 설명하는 latent factor를 다루어 좀 더 전체적인 목표를 가진 Collaborative Filtering으로 구성하는 대안적인 접근 방법이다. pLSA, neural networks, Latent Dirichlet Allocation을 포함한다. 우리는 초점을 유저-아이템 관찰 매트릭스에 SVD(Singular Value Decomposition)로 추론되는 모델에 맞춘다. 최근에 SVD 모델 들은 유명해졌다. 정확도와 확장성 덕분에; 전형적인 각 유저 u 를 유저-팩터 벡터 x_u 로 두고 아이템-팩터 벡터 y_i 를 연관하는 모델이다. 내적을 취함으로 예측이 된다. $x_u^T y_i = \hat{r}_{ui}$ 파라미터 추정과 관련성이 더 있다. 최근의 많은 연구에서 explicit feedback 데이터셋들에 적용되어서, 관측된 rating들 만으로 적당한 regularized 모델로 오버피팅을 하면서 직접적으로 모델링 하는 것이 추천 된다. 다음과 같이

$$\min_{x_\star, y_\star} \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2) \quad (2)$$

여기서 λ 는 모델을 regularizing을 하는데 사용된다. 파라미터들은 stochastic gradient descent에 의해 학습 된다. 큰 데이터셋인 Netflix 데이터셋에 보고된 것에 따르면, neighborhood 모델에 비교해 항상 더 좋은 결과를 보인다고 한다. 이 연구에서는 implicit feedback 데이터셋에 이 방식을 빌려온다. 모델 내부 공식과 최적화 기술 변경이 필요하기 는 하다.

4. Our model

이 섹션에서는 implicit feedback을 위한 모델을 설명한다. 우선 r_{ui} 변수가 측정하는 confidence의 개념을 공식화 해야한다. 공식화를 위해서 p_{ui} 라는 이진 변수 셋을 도입해서 사용자 u 의 아이템 i 에 대한 선호도를 나타낸다. p_{ui} 값들은 r_{ui} 를 이진화 해서 도출한다.

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

다르게 말하면 만약 사용자 u 가 아이템 i 를 소비한다면($r_{ui} > 0$), 우리는 사용자 u 가 아이템 i 를 좋아한다고 표현한다. ($p_{ui} = 1$) 그러나, 우리의 믿음은 매우 다양한 신뢰 구간과 연관되어 있다. 우선 p_{ui} 가 0 일때는 값은 낮은 신뢰도와 연관 되는데, 어떤 아이템에 대해 어떤 긍정적인 액션을 하지 않는 것은 그것을 좋아하지 않는 것 뿐만이 아닌 많은 다른 이유로부터 기인할 수 있기 때문이다. 예를 들어, 유저는 아이템의 존재를 인식하지 못했을 수도 있고, 아니라면 가격이나 사용이 제한 되었을 수도 있다. 추가로, 아이템을 소비하는 행위는 아이템을 선호하는 것과는 다른 요소가 될 수 있다. 예를 들어, 사용자는 이전에 본 쇼와 같은 채널을 그대로 두었기 때문에 그냥 TV 쇼를 볼 수도 있다. 혹은 소비자는 아이템을 구매해서 다른 사람에게 선물을 주었을 수도 있다. 그 스스로는 그 아이템을 좋아한다고 생각하지만, 따라서 사용자에게 의해서 선호된다고 나타나는 아이템들에 대해서 다른 신뢰 레벨을 가져야한다. 일반적으로는 r_{ui} 가 올라가면, 사용자가 진짜 아이템을 좋아한다고 강력하게 나타낼 수 있다. 결과적으로 변수들의 셋을 도입한다. c_{ui} , 관측된 p_{ui} 의 신뢰도를 측정하는. c_{ui} 에 대한 그럴 듯한 선택은

$$c_{ui} = 1 + \alpha r_{ui}$$

이런 방식으로, 모든 사용자-아이템 쌍에 대해 p_{ui} 에서 최소한의 신뢰도를 가지지만, 더 많은 긍정적인 선호에 대한 더 많은 증거를 관찰함에 따라 p_{ui} 에 대한 우리의 신뢰는 더욱 강력하게 증가한다. rate의 증가는 상수 α 로 컨트롤 될 수 있다. 실험적으로 $\alpha = 40$ 이 되는 것이 가장 좋았다.

우리의 목표는 각각의 사용자 u 에 대한 벡터 x_u 를 찾는 것이고 각각의 아이템 i 에 대한 사용자의 선호도를 고려한 벡터 y_i 를 찾는 것이다. 다르게 말하자면 선호도는 내적으로 가정할 수 있는데 $x_u^T y_i = \hat{p}_{ui}$ 다. 이러한 벡터들은 사용자-팩터 와 아이템-팩터 로 알려져있다. 기본적으로 벡터들은 사용자들과 아이템들을 직접적으로 비교할 수 있는 공통의 latent factor 공간으로 매핑하려고 한다. 이것은 explicit feedback 데이터에서 유명한 matrix factorization 기술과 유사하지만, 두가지 중요한 차별점을 가진다 : (1) 다양한 신뢰도 레벨을 처리해야하고 (2) 최적화는 관측된 데이터에 대응하는 u, i 쌍 뿐만 아니라 모든 가능한 u, i 쌍을 처리해야한다. 따라서 factor들은 다음과 같은 Cost 함수를 최소화해서 계산한다.

$$\min_{x_*, y_*} \sum_{u, i} c_{ui} (p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \quad (3)$$

람다는 λ 는 모델을 학습 데이터에 대해 오버피팅 하지 않게 하려고 정규화 할때 필요하다. 파라미터 λ 의 값은 데이터에 영향을 받는다. 그리고 cross validation으로 결정 된다.

Cost 함수는 $m \cdot n$ 를 포함하는 것에서 m 은 사용자의 수고 n 은 아이템의 수다. 일반적인 데이터 셋 $m \cdot n$ 는 쉽게 수천만이 된다. 이 거대한 수는 SGD(explicit feedback 데이터 셋에서는 널리 사용되는데)와 같은 직접적인 최적화 기술을 막는다. 그래서 다음과 같은 효율적인 최적화 프로세스를 제안한다.

사용자 팩터들이나 아이템 팩터들이 고정되었다면 cost 함수는 이차 방정식이 되고 global minimum이 쉽게 계산될 수 있다. alternating-least-squares 최적화 프로세스를 할 수 있게 된다. Alternating-Least-Squares는 유저-팩터와 아이템-팩터를 번갈아가며 재-계산하고, 각 스텝은 Cost 함수의 값을 낮출 수 있는게 보장되된다. Alternating least squares는 explicit feedback 데이터셋에서 사용되곤 했는데, 모르는 데이터를 누락된 값으로 보고, sparse objective 함수를 적용한다. Implicit feedback 구성은 신뢰도 레벨을 통합하고 dense cost 함수를 해결하기 위해서 다른 전략이 필요하다. 우리는 프로세스가 확장성이 높게 구현될 수 있도록 변수의 구조를 이용해서 문제를 해결했다.

첫번째 스텝은 모든 유저 팩터들을 재계산한다. 모든 아이템-팩터들을 $n \times f$ 매트릭스 Y 로 모여 있다고 가정하자. 모든 사용자에게 루프를 돌리기 전에, 우리는 $f \times f$ 매트릭스 $Y^T Y$ 를 $O(f^2 n)$ 시간에 계산한다. 각각의 유저 u 에 대해 $n \times n$ 대각 매트릭스 C^u 를 정의한다. C^u 는 $C_{ii}^u = c_{ui}$ 이고 벡터 $p(u)$ 는 사용자 u 의 모든 선호도(p_{ui} 값들)를 포함하고 있다. 차별화로 (3)의 cost 함수를 최소화하는 x_u 에 대한 분석 표현을 발견할 수 있다.

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u) \quad (4)$$

여기서의 연산적인 병목현상은 $Y^T C^u Y$ 를 계산할 때다. $O(f^2 n)$ 의 시간(각 m 사용자에게 대해)이 계산에 필요하다. 속도 업을 하기 위해 $Y^T C^u Y = Y^T Y + Y^T (C^u - I) Y$ 를 사용한다. 다음으로 $Y^T Y$ 는 u 에 독립적이고 이미 계산이 되었다. $Y^T (C^u - I) Y$ 를 위해, $C^u - I$ 는 오직 n_u 개의 0이 아닌 엘리먼트를 가지고 있다. n_u 는 $r_{ui} > 0$ 인 아이템 수이고 일반적으로 $n_u \ll n$ 하다. 비슷하게도, $C^u p(u)$ 는 0이 아닌 n_u 원소들을 포함한다. 결과적으로 x_u 의 재연산 시간은 $O(f^2 n_u + f^3 m)$ 이 걸린다. 여기서 matrix inversion $(Y^T C^u Y + \lambda I)^{-1}$ 하는데 $O(f^3)$ 이 걸린다고 가정하면, 더 효율적인 알고리

즘이 있다고 해도 일반적으로 f 는 작은 값들을 가지기 때문에 관련이 없을 것이다. 이 스텝이 모든 m 의 사용자에게 수행되게 되고, 그래서 전체 동작 시간은 $O(f^2 N + f^3 m)$ 이 된다. N 은 0이 아닌 관측값의 수이다. 입력의 크기만큼 실행 시간이 선형적으로 증가한다. 일반적인 f 의 값은 20에서 200의 사이이다.

유저-팩터들의 재연산은 병렬적인 방식으로 모든 아이템-팩터에 대한 계산이 된다. 모든 유저-팩터들을 $m \times f$ 매트릭스 X 로 재 배열한다. 우선 $f \times f$ 매트릭스 $X^T X$ 를 $O(f^2 m)$ 에 계산한다. 각각의 아이템 i 에 대해 $m \times m$ 대각 행렬 C^i 를 정의한다. C^i 는 $C_{uu}^i = c_{ui}$ 다. 벡터 $p(i)$ 는 아이템 i 에 대한 모든 선호도를 포함한다. 그러면 해결할 수 있다.

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i) \quad (5)$$

유저-팩터와 같은 기술을 사용해서 이 과정의 실행 시간은 $O(f^2 N + f^3 n)$ 이 된다. 안정화 될 때까지 유저-팩터와 아이템-팩터 쌍의 sweep를 한다. 일반적인 sweep 횟수는 10번이다. 이 전체 과정은 데이터 사이즈에 따라 선형적으로 확장한다. 유저-팩터와 아이템-팩터를 계산한 후에 우리는 사용자 u 에게 K 개의 사용가능한 아이템을 추천한다. $\hat{p}_{ui} = X_u^T y_i$ 를 사용해서 \hat{p}_{ui} 는 아이템 i 에 대해 사용자 u 의 예측된 선호도를 상징한다.

기술에 대한 기본적인 설명이 다 되었고, 더 의논해서, 결정한 것이 수정될 수 있다. 예를 들어 p_{ui} 가 0이 되지 않도록 r_{ui} 에 최소 임계값을 설정함으로써 p_{ui} 를 r_{ui} 와 다르게 도출할 수 있다. 마찬가지로, r_{ui} 를 신뢰도 레벨 c_{ui} 로 변환하는 많은 방법이 있다. 아래와 같은 구성이 잘 동작 했다.

$$c_{ui} = 1 + \alpha \log(1 + r_{ui}/\epsilon). \quad (6)$$

스키마의 변형에 상관없이, implicit feedback의 고유한 특성을 다루는 중요한 특성을 실현하는것은 중요하다.

1. raw한 관측값(r_{ui})을 독특한 해석으로 두 분리된 크기로 보낸다 : 선호도(p_{ui})와 신뢰도 레벨(c_{ui}). 이것이 실험적 연구에서 보듯이 데이터의 특성을 반영하고 예측 정확도를 개선하기에 필수적이다.
2. 알고리즘은 변수의 대수적 구조를 잘 이용해서 모든 가능한($n \cdot m$) 유저-아이템 결합을 linear 타임으로 해결한다.

5. Explaining Recommendations

좋은 추천이 설명을 동반 해야 한다는 것은 받아 들이기 쉽다. 설명은 유저에게 특정 제품이 왜 추천 되었는지 짧은 설명이다. 설명이 사용자의 시스템의 신뢰와 올바른 측면으로 추천 이 되는 능력을 향상시키는 데 도움이 된다. 추가로 시스템을 디버깅하고 예상하지 못한 행동의 소스를 트래킹하는 중요한 수단이다. neighborhood-기반(메모리-기반)으로 설명을 제공하는 것은 간단한게 추천 시스템들은 과거의 행동으로 부터 직접적으로 추론하기 때문이다. 그러나 latent factor 모델들의 경우 설명이 더 까다로워지는게 모든 과거 사용자 액션들이 사용자-팩터를 통해 추상화 되므로 과거 사용자 액션과 추천 출력의 직접적인 관계가 차단 된다. 흥미롭게도, 우리의 alternating least square 모델은 설명을 계산하는 획기적인 방법을 가능하게 한다. 키는 사용자-팩터를 수식 4인 $x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$ 로 교체 하는 것이고 그래서 사용자 u의 아이템 i에 대한 예측된 선호도는 $\hat{p}_{ui} = y_i^T x_u$ 은 $y_i^T (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$ 이 된다. 이 표현은 몇가지 새로운 개념을 도입해서 단순화 할 수 있다. $f \times f$ 메트릭스 $(Y^T C^u Y + \lambda I)^{-1}$ 를 W^u 라고 명시하면 W^u 는 사용자 u와 연관된 가중치 메트릭스라고 간주되어야한다. 따라서 사용자 u의 관점에서 아이템 i와 j의 weighted similarity는 $s_{ij}^u = y_i^T W^u y_j$ 로 표시된다. 이 새로운 개념을 사용해서 사용자 u의 아이템 i에 대한 예측된 선호도는 다음으로 새롭게 정의될 수 있다.

$$\hat{p}_{ui} = \sum_{j:r_{uj}>0} s_{ij}^u c_{uj} \quad (7)$$

이것이 latent factor 모델을 linear 모델로 감소시킨다. 아이템과 아이템 간의 유사도에 따라 가중치를 두고 과거 액션($r_{ui} > 0$)의 선형 함수의 형태로 예측하는 선형 모델로. 각 과거 행동은 별도의 용어를 받으므로 예측된 p_{ui} 를 형성하고 그래서 우리는 그것의 고유한 기여를 분리할 수 있다. 가장 높은 기여로 연관되는 액션들은 추천 이면의 주요한 설명으로 식별될 수 있다. 더하자면, 두가지로 구분되는 소스로 각 개인의 과거 행동의 기여를 더욱 분리할 수 있다. 사용자 u - c_{uj} 의 관계의 중요성 그리고 타겟 아이템 i와 s_{ij}^u 의 유사도 두가지.

이것은 item-oriented neighborhood 모델과 매우 유사하며, 계산된 예측을 설명을 가능하게 한다. 만약 우리가 이 관점을 채택한다면, 우리의 모델을 아이템 유사도들이 규칙이 있는 최적화 과정으로 학습되는 neighborhood 기반 메소드를 위한 강력한 사전 프로세서로 고려할 수 있다. 게다가 각 사용자가 어떤 아이템이 유사한지에 대해 완전히 동의하지 않는다는 사실을 반영하여, 아이템 간의 유사성은 해당 사용자에게 영향을 미치게 된다.

(7)으로 설명을 하는 것은 선형 시스템 $(Y^T C^u Y + \lambda I)^{-1} y_j$ 문제를 메트릭스 벡터 곱으로 해결하는 것을 포함하고 $O(f^2 n_u + f^3)$ 시간 내에 끝난다. $Y^T Y$ 이 미리 계산되었다는 것을 가정한다.

6. Experimental study

데이터 설명 우리의 분석은 디지털 TV 서비스로부터의 데이터에 기반한다. 30만 셋톱 박스에서 데이터를 수집할 수 있다. 모든 데이터는 수집된다. 적합한 엔드 유저의 동의와 개인 정보 정책에 대한 동의를 받아 모든 데이터를 수집된다. 분석은 aggregate 되거나 완전히 익명화된 데이터로 수행되었다. 이 연구에서 개인정보로 인식 될 수 있는 데이터는 수집되지 않았다.

우리는 이러한 유저들에 대해 모든 채널 튜닝 이벤트를 수집한다. 채널에 셋톱 박스를 조정하고 타임 스탬프를 찍는다. 거의 17,000 의 고유한 프로그램이 4주의 기간 동안 방송되었다. 학습 데이터는 r_{ui} 값들을 포함하고, 각 사용자 u 와 프로그램 i 에 대해서 사용자 u 가 프로그램 i 를 몇번이나 시청했느냐(특정 프로그램이 길이 기반 단위에 초점을 맞춘 시청된 시간(분))를 표현한다. r_{ui} 는 실제 값이고, 사용자들은 쇼의 부분들을 시청한다. 같은 프로그램에 대해 다수 시청을 aggregate 후에, 0이 아닌 r_{ui} 의 값은 32 백만 정도 였다.

게다가, 우리는 유사하게 구성된 4 주의 학습 기간 이후의 일주일 동안의 모든 채널 튜닝 이벤트에 기반한 테스트 셋을 사용했다. 우리 시스템은 다음 주에 어떤 시청자가 볼 것인지 예측을 생성하기 위해서 최근 4주간의 데이터를 사용해 학습 되었다. 4주의 학습 기간은 더 짧은 기간이 예측 결과를 악화 시키는 경향이 있지만, 긴 기간이 큰 가치를 부여하지는 않는다는 실험적인 연구로 선택 되었다. (TV 스케줄이 시즌에 따라 변하기 때문에, 긴 학습 기간이 장점을 가지지는 않고, 비록 우리가 우리의 핵심 모델이 계절성에 방해되는 것을 피할 수 있을 만큼 충분히 강력하다는 것을 발견 했음에도 불구하고). 테스트 셋의 관측치는 r_{ui}^t 로 표시된다.

tv 시청의 한 특징은 매주 같은 프로그램을 반복적으로 보는 경향이 있다. 최근에 시청하지 않았거나 잘 모르는 프로그램을 추천 받는 것이 사용자에게는 훨씬 더 가치 있는 일이다. 그래서 기본적인 세팅은 각 유저들에 대해, 학습 기간 동안 사용자가 시청한 쇼에 해당하는 테스트 세트에서 "실제" 예측을 제거한다. 테스트 셋을 더 정확하게 만들기 위해, $r_{ui}^t < 0.5$ 이 하인 모든 엔트리들을 제거하고, 프로그램의 절반 미만을 시청하는 것은 사용자가 프로그램을 좋아한다는 강한 징후가 아니기 때문이다. 이걸로 2백만 r_{ui}^t 가 테스트 셋에 남아있다.

반복적으로 같은 프로그램들을 보는 경향성은 넓은 범위에 걸쳐서 r_{ui} 를 크게 변화 시킨다. 0에 가까운 시청 이벤트가 많기 때문에(채널 변경), 1, 2 또는 3(영화 또는 시리즈의 에피소드들), 누적되면 수백이 되는 시청 이벤트도 있다(DVR이 4주 기간의 동안 많은 시간 동안 동일한 프로그램을 녹화할 수 있다). 그래서 로그 스케일링 방식을 사용한다.

또 하나의 중요한 조정이 필요하다. 우리는 단일 채널이 많은 시간동안 시청이 되는 많은 경우를 관측했다. 채널이 튜닝 되었을 때 처음 방영된 쇼는 시청자의 관심사이지만, 이후 바뀔 영되는 쇼들은 관심이 줄어들 가능성이 높다. 시청자가 다른 것을 할 동안 텔레비전은 그냥 켜져 있을 수도 있다. 우리는 이것을 모멘텀 효과라고 부르고, 모멘텀으로 인해 시청되는 프로그램들은 실제 선호도를 반영할 것으로 기대되지 않는다. 이런 효과를 극복하기 위해,

채널 튜닝 이벤트 이후의 두번째와 이후 시퀀스는 가중치를 줄인다. 더 구체화 하면, 채널 튜닝 후의 t 번째 쇼는 $\frac{e^{-(at-b)}}{1+e^{-(at-b)}}$ 가중치를 할당한다. 실험적으로 우리는 잘 작동하고 직관적인 $a=2$ 와 $b=6$ 를 찾았다. 채널을 튜닝하고 세번째 쇼는 r_{ui} 값이 반토막이 나고 채널 변경 없이 다섯 번째 쇼는 r_{ui} 가 99% 감소한다.

성능 검증 방법론 가장 선호될 것으로 예측되는 것에서 가장 덜 선호될 것으로 예측되는 쇼로 각 사용자들에 정렬된 시나리오를 성능 검증한다. 그런 다음, 추천된 쇼들로 사용자에게 리스트의 prefix를 제기한다. 우리가 프로그램을 시청하지 않는 것은 여러가지 다른 이유로부터 기인할 수 있기 때문에 어떤 프로그램들이 사랑받지 않는 것에 관해 신뢰 할 수 있는 피드백이 없어서 라는것을 알게되는 것은 중요하다. 추가로 우리는 현재 우리의 추천에서 사용자들의 리액션을 추적할 수 없다. 따라서 사용자가 원하지 않는 프로그램을 알아야 하기 때문에 정확도 기반 메트릭은 매우 적합하지 않다 그러나 시청 프로그램은 그것을 좋아한다는 표시로, recall-oriented 측정을 적용할 수 있게 한다.

우리는 $rank_{ui}$ 로 명시했다. 사용자 u 를 위해 준비된 모든 프로그램의 정렬된 리스트 내의 percentile-ranking이다. 이런 방식으로 $rank_{ui}=0$ 은 프로그램 i 는 사용자 u 에 가장 바람직한 것으로 예상된다. 그래서 리스트의 모든 다른 프로그램 보다 상위에 위치한다. 반면에 $rank_{ui}=100$ 은 프로그램 i 는 사용자 u 에게 가장 흥미가 적어서, 리스트의 가장 끝에 위치하게 된다. (우리는 프로그램의 수에 독립적이고 일반적인 결과를 만들기 위해 절대 순위 보단 percentile-rank를 사용해서 최적화를 했다.). 기본적인 성능 측정은 테스트 기간 내의 예상되는 percentila rank다.

$$\overline{rank} = \frac{\sum_{u,i} r_{ui}^t rank_{ui}}{\sum_{u,i} r_{ui}^t} \quad (8)$$

값이 낮을 수록 더 바람직한 것이다. 시청한 쇼가 추천 리스트의 상위에 가깝게 랭킹을 표시하기 때문이다. 랜덤 예측의 경우에 예상되는 $rank_{ui}$ 는 50%다. 값이 50% 보다 높아지면 랜덤보다 안좋은 것이다.

성능 검증 결과 10에서 200 범위를 가지는 서로 다른 수의 팩터들로 모델을 구현했다. 추가로 우리는 두 다른 경쟁할 모델을 구현했다. 첫번째 모델은 모든 쇼를 그들의 선호도로 정렬했고 그래서 추천된 쇼들의 순위는 가장 유명한 것들이다. 이 방법이 놀랄정도로 강력 한데, 군중이 수천 개의 가능한 쇼 중 일부에만 집중하기 때문이다. 이것을 베이스라인 값으로 한다.

두번째 모델은 section 3.1의 neighborhood 기반(아이템 - 아이템)이다. 이 스키마에 많은 변형을 했고 최고의 결과를 얻기 위해 두 가지 결정을 찾아냈다. (1) 가장 유사한 항목의 작은 부분 집합 뿐만이 아니라 모든 아이템들을 "neighbors"로 한다. (2) item-item의 유사도

를 측정하기 위해 코사인 유사도를 사용했다. 형식적으로 아이템 i 에 대해서 모든 사용자 m 과 연관된 r_{ui} 값을 r_i 내에서 재배열 하도록한다. 그래서 $s_{ij} = \frac{r_i^T r_j}{||r_i|| ||r_j||}$. 쇼 i 에 대한 사용자 u 의 예측된 선호도는 $\sum_j s_{ij} r_{uj}$ 다. 부연으로 explicit feedback 데이터에 적용을 할 때 neighborhood 기반 기술에 대해 매우 다른 세팅을 추천한다.

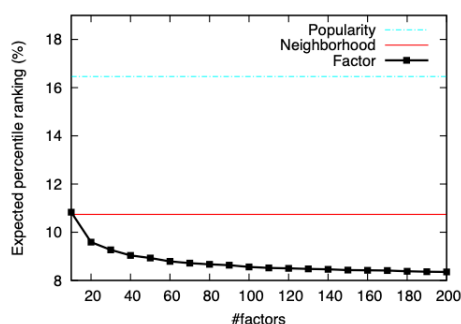


Figure 1. Comparing factor model with popularity ranking and neighborhood model.

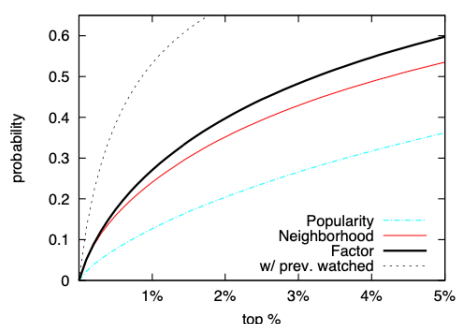


Figure 2. Cumulative distribution function of the probability that a show watched in the test set falls within top x% of recommended shows.

Figure 1은 측정된 rank 값을 다른 팩터 수들과 함께 보여주는데 그리고 popularity 랭킹과 neighborhood 모델을 보여준다. popularity에 오직 기반해서 본다면, rank는 16.46%를 달성했다. 랜덤한 예측기의 성능인 50%에 비해 훨씬 낮다. 그러나 popularity 기반 예측기는 개인화가 아니고 모든 유저들에게 동일하게 적용된다. neighborhood 기반 방법은 개인화된 추천에서 10.74%의 현저한 개선을 보인다. 훨씬 더 좋은 결과는 factor 모델에 의해 얻어진 것이며, 문제에 대해 보다 principled 접근방법을 제공한다. factor들이 증가 할수록 결과가 개선되고 200 팩터들에는 8.35%의 에 도달한다. 그래서 우리는 계산 한계 내에서 실현 가능한 최대 수의 요인을 사용할 것을 권장한다.

$rank_{ui}$ 의 누적 분포 함수를 연구해서 추천 성능을 좀더 판다. 여기서, 100 팩터들로 모델에 집중해서 Figure 2와 같이 popularity 기반과 neighborhood 기반 결과와 비교한다. 우리는 다음과 같이 묻는다면 : 테스트 세트에서 실제로 시청 되었던 쇼의 percentile의 분포는 무엇인가? 우리의 모델이 잘되면 모든 시청된 쇼들은 낮은 percentile을 가진다. Figure에서는 우리는 시청한 쇼가 우리의 모델의 약 27% 추천의 상위 1%에 있다는 것을 안다. 이러한 결과는 neighborhood 기반 접근 방법에 잘 비교되고 그리고 베이스라인인 popularity 기반 모델보다 훨씬 좋다.

이전에 본 모든 프로그램을 테스트 세트에 남겨두면 결과가 훨씬 좋아진다 (학습 기간에 이미 발생한 모든 사용자-프로그램 이벤트를 제거하지 않는다는). 프로그램을 재시청을 예측하는 것은 프로그램을 처음으로 시청하는 것을 예측하는 것보다 쉽다. 검정색 점으로 구성된 선으로 figure에서 볼 수 있다. 이전에 본 프로그램이 테스트 세트에서 제외되지 않을 때의 알고리즘을 평가한다. 비록 이전에 본 쇼를 제안하는 것은 별로 흥미롭지 않을 수 도 있지

만, 유용하다. 예를 들어 우리의 시스템이 사용자에게 흥미를 가질만한 오늘 진행되는 프로그램을 소개한다. 사용자들은 놀랄만한 것을 찾는 것이 아니라, 좋아하는 쇼를 놓치지 않게 리마인드 하고있다. 이전에 본 프로그램을 검색하는 높은 예측 정확도는 이 작업에 유용하다.

우리는 결정을 raw한 관측값 r_{ui} 를 선호-신뢰(p_{ui}, c_{ui}) 쌍으로 변환해서 성능 검증하기를 원한다. 다른 가능한 모델은 꽤 잘 연구되었다. 우선 모델을 주어진 관측에 바로 적용되는 걸 고려했다. 그래서 우리의 모델은 factor 모델로 교체되어 최소화하려고 한다.

$$\min_{x_*, y_*} \sum_{u, i} (r_{ui} - x_u^T y_i)^2 + \lambda_1 \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \quad (9)$$

이것이 regularized 버전의 dense SVD 다. Collaborative Filtering를 위해 구성된 접근 방법이다. 정규화를 제외한 결과($\lambda_1 = 0$)는 매우 안 좋고 인기도 기반 모델에 비해 개선되지 못했다. 더 좋은 결과는 모델을 정규화 할 때 획득 할 수 있다. 정규화를 하기 위해서 최고의 추천 딜리버리로 증명된 $\lambda_1 = 500$ 을 사용한다. 결과는 인기 모델을 꾸준히 능가하는 반면, neighborhood 모델에 비해 훨씬 안 좋았다. 예를 들어 50 팩터들일 때는 rank=13.63% 100일때 13.40%. 이 상대적으로 낮은 퀄리티는 이미 주장했었던 r_{ui} 를 raw한 선호도로 받아들이는 것은 합리적이지 않다고 했기 때문에 놀랍지는 않다.

그러므로 우리는 다른 모델도 트라이했다. 이전 선호도로 구성한. 결과적으로

$$\min_{x_*, y_*} \sum_{u, i} (p_{ui} - x_u^T y_i)^2 + \lambda_2 \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \quad (10)$$

모델은 $\lambda_2 = 150$ 로 정규화 된다. 결과들은 수식(9)를 사용한 모델보다 좋고, 50 팩터 일때 rank=10.72% 그리고 100 팩터 일때 rank=10.49% 이다. 이것은 neighborhood 모델로 얻어진 결과보다 훨씬 좋다. 그러나 여전히 완전한 모델 보다는 실질적으로 열등하다. rank가 50-100 팩터 일 때 8.93%-8.56 결과이기 때문이다. 이것이 (3)과 같은 신뢰도 레벨으로 (10)을 증가시키는 것이 중요하다는 것을 보여준다.

우리는 이제 분석한다 전체 모델의 성능을 (100 팩터로) 다른 타입의 쇼와 사용자들에 대해서. 다른 쇼는 다양한 시청 시간을 학습데이터에 가진 것을 보인다. 몇 쇼들은 유명하고 시청이 많이 된다. 반면에 다른 쇼들은 거의 시청되지 않는다. 쇼의 증가하는 인기도에 기반해서

긍정적인 관측을 15개로 동일하게 나누고, 동일하게 나눈 것들 각각의 에 모델의 성능을 측정한다 (1이 가장 낮은 인기 15가 가장 인기 좋은 쇼).

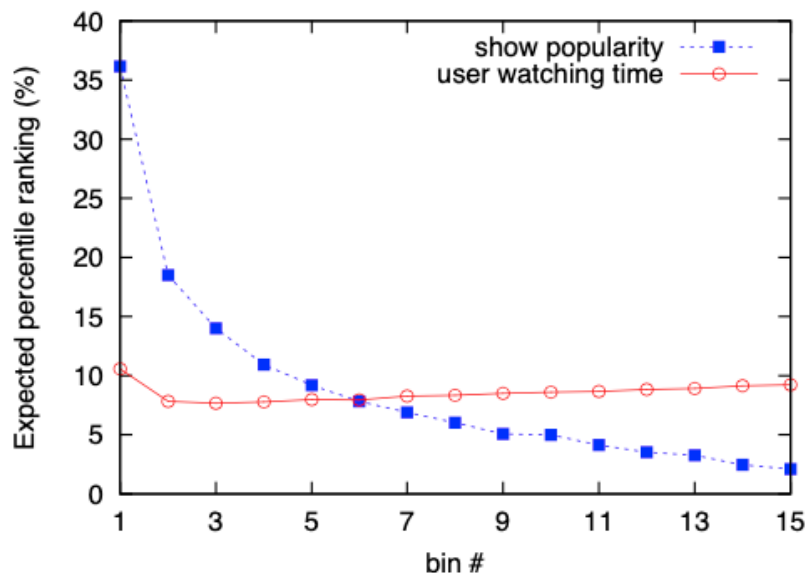


Figure 3. Analyzing the performance of the factor model by segregating users/shows based on different criteria.

Fig. 3(파랑)은 보이는데 인기 프로그램을 예측하기가 훨씬 쉬워지는 반면, 인기 없는 프로그램을 예측하는 것은 점점 더 어려워지는 모델의 정확도에 큰 차이를 보인다. 어느 정도까지는 모델은 충분한 데이터를 수집하고 잘 분석할 수 있는 익숙한 쇼를 안전하게 추천하는 것을 선호한다. 흥미롭게도, 이 효과는 사용자의 시청 시간에 따라 분할하는 사용자에게 해당되지 않는다. 시청 시간에 따라서 사용자를 나누면 Fig 3(빨강), 시청 히스토리가 없는 첫 번째를 제외하고 다른 그룹들은 꽤 비슷하다. 이건 다소 예상되지 않지만 explicit feedback 데이터셋에 대한 우리의 경험에 따르면, 우리가 사용자에게 대해 더 많은 정보를 수집하면, 예측 성능은 상당히 증가한다. 헤비한 시청자들에 대해 더 좋은 결과를 가지지 않는 것을 설명하면 한 티비로 여러 사람이 시청하는 것이다.

마지막으로 우리는 추천 설명의 효용성을 증명한다. 추천의 설명은 neighbor 메소드에 공통적이다 시스템이 항상 추천된 아이템의 가장 가까운 neighbors를 반환하기 때문에 그러나 matrix decomposition methods에 대해 어떻게 설명할 것인가는 논의된 적이 없다.

So You Think You Can Dance	Spider-Man	Life In The E.R.
Hell's Kitchen	Batman: The Series	Adoption Stories
Access Hollywood	Superman: The Series	Deliver Me
Judge Judy	Pinky and The Brain	Baby Diaries
Moment of Truth	Power Rangers	I Lost It!
Don't Forget the Lyrics	The Legend of Tarzan	Bringing Home Baby
Total Rec = 36%	Total Rec = 40%	Total Rec = 35%

Table 1. Three recommendations with explanations for a single user in our study. Each recommended show is recommended due to a unique set of already-watched shows by this user.

Table1은 한 사용자에게 세 추천된 쇼를 보인다. Section 5의 메소드에 따라, 프로그램을 설명하는 상위 다섯 시청된 프로그램을 보인다. 이 설명은 합리적이다. the reality show So You Think You Can Dance는 다른 리얼리티 쇼로 설명되고 Spider-Man은 다른 코믹쇼와 Life in E.R은 의학 문서들로 설명될 수 있다. 이러한 상식적인 설명은 특정 프로그램이 추천되는 이유를 사용자가 이해하는데 도움이 되며, 이웃의 방법에 의해 반환되는 설명과 유사하다. 우리는 또한 상위 5위 추천에 따른 추천의 총 퍼센트를 리포트한다. 이 경우에는 상위 다섯 쇼는 35-40%의 추천 만을 설명하고, 다른 많은 시청된 쇼들이 추천에 대한 입력을 제공한다는 것을 나타낸다.

7. Discussion

이 연구에서 매우 흔한 상황인 implicit feedback을 가지는 데이터셋에 대한 CF를 연구했다. 주요 연구 결과중 하나는 implicit 사용자 관찰을 두 개의 쌍으로 이루어진 선호도와 신뢰도 레벨로 변환해야 한다는 것이다. 즉, 각 사용자-아이템 쌍에 대해 우리는 입력 데이터로부터 사용자가 아이템을 좋아하는지 싫어하는지를 추론하고 이 추천치를 신뢰도 레벨과 결합한다. 이 선호도-신뢰도 파티션은 널리 연구된 explicit-feedback 데이터셋에서는 유사하지는 않지만, implicit feedback을 분석하는데 핵심적인 역할을 제공한다.

latent factor 알고리즘으로 선호도-신뢰도 패러다임을 직접적으로 해결한다. explicit 데이터셋과는 달리, 여기서의 모델은 모든 유저-아이템 선호도를 입력으로 여기에는 입력 관찰과 관련 없는 항목을 포함한다. 주어진 관찰은 본질적으로 긍정적 선호도에 치우쳐있어서 사용자 프로파일을 잘 반영하지 못하기 때문에 중요하다. 그러나 모든 사용자-아이템 값을 모델에 입력하는 것으로 심각한 확장성 문제를 야기한다. 일반적인 사용자가 사용 가능한 항목의 극히 일부에 대해서만 피드백을 제공하기 때문에 이러한 쌍의 수는 입력 크기를 상당히 초과하는 경향이 있다. 모델의 대수적 구조를 이용하여 이것을 다루며, 어떤 서브 샘플링에 의존하지 않고 전체 범위의 사용자-아이템 쌍을 다루면서 입력 크기에 따라 선형적으로 확장되는 알고리즘을 유도한다.

알고리즘의 흥미로운 특징은 최종 사용자에게 추천된 것을 설명할 수 있다는 것인데, latent factor 모델에서는 드물다. 이미 잘 알려진 item-oriented neighborhood 접근법에 놀랍고도 통찰력 있는 연결고리를 보여줌으로 설명이 가능해진다.

알고리즘은 대규모 티비 추천 시스템의 일부로 구현되어 테스트 되었다. 설계 방법론은 implicit feedback 데이터셋의 고유한 특성과 계산적 확장성 사이에서 절절한 균형을 찾으려고 한다. 우리는 현재 계산 복잡성을 증가시키는 비용을 감수하면서 정확성을 향상할 수 있는 잠재력을 가지고 수정을 시도하고 있다. 예를 들어, 모델에서 선호도 0과 연관된 모든 사용자-아이템 쌍을 동일한 신뢰도 레벨로 처리하기로 결정했다. 대 부분의 쌍들은 선호도 0과 연관되기 때문에, 이 결정은 많은 계산적인 노력을 절약했다. 그러나 좀더 면밀하게 분석하면 그런 0 값을 아이템의 가용성에 기반해서 서로 다른 신뢰도 레벨로 나눌 수 있을 것이다. TV 추천 시스템으로 예를 들면, 사용자가 프로그램을 시청하지 않았다는 것은 사용자가 프로그램을 인지하지 못했거나 ('비정상적인' 채널이나 시간에 있어서), 또는 동시에 또 다른 좋아하는 프로그램이 있거나, 또는 사용자가 단순히 흥미가 없을 수가 있다는 것을 의미할 수 있다. 이들 각각은 서로 다른 시나리오에 대응하며, 각각은 '선호도가 없다'라는 가정에서 구별이 되는 신뢰도 레벨을 보장할 수 있다. 이것이 우리를 또 다른 가능한 확장을 하게 만든다. 특정 시간에 TV를 시청하는 사용자의 경향을 설명하는 동적 시간 변수를 추가한다. 마찬가지로 특정 프로그램 장르가 하루 중 다른 시간대에 더 인기가 있다는 모델로 삼는다. 이것은 현재 진행 중인 연구의 일부로, 모델의 우수한 연산 능력을 유지하면서 모델에 추가 유연성을 도입하는 방법이 주요 과제이다.

마지막으로 우리는 표준 학습과 테스트 구성이 모델이 미래의 사용자 행동을 얼마나 잘 예측할 수 있는지를 평가 하기 위해 설계되었다는 점에 주목한다. 그러나, 추천 시스템의 목적은 다른 방법을 통해 구매하거나 아직 소비하지 않았을 수도 있는 아이템들을 추천하는 것이 아니다. 사용자 연구와 설문조사를 심층적으로 사용하지 않고는 목표를 어떻게 평가해야 하는지 알기 어렵다. 본 예제에서는 재방영되는 쇼의 "쉬운" 사례를 제거함으로써, 우리의 방법을 평가함으로써, 어떻게든 새로운 쇼의 사용자 발견을 포착하려는 방향에 가까워지려고 한다.