콘텐츠 기반 도서 추천 시스템 가속화 및 추천 경향 개선

이재정(*)
(*) 아주대학교 소프트웨어학과, jaejungscene@ajou.ac.kr

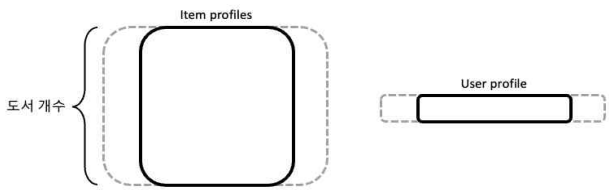
1. Abstract

일반적인 콘텐츠 기반 도서 추천 시스템은 각 도서에 대한 장르 바이너리 벡터와 사용자가 선호하는 장르에 대한 바이너리 벡터 간에 cosine similarity를 계산하여 높은 similarity값 순서대로 추천해 준다. 하지만 이와 같은 방법은 도서의 다양한 장르를 sparse한 장르 바이너리 벡터로 변환하여 similarity 계산에 있어서 효율적이지 못하다. 또한 사용자가 자신이 읽었던 도서를 기반으로 도서 추천을 원하는 등, 선호에 대한 정보가 다양해질 때에도 바이너리 벡터를 여러 개 생성하면서 메모리와 계산 복잡도 측면에서 효율적이지 못 하며, 선호에 대한 다양한 정보를 추천에 반영하지 못 하는 단점이 존재한다.

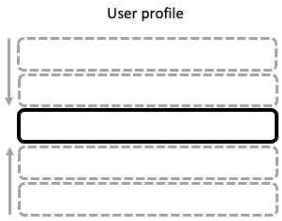
따라서 본 프로젝트에서는 LSH(Locally Sensitive Hashing)라는 방법론을 통해 도서 장르 바이너리 벡터를 압축하여 장르 바이너리 간의 similarity 계산을 가속화 할 수 있는지 검증해보고, 사용자가 자신이 읽었던 도서를 기반으로 도서 추천을 원할 때 사용자의 다양한 선호 정보를 최대한 user profile에 반영하면서 압축될 수 있는 방법과 새로운 similarity 계산을 제시한다. 결과는 LSH 방법론을 사용한 similarity 계산 가속화의 경우 가속화에는 유의미한 결과를 보이지 못 하지만, 다양한 선호 정보를 반영한 user profile 압축 및 새로운 similarity 계산은 가속화와 추천 경향 개선 모두에서 유의미한 결과를 보이며, 이는 도서 추천뿐만 아니라 또 다른 콘텐츠 기반 추천 시스템에 활용될 수 있을 것으로 기대된다.

2. Introduction

모든 사용자가 도서를 읽고 그 도서에 대해 rating을 하며 감상평을 남기는 행위에 적극적이지 않다. 즉, 특정 추천 시스템 환경 내에서 사용자가 적을 시 콘텐츠들에 대한 평가 정보는 더 적은 경우가 많으며, collaborative filtering에서 사용되는 utility matrix를 생성하기엔 한계가 있다. 따라서 이러한 환경에서는 처음에 사용자에게 선호하는 content를 선택하도록 강제하여 cold start 문제를 어느 정도 해결하는 콘텐츠 기반 추천 시스템이 필수적이다. 그리고 사용자의 다양한 선호를 받은 후에 similarity 계산이 이루어져야 하는 콘텐츠 기반 추천 시스템에서 사용자의 사용 경험을 개선하기 위해선 response타임을 줄여야 한다. response 타임을 줄일 방법은 크게 두 가지 방법이 있다. 하나는 Figure 1의 (1)과 같이 user profile 벡터 하나와 item profile 벡터들의 차원을 축소하여 계산 시간을 단축하는 것이고, 다른 하나는 Figure 1의 (2)와 같이 user profile로 여러 개의 vector가 생성되었을 때 이 vector들을 최소한의 vector로 압축하여 계산 시간을 단축하는 것이다. 본 프로젝트는 이 두 가지 방법을 통해 콘텐츠 기반 도서 추천 시스템에서의 계산 시간을 단축하려고 한다.



(1)

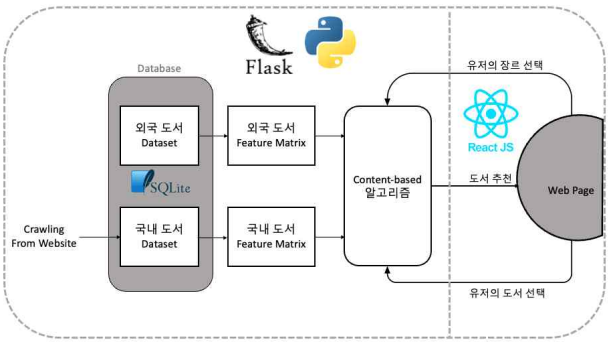


(2)

<Figure 1>

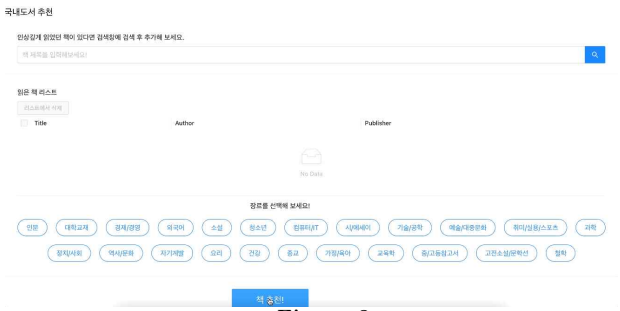
3. Method

3.1 도서 추천 시스템 구조 및 환경



<Figure 2>

본 도서 추천 프로젝트의 전체적인 구조는 Figure 2와 같이 백엔드에는 python 기반 flask 프레임워크, 데이터베이스에는 SQLite가 사용되었고,, 프론트엔드에서는 javascript 기반 react 프레임워크가 사용되었다. 국내 도서와 해외 도서는 각각 4만 개 도서 데이터를 가지고 있고 각각의 도서 데이터는 { id, title, genres, author, rating, publisher, p_date, pages, description, imgUrl } 로 구성되어 있다. 즉각적인 분석을 위한 도서 feature matrix는 Figure 5의 형태로 추출되어있다. 웹 페이지(프론트엔드)에서 사용자 선택으로 전달 받는 도서 선호 정보는 백엔드에서 python으로 분석되고, 분석된 결과는 다시 웹 페이지에 전달되어 추천 도서들이 웹 페이지에 보이게 된다. 사용자는 국내 도서 추천과 해외 도서 추천 중 하나를 선택할 수 있으며, Figure 3과 같은 인터페이스를 갖는다. Figure 3과 같은 웹 페이지에서 사용자는 적어도 하나 이상의 장르를 선택하여야 도서 추천을 받을 수 있으며, 자신이 읽은 책 중 인상 깊은 책을 검색 창을 통해 검색 및 추가하여 자신의 도서 선호 정보를 더할 수 있다.



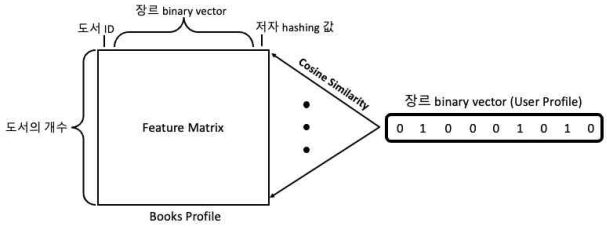
<Figure 3>

3.2 사용자로부터 장르 정보만을 받았을 시 도서 추천 알고리즘

이러한 전체적인 시스템에서 분석해야할 사용자의 도서 선호 정보는 두 가지의 형태로 존재할 수 있다. 첫 번째는 사용자로부터 받은 정보가 장르만으로 구성되어 있는 경우이고, 두 번째는 장르와 하나 이상의 인상 깊은 책으로 구성되어 있는 경우이다.

첫 번째의 경우 Figure 5와 같이 사용자로부터 받은 장르 정보를 binary vector로 변환하여 해외 도서면 해외 도서 feature matrix 국내 도서면 국내 도서 feature matrix의 각 도서들과 binary vector 간에 cosine similarity를 구해 큰 값 순서대로 도서를 추천해 준다. 본 프로젝트에서는 top 100개의 도서를 추천하도록 하였다. 그리고 Figure 1의 (1)과 같이 user profile 벡터 하나와 item profile 벡터들의 차원을 축소하여 계산 시간을 단축하기 위해 최대한 작은 k와 L값을 갖는 LSH setting을 찾았다. 찾는 과정에서 k와 L값을 정할 측정 기준은 같은 query에 대한 LSH search의 top 100 도서와 linear search의 top 100 도서 간에 cosine similarity가 0.8 이하가 되면 더 이상 k와 L을 줄이지 않도록 하였다. 또한 LSH로 추천 도서를 search할 시 가속화에 유의미한 영향을 미치는지를 측정하기 위해 같은 query에 대해 linear search 시와 LSH search 시 top 100 추천 도서를 결정하

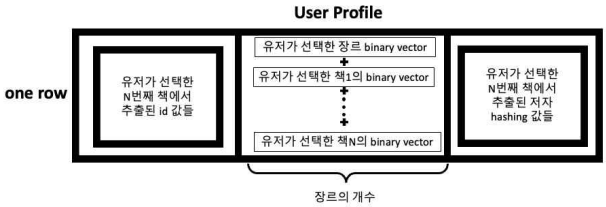
는 데까지 걸리는 시간을 측정하여 비교하였다.



<Figure 5>

3.3 사용자로부터 장르 정보와 도서 목록을 받았을 시 도서 추천 알고리즘

두 번째로 사용자의 도서 선호 정보가 장르와 하나 이상의 인상 깊은 책으로 구성되어 있는 경우 일반적인 방법론으로 user profile을 구성할 시 사용자가 인상 깊은 도서 목록을 증가 시키는 만큼 장르 binary vector가 생성되므로 search 속도는 이에 따라 linear하게 증가하게 된다. 따라서 이를 해결하기 위해 Figure 1의 (2)와 같이 user profile로 여러 개의 vector가 생성되었을 때 이 vector들을 최소한의 vector로 압축하여 계산 시간을 단축하였다. Figure 6과 같이 우선 사용자로부터 받은 장르 정보에 대한 binary vector와 인상 깊은 책들에 대한 장르 binary vector를 모두 element−wise로 더해 압축된 1개의 장르 vector를 생성하였고, search 시 중복 도서를 search 대상에서 제외하기 위해 vector의 맨 앞에는 일반적인 숫자 값이 아닌 사용자로부터 받은 도서 목록 id들의 list를 넣었으며, 사용자로부터 받은 도서 목록의 저자들 중 같은 저자의 도서가 있다면 가중치를 부여하기 위해 압축된 vector의 맨 끝에는 사용자로부터 받은 도서 목록의 저자들에 대한 list를 넣었다.



<Figure 6>

이렇게 사용자로부터 받은 도서 목록과 장르들을 기반으로 Figure 6과 같이 압축된 user profile vector를 가지고 feature matrix의 각각의 item profile들과 similarity를 계산한다. 여기서 장르 vector간의 similarity는 Figure 7과 같이 새롭게 정의하였다. 이와 같이 cosine similarity를 변형하여 similarity를 새롭게 정의한 이유는 겹친 장르들에 대해 더 높은 similarity를 부여하기 위해서 이다. user profile의 장르 벡터를 압축하는 과정에서 element−wise로 더하였기 때문에 압축된 user profile의 장르 벡터는 0을 포함한 양의 정수로 이루어져 있고, norm값을 구할 시에는 이 0을 포함한 양의 정수 벡터를 binary 벡터로 변환하기 때문에 inner product로 구해진 높은 similarity 값은 유지될 수 있는 것이다.

$$\frac{x \cdot y}{\| x \| \cdot \| (y > 0) \|}$$ (y는 0을 포함한 양의 정수)

<Figure 7>

사용자의 도서 선호 정보가 장르와 하나 이상의 인상 깊은 책으로 구성되어 있는 경우 user profile 압축과 새롭게 정의한 similarity가 가속화에 유의미한 결과를 가져다주는지 측정하기 위해 같은 query에 대해 linear search 시 평균 타임 시간과 새롭게 정의한 user profile과 similarity로 search 시 평균 타임 시간을 비교하였다. 또한 추천 경향에도 유의미한 결과를 미치는지 측정하기 위해 top 10 similarity 도서에 사용자로부터 받은 도서 목록에서의 저자의 책이 존재하는지를 그 비율을 측정하였다.

4. Results

결과는 github link에서 확인 가능

Korean book

Algorithm	Average Time	Similarity
linear search (cosine similarity)	0.957(s)	1.000
LSH search	1.012(s)	

Foreign book

Algorithm	Average Time	Similarity
linear search (cosine similarity)	0.635(s)	1.000
LSH search	0.680(s)	

Table 1: 사용자로부터 장르만 받았을 경우, Average Time은 linear search 및 LSH search 평균 시간, Similarity는 Top 100의 결과 벡터 간에 평균 cosine similarity

Table 1의 결과는 LSH search의 정확도가 linear search와 동일하지만 속도를 가속화하지 못하는 것을 보여준다. 즉, LSH search는 가속화 방법으로 적합하지 않다는 결론을 얻을 수 있다. 이에 대한 이유는 LSH search 시 hash된 벡터를 추출하는 과정과 hash된 벡터들을 통해 비교 후 원래의 벡터 간에 cosine similarity를 구하는 과정이 들어가면서 시간이 늘어나는 것으로 보인다. 따라서 장르 벡터 하나에 대한 도서 추천 검색은 간단한 numpy array 간 연산, 미리 또 다른 feature를 setting할 필요가 없는 linear search가 LSH search 보다 더 합리적인 선택으로 보인다. 이를 기반으로 프로젝트 구현 시 사용자로부터 장르만 받을 경우 linear search로 분석하도록 하였다.

Korean book

Algorithm	Average Time
linear search (cosine similarity)	-
my search	1.682(s)

Foreign book

Algorithm	Average Time
linear search (cosine similarity)	-
my search	1.537(s)

Table 2: 사용자로부터 장르와 도서 목록을 받았을 경우, linear search 및 새롭게 정의한 my search 평균 시간

Algorithm	Similarity
cosine similarity	0.722
my similarity	0.913

Table 3: 하나의 데이터 셋을 통해 도서의 id는 다르지만 저자가 같은 두 개의 도서 간에 similarity 값

Table 2의 결과에서 linear search는 정확히 장르 벡터가 하나 추가될 때마다 2배 씩 linear하게 Average Time이 증가하는 것을 확인할 수 있었고, 따라서 전체

Average Time을 구하는 의미가 없었다. 반면에 벡터를 압축한 user profile은 정보가 많아져 user profile의 벡터가 많아질수록 Average Time이 일관되게 증가하지 않았기 때문에 전체 Average Time을 구할 수 있었다. 즉, 압축한 user profile 벡터와 새롭게 정의한 similarity가 계산 량을 줄이는 데에 효과적이었음을 확인할 수 있었다. 또한 Table 3의 결과를 통해 도서는 다르지만 hashing한 저자의 값이 같을 시 similarity score가 올라가는 것을 확인함으로써 사용자의 도서 선택에 대한 추천 경향이 적절히 반영되어 개선되었음을 확인할 수 있었다. 이를 기반으로 프로젝트 구현 시 사용자로부터 장르와 도서 목록을 받을 경우 my search로 추천 도서를 분석하도록 하였다.

5. Conclusion

도서 구매 사이트들은 많이 존재하지만 사용자 개인이 선호하는 도서의 경향을 파악하여 추천하는 시스템은 존재하지 않는다. 또한 초기 유저가 충분치 않아 평가 정보가 별로 없는 환경에서 콘텐츠 기반 추천 시스템은 좋은 방법으로 여겨진다. 이러한 이유로 콘텐츠 기반 도서 추천 시스템을 만들어 보고 싶었으며, 콘텐츠 기반 추천 시스템에서의 고질적 문제인 response 타임을 최대한 줄이면서 추천 또한 잘 이루어 질 수 있는 콘텐츠 기반 추천 시스템을 만들어보고 싶었다.

본 프로젝트에서는 response 타임을 최대한 줄이면서 추천 결과에 큰 영향을 끼치지 않기 위해 LSH search와 my search라는 두 가지 알고리즘을 제시한다. LSH search는 document similarity를 측정하는 LSH 알고리즘을 장르 binary 벡터를 압축하여 빠르게 search하는 데에 사용된 search 기법이며, 이는 추천 가속화에 유의미한 영향을 미치지 못 했다. 반면에 my search는 사용자의 도서 관련 정보가 많을 시 정보를 보존하면서 user profile의 벡터를 압축하고 압축한 벡터로 새롭게 정의한 my similarity로 similarity를 계산 하여, linear search보다 search 시간을 획기적으로 줄이면서 추천 결과 또한 해치지 않을 수 있었다. 이러한 콘텐츠 기반 추천 알고리즘은 도서 콘텐츠뿐만 아니라 영화와 같은 장르가 다양한 콘텐츠 기반 추천 시스템에 적용 및 확장할 수 있을 것으로 기대된다.

References

[1] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul lslam, Chidambaram Crushev, “A Survey on Locality Sensitive Hashing Algorithms and their Applications”, arXiv:2102.08942, Feb, 2021.