Metric space indexing을 이용한 욕설 필터

A Profanity Fillter by using Metric Space Indexing

윤태진
Yoon Taijin
부산대학교 컴퓨터공학과
ytj@pusan.ac.kr

ABSTRACT

Alignment를 이용한 욕설 필터에서 가장 문제되는 점은 많은 연산량으로 인한 연산속도의 문제이다. 금칙어 데이터 베이스에 등록된 모든 단어와 Alignment하여 가장 높은 유사도를 가지는 단어를 찾는 다면 가장 확실한 방법이나 수천에서 수만개의 단어 모두와 alignment를 수행한다면 대단히 많은 연산량을 필요로 하여 채팅시스템에 렉을 유발 시킬 수 있다. Binary search나 Hash 같은 방법으로는 이러한 유사단어 검색에 도움을 줄 수 없다. 우리는 이 문제를 해결하기 위하여 Edit Distance의 metric space를 만족하는 특성을 이용하여 R*tree의 다차원 데이터 구조를 이용한 Range search를 이용하여 alignment에 사용될 후보군을 선택하는 방법에 대해 서술하고자 한다.

KEYWORDS Indexed approximate string matching, Metric spaces, Metric indexes

1 서 론

욕설 필터링 시스템을 구성하기 위하여 우리는 한글을 자소로 분리하여 단어간 Alignment 값을 측정하여 유사도를 판단하는 방법을 사용하였다. 그러나 Alignment는 많은 연산량을 필요로 하는 작업이다. 일반적으로 게임 포탈 등에서는 수천에서 수만의 금칙어를 설정하여 필터링을 수행하는데 채팅창에 입력되는 모든 단어를 수만개의 단어와 Alignment 값을 측정하여 가장 가까운 값을 찾는 방식은 채팅에 대단히 긴 지연시간을 유발하여 사용자에게 불편함을 끼칠 수 있다.

그러므로 실재 채팅 시스템에 적용하기 위해서는 Alignment 값을 측정할 대상의 수를 줄일 필요성이 있다. 일반적인 사전 검색 방식으로는 이러한 Approximate String Matching에 대응할 수 없다. 기존에 가장 많이 쓰이는 방식인 Hash나 tree구조를 이용한 binary search로는 유사한 단어를 검색해낼 수 없기 때문이다.

우리는 이러한 문제점을 해결하기 위하여 유사한 발음과 형태의 글자를 통합하여 대표 글자로 치환한 단어를 index key로 삼는 방식을 이용하여 단어의 후보군을 추출하는 방법을 사용하였다. 그러나 최근의 인터넷 욕설은 변화 형태가 매우 다양하여 대표 글자로 치환하는 방식으로도 미쳐 대응할 수 없는 단어가 많다. 예를 들어 "씹새끼"를 "띕때끼"로 변형하여 사용하는 경우가 많은데 이 경우

"ㅆ"과 "ㅍ"를 유사한 발음으로 보고 하나의 글자로 통합하기에는 무리가 있다.

즉 기존의 string indexing에서 사용되는 1차원적인 방법으로는 비슷한 단어를 찾아내는데 어려움이 있다. 단어를 변형시키는 방법에는 삽입, 교환, 삭제 등이 순서에 관계없이 사용되기 때문에 Alphabet 순의 정렬 방식으로는 이러한 변화에 대응하기 어렵다. 그러나 다차원 Metric Space에 단어를 정렬 시킨다면 이러한 1차원 index로 인해 생겨나는 문제점을 해결할 수 있다.

본 보고서에서는 다차원 자료구조와 Range Query를 이용하여 욕설 필터링을 수행하는 방법에 대해서 서술하고 그 성능에 대한 실험과 본 시스템의 활용방법에 대하여 서술하도록 하겠다.

2 A metric index for approximate string matching

metric space는 triangle inequality를 만족하는 black-box obeject의 집합과 그들간의 함수이다.[1] 정규적으로 metric space는 (X, d)의 쌍으로 X는 object의 전체 집합이고 $d : X \times X \rightarrow R^+$은 거리 함수로 양수만 반환한다. 이 거리는 reflexivity($d(x, x) = 0$, strict postiveness($x \neq y \Rightarrow d(x, y) > 0$), symmetry (d(x, y) = d(y, x)) 그리고 triangle inequality($d(x, y) \leq d(x, z) + d(z, y)$)를 만족한다.

X의 부분집합 U는 우리가 찾고자 하는 object의 집합이다. metric space의 많은 query중에 우리가 주목하는 것은 range querys: 주어진 query $q \in X$, 허용 반지름 r, U는 q에서 r거리 안에 있는 원소의 집합이다. 정규적으로 query의 출력은 $(q, r)_d = \{u \in U, d(q, u) \leq r\}$ 이다.

metric space를 index하는 방법은 크게 2가지로 나뉜다. 첫째 pivot기반의 기술로 하나의 일반적인 아이디어로 구성된다. U에서 k개의 원소를 선택하여 $\{p_1, ..., p_k\}$ 각 원소 $u \in U$를 k차원의 점$(d(u, p_1), ..., d(u, p_k))$ 으로 인식하는 것이다. 이 정보를 가지고, triangle inequality를 이용해 어떤 원소 u를 어떤 pivot p_i에 대해서 $|d(q, p_i) - d(u, p_i)| > r$로 필터링할 수 있다. 그 경우 우리는 $d(u, q)$를 평가하지 않고도 $d(q, u) > r$ 이라는 것을 알 수 있다. 이 규칙을 이용해 필터링하지 못한 원소들은 q와 직접적으로 비교하게 된다. 더 많은 pivot을 사용할 수록 더 많은 원소가 제거 되지만 index에 더 많은 메모리를 필요로 하게 되고 좌표 계산에 더 많은 계산량을 요구하게 된다. 효율적인 range search method 로는 kd-tree, R-tree, X-tree등에 포함된 방법이 있다.

2.1 R*tree

우리는 다수의 pivot과 단어와의 Edit distance를 이용한 다차원 metric space를 저장할 자료 구조로 R*tree를 선택하였다. R*tree는 다차원 데이터 처리에서 가장 널리 사용되는 데이터 구조이다. R*Tree는 R-tree의 발전된 형태로 R-tree는 우리가 흔히 사용하는 B-tree를 다차원 정보 indexing에 맞게 개량한 것이다.[2] 기본적인 구조는 B-tree와 같이 삽입과 삭제가 일어나더라도 균형을 유지할

수 있는 방법을 취하고 있다.

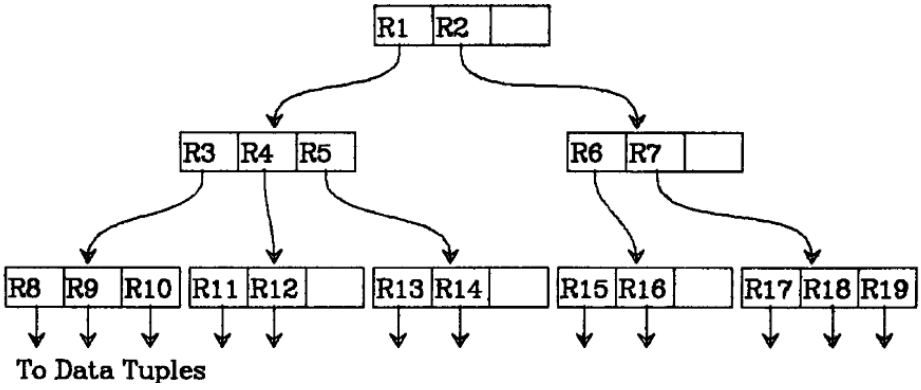

그림 1. R-tree의 구조

R-tree는 다차원의 space를 minimum bounding rectangles로 나눈다. 각 노드는 가변적인 수의 원소를 가진다. leaf노드 외에는 자식노드의 주소와 모든 원소를 포함하는 MBR의 정보를 가지고 있다. insertion과 deletion은 가장 가까운 MBR을 이용하여 이루어진다.

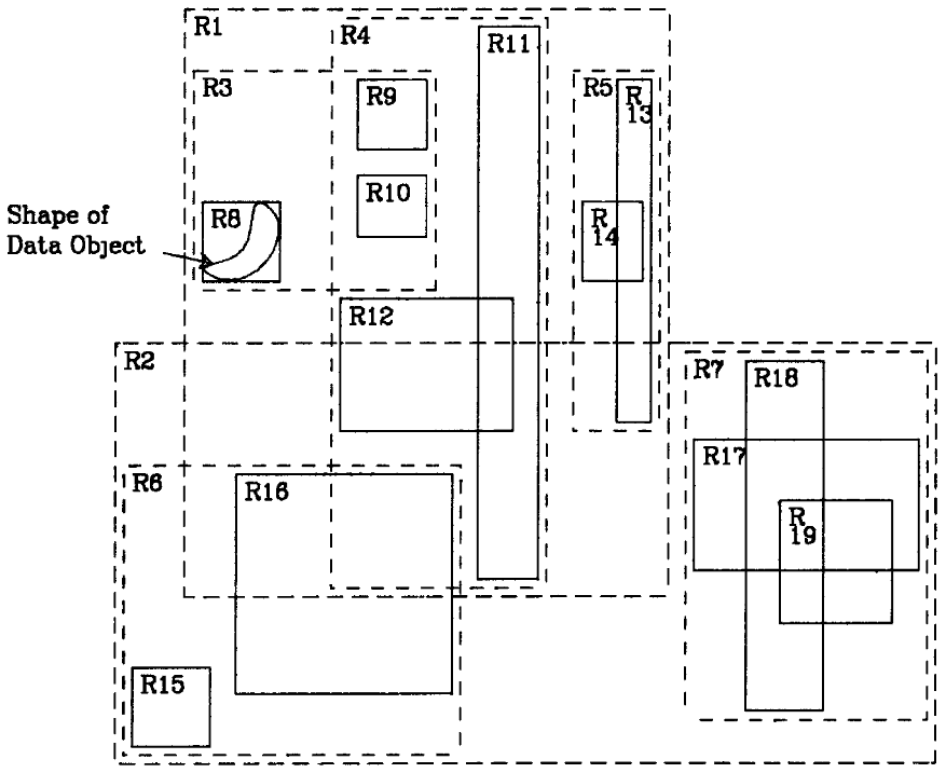

그림 2. R-tree의 MBRs

R*tree는 이 R-tree의 MBR의 빈 공간을 최소화하기 위하여 원소가 삽입 될때 MBR의 원소가 overflow를 일으킬 경우 MBR의 중심에서 가장 먼 원소를 reinsertion하여 공간효율을 높이는 방법이다. 공간효율이 높아지면 overlab이 줄어 들어 자료구조의 성능이 높아지게 된다.[3]

3 한글을 위한 **Approximate String Matching**

본 시스템의 목적은 입력단어와 유사한 단어의 집합을 기존의 금칙어 리스트에서 빠르게 찾아내는 것이다. 이미 이 Approximate String Indexing 방법은 앞에서 기술한 것처럼 많은 연구가 수행되어 왔다. 그러나 한글의 경우 자음과 모음 phoneme의 조합으로 하나의 글자가 이루어지는 방식이기 때문에 영어와는 다른 접근 방식이 필요하다.

3.1 한글 **Edit Distance**

한글 단어간의 Edit Distance를 계산하고자 할때 글자 단위로 처리할 경우 Alphabet과는 달리 문제가 있다. 예를 들어 "강남"과 "전남", "감낭"의 세 단어를 비교해 보자. "강남"과 "전남"보다 "강남"과 "감낭"이 발음상으로 더 유사해 보이나 Edit Distance는 전자가 1이고 후자가 2로 후자의 경우가 더 먼 거리로 측정된다.

이 문제를 해결하기 위해서는 한글 단어를 phoneme단위로 분리 시켜 alignment를 수행할 필요가 있다. 위의 예를 자모로 분리시키면 "ㄱ ㅏㅇㄴ ㅏㅁ"과 "ㅈ ㅓㄴㄴ ㅏㅁ", "ㄱ ㅏㅁㄴ ㅏㅇ"으로 전자의 Edit Distance는 3, 후자는 2로 보다 발음이 가까운 단어 끼리 가까운 거리로 측정되어 더욱 합리적인 결과를 얻을 수 있다.

표 1. 한글 Edit Distance의 두가지 측정법

글자 단위		자소 단위	
비교 대상	Edit Distance	비교 대상	Edit Distance
강남, 전남	1	ㄱ ㅏㅇㄴ ㅏㅁ, ㅈ ㅓㄴㄴ ㅏㅁ	3
강남, 감낭	2	ㄱ ㅏㅇㄴ ㅏㅁ, ㄱ ㅏㅁㄴ ㅏㅇ	2
전남, 감낭	2	ㅈ ㅓㄴㄴ ㅏㅁ, ㄱ ㅏㅁㄴ ㅏㅇ	4

한글 비속어의 변형은 유사한 발음과 형태의 글자로 치환하여 이루어지는 경우가 많다. "개구라"라는 비속어를 금칙어로 입력 한다면 "캐구라", 혹은 "ㄱH구라" 등 뜻을 전달할 수 있는 한도 내에서 여러가지 변형이 이루어진다. 그러므로 비속어를 indexing하기 위해서는 pivot에 다양한 글자가 입력되어야 하는데 이 경우 오히려 indexing 효율을 떨어 뜨리게 된다. 그러므로 효과적인 indexing

을 위해서는 사용되는 문자의 수를 줄여 줄 필요가 있다.

표 2. 효율적인 indexing을 위한 대표형 변환 규칙

초성		중성		종성	
원래 자소	바뀐 자소	원래 자소	바뀐 자소	원래 자소	바뀐 자소
ㄱ,ㄲ,ㅋ,g	ㄱ	ㅏ, ㅑ,a	ㅏ	ㄱ,ㄲ,ㅋ	ㄱ
ㄷ,ㄸ,ㅌ,d,t	ㄷ	ㅐ, ㅒ, ㅔ, ㅖ,ㅙ,ㅚ,ㅝ,ㅞ,H,e	ㅐ	ㄷ,ㅌ,ㅅ,ㅈ,ㅉ,ㅊ	ㄷ
ㅂ,ㅃ,ㅍ,b,v,p,f	ㅂ	ㅓ, ㅕ	ㅓ	ㅂ,ㅍ	ㅂ
ㅅ,ㅆ,s	ㅅ	ㅗ,ㅛ,o	ㅗ		
ㅈ,ㅉ,ㅊ,j,z	ㅈ	ㅜ,ㅠ,u	ㅜ		
		ㅟ,ㅢ, ㅣ,ㅡ,i	ㅣ		

위의 표 2는 자소의 대표형 변환을 위한 규칙이다. indexing의 좌표를 단순화 할 수 있어 효과적인 searching이 가능해진다. 복잡한 변환단어가 입력되더라도 효율적으로 Edit Distance를 측정하여 알맞은 좌표를 생성할 수 있는 것이다.

3.2 pivot의 선택 알고리즘

pivot의 선택 방법은 본 시스템의 성능을 크게 좌우하는 요소이다. 가능한 많은 단어를 pivot으로 사용할 수록 indexing의 정확도는 높아지지만 검색할때 입력단어의 좌표를 계산하기 위한 Edit Distance 계산 시간이 증가하여 시간 효율이 감소하게 된다. 그러므로 정확도와 검색 시간 간의 타협점을 찾을 필요가 있다.

이 pivot을 선택할 때에 단순히 랜덤으로 단어를 선택하여 R*tree를 구성한다면 높은 성능을 기대하기는 어렵다. 위 섹션에서 언급한 대표형에 해당하는 가능한 모든 자소가 포함되어야 하기 때문이다. 일부의 자소만 pivot에 포함될 경우 특정 단어가 일정 지역에 집중되는 현상이 일어날 수 있기 때문이다. 만약에 "ㄱ, ㅎ"이 pivot에 포함되어 있지 않다면 "개장"과 "해장"은 동일한 좌표에 위치하게 될 것이다.

pivot을 선택하는 방법에는 kMean 알고리즘을 활용할 경우 만족할 만한 성능을 얻을 수 있을 것이다. 그러나 kMean알고리즘을 활용하기 위해서는 모든 금칙어 간의 Edit Distance를 측정하여야 하므로 금칙어의 숫자가 많아질 경우 지나치게 많은 연산시간을 필요로 한다. 만약에 금칙어의 DB가 자주 바뀌는 시스템이라면 크게 문제가 된다.

그래서 금칙어의 숫자가 많고 변경이 자주 이루어지는 시스템에서는 임의의 단어를 선정한 후 그 단어와 Edit Distance가 먼 단어들을 선택하는 방법을 사용한다. 이미 pivot으로 단어들과 Edit Distance가 기준치 이상인 단어를 pivot으로 추가하여 다양한 자소를 pivot으로 선택하는 것이다. 실제 사용되는 시스템에서는 단어의 사용빈도를 측정하여 자주 사용되는 단어를 우선적으로 pivot으로 선택한다면 전체적인 시스템의 성능을 높일 수 있을것이다.

4 결론

본 보고서에서는 연구 중인 비속어 필터링 시스템에 approximate string matching를 위한 metric index를 적용하는 방법에 대해 설명하였다. 기존 비속어 필터링 시스템의 문제점인 합리적인 후보군 선정 문제를 상당히 해결할 수 있었다. 예를 들어 "내장"을 입력하더라도 "개장"을 검색해낼 수 있다.

본 시스템의 indexing방법은 비속어 필터링 뿐만 아니라 사전의 단어 검색 혹은 네비게이션 시스템의 지명 검색에도 유용하게 활용될 수 있을 것이다. 기억이 확실하지 않아 애매한 단어가 입력되더라도 유사한 단어를 빠르게 검색하여 출력할 수 있기 때문이다. 이 경우 pivot을 단어 내에서 검색하기 보다 범용적으로 사용할 수 있는 한글 pivot을 개발한다면 더욱 효과적인 indexing이 가능해질 것이다.

참고 문헌

1. Gonzalo Navarroa and Edgar Chávez, "A metric index for approximate string matching," *Theoretical Computer Science*, vol. 352, no. 1, pp. 266–279, 2006.
2. Antonin Guttman, "R-trees: a dynamic index structure for spatial searching," pp. 599–609, 1988.
3. Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger, "The r*-tree: an efficient and robust access method for points and rectangles," in *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 1990, pp. 322–331, ACM.