JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 21, No. 5 : 974~983 May. 2017

스마트폰 애플리케이션을 위한 임베디드형 피드백 지원 검색체

강문중[1] · 황민태[2*]

Embeded-type Search Function with Feedback for Smartphone Applications

Moonjoong Kang[1] · Mintae Hwang[2*]

[1]School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, 123 Cheomdangwagi-ro, Buk-gu, Gwangju 61005, Korea
[2]Dept. of Information and Communication Engineering, Changwon National University, 20 Changwondaehak-ro, Uichang-gu, Changwon 51140, Korea

요 약

본 논문에서는 안드로이드 기반의 각종 어플리케이션에 내장시켜 사용가능한 검색체에 대해 연구하였다. 이를 위해 조사와 같이 무의미하지만 자주 사용되는 단어를 빈도수에 따라 억제하는 BM25, 아이템의 길이 편차에 따른 검색 순위 문제를 해결하기 위해 아이템의 길이에 따라 중요도를 보정하는 Pivoted Length Normalization, 그리고 벡터 공간 모형 상에서 쿼리 벡터를 관련 있는 것으로 판정된 아이템들의 벡터 그룹으로 끌어당겨 보정하는 Rocchio's Method를 사용해 묵시적 피드백 기능을 지원하도록 하였다. 그리고 색인 작업은 오프라인 동작을 위한 간단 색인과 온라인 동작을 위한 정밀 색인의 두 단계로 나누어 동작성을 보장하도록 하였다. 본 논문에서 연구한 피드백 지원 검색체는 쿼리 유추를 통해 사용자의 입력을 색인된 자료와 비교해 입력 내용을 예측하고 오타 등의 사용자 실수에 대해서도 대응하므로 스마트폰 어플리케이션에 손쉽게 탑재되어 검색 기능을 향상시킬 수 있을 것으로 기대한다.

ABSTRACT

In this paper, we have discussed the search function that can be embedded and used on Android-based applications. We used BM25 to suppress insignificant and too frequent words such as postpositions, Pivoted Length Normalization technique used to resolve the search priority problem related to each item's length, and Rocchio's method to pull items inferred to be related to the query closer to the query vector on Vector Space Model to support implicit feedback function. The index operation is divided into two methods; simple index to support offline operation and complex index for online operation. The implementation uses query inference function to guess user's future input by collating given present input with indexed data and with it the function is able to handle and correct user's error. Thus the implementation could be easily adopted into smartphone applications to improve their search functions.

키워드 : 검색 기능, 스마트폰 응용, 안드로이드, 텍스트 검색, 피드백

Key word : Search Function, Smartphone Application, Android, Text Retrieval, Feedback

Received 12 December 2016, **Revised** 14 December 2016, **Accepted** 28 December 2016

* **Corresponding Author** Mintae Hwang(E-mail:professorhwang@gmail.com, Tel:+82-55-213-3832)
Department of Information and Communication Engineering, Changwon National University, Changwon 641-773, Korea

Open Access https://doi.org/10.6109/jkiice.2017.21.5.974 print ISSN: 2234-4772 online ISSN: 2288-4165

ⓒThis is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(http://creativecommons.org/li-censes/ by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Copyright Ⓒ The Korea Institute of Information and Communication Engineering.

Ⅰ. 서 론

정보의 양이 폭증하는 요즈음 웹이나 데이터베이스의 수많은 데이터 중에서 원하는 정보를 찾아주는 검색 기능은 이제 방대한 자료를 다루기 위한 필수적인 요소로 자리 잡고 있다. 빅 데이터를 대상으로 하는 구글, 네이버 등의 검색 엔진 사이트는 물론이고 쇼핑몰 등에서도 항상 상단 중앙에는 검색 기능이 자리 잡고 있으며, 카카오톡이나 텔레그램, 어도비 PDF 리더기 등의 스마트폰 어플리케이션들 또한 모두 검색 기능을 지원한다. 그러나 전문적인 웹 검색 엔진 사이트들을 제외한다면 어플리케이션에 탑재되는 검색 기능들은 단순히 검색어로 주어진 문자열을 포함하는 데이터를 찾고 또 그 문자열의 위치를 알아내는 초보적인 수준에 그친다. 그리고 이러한 검색 기능은 텍스트 그 자체가 똑같은 동음이의어나 다의어를 대상으로 한 검색에서는 취약하며, 그렇지 않은 단어를 대상으로 한 검색에서도 사용자가 원하는 데이터를 정확하게 높은 순위로 찾아준다는 보장은 없다.

사용자가 원하는 바를 문자열에 불과한 검색어만으로 정확하게 반영할 수는 없겠지만, 사용자가 검색 결과에 대해 보이는 반응을 피드백 받아서 그 이후의 검색 결과에 반영한다면 사용자의 묵시적인 의도를 좀 더 정확하게 이해할 수 있는 검색이 가능할 것이다. 그리고 여기에 더해 특정 사용자의 피드백이 계속 검색 결과에 축적된다면 그 사용자의 취향에 맞춘 검색 결과를 내놓을 수도 있을 것이다.

이러한 피드백 기술은 당연히 상용 웹 검색 엔진 등에서 활용되고 있으며, 대형 데이터베이스 등의 검색 기능에서도 이용된다. 그러나 스마트폰 등에 내장되는 검색 기능들은 단순한 텍스트 필터링 수준에 그친다.

또한 이러한 피드백 기능을 지원하는 검색 기능의 구현체들 중에서 오픈 소스인 것들은 모두 대형 서버나 최소한 PC 정도의 플랫폼을 대상으로 한다. 아직까지는 스마트폰 내에서 이용할 수 있는 SQLite와 같은 자체 데이터베이스를 기반으로 피드백 검색 기능을 제공하는 오픈 소스는 공개된 바가 없다. 애플의 Spotlight는 비슷한 기능을 지원하고는 있으나 비공개 독점 소스 기반이다.

만일 스마트폰을 대상 플랫폼으로 하고서 간편하게 여러 어플리케이션에 탑재되어 피드백 기능을 제공하는 검색 구현체를 만들 수가 있다면 이를 통해 검색 결과의 품질을 향상시키고 어플리케이션들의 검색 성능 향상에 보탬이 될 수 있을 것이다.

따라서 본 논문에서는 안드로이드 기반의 스마트폰을 대상 플랫폼으로 하여 피드백을 지원하는 검색 기능의 구현에 대해 연구하였다. 이를 위해 수학적으로 단어의 의미를 표현하는 벡터 공간 모형(Vector Space Model), 단어의 빈도에 따른 보정 기능을 위해 Okapi BM25, 그리고 과도한 피드백을 억제하기 위한 Pivoted Length Normalization 알고리즘을 동시에 적용하는 검색 기능 구현체를 제안하고서 이를 Java 프로그래밍을 이용해 프로토타입을 구현하였다.

본 논문의 구성은 다음과 같다. II장에서는 검색 구현체에 적용한 여러 가지 이론 및 기법에 대해 설명하며, III장에서는 직접 구현한 검색 기능 구현체에 대해 소개한다. 그리고 IV장에서는 구현한 검색 구현체의 평가 모델을 소개하고, 마지막으로 V장에서 결론을 기술하였다.

Ⅱ. 검색 구현체 기반 이론

2.1. 벡터 공간 모형(VSM, Vector Space Model)

문맥 상 단어의 의미는 각 단어에 대한 선별적 선호도를 결합시킨 벡터 공간 모형으로 표현 가능하다[1]. 또한 각각의 단어를 사전적 의미에 대응하는 단일 벡터와 그 단어가 지닐 수 있는 문법적인 기댓값 및 선별적 선호도를 대표하는 벡터들의 집합으로 표현 가능하며, 이 벡터들을 서로 비교하여 얼마나 문맥상으로 유사한 의미를 가지는지를 판별하기 위해서는 두 벡터 간 유사성 비교를 위해 자주 쓰이는 코사인 유사도를 사용한다[2-5]. 이처럼 문맥 상 단어의 의미를 수학적으로 모델링함으로서 검색 구현체로 하여금 단순히 형태가 아닌 단어들의 문맥상 의미에 따라 검색 결과를 제시할 수 있을 것으로 기대한다.

기본적으로 본 논문에서 제안하는 검색 구현체는 이러한 벡터 공간 모형 이론에 기반을 두고 있다. 사용자가 보고 호출할 수 있는 아이템들은 구현체 내에서 ActionMacro 객체에 대응된다. 그리고 ActionMacro 객체는 ActionWord 객체를 여러 개 포함할 수 있으며, ActionWord는 벡터 공간 모형 상에서 더 이상 분해할

수 없는 하나의 독립된 단위이자 하나의 차원으로 간주되므로 전체 n개의 ActionWord가 있는 데이터베이스는 n차원 공간으로 취급된다. 그리고 모든 아이템과 쿼리문은 이러한 각각의 단위에 대한 연관성 값의 벡터의 합으로 취급된다. 그리고 각각의 독립된 단위의 차원은 서로 직교해야 가장 이상적이다. 제안하는 검색 구현체는 문장을 문장 속 단어의 선후 관계 등은 고려하지 않으며 단지 각각의 아이템을 순서를 구별할 수 없는 최소 단위의 집합으로만 간주하는 “Bag of Words”로 취급한다.

그러나 검색 구현체는 각각의 차원이 직교관계에 놓인 두 벡터처럼 엄밀히 구분된다고는 볼 수 없다. 인간이 사용하는 자연어의 특성상 각 단어의 연관성을 엄밀하게 구별할 수 없어 완벽한 구분을 보장할 수 없기 때문이다. 예를 들어 스마트폰 자체에서 진행되는 간단 색인만으로는 형태소 단위까지의 분석이 이루어지지 않으므로 “풋사과”와 “사과”의 연관성을 알아낼 수 없다. “풋사과”는 “풋” + “사과”로 분리될 수 있다. 따라서 형태만을 보아도 분명 연관성이 있는 단어들임에도 서로 다른 ActionWord로 처리된다. 이러한 예시 말고도 형태가 다른 유의어, 동의어 등의 경우까지 포함한다면 형태소 분석을 거쳐도 최종적으로 분해된 모든 단위가 서로 직교한다는 보장은 불가능하다.

2.2. 단어의 빈도에 따른 보정

ActionMacro에서 어떤 특정 ActionWord가 많이 반복되는 경우가 있다. 간단한 예시를 들자면 우리말의 조사나 영어의 be동사와 같은 것들이 있다. 이런 ActionWord들은 그 빈도수는 매우 높지만 그 ActionMacro의 중요한 뜻을 담는 경우는 거의 없기 때문에 실제로 검색어로서의 가치는 없다. 즉, 어떤 단어가 어떤 아이템 내에서 많이 반복된다고 해서 그 단어가 그 아이템과 깊게 연관성이 있는 것은 아니라는 뜻이다.

따라서 이런 경우를 대비하기 위해 한 ActionMacro 내에서 어떤 ActionWord가 너무 자주 반복되는 경우 그 ActionWord의 중요도에 대해 패널티를 가할 필요가 있다. 단 하나의 ActionWord가 다른 모든 ActionWord에 대해 너무 중요도를 높게 가지는 경우를 막아야 하는 것이다.

이렇게 중요도에 대한 제한을 수학적으로 두기 위해 이 구현체의 평가 공식에는 Okapi BM25(Best Matching) 알고리즘이 적용되었다. BM25는 검색 엔진, 추론 시스템 등에서 사용되는 스코어링 알고리즘이다[6].

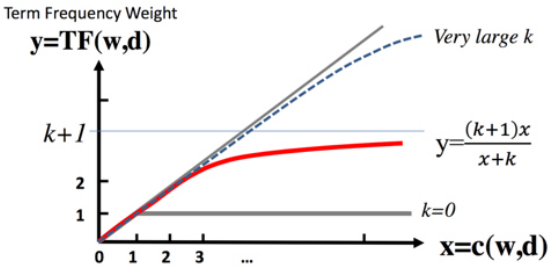

Fig. 1 TFW Transformation by BM25 [6]

먼저 빈도에 따른 중요도 평가 중 가장 간단한 방법으로는 그림 1에서 직선 그래프로 보이는 선형적인 평가법이 있다. 그러나 이 방법은 앞 문단에서 언급한 대로 조사와 같이 의미는 없으나 몹시 자주 사용되는 단어들의 중요도가 과도하게 평가된다.

반면 그림 1에서 굵은 실선으로 표시된 BM25는 빈도가 높아질수록 중요도가 증가하기는 하나, 빈도가 무한대로 높아지면 특정값 k+1에 수렴한다. 이렇게 되면 로그함수나 일차함수와 달리 값이 발산하지 않고 특정값 한계 내로 수렴하므로 안정성이 보장되며, 빈도가 과도하게 높아지는 경우에도 그 중요도의 증가가 훨씬 적어짐으로서 많은 빈도로 사용되는 특정 단어가 너무 큰 중요도를 가지는 경우를 방지한다[6-8].

2.3. 아이템의 길이에 따른 보정

상대적으로 다른 ActionMacro들에 비해 너무 긴 ActionMacro의 경우, 그만큼 쿼리에 매치될 확률이 평균적으로 높아진다. 따라서 ActionMacro의 길이에 따른 패널티를 부과해 길이에 따른 정규화를 시킬 필요가 있다. 이를 위해 사용되는 것이 그림 2에 제시한 PLN(Pivoted Length Normalization)이다[9,10]. PLN은 주어진 특정 길이를 기준으로 삼아 그 보다 큰 길이를 가지는 아이템에 대해서는 그 평가값에 패널티를 주고, 반대로 기준값보다 짧은 아이템에 대해서는 보너스를 줘서 그 평가값을 끌어올린다.

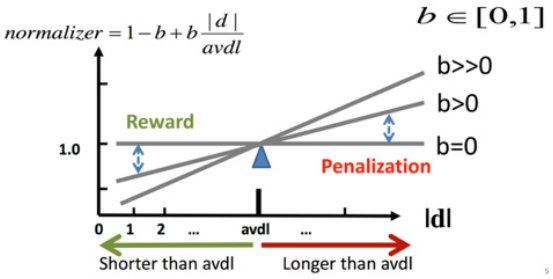

Fig. 2 Compensation of PLN by Item Length [9,10]

본 논문에서 고려하는 검색 구현체는 PLN의 Pivot 지점을 평균 ActionMacro 길이로 결정한다. 여기서 길이는 ActionMacro를 구성하는 ActionWord의 수를 말한다. 따라서 평균 길이를 가지는 ActionMacro는 PLN 보정계수에 보정이 가해지지 않으며, 이에 대해 차이가 클수록 보정치의 크기가 커진다.

2.4. 쿼리 벡터의 보정

본 논문에서 고려하는 검색 구현체는 사용자의 피드백을 받으며, 사용자의 반응이 미래의 검색 결과에 영향을 준다. 그리고 사용자의 검색 기록은 데이터베이스에 저장되며 점차 사용자의 선호에 따라 검색 결과가 조정된다.

이를 위해 Rocchio 알고리즘[11,12]을 사용하였으며, 이는 "관련 있음"으로 평가된 아이템들을 벡터 쪽으로 끌어당기며, 반대로 "관련 없음"으로 평가된 아이템들을 벡터로부터 멀어지게 보정한다.

$$\overrightarrow{q_m} = \alpha\vec{q} + \frac{\beta}{|D_r|}\sum_{\forall \vec{d_j} \in D_r}\vec{d_j} - \frac{\gamma}{|D_n|}\sum_{\forall \vec{d_j} \in D_n}\vec{d_j} \qquad (1)$$

아래 표 1은 Rocchio 알고리즘에 적용된 수식의 인수들과 그 의미를 제시하고 있다.

Table. 1 Parameters of Rocchio's Algorithm

$\vec{q_m}$	New query
$\vec{q}$	Original query
D_r	Relative documents
D_n	Non-relative documents
α, β, γ	Parameters

Ⅲ. 검색 구현체의 설계 및 구현

3.1. 쿼리 유추 기능

본 논문에서 구현한 쿼리 유추 기능은 사용자가 검색어를 입력하면 입력된 내용을 기반으로 쿼리 유추를 수행한다. 쿼리 유추는 두 가지 기능이 있다. 첫 번째는 단순한 문자열에 불과한 검색어를 색인된 정보와 연관된 쿼리 벡터로 변환하는 것이고, 두 번째는 현재 주어진 검색어를 기반으로 사용자가 의도하는 쿼리가 무엇인지 유추하여 사용자의 키 입력 횟수를 줄이는 것이다.

$$s(C_1, C_2) = \frac{M}{N(C_1), N(C_2)} \qquad (2)$$

아래 표 2는 상기 수식에 사용된 인수들이다.

Table. 2 Parameters of Similarity Formula

M	Count of matching initial/medial/final sound
$N(C_1)$	Count of consonants and vowels of C_1

쿼리와 각 단어 간의 유사도는 코사인 유사도 방식을 변형해서 이용한다. 먼저 비교할 두 단어를 각 음절 단위로 서로 비교하게 된다. 일례로 "고길동", "홍길동"이란 단어를 비교하게 될 경우, "고"와 "홍", "길"과 "길" 그리고 "동"과 "동"이 서로 짝을 이루게 된다. 그리고 비교대상이 되는 각 음절을 유니코드 정규화 방식을 이용해 초성, 중성 및 종성을 분리한다. "고"와 "홍"의 경우 <ㄱ,ㅗ>, <ㅎ,ㅗ,ㅇ>이 된다. 그리고 이들을 각각 비교하되 자모가 같으면 1, 틀리면 0으로 처리해 그 값들을 합해 분자로 사용한다. "고" 와 "홍"의 경우 중성만 같으므로 중성은 1, 나머지 초성과 종성은 모두 0이 된다. 따라서 분자는 이들을 합한 값인 1이 된다. 그리고 뒤의 두 음절은 완전히 같으므로 둘 다 1이 나오며, 유사도 결정에는 이들 3가지 값의 평균값을 이용한다.

그리고 분모에는 코사인 유사도와 비슷하게 각 음절별 자모 수의 제곱근들간 곱을 계산해 넣는다. 따라서 "고길동"과 "홍길동"의 유사도 평가값은 0.8027494로서 약 80.27%의 유사도를 가진다고 볼 수 있다. 이 방식은 코사인 유사도와 비슷하게 결과값의 범위가 0~1로 고정된다.

이렇게 초성, 중성 및 종성 단위까지 분석하게 되는 경우 “ㄱ”, “ㄴ”과 같은 자음 하나만 쿼리로 입력해도 비교 작업을 진행하는 것은 가능하다. 그러나 이렇게 될 경우 비교해야 할 아이템의 수가 너무 많아지므로 사용자의 입력 속도에 대한 대응이 떨어질 수 있다. 따라서 본 논문에서는 한 음절이라도 일치하는 것들에 대해서만 비교를 수행하도록 필터링 작업을 거친 후 반환된 아이템들에 대해서만 일치 작업을 수행하도록 구현하였다.

이렇게 쿼리 유추 단계를 거친 후 만들어진 쿼리 벡터는 VSM에 따라 색인된 정보와 비교할 수 있게 된다.

3.2. 색인

본 논문에서 구현한 검색 구현체에 포함된 색인 작업은 주어진 ActionMacro를 형태소 단위까지 쪼갤 수 있다. 그러나 형태소 단위까지의 분석에 Java 기반의 Komoran 2.4e 라이브러리[13]와 한국어 위키피디아의 모든 학습 데이터를 적용해 본 결과 메모리 요구량이 너무 커 스마트폰 어플리케이션에서 자체적으로 형태소 분석을 처리하는 것이 부적합하였다.

따라서 주어진 ActionMacro를 스마트폰 자체에서 공백 단위로 분리하는 간단 색인 방법과 형태소 분석 서버에 접속해 형태소 단위로 분리하는 정밀 색인 방법의 두 가지 방법으로 색인이 가능하도록 구현하였다.

1) 간단 색인 – 공백 단위 분리

이 방식은 형태소 분석 서버에 접속이 불가능한 경우 스마트폰에서 자체적으로 처리 가능한 간단 색인 방법으로서 사용자가 ActionMacro를 추가하면 간단하게 공백을 구분자로 삼아 단위를 분리한다. 예를 들면 “I like yours”는 “I”, “like” 그리고 “yours”로 구분하고서 이를 스마트폰 내부의 데이터베이스에 기록한다.

그러나 이러한 간단 색인은 의미 있는 형태소 단위까지 정확한 분리가 되지 않는다. 예를 들어 “풋사과”란 단어가 추가된다면, “풋사과”는 “풋”과 “사과”로 형태소 분리가 가능하나 간단 색인 방법으로는 이를 수행할 수 없다. 따라서 “풋사과”와 “사과”는 서로 독립된 ActionWord로 데이터베이스에 추가된다.

그러나 앞서 언급한 쿼리 유추 기능으로 인해 간단 색인만으로도 이 문제는 다소 해결된다. 그림 3에서 예를 든 바와 같이 간단 색인만 된 상태로 “사과는”, “사과를” 그리고 “사과가”가 데이터베이스에 있는 상태를 가정하자. 정밀 색인 처리가 된다면 “사과”, “는”, “를”, “가”로 분리해 낼 수 있지만 간단 색인 기능에서는 그럴 수가 없다. 하지만 사용자가 “사과”를 입력하면 쿼리 유추 단계에서 세 가지 ActionWord에 대해 유사도를 측정하게 된다. 그리고 세 가지 ActionWord는 모두 문자열 “사과”와 비교 시 높은 유사도를 보이며 0.66667 값을 가지는 쿼리 벡터로 만들어진다. 그리고 이 쿼리 벡터를 통해 세 단어는 서로 연관된 ActionMacro로서 검색되게 된다. 비록 유사도 값이 차이가 나지만 결국 형태소 분석이 없더라도 어느 정도는 사용자가 입력한 단어에 대해 유사한 단어들을 찾아낼 수 있는 것이다.

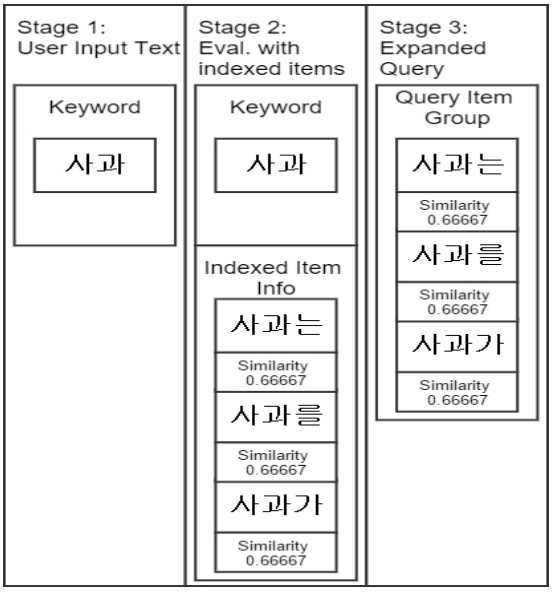

Fig. 3 Compensation of Simple Indexing by Query Estimate

2) 정밀 색인 – 형태소 단위 분리

앞서 언급한 바와 같이 형태소 분석에 이용 가능한 라이브러리는 용량이 커서 스마트폰 자체에서 구동이 불가능하므로 원격 서버에서 동작하는 형태소 분석 라이브러리를 이용한 정밀 색인 방법을 고려하였다.

형태소 분석 서버는 FreeBSD 9.3-RELEASE-p25 운영체제 상에서 동작하며, 본 논문에서는 Ecilpse IDE와 openjdk8-8.45.14_2 환경에서 Java 8 프로그래밍 언어를 이용해 구현하였다. 또한 멀티스레딩과 오브젝트 풀링 기능을 넣어 최대한 최적화를 시도했으며, 서버-클

라이언트 간 최대한 패킷을 절약해 통신할 수 있도록 객체 직렬화를 위해 Kryo 라이브러리[14]를 사용했다.

정밀 색인에서는 앞서 언급한 “풋사과”의 문제가 해결된다. “풋사과”는 서버에서 “풋” + “사과”로 분리되고, 스마트폰에는 “풋”과 “사과”가 저장된다. 그리고 사용자가 추가한 “사과”는 앞서 추가된 “사과”와 같은 단어임을 확인하고 새로운 ActionWord를 만드는 대신 기존 “사과”의 카운터를 1 증가시키는 것으로 끝난다.

형태소 분석은 공백 포함 분석과 공백 미포함 분석의 두 가지 방법으로 결과를 얻은 후 결과를 병합해 사용한다. 이런 방식을 채택하는 이유는 공백을 포함하는 고유명사의 존재 때문이다. 예를 들어 “나는 친구가 적다”라는 ActionMacro가 있다고 가정하자. 형태소 분석 서버에 탑재된 Komoran 라이브러리에서는 공백 포함 분석 시 “나는 친구가 적다”를 고유 명사로 판단하며, 공백 미포함 분석 시에는 “나” + “는” + “친구” + “가” + “적” + “다”로 반환한다. 이처럼 분석 방법에 따라 판단 결과가 다른 것이다.

아이템을 수정하거나 새로 추가하는 이벤트가 발생할 경우 구현체는 먼저 인터넷에 접근이 가능한지 확인한 후, 접근이 가능하면 서버로 아이템을 보내 처리 결과를 받는다. 그 후 서버에서 결과 값을 회신 받으며 처리를 맡겼던 해당 아이템이 그 사이에 변경되었는지 확인한다. 아이템이 만일 사용자에 의해 변경되었다면 다시 분석을 해야 하므로 회신받은 결과 값을 버리지만 그렇지 않다면 분석 결과 값들은 아이템의 최소 단위로서 데이터베이스에 업데이트된다. 이러한 송신-수신-업데이트 과정을 그림 4에 제시하였으며 모두 백그라운드 스레드에서 진행된다. 만일 서버로의 접속이 불가능할 경우 다시 데이터베이스 업데이트 이벤트가 발생할 때까지 대기 후 미처리된 아이템에 대한 처리를 재시도한다.

3.3. 쿼리 보정과 벡터 형성

앞서 언급한 형태소 분석을 거치면 “풋사과”는 접두사 “풋” + 명사 “사과”의 형태소 집합으로 분해될 수 있다. 쿼리 확장의 첫 단계에서 쿼리에 대응하는 원래 단어와 형태소들을 포함하는 목록인 “풋사과, 풋, 사과”가 생성된다. “풋사과”를 구성하는 형태소만이 아니라 단어 “풋사과” 자체도 포함된 이유는 형태소 분석이 불가능한 오프라인 환경에서의 검색을 고려해서이다. 오프라인 환경에서는 구성 형태소를 알 수 없으므로 단순히 “풋사과”만 있는 목록이 생성된다.

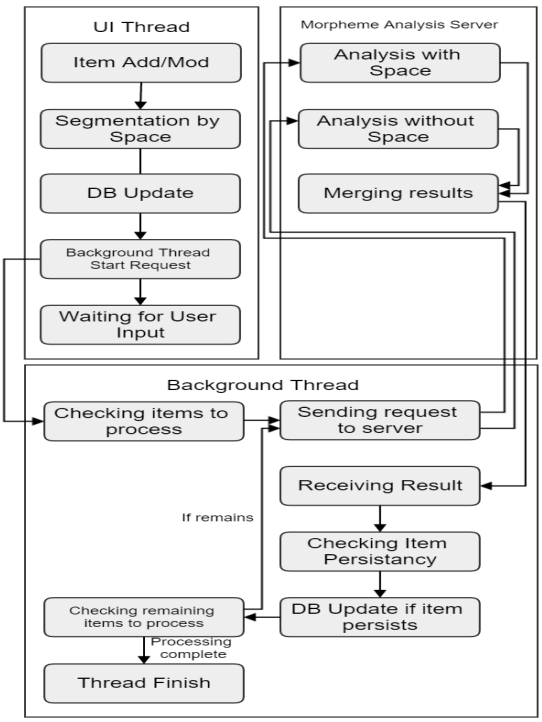

Fig. 4 Entire Indexing Flowchart

두 번째 단계에서는 내부 DB에 저장된 각 형태소와 연관된 다른 형태소들을 목록에 추가한다. 예를 들어 이전에 “풋”이라는 검색어를 제시하고, “풋김치”라는 검색 결과를 선택한 적이 있다면 “풋”이라는 형태소는 Rocchio’s Method에 따른 벡터 보정에 의해 데이터베이스 상에 “김치”라는 단어와 연관된 것으로 기록되어 있을 것이다. 따라서 “김치” 쿼리에 대응하는 형태소 목록은 “풋사과, 풋, 사과, 김치”로 확장된다.

마지막 단계에서는 이러한 형태소 목록을 검색체에 기록된 모든 형태소마다 하나의 차원이 대응되는 n차원 벡터로 전환하게 된다. 목록에 없는 형태소들은 연관이 없으므로 쿼리 벡터의 해당 차원의 크기는 0으로 결정되며, 목록에 존재하는 형태소들에 대해서는 저장된 평가값을 데이터베이스에서 불러와 쿼리를 쿼리 벡터로 확장하는 단계가 마무리된다.

최종적으로 생성된 쿼리 벡터는 예를 들어 <0, 0, ... ,

0.1(풋사과 차원), 0.2(풋 차원), 0.3(사과 차원), 0.4(김치 차원), 0, ... , 0> 과 같은 n차원 벡터가 된다.

3.4. 묵시적 피드백

피드백은 피드백 대상 쿼리의 각 단위에 대해 사용자와 상관있는 것으로 판별된 아이템들을 참조에 추가하되, 참조 보정값은 쿼리 내 각 단위의 코사인 유사도와 그 쿼리의 기존 피드백 보정값에 따라 결정되어 기존 보정값에 더해진다. 또한 반대로 상관없는 것으로 판별된 아이템들에 대해서는 동일한 계산 과정을 거쳐 보정값이 감소된다. 그리고 이렇게 저장된 피드백 정보는 검색 시작 단계에서 쿼리 유추가 끝난 뒤의 쿼리에 대해 쿼리의 구성 단위와 연관된 피드백 정보들을 가져와 쿼리를 한 번 더 보정하는데 사용된다. 그림 5에 이러한 쿼리 벡터 확장 과정을 단계별로 보여주고 있다.

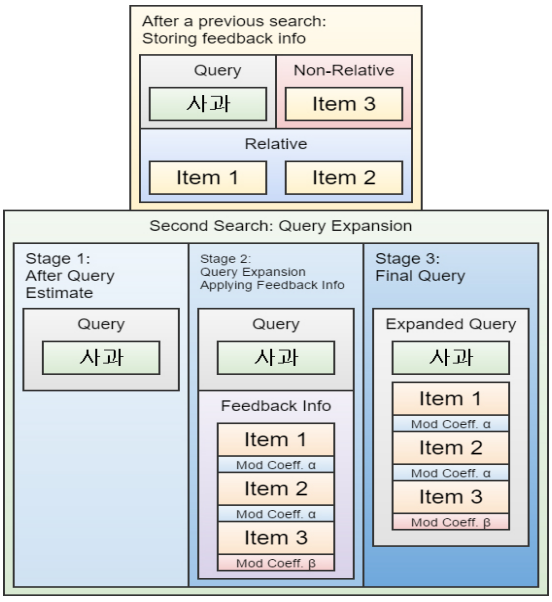

Fig. 5 Query Vector Expansion by Rocchio's Algorithm

피드백 방식을 위해서 먼저 조건을 두었다. 첫 번째는 검색 결과는 평가값을 기준으로 내림차순으로 결과가 출력된다는 것이다. 그리고 두 번째는 사용자가 맨 위에서부터 차례차례로 검색 결과를 훑으며 자신이 원하는 값을 찾는다는 것이다. 결과 출력은 첫째 조건에 정확하게 부합되도록 만들 수 있었지만, 두 번째 조건은 사용자의 행동 패턴이므로 단순 가정으로 넘어갈 수 밖에 없었다. 피드백 방식은 사용자의 번거로움을 피하기 위해 그림 6에서 보는 바와 같이 묵시적 피드백(Implicit Feedback)으로 정했다. 사용자가 위에서부터 아래로 목록을 훑어 내려가며 클릭하는 아이템을 쿼리와 상관있는 아이템, 클릭하지 않고 넘어가는 아이템을 상관없는 아이템으로 결정하고, 맨 마지막으로 클릭된 아이템까지만 사용자가 검토한 것으로 간주했다. 그 아래의 클릭되지 않은 아이템에 대해서는 피드백을 할 수가 없다. 명시적 피드백(Explicit Feedback)이 아닌 이상 사용자의 검토 여부가 확실하지 않기 때문이다. 탐색 결과가 나온 직후부터 사용자가 결과의 탐색을 종료하는 시점, 즉 다른 메뉴로 넘어가거나 새로운 검색어를 입력해 결과 목록을 갱신해야 할 때까지 사용자의 탐색 결과를 모은 뒤 한 번에 피드백된다.

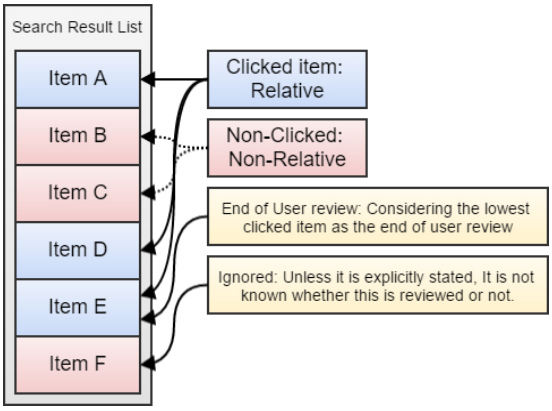

Fig. 6 Implicit Feedback

3.5. 피드백 지원 검색체 구현 결과

그림 7은 구현한 애플리케이션에서 "사과"를 검색했을 경우 "사과를", "사과가", "사과는" 등의 단어가 포함된 문장들을 검색 결과로 보여주는 모습이다.

서버와의 연결이 불가능한 오프라인 상태에서는 정밀 색인 작업이 진행되지 않으므로 단순히 "사과"라는 단어를 포함하고 있는 문장에 대한 패턴 매칭을 통해 결과를 검색하며, 온라인 상황에서는 서버 로그(Server Log) 기록에서 보는 바와 같이 정밀 색인을 통해 스마트폰 클라이언트가 보낸 문장을 Komoran 2.4e 형태소 분석 라이브러리를 이용해 형태소를 구분한 다음 "사과를", "사과가", "사과는" 등의 단어에서 어간 "사과"를 잘라내어 명확하게 인식하는 것을 볼 수 있다.

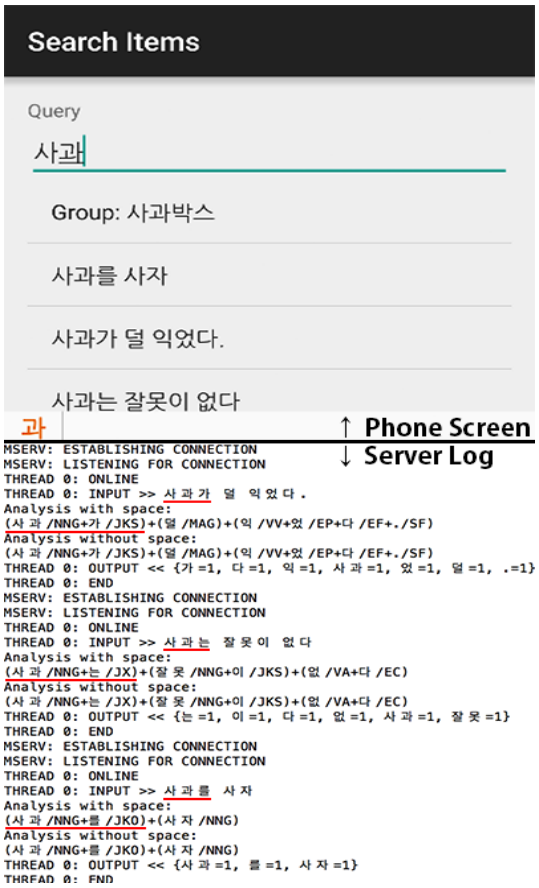

Fig. 7 Stem Matching via Morpheme Analysis

피드백은 피드백 대상 쿼리의 각 단위에 대해 사용자와 상관있는 것으로 판별된 아이템들을 참조에 추가하되 참조 보정값은 쿼리 내 각 단위의 코사인 유사도와 쿼리의 피드백 보정값에 따라 결정되어 기존 보정값에 더해진다. 반대로 상관없는 것으로 판별된 아이템들에 대해서는 동일한 계산 과정을 거쳐 보정값이 감소된다. 그리고 이렇게 저장된 피드백 정보는 검색 시작 단계에서 쿼리 유추가 끝난 뒤의 쿼리에 대해 쿼리의 구성 단위와 연관된 피드백 정보들을 가져와 쿼리를 한 번 더 보정하는데 사용된다.

묵시적 피드백은 주어진 쿼리에 대한 검색 결과에 대해 사용자의 반응에 따라 결과 값들의 벡터 가중치들을 조절할 수 있다. 그림 8에서는 선택한 "우유가 냄새나요."를 구성하는 단위 벡터들의 가중치가 조절되고 있는 모습을 확인할 수 있다.

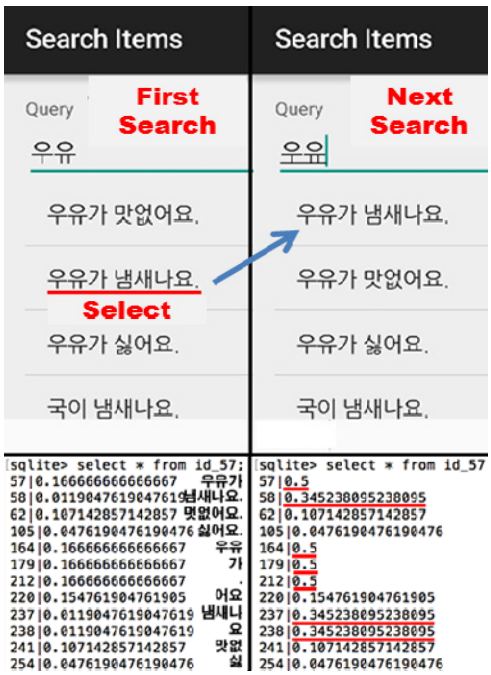

Fig. 8 Search Result Changes by Implicit Feedback

Ⅳ. 평가 모델

앞서 언급한 BM25, Pivoted Length Normalization을 적용해 만들어진 최종 평가식은 다음과 같다[15,16].

$$f(q,d) = \sum_{w \in q \cap d} c(w,q) \frac{\ln[1+\ln[1+c(w,d)]]}{1-b+b\frac{|d|}{avdl}} \cdot \log\frac{M+1}{df(w)} \quad (3)$$

표 3은 상기 평가식에 포함된 인수들과 그 의미를 보여주고 있다.

Table. 3 Parameters of Ranking Fomula

q	Query Vector
d	Document (ActionMacro)
c(c1, c2)	Count of c1 in c2
|d|	Length of Document
avdl	Average of total documents
M	Count of total documents
df(w)	Count of total documents containing w

본 논문에서는 구현한 검색 구현체의 정확도를 평가할 방법으로 Mean Average Precision(MAP)을 선택했다. 그리고 구현한 검색 구현체는 스마트폰에서 구동함을 전제로 하였기에 검색 결과에서 문서의 관련성을 체크하는 최하위 순위는 한 화면 상에 보여줄 수 있는 가장 아래쪽에 있는 문서로 결정했다. 예를 들어 한 화면에 10개의 검색 결과가 나올 수 있다면 그 이후에는 관련 있는 문서가 나와도 Precision(n 개중 몇 개가 관련 아이템인가)을 0으로 계산한다.

$$MAP = \sum_{n=1}^{M} \frac{\mathrm{Precision\,at\,Rank}(n)}{M} \tag{4}$$

여기서 M은 한 화면에 출력할 수 있는 쿼리의 개수 혹은 검색 결과의 개수 중에서 더 큰 수를 사용한다.

예를 들어 화면에 출력된 상위 10개의 검색 결과 중 1, 2, 5, 8번째 결과가 관련 있는 아이템의 경우 MAP는 아래와 같다.

$$\frac{\frac{1}{1}+\frac{2}{2}+\frac{3}{5}+\frac{4}{8}}{10} = 0.41 \tag{5}$$

다만 이 평가 모델은 자연어를 대상으로 한 구현체의 특성상 모든 검색 대상을 상대로 한 객관적인 절대평가를 제공할 수는 없으며, 같은 구현체라도 평가에 사용한 데이터 셋에 따라 MAP 값은 차이가 난다[17].

Ⅴ. 결 론

본 논문에서는 안드로이드 기반의 스마트폰 어플리케이션을 위한 임베디드형 피드백 지원 검색 기능 구현체에 대해 연구하였다.

벡터 공간 모형을 기반으로 하되 조사와 같이 의미 없이 빈도수가 높은 단어들을 억제하기 위해 BM25 식으로 빈도수-중요도의 관계를 변형시켰다. 그리고 아이템간의 길이 편차에서 길이가 긴 아이템이 중요도를 더 높게 받는 경향을 억누르기 위해 PLN을 써서 길이에 따른 보정이 가해지도록 했다. 그리고 피드백 기능을 위해 쿼리 벡터를 관련 있는 아이템들의 벡터로 끌어당기고, 반대로 관련없는 아이템들의 벡터로부터는 밀어내어 보정하는 방식인 Rocchio's Method를 도입했다.

사용자가 입력한 문자열을 실제로 아이템 벡터에 대응하는 쿼리 유추 단계를 통해 사용자가 입력하려고 하는 단어를 예측하는 동시에 문자열을 벡터로 변환한 후 피드백 정보를 적용해 쿼리 벡터를 보정했다. 또한 사용자의 반응은 별도의 피드백을 위한 조작이 필요없도록 묵시적 피드백을 사용해 번거로움을 줄였다. 또한 주어진 데이터를 기반으로 이 구현체를 테스트할 수 있는 평가 모델을 제시하였다.

제안하는 검색체의 프로토타입은 Java 프로그래밍을 이용해 구현되었으므로 프로그래머는 다른 안드로이드 애플리케이션 개발 시에 임베디드 형식으로 탑재해 사용할 수 있어 해당 소프트웨어의 검색 기능을 향상시키는 역할을 제공할 것이다.

REFERENCES

[1] K. Erk, and S. Pado, "A structured vector space model for word meaning in context," in *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, Honolulu, pp. 897-906, 2008.

[2] Y. Jun. "Cosine similarity measures for intuitionistic fuzzy sets and their applications," *Mathematical and Computer Modelling*, vol. 53, no. 1, pp. 91-97, 2010.

[3] L. Muflikhah, and B. Baharudin. *Document clustering using concept space and cosine similarity measurement*, Institute of Electrical and Electronics Engineers, 2009.

[4] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613-620, Nov. 1975.

[5] C. Zhai. (2015, March). Vector Space Model : Basic Idea. Text Retrieval and Search Engines [Internet]. Available: https://www.coursera.org/course/textretrieval.

[6] S. Robertson, and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333-89, April 2009.

[7] C. Zhai. (2015, March). 1.8 TF Transformation : Text Retrieval and Search Engines [Internet]. Available: https://www.coursera.org/course/textretrieval.

[8] S. Buttcher, C. Clarke, and B. Lushman. "Term proximity scoring for ad-hoc retrieval on very large text collections," in *Proceedings of the 29th annual international ACM SIGIR*

conference. ACM, 2006.

[9] A. Signal, C. Buckley, and M. Mitra, “Pivoted document length normalisation,” in *Proceedings of ACM SIGIR*, New York: NY, pp. 21-29, 1996.

[10] T. L. Chung, R. W. Luk, K. F. Wong, K. L. Kwok, and D. L Lee, “Adapting pivoted document- length normalization for query size: Experiments in chinese and english,” *ACM Transactions on Asian Language Information Processing (TALIP)*, vol. 5, no. 3, pp. 245-263, 2006.

[11] Rocchio’s Algorithm [Internet]. Available: http://www.cs.cmu.edu/~wcohen/10-605/rocchio.pdf.

[12] G. Salton and C. Buckley, “Improving retrieval performance by relevance feedback,” *Journal of the American Society for Information Science* [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.92.3553.

[13] Shineware. Komoran ver 2.4 Java Korean Language Morphological Analyzer [Internet]. Available: http://shineware.tistory.com/entry/KOMORAN-ver-24.

[14] Kyro - Java Class Serialization Library [Internet]. Available: http://edoli.tistory.com/87.

[15] C. Zhai. (2015, March). 1.9 Doc Length Normalization: Text Retrieval and Search Engines [Internet]. Available: https://www.coursera.org/course/textretrieval.

[16] M. E. Voorhees, “Variations in relevance judgments and the measurement of retrieval effectiveness,” *Information processing & management*, vol. 36, no. 5, pp. 697-716, 1999.

[17] C. Zhai. (2015, March). 2.6 Evaluating Ranked Lists: Part 1. Text Retrieval and Search Engines [Internet]. Available: https://www.coursera.org/course/textretrieval.

강문중(Moonjoong Kang)

2016 창원대학교 정보통신공학과 공학사
현재 광주과학기술원 전자전기컴퓨터공학부석사과정
※ 관심분야 : 클라우드컴퓨팅, 안드로이드 응용

황민태(Mintae Hwang)

1990 부산대학교 컴퓨터공학과 공학사
1992 부산대학교 컴퓨터공학과 공학석사
1996 부산대학교 컴퓨터공학과 공학박사
1996 한국전자통신연구원 표준연구센터 선임연구원
1999 인제대학교 정보컴퓨터공학부 교수
현재 창원대학교 정보통신공학과 교수
※ 관심분야 : 안드로이드 응용, 매체접속제어 프로토콜, 사물인터넷 응용 기술