효율적인 kNN 알고리즘

이 재 문†

요 약

본 논문은 문서분류 방법인 kNN의 실행속도를 개선하는 알고리즘을 제안한다. 제안된 알고리즘은 기존의 kNN이 사용하는 <용어, 가중치> 쌍의 목록 대신, <문서, 가중치> 쌍의 목록을 사용하여 유사성 계산을 빠르게 함으로써 실행속도를 개선하는 것이다. <문서, 가중치>의 목록은 문서분류의 학습단계에서 <용어, 가중치>의 목록을 행렬 전치함으로써 구한다. 본 논문에서는 제안된 알고리즘을 시간복잡도 측면에서 분석하고 기존의 kNN과 비교 하였으며, 로이터-21578 데이터를 사용하여 실험적으로 성능을 비교 하였다. 실험결과, 본 논문에서 제안한 알고리즘이 기존의 kNN보다 실행속도측면에서 약 90% 정도의 우수함을 알 수 있었다.

An Efficient kNN Algorithm

Jae Moon Lee†

ABSTRACT

This paper proposes an algorithm to enhance the execution time of kNN in the document classification. The proposed algorithm is to enhance the execution time by minimizing the computing cost of the similarity between two documents by using the list of <document, weight> pairs, while the conventional kNN uses the list of <term, weight> pairs. The list of <document, weight> pairs can be obtained by applying the matrix transposition to the list of <term, weight> pairs at the training phase of the document classification. This paper analyzed the proposed algorithm in the time complexity and compared it with the conventional kNN. And it compared the proposed algorithm with the conventional kNN by using reuters-21578 data experimentally. The experimental results show that the proposed algorithm outperforms kNN about 90% in terms of the execution time.

키워드 : 문서 분류(Document Classification), 학습 문서(Training Document), 시험 문서(Testing Document), kNN, NaiveBayes, SVM, 문서 벡터(Document Vector)

1. 서 론

최근 웹 문서 등 전자 문서의 급증으로 이들을 관리하는 정보관리시스템 분야에서 기계 학습에 의한 문서분류 연구가 활발히 진행되고 있다[1-4, 9, 11, 12]. 문서분류란 미리 정해진 분류의 집합이 있을 때 특정 문서가 어느 분류에 속하는지를 판단하는 것을 말한다. 문서분류에 대한 연구의 방향은 크게 두 가지로 나뉜다. 하나는 분류의 정확도를 높이는 기술에 관한 연구이고[1-2, 4, 6], 다른 하나는 문서분류의 실행속도를 개선하는 기술에 관한 연구이다[3, 13]. 문서분류에 대한 연구는 대부분 전자에 집중되어 왔다. 주요 연구의 결과로는 k-Nearest Neighbor(kNN)[1], NaiveBayes[2], Support Vector Machine (SVM)[3] 등이 있는데, kNN은 가장 간단한 기법 중의 하나이면서도 비교적 높은 정확도를 보이지만 실행속도가 매우 느린 단점을 가지고 있다. 본 논문은 kNN의 정확도에 대한 어떠한 손실 없이 느린 실행속도를 개선하는 방법을 제안하는 것이다.

일반적으로 기계학습 기법에 의한 문서분류는 학습단계와 분류단계로 나뉜다[10]. 학습단계는 전문가에 의하여 잘 분류된 분류집합 및 학습문서집합을 사용하여 학습하는 단계이고, 분류단계는 학습단계에서 가공된 데이터를 기반으로 임의의 문서를 특정 분류로 분류하는 단계이다. 일반적인 문서분류 시스템은 특정 분류집합 및 학습문서집합에 대하여 오직 한번만 학습단계를 실행하고, 분류하고자 하는 문서가 발생할 때마다 분류단계를 반복적으로 실행한다. 이것은 학습단계보다는 분류단계의 효율성이 더욱 중요하다는 것을 의미한다. 특히, 최근 웹기반 뉴스 분류, 메일 자동 분류 시스템 등에서는 분류단계의 실행을 실시간으로 요구하고 있다[10, 12]. 따라서 문서분류의 실행속도를 개선하는 기술에 관한 연구는 대부분 분류단계의 개선에 초점을 맞추고 있다[3, 13]. 본 연구도 kNN 학습단계에서는 오히려 약간의 오버헤드가 주어지나, 분류단계의 효율성을 크게 개선하는 알고리즘을 제안하는 것이다.

※ 본 논문은 2004학년도 한성대학교 교내연구비 지원과제임.
† 정 회 원 : 한성대학교 컴퓨터공학부 교수
논문접수 : 2004년 7월 16일, 심사완료 : 2004년 11월 2일

제안하는 알고리즘의 핵심 아이디어는 kNN에서 유사성 계산 시 탐색 비용을 줄여 실행속도를 개선하는 것이다. 이를 위하여 기존의 kNN이 문서별 <용어, 가중치> 쌍의 집합으로 구성된 학습문서집합을 사용하는 반면, 제안하는 방법에서는 용어별 <문서, 가중치> 쌍의 집합으로 구성된 학습용어집합을 사용한다. 용어별 <문서, 가중치> 쌍의 집합은 문서별 <용어, 가중치> 쌍의 집합을 행렬전치(Matrix Transposition)[14] 함으로써 쉽게 구할 수 있다. 따라서 본 논문에서는 특별히 학습단계의 알고리즘은 제시하지 않고, 기존의 kNN에서 학습단계 알고리즘에 행렬전치 기능이 추가되어 학습단계의 결과물이 용어별 <문서, 가중치> 쌍의 집합이라고 가정하고, 분류단계의 효율성을 개선하는 알고리즘만 제시한다.

2장은 kNN에 대한 소개와 실행 속도에 대한 분석을 설명하며, 3장에서는 새로운 kNN을 설명하고 성능을 분석한다. 4장에서는 기존의 kNN과 제안하는 알고리즘 사이의 성능 비교를 하며, 5장에서 결론을 논한다.

2. 기존의 kNN 및 성능

2.1 기존의 kNN 알고리즘

kNN, NaiveBayes, SVM등 많은 문서분류 방법들이 연구되어 왔다[1-3, 9]. 이들은 각각의 특성이 있어, 그 특성이 잘 반영되는 환경에서 최고의 성능을 발휘한다. kNN 알고리즘의 특성은 방법이 간단하다는 것과 학습단계에서 최소한의 처리 작업을 한다는 것이다. 문서분류 관점에서 kNN의 정확도는 매우 우수하나, 분류단계에서 실행 속도가 매우 느린 단점을 갖고 있다.

kNN에서 학습단계는 학습문서집합을 입력받아 문서별로 문서에 나타나는 모든 용어들을 찾아내고 이러한 용어들에 대하여 가중치를 계산하여, 문서별 <용어, 가중치> 쌍의 목록을 생성한다. 따라서 kNN의 학습단계는 각 문서에 대하여 단순히 <용어, 가중치>쌍의 목록만 생성하므로 NaiveBayes, SVM 등에 비하여 매우 단순하다. kNN의 분류단계는 분류집합, 학습단계에서 생성된 학습문서집합(문서별 <용어, 가중치> 쌍의 목록), 분류되어야하는 시험문서 및 이웃상수 k를 입력받는다[1, 10]. 이러한 입력을 이용하여 학습문서집합의 각 문서에 대하여 시험문서와의 유사성을 계산하여 유사성이 가장 큰 k개의 학습문서를 선택하고, 선택된 학습문서가 속하는 분류들에 대하여 등급을 정하여 이들을 출력한다[10]. (그림 1)은 [1-2, 10]에서 제시한 내용을 기초로 한 기존의 kNN에 대한 기술이다.

(그림 1)에서 normalize_sort 함수는 시험문서 벡터를 입력받아 벡터의 크기가 1이 되도록 정규화를 하며, 또한 용어 순서로 목록을 정렬하는 함수이다. 스텝 2~3은 시험문서 d에 대하여 모든 학습문서의 유사성을 계산하는 것으로 각 유사성의 계산은 compute_similarity에서 두 문서 벡터의 내적을 계산하는 것이다.

kNN(입력 : 분류집합 C, 학습문서집합 D, 시험문서 d, 이웃상수 k, 출력 : 등급화된 c)
스텝 1 : d = normalize_sort(d) ;
스텝 2 : for each d_i in D {
스텝 2.1 : $s[d_i]$ = compute_similarity(d, d_i) ;
스텝 3 : }
스텝 4 : D_k = find_k_documents($s[\,]$, k) ;
스텝 5 : $c[c_i]$= 0 for each c_i in C ;
스텝 6 : for each d_j in D_k {
스텝 6.1 : $c[c_{d_j}]$++ for each c_{d_j} in { c_{d_j} | c_{d_j} in C and d_j is classified to c_{d_j} }
스텝 7 : }
스텝 8 : return $c[\,]$

(그림 1) kNN 알고리즘

벡터의 구성이 <용어, 가중치>의 목록으로 구성되어 있고, 이들이 용어 순서로 정렬되어 있기 때문에 합병(merge) 방법으로 하는 것이 가장 효율적이다. 이러한 과정은 (그림 2)의 compute_similarity에서 보이고 있다.

compute_similarity(입력 : 시험문서 d_c, 학습문서 d_t, 출력 : 유사성 w)
스텝 1 : $w = 0$;
스텝 2 : $< t_c,\ w_c >$ = shift d_c ; $< t_t,\ w_t >$ = shift d_t ;
스텝 3 : while(t_c != { ϕ } and t_t != { ϕ }) {
스텝 3.1 : if($t_c == t_t$) w += $w_c * w_t$;
스텝 3.2 : else if($t_c < t_t$) $< t_c,\ w_c >$ = shift d_c ;
스텝 3.3 : else $< t_t,\ w_t >$ = shift d_t ;
스텝 4 : }
스텝 5 : return w ;

(그림 2) kNN에서 유사성 계산

〈표 1〉 분류 집합 C

분 류	c_1	c_2	c_3	c_4
소속문서	d_1, d_5	d_1, d_2, d_3	d_2, d_4	d_3, d_4, d_5

〈표 2〉 학습문서 집합 D 및 시험문서 d

문 서	벡터(<용어, 가중치>의 쌍)
d_1	{<c, 30><d, 10><e, 30><h, 30>}
d_2	{<a, 20><e, 20>}
d_3	{<a, 10><b, 10><c, 20><f, 30><h, 40>}
d_4	{<a, 10><b, 10><c, 10>}
d_5	{<e, 10><g, 20>}
d	{<a, 20><c, 10><f, 20><h, 10>}

〈표 3〉 문서별 유사성 $s[d_i]$

문 서	a	b	c	d	e	f	g	h	계
$s[d_1]$			10*30					10*30	600
$s[d_2]$		20*20							400
$s[d_3]$	20*10		10*20			20*30		10*40	1400
$s[d_4]$	20*10		10*10						300
$s[d_5]$									0

kNN의 동작을 설명하기 위하여 <표 1>, <표 2>에서 주어진 데이터를 사용하여 예를 들어 보기로 한다. <표 1>은 분류집합을 의미하는 것으로 분류별 이들이 속하는 학습문서의 정보를 포함하고 있다. <표 2>는 학습문서집합과 시험문서를 표시하는 것으로 문서별 <용어, 가중치>의 쌍으로 구성되어 있고, 이들은 용어의 순으로 정렬되어 있다. 예를 들어 문서 d에서 {<a, 20><c, 10><f, 20><h, 10>}의 의미는 문서 d는 용어 a, c, f, h로 구성되어 있고, 각 용어의 가중치는 20, 10, 20, 10과 같다는 것이다. 여기서 k는 2라고 하고, 스텝 1에서와 같이 문서 벡터의 크기를 일정하게 하는 정규화를 하면 소수점이 발생하여 설명이 복잡해지므로 정규화는 생략하기로 한다. <표 3>은 스텝 2~3에서 구해진 유사성을 표시하고 있고, <표 4>는 <표 3>으로부터 구해진 $D_k=\{d_1, d_3\}$을 이용하여 스텝 6~7에서 분류별 계산된 가중치이다. <표 4>로부터 시험문서 d는 분류 c_2에 가장 가깝다는 것을 알 수 있다.

〈표 4〉 $D_k=\{d_1, d_3\}$에 따른 분류의 가중치

c_i	c_1	c_2	c_3	c_4
$c[c_i]$	1	2	0	1

2.2 기존의 kNN 시간 복잡도

앞에서 설명한 kNN 알고리즘의 시간 복잡도를 계산해 보기로 한다. 시간 복잡도를 간략히 표현하기 위하여 임의의 집합 P에서 그 요소의 수를 $|P|$로 표현하고, a_1, a_2, …, a_n은 임의의 상수라고 한다. 스텝 1은 시험문서를 정규화하고 용어 순서로 정렬하는 것이므로 $a_1*|d|*\log(|d|)$의 비용을 가진다. 스텝 2~3은 $|D|$번 반복하며, 스텝 2.1은 합병 비용이므로 $a_2*(|d|+|d_i|)$이다. 따라서 스텝 2~3의 비용은 $\sum_{i=1}^{|D|} a_2*(|d|+|d_i|)$가 된다. $|\overline{d}|$를 학습문서의 평균 크기라고 하면 $a_2*(|d|+|\overline{d}|)*|D|$가 된다. 스텝 4는 유사성이 가장 큰 k개의 문서를 찾는 것이므로 $a_3*k*|D|$가 된다. 스텝 5는 $a_4*|C|$이며, 스텝 6은 $a_5*k*|D|$가 된다. 모든 스텝의 비용을 합하면 다음과 같이 표현할 수 있다.

$$T_{kNN} = a_1*|d|*\log(|d|)+ a_2*(|d|+|\overline{d}|)*|D|+ a_3*k*|D|+ a_4*|C|+ a_5*k*|D| \quad (1)$$

일반적으로 $|C|<<|D|$이며, $|d|<<|D|$이다. 이러한 관계를 적용하면 식 (1)은 다음과 같이 표현될 수 있다

$$T_{kNN} = O((|d|+|\overline{d}|+k)*|D|) \quad (2)$$

식 (2)에서 나타난 항목은 상기 알고리즘에서 유사성을 계산하는 스텝 2~3과 유사성이 큰 k개의 문서를 찾는 스텝 4이다. 이것의 의미는 kNN 알고리즘의 대부분 비용은 유사성의 계산과 k개의 유사성이 가장 큰 문서를 찾는데 있다는 것이다. 이것은 역으로 kNN 알고리즘의 실행속도를 개선하기 위해서는 이러한 상기 두 요소를 개선하여야 한다는 것을 의미한다. 문서분류에서 대부분 k가 20 이하인 점을 감안한다면 상기 두 요소 중 특히, 유사성을 계산하는 과정을 효율적으로 개선할 필요가 있다. [13]에서는 이러한 점에 착안하여 유사성을 계산할 때 휴리스틱을 사용하여 성능을 향상 시켰다.

3. 효율적인 kNN 알고리즘

3.1 효율적인 kNN 알고리즘

kNN의 분류단계가 느린 이유는 유사성을 계산하기 위하여 시험문서에 포함된 용어와 관계없이 학습문서에 포함된 모든 용어들을 액세스하여야 한다는 것이다. 즉, 시험문서에 포함되지 않은 용어도 액세스 되어 비교되어야 하는 것이다. 예를 들어, <표 2>에서 시험문서에 포함된 용어는 {a, c, f, h}이다. 따라서 학습문서들에 포함된 {b, d, e, g} 용어들은 유사성의 계산에 아무런 기여도 하지 못하면서 유사성의 계산 시 액세스되고 또한 비교된다. 극한 예로는 시험문서 d_5는 {a, c, f, h}에 속한 어떠한 용어도 포함하지 않음에도 유사성은 계산되어야 한다. 이것은 kNN 알고리즘에 아주 큰 오버헤드이다.

본 본문은 이러한 오버헤드를 제거하는 알고리즘을 제안한다. 기존의 kNN에서 학습문서집합을 문서별 <용어, 가중치>의 쌍으로 표현하였다. 따라서 하나의 문서에 포함된 모든 용어를 검색하여야 유사성을 계산할 수 있다. 본 논문에서는 이러한 학습문서집합을 용어별 <문서, 가중치>의 쌍으로 표현함으로써 유사성의 계산을 효율적으로 수행하도록 한다. 본 논문에서는 이를 학습용어집합이라 표현한다. 용어별 <문서, 가중치> 쌍의 집합은 문서별 <용어, 가중치> 쌍의 집합을 행렬전치 함으로써 쉽게 구하여 진다. 예를 들어 <표 5>는 <표 2>에 대한 학습 데이터에 대한 정보를 용어별 <문서, 가중치>의 쌍으로 표현한 것이다. <표 2>로부터 <표 5>를 구하는 과정은 먼저 <표 2>에 나타나는 모든 용어들(a, b, c, d, e, f, g, h)에 대하여 각각 공집합으로 된 <문서, 가중치> 쌍의 목록을 만든다. 다음, 문서 d_1의 {<c, 30><d, 10><e, 30><h, 30>} 집합에서 각 용어 c, d, e, h에 대하여 각각 $<d_1, 30>$, $<d_1, 10>$, $<d_1, 30>$, $<d_1, 30>$의 쌍을 만들고 이들을 각각 c, d, e, h의 <문서, 가중치> 쌍의 목록에 삽입한다. 그러면 <표 5>에서 음영을 가진 요소들만 나타나게 된다. 이러한 과정을 d_2, d_3, d_4, d_5에 대하여 반복하면 <표 5>와 같은 결과를 얻는다. 이것은 문서별 <용어, 가중치> 쌍의 목록이 "문서×용어"로 구성된 희소행렬에 대한 표현[14]이라면, 용어별 <문서, 가중치> 쌍의 목록은 "용어×문서"로 구성된 희소 행렬에 대한 표현이다. 따라서 문서별 <용어, 가중치> 쌍의 목록으로부터 용어별 <문서, 가중치> 쌍의 목록을 얻는 것은 단

순한 행렬전치이라고 할 수 있다. 본 논문에서는 용어별 <문서, 가중치> 쌍의 목록을 구하는 과정은 학습단계에서 실행한다고 가정한다. 이것은 일단 용어별 <문서, 가중치>의 목록이 만들어지면, 분류 단계에서는 절대로 변하지 않기 때문에 학습단계에서 실행되어야 하는 것은 당연하다. 따라서 제안하는 방법을 사용하기 위해서는 기존의 학습단계에 상기와 같은 행렬전치 기능이 더해져야 한다.

〈표 5〉 학습문서에 대한 용어별 〈문서, 가중치〉 쌍의 목록

용어	벡터(<문서, 가중치>의 쌍)
a	{< d_2, 20>< d_3, 10>< d_4, 10>}
b	{< d_3, 10>< d_4, 10>}
c	{< d_1, 30>< d_3, 20>< d_4, 10>}
d	{< d_1, 10>}
e	{< d_1, 30>< d_2, 20>< d_5, 10>}
f	{< d_3, 30>}
g	{< d_5, 20>}
h	{< d_1, 30>< d_3, 40>}

<표 5>로부터 시험문서 d와 문서별 유사성을 구하는 것은 단순하다. 시험문서 d에 포함된 각 용어에 대하여 해당 <문서, 가중치> 쌍의 목록을 <표 5>로부터 찾는다. 각 <문서, 가중치> 쌍에 대하여 시험문서의 해당 용어 가중치를 곱하여 이를 해당 문서의 유사성에 축적함으로써 문서별 유사성을 구한다.

〈표 6〉 〈문서, 가중치〉 쌍의 목록을 이용한 유사성 $s[d_i]$ 계산

용어	$s[d_1]$	$s[d_2]$	$s[d_3]$	$s[d_4]$	$s[d_5]$
a		20*20	20*10	20*10	
c	10*30		10*20	10*10	
f			20*30		
h	10*30		10*40		
합계	600	400	1400	300	0

<표 6>은 <표 5>와 시험문서 d = {<a, 20><c, 10><f, 20><h, 10>}와의 유사성을 구하는 과정을 내타내고 있다. 먼저 시험문서 d에서 <a, 20>을 추출하고 <표 5>에서 a에 해당하는 {< d_2, 20>< d_3, 10>< d_4, 10>} 목록을 추출한 후 a의 가중치 20을 문서 d_2, d_3, d_4의 가중치 20, 10, 10에 곱한 후 $s[d_2]$, $s[d_3]$, $s[d_4]$에 저장함으로써 용어 a에 대한 문서별 유사성을 계산하는 것이다. 용어 c, f, h에 대해서도 같은 방법으로 구하면 된다. 이 과정에서 중요한 것은 용어 b, d, e, g에 대한 목록은 전혀 액세스되지 않는다는 것이다. <표 2>와 <표 5>를 고려할 때 두 벡터의 그룹에서 요소의 수는 동일하다. 하지만 기존의 kNN 방법은 유사성을 계산하기 위하여 <표 2>의 모든 요소를 액세스 하였으나, 제안하는 방법은 <표 5>에서 부분적인 요소들만 액세스 하면 된다는 것이다. <그림 3>과 <그림 4>는 상기 예제를 기반으로 제안하는 알고리즘이다. 본 논문에서를 이를 EkNN이라고 하기로 한다.

EkNN(입력 : 분류집합 C, 학습용어집합 T, 시험문서 d, 이웃상수 k, 출력 : 등급화된 c)
스텝 1 : d = normalize_sort(d) ;
스텝 2 : for each < t_c, w_c> in d {
스텝 2.1 : t_i = find_list(t_c, T) ;
스텝 2.2 : compute_similarity(w_c, t_i, s) ;
스텝 3 : }
스텝 4 : D_k = find_k_documents($s[\,]$, k) ;
스텝 5 : $c[\,c_i]$ = 0 for each c_i in C ;
스텝 6 : for each d_j in D_k {
스텝 6.1 : $c[\,c_{d_j}]$++ for each c_{d_j} in { c_{d_j} | c_{d_j} in C and d_j is classified to c_{d_j}}
스텝 7 : }
스텝 8 : return $c[\,]$

〈그림 3〉 EkNN에서 유사성 계산

compute_similarity(입력 : 가중치 w, 학습용어 t_d, 축적가중치 $s[\,]$)
스텝 1 : for each < d_i, w_i> in t_d
스텝 1.1 : $s[\,d_i]$ += w* w_i ;
스텝 2 : }

〈그림 4〉 EkNN에서 유사성 계산

kNN과 EkNN의 차이는 단지 알고리즘의 입력과 스텝 2~3에만 나타난다. kNN에서는 학습문서집합 D가 입력인 반면, EkNN에서는 학습용어집합 T가 입력이다. EkNN에서 스텝 2~3은 시험문서 d에 있는 각 < t_c, w_c>에 대하여 T에서 해당 용어 w_c의 <문서, 가중치>의 목록인 t_i를 찾아서 compute_similarity를 사용하여 유사성을 계산하는 과정이다. find_list는 학습용어집합 T에서 주어진 용어에 대한 <문서, 가중치> 쌍의 목록을 찾는 함수 이며, <그림 4>에서 compute_similarity는 유사성을 계산하는 함수이다.

3.2 EkNN의 시간 복잡도

시간 복잡도에 사용된 용어들의 표현은 2.2절에서 사용한 것과 동일하다. 스텝 1과 스텝 4~8은 kNN과 동일하므로 앞장에서 설명한 것을 그대로 도입한다. 스텝 2~3은 $|d|$번 반복되며, 스텝 2.1은 T에서 t_c에 해당하는 목록을 찾는 것이므로 T가 용어의 순으로 정렬되어 이진탐색을 한다고 하면 a_{21} * log $|T|$가 된다. 스텝 2.2에서 compute_similarity는 t_i의 모든 요소에 대하여 계산을 하는 것이므로 a_{22}*$|t_i|$의 비용을 갖게 된다. 따라서 스텝 2~3의 전체 비용은 a_{21}*$|d|$* log $|T|$ + $\sum_{i=1}^{|d|} a_{22}$*$|t_i|$가 된다. $|\overline{t}|$가 T에서의 목록들의 평균 요소의 수라고 하면 스텝 2~3의 비용은 (a_{21}* log $|T|$+ a_{22}*$|\overline{t}|$)*$|d|$로 표현된다. 따라서 EkNN의 전체 비용은 다음과 같다.

$$T_{EkNN} = a_1*|d|*\log(|d|)+(a_{21}*\log|T|+a_{22}*|\overline{t}|)*|d|+ a_3*k*|D|+a_4*|C|+a_5*k*|D| \quad (3)$$

일반적으로 $|C|<<|D|$이며, $|d|<<|D|$이고, $\log|T|<<|\overline{t}|$이다. 이러한 관계를 적용하면 식 (3)은 다음과 같이 표현될 수 있다.

$$T_{EkNN} = O(|\overline{t}|*|d|+k*|D|) \quad (4)$$

4. 성능 비교

4.1 시간 복잡도 비교

이 절에서는 제안된 EkNN과 기존의 kNN을 시간 복잡도 관점에서 비교하여 보기로 한다. 계산을 간단히 하기 위하여 식 (1)과 식 (3)을 사용하여 비교하기로 하고, 두 수식에 사용된 모든 상수 값은 동일하게 1라고 가정한다. 다음 식은 두 방법의 시간 복잡도의 차이다.

$$T_{kNN} - T_{EkNN} = (|d|+|\overline{d}|)*|D|-(\log|T|+|\overline{t}|)*|d| \quad (5)$$

식 (5)에서 $|\overline{d}|$는 학습문서에 나타나는 용어의 평균수이고, $|d|$는 시험문서에 나타나는 용어의 수이다. 일반적으로 학습문서와 시험문서가 동일 종류(예를 들어 로이터통신의 기사 등)의 문서이므로 $|\overline{d}|$과 $|d|$는 같다고 가정할 수 있다. 또한 $|\overline{d}|*|D|$는 학습문서집합에 나타난 모든 요소의 수이므로 학습용어집합의 요소수인 $|\overline{t}|*|T|$와 같다. 따라서 $|\overline{t}|=|\overline{d}|*|D|/|T|$가 성립한다. 식 (5)에 $|d|=|\overline{d}|$, $|\overline{t}|=|\overline{d}|*|D|/|T|$를 적용하면 다음의 부등식을 얻을 수 있다.

$$T_{kNN} - T_{EkNN} = |d|*(|D|-\log|T|)+|d|*(|D|-|D|/|T|)>= |d|*(|D|-\log|T|) \quad (6)$$

식 (6)은 $|D|>\log|T|$의 조건을 만족한다면, $T_{kNN}>T_{EkNN}$이 되어 제안된 알고리즘이 우수하다 것을 의미한다.

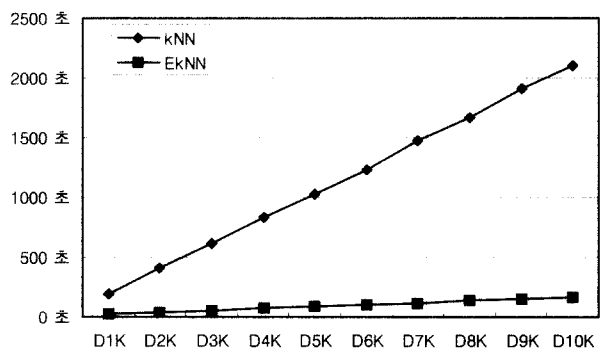
4.2 로이터-21578 데이터를 사용한 실험적 비교

제안된 알고리즘과 기존의 알고리즘을 문서분류의 실험 데이터로 잘 알려진 로이터-21578 데이터를 사용하여 실험적 비교를 한다[8, 11]. 이를 위하여 kNN, EkNN을 [10]에서 개발한 AI::Categorizer 프레임웍을 사용하여 구현하였다. AI::Categorizer는 객체 지향 문서분류 프레임웍으로 문서분류에 필요한 다양한 기능 및 일반적으로 잘 알려진 문서분류 알고리즘의 구현하여 제공한다. 이 프레임웍을 사용함으로써 문서의 토큰화, 벡터 모델, 차원 축소등 부수적인 구현을 생략할 수 있었으며, 실험을 위해서 AI::Categorizer::Learner 객체로부터 상속된 kNN, EkNN만 구현하였다. EkNN에 대한 학습단계는 단순히 kNN의 학습단계에 행렬전치 기능을 추가함으로써 구현되었다. 실험은 펜티엄III 512MB의 리눅스 시스템 상에서 실행되었으며, 사용한 데이터는 로이터-21578 ApteMod 버전[8]이다. 시험문서는 전체 10,788 문서로부터 임의적으로 선택한 788개의 문서로 고정하였으며, 학습문서집합은 10가지 다른 문서집합을 시험문서를 제외한 10,000 문서로부터 임의적으로 선택하여 구성하였다. 학습문서집합은 DXK로 표시되는데 이것의 의미는 이 학습문서집합에는 X천개의 문서가 학습문서로 사용되었다는 것이다. 즉, D2K는 학습문서가 2,000개라는 것을 의미한다.

<표 7>은 로이터-21578 데이터에 대한 통계이다. 각 학습문서집합별 출현하는 용어의 수를 표시하고, 식 (6)의 값을 추정하기 위하여 $|D|-\log_2|T|$를 계산 하였다. 학습문서집합에서 학습문서의 수가 증가함에 따라 $|T|$도 증가하나 그 증가치가 매우 적다. 이것은 학습 문서의 수는 증가하나 문서에 포함된 용어들이 대부분 반복되어 나타나므로 새로운 용어의 출현이 매우 적다는 것을 의미한다. 따라서 <표 7>에서 볼 수 있듯이 $|D|-\log_2|T|$는 $|D|$가 증가함에 따라 더욱 커지게 되고, 이것은 식 (6)에 따라 제안된 알고리즘의 성능이 더욱 좋아지게 된다는 것을 의미한다.

〈표 7〉 문서에 따른 용어의 수 $|T|$ 및 $|D|-\log_2|T|$

문서	D1K	D2K	D3K	D4K	D5K	D6K	D7K	D8K	D9K	D10K
$\|T\|$	1292	1952	2469	2850	3212	3531	3829	4087	4320	4562
$\|D\|-\log_2\|T\|$	990	1989	2989	3989	4988	5988	6988	7988	8988	9988



〈그림 5〉 학습문서의 량에 따른 kNN, EkNN의 실행시간

〈표 8〉 kNN의 실행시간 및 EkNN의 상대적 효율성(η)

문서	D1K	D2K	D3K	D4K	D5K	D6K	D7K	D8K	D9K	D10K
T_{kNN}	192초	406초	609초	827초	1031초	1232초	1472초	1672초	1911초	2100초
η_{EkNN}	89.6%	90.6%	91.1%	91.4%	91.5%	91.6%	91.8%	91.8%	92.0%	92.0%

실험은 각각의 학습문서집합에 대하여 한 번의 학습단계를 실행하고, 각 시험문서에 대하여 분류단계를 실행하였다. 따라서 동일한 학습문서집합에 대하여 분류단계를 788개의 다른 시험문서로 788번 반복적으로 실행한 것이다. 실험에서 k값은 20으로 고정하였다. 이것은 일반적으로 kNN 실험에서 사용하는 k값이다[10]. 기존의 kNN과 EkNN의 성능 차이를 보이기 위하여 각 실험에서 788번의 분류단계의 실행시간을 합하여 초단위로 측정하였다.

<그림 5>는 kNN, EkNN에 대하여 시험문서를 분류하는 분류단계에 대한 실행 시간을 그래프로 표시한 것이다. 그래프에서 x축은 실험에 적용한 학습문서의 크기를 표시하고, y축은 788개의 시험문서를 분류하는 총 실행시간을 나타낸다. <그림 5>는 두 알고리즘 모두 학습문서집합의 크기에 따라 실행 시간이 선형적으로 증가한다는 것을 보인다. 이것은 kNN이 경우 실행 시간이 직접적으로 $|D|$ 비례하기 때문이고, EkNN의 경우 $|D|$가 증가함에 따라 $|T|$는 거의 고정이고 따라서 $|\overline{t}|$가 증가하게 되는데, 실행시간이 $|\overline{t}|$에 비례하기 때문이다. <그림 5>의 그래프로부터 제안하는 알고리즘의 성능이 기존의 kNN보다 월등히 앞선다는 것을 알 수 있다.

<표 8>은 EkNN이 kNN에 대하여 상대적으로 얼마나 효율적인가를 보인다. 여기서 상대적 효율성(η_{EkNN})는 식 (7)과 같이 계산하였다.

$$\eta_{EkNN} = \frac{T_{kNN} - T_{EkNN}}{T_{kNN}} \times 100(\%) \qquad (7)$$

<표 8>은 제안한 EkNN이 kNN에 비하여 상대적으로 약 90%정도 실행속도의 개선이 있음을 보이고 있다. 이것은 식 (7)에 의하면 EkNN이 kNN에 비하여 약 10배 정도 빠르다는 것을 의미한다.

5. 결 론

본 논문은 kNN의 유사성 계산에서 <문서, 가중치> 쌍의 목록으로 구성된 학습용어집합을 사용하여 kNN에서 분류단계의 효율성을 크게 개선하는 알고리즘을 제안하였다. 제안된 알고리즘은 시간복잡도 측면에서 분석되고 기존의 알고리즘과 비교되었으며, 또한 로이터-21578 데이터를 사용하여 실험적으로 성능이 비교되었다. 이 실험으로부터 제안한 알고리즘 EkNN이 기존의 kNN보다 약 90% 정도의 효율성 개선이 있음을 알 수 있었다.

참 고 문 헌

[1] Y. Yang, "Expert Network : Effective and efficient learning from human decisions in text categorization and retrieval," In 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1994.

[2] S. T. Dumais, J. Platt, D. Heckerman, and M. Sahami, "Inductive learning algorithms and representations for text categorization," In *CIKM*, 1998.

[3] Y. Yang and X. Liu., "A re-examination of text categorization methods," In 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Berkley, August, 1999.

[4] Calvo, R. A. and H. A. Ceccatto, "Intelligent Document Classification," Intelligent Data Analysis, Vol.4, No.5, 2000.

[5] Calvo R. A., Classifying financial news with neural networks, In 6th Australian Document Symposium, p.6, December, 2001.

[6] Tom Ault and Y. Yang, "kNN, Rocchio and Metrics for Information Filtering at TREC-10," In The 10th Text Retrieval Conference(TREC-10), NIST, 2001.

[7] Y. Yang, A Study on Thresholding Strategies for Text Categorization, In 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New York, 2001.

[8] Reuters-21578 Document Collection, http://about.reuters.com/researchandstandards/corpus.

[9] Sebastiani F., "Machine learning in automated text categorization," ACM Computing Surveys, Vol.34, No.1, pp.1-47, 2002.

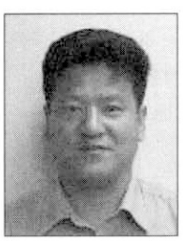
[10] Williams K. and R. A. Calvo, "*A Framework for Text Categorization*," 7th Australian Document Computing Symposium. December, 2002.

[11] 김한준, "텍스트 마이닝 기술을 적용한 대용량 온라인 문서 데이터의 계층적 조직화 기법", 서울대학교 대학원 박사학위 논문, 2002.

[12] Calvo, R. A. and J. M. Lee, "Coping with the News : the machine learning way," The 9th Australian World Wide Web Conference(AUSWEB 03), 2003.

[13] 이재문, "휴리스틱을 이용한 kNN의 효율성 개선", 정보처리학회논문지B, 제10-B권 제6호, 2003.

[14] 이석호, "C로 쓴 자료구조론", 사이텍미디어, pp.68-88.



이 재 문

e-mail : jmlee@hansung.ac.kr

1996년 한양대학교 전자공학과(학사)

1998년 한국과학기술원 전기 및 전자공학과(석사)

1992년 한국과학기술원 전기 및 전자공학과(박사)

1992년~1994년 한국통신 연구개발단 선임연구원

1994년~현재 한성대학교 컴퓨터공학부 부교수

관심분야 : 데이터베이스, 데이터 마이닝, 멀티미디어, 문서 분류, 정보처리임