# 휴리스틱을 이용한 kNN의 효율성 개선

이 재 문†


## 요 약

이 논문은 kNN의 정확도의 손실 없이 kNN의 효율성을 개선하는 휴리스틱을 제안한다. 제안된 휴리스틱은 kNN 실행 시간의 주요 요소인 두 문서간 유사성 계산을 최소화하는 것이다. 이것을 위하여 본 논문은 유사성의 상한값을 계산하는 방법과 훈련 문서를 정렬하는 방법을 제안한다. 제안된 휴리스틱을 문서 분류 프레임웍인 AI :: Categorizer 상에서 구현하였으며, 잘 알려진 로이터-21578 데이터를 사용하여 기존의 kNN과 비교 하였다. 성능 비교의 결과로부터 제안된 휴리스틱을 적용한 방법이 기존의 kNN보다 실행 속도측면에서 약 30~40%의 개선 효과가 있음을 알 수 있었다.


# An Improvement Of Efficiency For kNN By Using A Heuristic

Jae-Moon Lee†


## ABSTRACT

This paper proposed a heuristic to enhance the speed of kNN without loss of its accuracy. The proposed heuristic minimizes the computation of the similarity between two documents which is the dominant factor in kNN. To do this, the paper proposes a method to calculate the upper limit of the similarity and to sort the training documents. The proposed heuristic was implemented on the existing framework of the text categorization, so called, AI :: Categorizer and it was compared with the conventional kNN with the well-known data, Reuter-21578. The comparisons show that the proposed heuristic outperforms kNN about 30~40% with respect to the execution time.


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

## 1. 서 론

최근 웹 정보관리시스템 분야에서 기계 학습에 의한 문서 분류 연구가 활발히 진행되고 있다[2-4, 9, 11, 12]. 문서 분류란 미리 정해진 분류의 집합이 있을 때 특정 문서가 어느 분류에 속하는지를 판단하는 것을 말한다. 문서 분류의 초기 연구에서는 규칙에 기초하여 문서를 분류 하였으나 최근에는 컴퓨팅 기술의 발전으로 기계학습 방법에 의한 연구가 활발히 진행되고 있다[9].

문서 분류에 대한 연구의 방향은 크게 두 가지로 나뉜다. 하나는 분류의 정확도를 높이는 기술에 관한 연구이고[1, 2, 4, 6], 다른 하나는 문서 분류의 속도를 높이는 기술에 관한 연구이다[3]. 문서 분류에 대한 대부분의 연구는 전자에 집중되어 왔다. NaiveBayes, SVM, kNN 등 여러 기법들이 활용되어 왔고, 각 기법은 상대적인 장단점을 가지고 있다. NaiveBayes[2] 방법은 단순하고, 빠른 응답을 주는 반면, 분류의 정확도가 다른 기법에 비해 떨어지는 편이다. SVM[3]은 상당히 높은 정확도를 주는 기법이나 알고리즘이 복잡하고 속도가 느리다. kNN[1]은 가장 간단한 기법 중의 하나이면서 비교적 높은 분류 정확도를 보이지만, 실행 속도가 매우 느리다.

기계학습에 의한 문서 분류는 학습(훈련) 단계와 분류(시험) 단계로 나뉜다. 학습 단계는 미리 정해진 분류 집합과 각 분류별로 전문가에 의하여 정확히 분류된 학습 문서 집합을 입력으로 받아 학습하는 과정이다. 이 단계의 결과는 알고리즘별로 차이는 있으나 궁극적으로 시험 단계에서 최적의 성능을 얻을 수 있도록 입력 데이터를 가공하는 것이다. 분류 단계는 이러한 가공된 데이터와 새로운 문서를 입력받아 입력된 문서가 어느 분류에 속하는지를 판단하는 단계이다. 따라서 학습 단계는 대부분의 경우 한번 실행되고, 분류 단계는 새로운 문서가 발생할 때 마다 실행되어야 한다. 이러한 관계로 최근 뉴스 문서, 전자 메일 문서 등에서는 분류 단계의 실행을 실시간적으로 요구하고 있다[12].

앞에서 언급 하였듯이 kNN의 장점은 알고리즘의 단순성과 높은 정확도이며, 단점은 느린 실행 속도이다. kNN은 학습 단계에서 최소의 작업만 한다. 따라서 상대적으로 분

※ 본 연구는 2003학년도 한성대학교 공학연구센터 특별연구비 지원과제임.
† 정 회 원 : 한성대학교 정보전산학부 교수
논문접수 : 2003년 5월 29일, 심사완료 : 2003년 8월 7일

류 단계에서 많은 작업을 해야 하고 결과적으로 실행 속도가 매우 느리다. 본 논문은 이러한 kNN의 단점을 보완하여 분류 단계의 실행 속도를 높이는 것에 관한 연구이다. kNN은 하나의 시험 문서에 대하여 모든 학습 문서와 유사성을 계산한다. 따라서 kNN은 학습 문서의 수가 증가하면, 그 증가하는 비율에 따라 분류 단계의 속도는 느려지게 된다. 본 논문은 휴리스틱을 적용하여 학습 문서와의 유사성 계산을 가능한 최소화함으로써 실행 속도를 높이는 것이다. 물론 이러한 과정에서 kNN의 정확도에 대한 어떠한 희생도 없다.

2장은 kNN에 대한 소개와 실행 속도에 대한 분석을 설명하며, 3장에서는 휴리스틱을 적용한 개선된 kNN을 설명한다. 4장에서는 기존의 kNN과 개선된 kNN 사이의 성능 비교를 하며, 5장에서 결론을 논한다.

## 2. KNN 알고리즘

### 2.1 kNN 알고리즘

문서 분류는 임의의 문서가 미리 정해진 분류의 집합 중 어느 분류에 속하는 지를 찾아내는 문제이다[9, 11, 12]. 여기서 하나의 문서는 여러 분류에 속할 수도 있고, 반대로 하나의 분류는 여러 문서를 포함할 수도 있다. 전통적으로 이 문제는 규칙에 근거하여 해결하여 왔으나, 최근에는 컴퓨팅 기술의 발전에 힘입어 기계 학습 방법에 의하여 해결한다. 기계학습에 의한 문제의 해결은 먼저 분류 집합을 정의하고, 그 분류 집합의 특성을 잘 포함하는 문서(학습 문서)들을 전문가에 의하여 수집하여 이들을 통하여 기계 학습을 한다. 다음 이러한 학습된 데이터를 사용하여 문서(시험 문서)들을 적절히 분류하는 것이다.

기계 학습에는 많은 방법들이 연구되어 왔다[1-3, 9]. 이들은 각각의 특성이 있어, 그 특성이 잘 반영되는 환경에서 최고의 성능을 발휘한다. kNN은 [1]에 의하여 제안되었다. 이 알고리즘의 특징은 아주 간단하다는 것과 학습 단계에서 최소한의 처리 작업을 한다는 것이다. 문서 분류 관점에서 kNN의 정확도는 매우 우수하나, 분류 단계에서 실행 속도가 매우 느린 단점을 갖고 있다.

kNN은 학습 단계에서는 학습 문서에 대한 벡터화정도만 한다. 따라서 kNN이 어떻게 동작하는 가는 분류 단계만의 설명으로 충분하다. kNN은 분류 집합, 학습 문서 집합, 시험 문서 및 상수 $k$를 입력받는다. 이러한 입력으로부터 모든 학습 문서에 대하여 시험 문서와의 유사성을 계산하여 유사성이 가장 큰 $k$개의 학습 문서를 선택하고, 선택된 학습 문서가 속하는 분류들에 대하여 등급을 정하여 이들을 출력한다. 다음은 [1, 2, 10]에서 제시한 내용을 기초로한 kNN에 대한 기술이다.

```
스텝 1  : 시험문서 d를 정규화 한다.
스텝 2  : D_k = { };
스텝 3  : (d_m, m_s) = (φ, 0);                    // m_s : 최소의 유사성
스텝 4  : foreach d_i ∈ D {
스텝 4.1 :     s_i = compute_similarity(d, d_i)
스텝 4.2 :     if (m_s < s_i){
스텝 4.3 :         D_k = D_k ∪ {<d_i, s_i>} - {d_m, m_s}
스텝 4.4 :         (d_m, m_s) = find_minimal_similarity(D_k)
스텝 4.5 :     }
스텝 4.6 : }
스텝 5  : C의 계수를 0으로 초기화 한다.
스텝 6  : foreach d_si ∈ D_k{
스텝 6.1 :     d_si가 분류되는 c_i = {c_i1, c_i2, ···, c_i∈ C}의 모든
          요소에 대하여 C의 계수를 증가 시킨다.
스텝 6.2 : }
스텝 7  : C를 계수값으로 정렬하여 출력한다.
```

(알고리즘) kNN(입력 : 분류 C, 학습문서 D, 시험문서 d, 이웃상수 k, 출력 : 등급화된 C)

상기 알고리즘에서 입력 데이터는 분류 집합 $C$, 학습 문서 집합 $D$, 시험 문서 $d$와 이웃 상수 $k$이다. 시험 문서라 함은 분류되어야 할 문서를 말한다. 스텝 1에서 정규화는 모든 문서 벡터의 크기를 동일하게 하는 것으로 대부분의 경우 크기를 1로 한다[1, 9]. $m_s$는 $D_k$에서 가장 작은 유사성을 의미한다. 따라서 스텝 3에서 $m_s$는 0이 된다. 스텝 4.1에서 *compute_similarity*($d$, $d_i$) 함수는 두 입력 문서 $d$, $d_i$의 유사성($s_i$)을 계산하는 함수이다. 스텝 4.3에서 $D_k$는 $k$개의 가장 큰 유사성을 가지는 문서의 집합이고 *find_minimal_similarity*($D_k$)는 $D_k$에서 가장 작은 유사성을 가지는 문서와 유사성을 찾는 함수이다. 대부분의 문서 분류에서 문서의 표현은 <용어, 가중치>의 쌍으로 구성되는 벡터[9, 10, 12]를 사용하며, kNN에서 유사성의 계산으로 두 문서에 대한 벡터의 내적으로 한다[1, 9].

kNN의 동작을 설명하기 위하여 <표 1>, <표 2>에서 주어진 데이터를 사용하여 예를 들어 보기로 한다. <표 1>은 분류 집합을 의미하는 것으로 학습 문서별 이들이 속하는 분류들을 표시하고 있다. <표 2>는 학습 및 시험 문서를 표시하는 것으로 <용어, 가중치>의 쌍으로 구성되어 있다. 예를 들어 문서 $d$에서 {< a, 20 >< c, 10 >< e, 20 >< h, 10 >}의 의미는 문서 $d$는 용어 a, c, e, h로 구성되어 있고 이 문서에서 각 용어의 가중치는 20, 10, 20, 10과 같다는 것이다. 여기서 $k$는 2라고 하고, 스텝 1에서와 같이 문서 벡터의 크기를 일정하게 하는 정규화를 하면 소수점이 발생하여 설명이 복잡해지므로 정규화는 생략하기로 한다. 스텝 4의 반복문에 대한 실행 결과는 <표 3>과 같이 된다. <표 3>에서 알 수 있듯이 스텝 4의 결과 $D_k = \{d_1, d_3\}$와 <표 1>을 이용하여 스텝 6을 실행하면, $< c_2, 2>$, $< c_1, 1>$, $< c_4, 1>$의 결과를 얻게 되어 최종적으로 이들을 출력하게 된다.

〈표 1〉 분류 집합 $C$

| 문 서 | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ |
|---|---|---|---|---|---|
| 소속분류 | $C_1$, $C_2$ | $C_2$, $C_3$ | $C_2$, $C_4$ | $C_3$, $C_4$ | $C_1$, $C_4$ |

〈표 2〉 학습 문서 집합 $D$ 및 시험 문서 $d$

| 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 >< e, 20 >< h, 10 >} |

〈표 3〉 $s_i$, $m_s$ 및 $D_k$의 변화

| $d_i$ | 유사성($S_i$) | $m_s$ | $D_k$ |
|---|---|---|---|
| $d_1$ | 1200 = 10×30 + 20×30 +10×30 | 0 | {$d_1$} |
| $d_2$ | 400 = 20×20 | 0 | {$d_1$, $d_2$} |
| $d_3$ | 800 = 20×10 +10×20 +10×40 | 800 | {$d_1$, $d_3$} |
| $d_4$ | 300 = 20×10 +10×10 | 800 | {$d_1$, $d_3$} |
| $d_5$ | 300 = 20×10 | 800 | {$d_1$, $d_3$} |

### 2.2 kNN의 성능 분석

kNN의 성능 분석의 목적은 시간 복잡도 계산과 잘 알려진 데이터를 사용하여 실제 실행 시간을 측정하여 kNN 방법의 병목 현상을 주는 요소가 무엇인지를 보이는 것이다. 먼저 kNN의 시간 복잡도를 계산하기로 한다. 스텝 1의 비용은 단순히 시험 문서의 값을 특정 값으로 정규화 하는 것이므로 $|d|$의 비용이라 할 수 있다. 여기서 $|d|$는 문서 $d$에서 용어의 개수이다. 스텝 4.1의 경우 모든 학습 문서에 대하여 유사성을 계산하는 것이다. 유사성을 두 벡터의 내적이라고 할 때, 이의 최적의 방법은 정렬된 두 벡터의 값을 합병(merge) 함으로 찾을 수 있다. 따라서 두 문서의 내적 계산 비용은 ($|d|+|d_i|$)이 된다. 스텝 4.2, 스텝 4.3의 비용은 방법에 따라 여러 가지 형태로 나타날 수 있으나, 단순히 $D$에서 $k$번의 최대값 찾기를 실행한다고 하면 $k|D|$가 된다. 스텝 6의 비용은 해쉬를 사용하여 $C$에서 $c_{ij}$의 위치를 $O(1)$ 비용으로 찾을 수 있다고 하면 $\sum_{i=1}^{k}|c_i|$가 된다. 스텝 7의 비용은 전통적인 정렬 비용과 같다. 이러한 비용을 요약하면 다음과 같은 식으로 표현 할 수 있다.

$$T=\sum_{i=1}^{|D|}(|d|+|d_i|)+k|D|+\sum_{i=1}^{k}|c_i|+|C|\log_2|C| \quad (1)$$

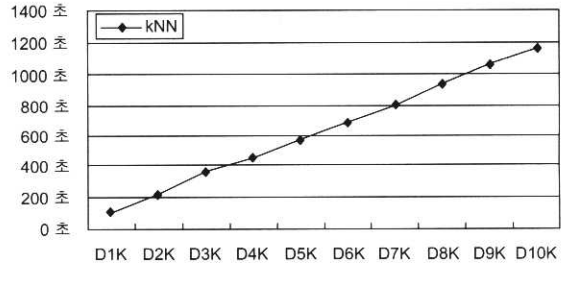
식 (1)에서 학습 문서의 평균 길이를 $|\overline{d}|$라고 하고, $|c_i|$의 평균 개수를 $|\overline{c}|$라고 한다면 식 (1)을 다음과 같이 쓸 수 있다.

$$T_{avg}=2|\overline{d}||D|+k|D|+k|\overline{c}|+|C|\log_2|C| \quad (2)$$

식 (2)에서 $k$의 값은 대체로 5~20이며, $|C|<<|D|$ 관계가 성립하는 것이 일반적이다. 이러한 조건을 고려하면 상기 알고리즘에서 스텝 4가 가장 많은 시간을 요구하는 요소가 되며, 시간 복잡도는 $T_{avg}=2|\overline{d}||D|=O(|D|)$으로 단순화 할 수 있다. 이것의 의미는 kNN의 분류 단계는 학습 문서의 수에 그 효율성이 선형적으로 의존한다는 것을 의미한다.

다음은 실제 실험 데이터를 사용하여 kNN의 실행 시간을 측정하였다. 실험을 위하여 사용된 데이터는 로이터-21578 ApteMod 버전[8]이다. 이것은 약 40M 바이트의 데이터로 10,788 뉴스 문서로 구성되어 있다. 전체 분류 집합에서 분류의 수는 90이다. kNN의 실행 시간을 측정하기 위해서 ApteMod는 하나의 시험 문서 집합과 다수의 학습 문서 집합으로 나누어 졌다. 시험 문서로는 ApteMod로부터 788개의 문서가 임의적으로 선택되었으며, 나머지 10,000개의 문서는 학습 문서로써 사용되었다. 학습 문서는 포함된 문서의 수에 따라 10종류로 나뉘었다. 문서는 DXK로 표시되는데 이것의 의미는 이 학습 문서에는 X천개의 문서가 학습 문서로 사용되었다는 것이다. 즉 D1K는 1,000개의 학습 문서가 있는 문서 집합을 의미한다. 실험은 펜티움Ⅳ 1GMB 메모리를 가진 리눅스 시스템에서 수행되었으며, 문서 분류를 위하여 사용된 프레임웍은 [10, 12]에서 개발된 객체지향 문서 분류 프레임웍을 이용하였다.

(그림 1)은 실험 결과를 보이고 있다. (그림 1)에서 볼 수 있듯이 실행 시간은 정확히 학습 문서의 수에 비례한다. 이는 시간 복잡도에서 예측한 것과 정확히 같은 결과이다.




(그림 1) 학습 문서의 량에 따른 kNN의 실행시간

kNN은 가장 단순한 문서 분류 기법중의 하나이다. 단순한 기법임에도 불구하고 우수한 분류 정확도를 주는 장점이 있으나 실행 속도가 매우 느리다는 단점이 있다. 이것은 대부분의 문서 분류에서 정확한 분류 결과를 얻기 위하여 많은 학습 문서를 채택하기 때문이다. 학습 문서가 많다는 것은 앞의 성능 분석에서 볼 수 있듯이 kNN의 경우 그 성능이 학습 문서의 양($|D|$)에 비례하여 늦어지기 때문에

kNN 기법에서는 치명적이라 할 수 있다.

## 3. 휴리스틱을 적용한 kNN의 개선

### 3.1 기본 개념

kNN이 단순한 알고리즘임에도 속도가 느린 이유는 kNN이 학습 단계에서 학습 문서의 벡터화 처리 외에는 Naive Bayes, SVM등 다른 방법에 비하여 특별한 일을 하지 않기 때문이다. 본 논문은 비교적 실시간적인 특성이 적은 학습 단계에서 많은 처리를 하게 함으로써 분류 단계에서 보다 효율적으로 동작하도록 하여 kNN의 실행 속도를 향상 시키는 방법을 제안한다.

본 논문에서 제안하는 기본 개념은 매우 간단한다. 2.1 절의 마지막에서 주어진 예제를 다시 한번 관찰하면, $d_4$, $d_5$과의 유사성은 최종 결과를 도출하는데 있어 아무런 영향을 미치지 못하였다. 즉, 이것은 이들의 유사성은 계산할 필요도 없었다는 것을 의미한다. 만약 $d_4$, $d_5$와의 유사성 계산에서 계산 전에 이들의 값이 800보다 클 수 없다는 것을 미리 알 수 있다면, $d_4$, $d_5$와의 유사성을 계산하지 않고도 $d_1$, $d_3$을 선택할 수 있을 것이다. 이 경우 $d_4$, $d_5$과의 유사성 계산 비용을 절약 할 수 있다. 기본 개념은 임의의 두 벡터에 대한 유사성을 계산하기 전에 그 유사성의 가능한 상한값을 계산하여 그 상한값이 현재까지 계산된 유사성들의 $k$ 번째 큰 값보다 작다면 유사성을 계산하지 않음으로써 효율성을 향상시키는 것이다.

〈표 4〉 BkNN에서 $s_i$, $m_i$, $m_s$ 및 $D_k$의 변화

| $d_i$ | 유사성($s_i$) | $m_i$ | $m_s$ | $D_k$ |
|---|---|---|---|---|
| $d_1$ | 1200 = 10×30 + 20×30 +10×30 | 2400 | 0 | {$d_1$} |
| $d_2$ | 400 = 20×20 | 800 | 0 | {$d_1$, $d_2$} |
| $d_3$ | 800 = 20×10 +10×20 +10×40 | 3200 | 800 | {$d_1$, $d_3$} |
| $d_4$ | - | 600 | 800 | {$d_1$, $d_3$} |
| $d_5$ | - | 800 | 800 | {$d_1$, $d_3$} |

상한 값에 대한 다양한 방법이 존재할 수 있다. 그러나 명백한 것은 이 상한 값을 구하는 비용이 최소한 두 벡터의 유사성을 구하는 비용보다는 작어야 한다. 본 논문에서는 두 문서 $d$, $d_i$ 사이의 상한 값은 $M_d \times M_{d_i} \times \min(CNT_d, CNT_{d_i})$으로 한다. 여기서 $M_d$, $M_{d_i}$는 두 문서 $d$, $d_i$에서 가장 큰 가중치를 의미하며, $CNT_d$, $CNT_{d_i}$는 두 문서 $d$, $d_i$ 포함하고 있는 용어의 개수이다. <표 4>는 <표 2>의 입력 문서에 대한 유사성, 상한 값($m_i$), $m_s$, $D_k$의 변화를 보이고 있다. <표 4>에서 볼 수 있듯이 $d_4$, $d_5$에 대한 유사성은 상한 값이 $m_s$ 800보다 작거나 같은 600, 800이므로 계산되지 않았다. 다음은 kNN을 개선한 BkNN 알고리즘이다.

```
스텝 1    : 시험문서 d를 정규화 및 M_d, CNT_d를 찾는다.
스텝 2    : D_k = { } ;
스텝 3    : (d_m, m_s) = (φ, 0) ;
스텝 4    : foreach d_i ∈ D {
스텝 4.0.1 :   m_i = compute_maximal_similarity(M_d, CNT_d, d_i)
스텝 4.0.2 :   if (m_s >= m_i) continue ;
스텝 4.1  :   s_i = compute_similarity(d, d_i)
스텝 4.2  :   if (m_s < s_i){
스텝 4.3  :       D_k = D_k ∪ {<d_i, s_i>} - {d_m, m_s}
스텝 4.4  :       (d_m, m_s) = find_minimal_similarity(D_k)
스텝 4.5  :   }
스텝 4.6  : }
스텝 5    : C의 계수를 0으로 초기화 한다.
스텝 6    : foreach d_si ∈ D_k {
스텝 6.1  :   d_si가 분류되는 c_i = {c_i1, c_i2, ···, c_im ∈ C}의
              모든 요소에 대하여 C의 계수를 증가 시킨다.
스텝 6.2  : }
스텝 7    : C를 계수값으로 정렬하여 출력한다.
```

(알고리즘) BkNN(입력 : 분류 C, 학습문서 D, 시험문서 d, 이웃상수 k, 출력 : 등급화된 C)

상기 알고리즘은 kNN 알고리즘에서 스텝 1에서 시험 문서 $d$의 $M_d$, $CNT_d$를 찾는 것이 추가 되었으며, 스텝 4.0.1, 4.0.2도 추가되었다. 스텝 4.0.1에서 *compute_maximal_similarity*($M_d$, $CNT_d$, $d_i$)는 두 문서 $d$, $d_i$에서 $M_d \times M_{d_i} \times \min(CNT_d, CNT_{d_i})$를 찾는 함수이다. 이 함수를 효율적으로 실행하기 위해서는 두 문서에 대한 $M$, $CNT$를 알아야 한다. 시험 문서 $d$에 대해서는 스텝 1에서 찾아졌으며, 학습문서 $d_i$에 대해서는 학습 단계에서 찾아진다고 가정한다. *compute_maximal_similarity*($d$, $d_i$)에서는 특별히 이들을 찾는 과정은 없으며, 따라서 이 함수의 시간 복잡도는 $O(1)$이 된다.

### 3.2 개선된 알고리즘

기본 알고리즘은 4장에서 실험을 통하여 보이겠지만 성능을 크게 향상 시키지 못하였다. 그 이유는 2장의 예에서 $d_2$와 같이 실제로 유사성의 값은 작으나 순서상 먼저 계산된 관계로 스텝 4.0.2를 통과하여 유사성을 계산하여야 하는 문서가 많기 때문이다. 이것은 학습 문서의 순서가 알고리즘의 성능에 영향을 미친다는 것을 의미한다. 즉, 스텝 4에서 $d_1$, $d_3$, $d_2$, $d_4$, $d_5$의 순서로 학습 문서가 처리된다면 $d_2$에 대해서도 유사성 계산을 하지 않아도 될 것이다. 따라서 정확한 유사성을 계산하지 않아도 유사성이 클 가능성이 있는 문서를 먼저 처리하도록 학습 문서를 정렬하여야 할 필요가 있다. 본 논문에서는 이를 위하여 용어-문서 목록의 사용을 제안한다. 용어-문서 목록에 근거하여 학습 문서를 정렬하고 그 정렬된 순서에 따라 스텝 4를 실행하도

록 알고리즘을 개선한다. 용어-문서 목록이란 전체 학습 문서에 대하여 용어별 그 용어를 포함하는 문서의 목록을 말한다. <표 2>의 학습 문서에 대한 용어-문서 목록은 <표 5>와 같다.

<표 5> <표 2>의 학습 문서에 대한 용어-문서 목록

| 용 어 | 문서 목록 | 용 어 | 문서 목록 |
|---|---|---|---|
| a | $d_2$, $d_3$, $d_4$ | e | $d_1$, $d_2$, $d_5$ |
| b | $d_3$, $d_4$ | f | $d_3$ |
| c | $d_1$, $d_3$, $d_4$ | g | $d_5$ |
| d | $d_1$ | h | $d_1$, $d_3$ |

문서의 정렬은 용어-문서 목록을 이용하여 모든 학습 문서에 우선순위를 부여하고 이 우선 순위를 기준으로 정렬한다. 각 학습 문서에 대한 우선순위는 시험 문서에 포함된 용어들을 포함하는 개수와 같다. 이것은 시험 문서가 포함하고 있는 용어를 많이 포함하면 할수록 높은 우선순위를 갖는다는 것을 의미한다. <표 3>의 시험 문서 $d$에 대하여 <표 5>의 용어-문서 목록을 이용하여 각 학습 문서의 우선순위를 계산하면 시험 문서 $d$가 용어 a, c, e, h를 포함하고 있으므로 {$<d_1, 3>$, $<d_2, 2>$, $<d_3, 3>$, $<d_4, 2>$, $<d_5, 1>$}와 같은 우선순위를 얻게 된다. 우선순위를 기준으로 학습 문서를 정렬 하면 $d_1 - d_3 - d_2 - d_4 - d_5$ 순서를 얻을 수 있다. 이러한 순서로 학습 문서와의 유사성을 계산하면, $d_2$의 유사성도 계산할 필요가 없게 된다. 두 문서 $d$와 $d_i$ 사이의 우선순위를 $P_i$라고 하면 이것은 앞에서 언급하였듯이 두 문서 $d$와 $d_i$ 사이에 공통 용어의 개수를 의미한다. 따라서 $P_i \leq CNT_d$, $CNT_{d_i}$가 성립하므로 보다 유사성에 가까운 상한 값을 구하기 위하여 $CNT_d$, $CNT_{d_i}$ 대신에 $P_i$를 사용하는 것이 좋다. 다음은 용어-문서 목록을 이용하는 개선된 EkNN 알고리즘이다.

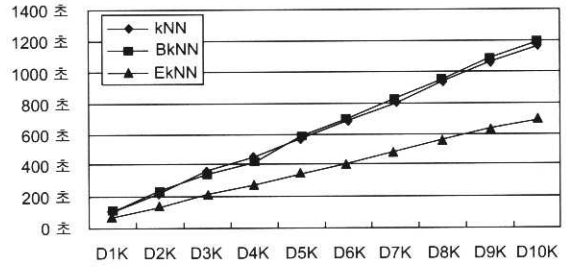
```
스텝 1    : 시험문서 d를 정규화 및 M_d, CNT_d를 찾는다.
스텝 1.0  : (D_ordered, P) = build_order_document(d, M)
스텝 2    : D_k = { } ;
스텝 3    : (d_m, m_s) = (φ, 0) ;
스텝 4    : foreach d_i ∈ D_ordered {
스텝 4.0.1 :    m_i = compute_maximal_similarity(M_d, P_i, d_i)
스텝 4.0.2 :    if(m_s >= m_i) continue ;
스텝 4.1  :    S_i = compute_similarity(d, d_i)
스텝 4.2  :    if (minimalSimilarity < s_i){
스텝 4.3  :        D_k = D_k ∪ {<d_i, s_i>}-{d_m, m_s}
스텝 4.4  :        (d_m, m_s) = find_minimal_similarity(D_k)
스텝 4.5  :    }
스텝 4.6  : }
스텝 5    : C의 계수를 0으로 초기화 한다.
스텝 6    : foreach d_si ∈ D_k {
스텝 6.1  :    d_si가 분류되는 C_i = {C_i1, C_i2, ···, C_im ∈ C}의 모든
               요소에 대하여 C의 계수를 증가 시킨다.
스텝 6.2  : }
스텝 7    : C를 계수값으로 정렬하여 출력한다.
```

(알고리즘) EkNN(입력 : 분류 C, 학습문서 D, 시험문서 d, 이웃상수 k, 용어문서목록 M, 출력 : 등급화된 C)

상기 알고리즘에서 용어-문서 목록은 입력으로 받는다. 이것은 용어-문서 목록은 학습 단계에서 계산된다고 가정한다. BkNN에서 스텝 1.0이 추가되었다. 이 스텝에서 *build_order_document(d, M)* 함수는 스텝 4에서 처리할 문서의 순서를 정하는 함수이다. 이 함수의 비용은 학습 문서 $|D|$개를 각각 0, 1, 2, ···, $|D|$-1와 같이 번호로 표기하면 $O(|D|\log|D|)$으로 할 수 있다. 스텝 4.0.1에서는 보다 우수한 상한값을 얻기 위하여 *compute_maximal_similarity*는 $CNT_d$ 대신에 $P_i$를 입력으로 사용한다.

## 4. 성능 비교

제안된 휴리스틱의 효과를 보이기 위하여 kNN, BkNN, EkNN을 구현하여 그 성능을 측정 하였다. 구현의 효율성 및 일반성을 위하여 [10]에서 개발한 AI :: Categorizer 프레임웍을 사용하였다. AI :: Categorizer는 객체 지향 문서 분류 프레임웍으로 문서 분류에 필요한 다양한 기능 및 일반적으로 잘 알려진 문서 분류 알고리즘의 구현을 제공한다. 이 프레임웍을 사용함으로써 문서의 토큰화, 벡터 모델, 차원 축소등 부수적인 구현을 생략할 수 있었으며, 실험을 위해서 AI :: Categorizer :: Learner 객체로부터 상속된 kNN, BkNN, EkNN만 구현 하였다. 실험은 펜티움IV 512MB의 리눅스 시스템 상에서 실행되었으며, 사용한 데이터는 2장에서 설명한 것과 같이 로이터-21578 ApteMod 버전이다. 시험 문서는 10,788 문서로부터 임의적으로 선택한 788개의 문서로 고정하였으며, 학습 문서는 10가지 다른 데이터를 시험 문서를 제외한 10,000 문서로부터 임의적으로 선택하였다. 학습 문서는 DXK로 표시되는데 이것의 의미는 이 학습 문서에는 X천개의 문서가 학습 문서로 사용되었다는 것이다.




(그림 2) 학습 문서의 량에 따른 kNN, BkNN, EkNN의 실행 시간

〈표 6〉 kNN의 실행 시간 및 BkNN, EkNN의 상대적 효율성( $\eta$)

| 데이터 | $T_{kNN}$ | $\eta_{BkNN}$ | $\eta_{EkNN}$ | 데이터 | $T_{kNN}$ | $\eta_{BkNN}$ | $\eta_{EkNN}$ |
|---|---|---|---|---|---|---|---|
| D1K | 106.0초 | -2.0% | 33.5% | D6K | 576.0초 | -1.7% | 40.0% |
| D2K | 224.2초 | -2.4% | 37.8% | D7K | 686.7초 | -2.3% | 40.2% |
| D3K | 360.6초 | 4.6% | 38.0% | D8K | 805.5초 | -2.0% | 40.8% |
| D4K | 454.5초 | 7.1% | 33.4% | D9K | 935.1초 | -1.8% | 40.8% |
| D5K | 576.0초 | -1.5% | 39.7% | D10K | 1060.9초 | -2.3% | 41.0% |

(그림 2)는 kNN, BkNN, EkNN에 대하여 실행 시간 측정치이다. <표 6>은 kNN에 대한 BkNN, EkNN의 상대적 효율성( $\eta$)을 보인다. 상대적 효율성은 다음과 같이 계산하였다.

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

여기서 $T_{algorithm}$은 특정 알고리즘의 실행 시간이다. (그림 2)는 3가지 알고리즘 모두 학습 문서의 수에 따라 실행 시간이 선형적으로 증가한다는 것을 보인다. 이것은 3가지 알고리즘 모두 실행 시간이 학습 문서의 수에 비례하기 때문이다. (그림 2)와 <표 6>은 BkNN이 대부분의 경우 오히려 kNN보다 우수하지 못하다는 것을 보인다. 이것의 이유는 BkNN에서 스텝 4.0.2를 통과하지 못하는 문서가 많기 때문이다. EkNN의 경우 kNN에 비하여 확실히 성능 개선이 되었음을 볼 수 있다. 흥미로운 것은 학습 문서의 수가 증가함에 따라 효율성이 보다 증가한다는 것이다. 즉 <표 6>에서 볼 수 있듯이 D1K에 대해서는 효율성이 33.5%이나 D10K에서는 41%로 증가 하였다. 이것은 문서의 수는 증가하나 $k$의 값이 고정되었기 때문에 문서의 수가 증가하는 경우 상대적으로 스텝 4.0.2를 통과하는 비율이 높아지기 때문이다.

## 5. 결 론

본 논문은 kNN의 효율성 개선을 위하여 하나의 휴리스틱을 제안하였다. 제안된 휴리스틱은 유사성의 상한값을 이용하여 유사성의 계산을 가능한 최소화하는 것이다. 이를 위하여 용어-문서 목록을 이용하였으며, 이 용어-문서 목록을 학습 단계에서 생성하도록 함으로써 분류 단계를 효율적으로 실행하도록 하였다. 제안한 알고리즘은 잘 알려진 AI :: Categorizer 프레임웍상에서 구현되었으며, 성능의 우수성을 보이기 위하여 기존의 kNN와 실행 속도측면에서 비교되었다. 성능 비교의 결과는 제안한 알고리즘이 30~40%까지 기존의 kNN보다 우수함을 보였다.

## 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, 4(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, 34(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.



**이 재 문**

e-mail : jmlee@hansung.ac.kr

1986년 한양대학교 전자공학과(학사)

1988년 한국과학기술원 전기 및 전자공학과 (공학석사)

1992년 한국과학기술원 전기 및 전자공학과 (공학박사)

1992년~1994년 한국통신 연구개발단 선임연구원

1994년~현재 한성대학교 정보전산학부 부교수

관심분야 : 데이터베이스, 데이터 마이닝, 멀티미디어, 정보검색, XML