구직자를 위한 추천 알고리즘 연구

박준호*, 김지용**, 박진호*
*숭실대학원 컴퓨터학과
,*숭실대학교 소프트웨어학부
e-mail : park1058@ssu.ac.kr

A Study of Recommendation Algorithms for Job Seekers

Joon-Ho Park*, Kim ji yong**, Jin-Ho Park**
*Dept. of Computer Science, Soongsil University
,* Dept. of Software, Soongsil University

요 약

구직자는 취업을 하기 위해 다양한 채용 공고를 확인하고 이력서를 제출하는데 많은 시간을 소비한다. 만약 채용 추천 시스템을 통해 사용자에게 알맞는 회사를 추천해 준다면 구직자는 구직 활동 시간을 절약할 수 있다. 이를 위해 본 논문에서는 구직자에게 알맞는 회사를 추천하기 위한 알고리즘을 알아본다.

1. 서론

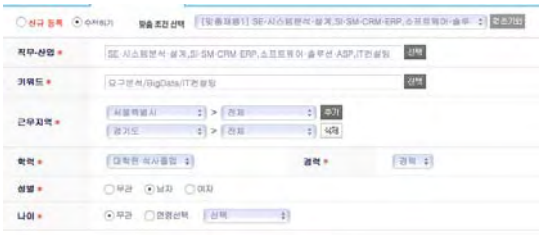
구직자는 취업을 하기 위해 취업 포털에 자신의 이력서를 등록한다. 이력서에는 구직자의 개인정보를 비롯하여 다양한 정보를 입력한다. 취업 포털은 구직자가 작성한 정보를 분석하여 구직자에게 가장 알맞은 회사를 추천해 주고 있다. 하지만 아직까지 제공되는 정보는 사용자의 요구를 충족하지 않고 있다. 본 논문에서는 구직자에게 알맞는 회사를 추천하기 위한 추천 알고리즘과 개선 방안에 대해 다룬다.

2. 본론

본 논문에서는 구직자들이 취업 포털에서 활동하는 이력 정보, 이력서에 작성된 기본 정보를 활용하여 학습한다. 구직자가 취업 포털에 작성된 이력서는 기본사항, 학력사항, 경력사항, 취업우대사항, 교육이수내용, 자격증, 컴퓨터 활용능력, 어학시험 및 외국어 구사능력, 자기 소개서를 작성한다. 하지만 구직자의 정보 중 경력사항과 자기 소개서는 일반화를 통해 알고리즘에 적용하기 어렵다. 따라서 본 논문에서는 추천 알고리즘에 적용이 가능한 항목을 도출한다.

2.1 기존의 추천 방안

취업 포털은 구직자의 정보를 분석하여 사용자에게 가장 알맞는 회사를 추천해 주고 있다. 취업 포털은 구직자가 이력서를 작성할 때 직무와 산업 그리고 키워드를 선택한다. 이를 통해 취업 포털은 1 차적으로 키워드를 통해 가장 유사한 기업의 채용 정보를 추천한다. 그림 1 은 국내 취업 포털에서 구직자에 추천하기 위한 조건 설정 화면이다. 일반적인 추천 알고리즘 중 SEG 방식은 구직자의 희망직종, 희망지역, 학력을 사용한다.



(그림 1) 맞춤 채용정보 조건 설정

*** 교신저자,

※ 이 논문은 서울어코드 활성화 사업에서 지원되었음.

SEG 방식을 통해 구직자에게 추천하는 경우에는 다양한 시나리오를 구성한다. 시나리오 구성은 'A 와 연관성이 높고 A 와 직종은 동일한 공고를 추천' , 'A 와 연관성이 높고, A 와 직종연관성이 가장 높은 공고추천' 등의 시나리오를 구성한다. 다양한 시나리오 구성으로 추가 및 변경이 발생한 경우 민첩하게 대응이 어렵다.

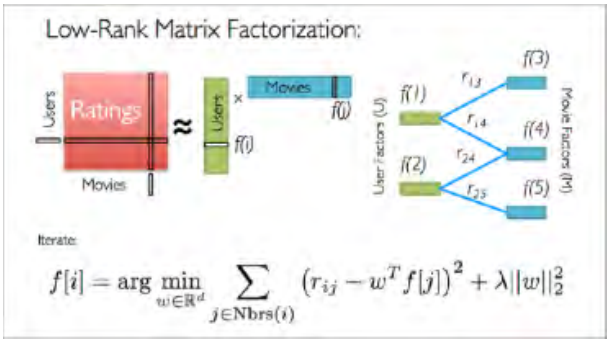
2.2 제안하는 알고리즘

본 논문에서 기존에 운영하고 있는 시나리오 기반에서 추천 알고리즘을 이용한 방안을 제시한다. 제시하는 알고리즘은 User-Based, Item-Based, Keyword-Based, Apache Spark 의 ALS 을 이용한다. User 는 구직자를, Item 은 채용 공고, Keyword 는 검색 키워드로 정의한다. 본 논문에서는 구직자의 기본 행동을 정량화 하여 구직자간의유사성을 계산하다. 유사성(Similarity) 계산을 위해 jaccard 을 기본으로 사용하고, 각 결과에 따라 cosine, pearson 을 사용한다. 또한 Apache Spark 의 ALS(그림 2)을 제외한 알고리즘은 IPython 을 사용했다. 본 논문에서는 각 알고리즘의 동일한 결과 도출을 위해 <표 1>과 같이 User_id, Item_id, Similarity, top_k, rating 정보를 사용한다.

<표 1> 알고리즘 공통 함수 필드

Algorithm	User_id	Item_id	Similarity	only_top_k	rating
User-Based	111	222	jaccard	TopN	binary
Item-Based	AAA	BBB	jaccard	TopN	binary
Keyword-Based	abc	abc	jaccard	TopN	binary

user_id='user_id', item_id='item_id', type='item-based', similarity_type='jaccard', only_top_k=10) 으로 정의하여 확인한다.



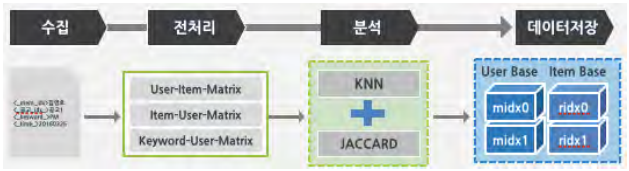
(그림 2) Apache Spark ALS 알고리즘의 Matrix Factorization

3. 실험

본 장에서는 2 장에서 제안한 각각의 알고리즘을 수행한다. 구직자에게 가장 알맞는 추천을 할 수 있는 시나리오를 설계하여 수행한다. 이를 통해 가장 알맞는 알고리즘을 제시한다.

3.1 실험 환경

본 논문에서는 국내 구직사이트에서 POC 를 위해 제공한 LogData 를 기반으로 실험을 진행했다. LogData 의 형식은 CVS 형식으로 User_id, Item_id, Rating 값을 기본으로 하고 있다. IPython 기반의 GraphLab 을 이용하여 알고리즘을 수행하였다. 알고리즘을 수행한 환경은 메모리 8 기가, SSD 디스크, Mac 에서 수행했다. 알고리즘 수행 프로세스는 데이터 수집, 전처리(User-item Matrix, Item-User Matrix, Keyword-User Matrix), 알고리즘 수행(User-base) 순으로 진행한다.(그림 3)



(그림 3) 단계별 알고리즘 프로세스

개발 코드는 자바소스 (그림 4)를 이용한 코드와 Ipython 기반으로 구현했다. (그림 5) 와 같이 같은 설계 내용이지만 코드의 길이가 다른 것을 알수 있다.

```
import java.io.*;
import java.util.*;

class BaseCF
{
    public static void makeRecList(HashMap<Integer,VRArr> user_item_map, HashMap<Integer,
        VRArr> item_user_map)
    {
        if (CFType.isUB()) {
            System.out.println("#### User-Based CF ####");
            executeWorker(user_item_map, item_user_map, "ub");
        }

        if (CFType.isIB()) {
            System.out.println("#### Item-Based CF ####");
            executeWorker(user_item_map, item_user_map, "ib");
        }
    }

    private static void executeWorker(HashMap<Integer,VRArr> user_item_map, HashMap<Integer
        ,VRArr> item_user_map, String type)
    {
        int i;
        int cnt;
        int n_thread = Configuration.getConfValue(Configuration.NO_OF_THREADS);
        int total;

        HashMap<Integer,int[]>  top_k_cache_map = null;

        (new File("out")).mkdir();

        if (type.equals("ib")) {
            total = item_user_map.size();

            top_k_cache_map = new HashMap<Integer,int[]>();

            cnt = 0;
            IUR[] iur = new IUR[total];
            for (Map.Entry<Integer,VRArr> ent: item_user_map.entrySet()) {
                iur[cnt++] = new IUR(ent.getKey(), ent.getValue());
            }

            System.out.printf("Total items: %d\n", total);
            Worker2[] ws = new Worker2[n_thread];
            for (i=0; i<n_thread; i++) {
                ws[i] = new Worker2(i, n_thread, user_item_map, item_user_map, iur);
                ws[i].start();
```

(그림 4) Base CF 알고리즘 코드

```
from pyspark import SparkContext
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating

sc = SparkContext()

# Load and parse the data
data = sc.textFile("ui.txt")
ratings = data.map(lambda l: l.split(','))\
    .map(lambda l: Rating(int(l[0]), int(l[1]), int(l[2])))
ratings.cache()

# Build the recommendation model using Alternating Least Squares
rank = 200
numIterations = 10
model = ALS.trainImplicit(ratings, rank, numIterations, 0.01, alpha=0.01, nonnegative=
    True)

# Evaluate the model on training data
testdata = ratings.map(lambda p: (p[0], p[1]))
predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
print("Mean Squared Error = " + str(MSE))

# Save model
model.save(sc, "./rec_model")
```

(그림 5) IPython 기반 CF 알고리즘 코드

3.2 실험 결과 및 해석

(그림 6)은 GraphLab 을 이용하여 Logdata 을 분석한 결과를 나타낸 그림이다. (그림 6)을 통해 추천되는 공고가 일부 사용자에게 동일하게 추천됨을 확인할 수 있었다. 이경우 Score 는 향상되지만 중복 추천에 문제가 발생하였다.

(그림 6) 동일한 사용자별 추천 아이템 리스트

이를 해결하기 위해 각 변수별 가중치를 조절하여 문제를 해결하였으며, Rating 값과 TopN 값을 조절하여 일부 문제를 해결하였다.

4. 결론

User_id	item_id	score	rank
1003091	26349455	0.0833333333333	1
1003091	26569098	0.0666666666667	2
1003091	26373707	0.0555555555556	3
1003091	26357070	0.0555555555556	4
1003091	26291758	0.0555555555556	5
1003091	26262477	0.0555555555556	6
1003091	25862519	0.0555555555556	7
1003091	24997182	0.0555555555556	8
1003091	24869859	0.0555555555556	9
1003091	24229809	0.0555555555556	10

User_id	item_id	score	rank
1003091	26169964	0.125	1
1003091	26022918	0.125	2
1003091	24006244	0.125	3
1003091	26139965	0.1	4
1003091	26072065	0.0833333333333	5
1003091	26418063	0.0625	6
1003091	26458407	0.0555555555556	7
1003091	25774308	0.05	8
1003091	26529612	0.0416666666667	9
1003091	25416212	0.0416666666667	10

User_id	item_id	score	rank
1003091	26423797	1.4009190308	1
1003091	26376752	1.3820045825	2
1003091	18950456	1.3777214887	3
1003091	10916897	1.3754853062	4
1003091	26570530	1.3717120898	5
1003091	26401691	1.3706995448	6
1003091	26514073	1.3682259173	7
1003091	26520009	1.3681279209	8
1003091	26545291	1.3618952359	9
1003091	26375274	1.3611462977	10

본 논문에서는 다양한 추천 알고리즘을 통해 구직자에게 가장 알맞는 채용 공고를 제공하고자 하였다. 이를 위해서는 다양한 변수 활용이 중요하다. 또한 구직 사이트는 사용자와 채용정보가 실시간으로 업데이트 되고 있다. 이에 대한 데이터 처리 방안이 필요하다. 또한 일반 사용자가 구직활동을 하는 경우도 있지만 로봇을 통한 활동, 채용을 올린 회사의 명이 변경된 경우 등의 상황을 고려한 알고리즘 설계가 필요할 것 같다.

본 논문에서는 CF 알고리즘이 아닌 Contents-based Filtering 기반의 추천 시스템을 활용하면 구직자에게 더 알맞는 정보를 제공할 수 있을 것으로 기대되며, 구직 사이트의 경우 맞춤형 채용 정보를 위해 각 개인 정보를 데이터 베이스화 하는 것도 데이터의 정확도를 높이는데 중요하다.

참고문헌

[1] Balabanovic , Marko, and Yoav Shoham, “Content-Based Collaborative Recommendation.”, Communications of the ACM, 40(3) : 66-72, 1997

[2] Juan Ramos, "Using TF-IDF to determine word relevance in document queries," ICML, 2003.

[3] Micheline Kamber “Data Mining: Concepts and Techniques,” Morgan Kaufmann Pub, 2011.

[4] Billsus, Daniel, and Michael J Pazzani. Learning Collaborative Information Filters. Berkeley : University of California Press. 1998