행렬 전치를 이용한 효율적인 NaiveBayes 알고리즘

이 재 문†

요 약

본 논문은 NaiveBayes에서 정확도의 손실 없이 효율적으로 동작하는 NaiveBayes에 대한 새로운 알고리즘을 제안한다. 제안된 방법은 분류 벡터에 대한 행렬 전치를 사용하여 NaiveBayes의 확률 계산량을 최소화 하는 것이다. 제안된 방법을 문서 분류 프레임웍인 AI::Categorizer 상에서 구현하였으며, 잘 알려진 로이터-21578 데이터를 사용하여 기존의 NaiveBayes 방법과 비교하였다. 성능 비교의 결과로부터 제안된 방법이 기존의 NaiveBayes 방법보다 실행 속도측면에서 약 2배 정도의 성능 개선 효과가 있음을 알 수 있었다.

An Efficient Algorithm for NaiveBayes with Matrix Transposition

Jae Moon Lee†

ABSTRACT

This paper proposes an efficient algorithm of NaiveBayes without loss of its accuracy. The proposed method uses the transposition of category vectors, and minimizes the computation of the probability of NaiveBayes. The proposed method was implemented on the existing framework of the text categorization, so called, AI::Categorizer and it was compared with the conventional NaiveBayes with the well-known data, Reuter-21578. The comparisons show that the proposed method outperforms NaiveBayes about two times with respect to the executing time.

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

1. 서 론

최근 웹 문서 등 전자 문서의 급증으로 이들을 관리하는 정보관리시스템 분야에서 기계 학습에 의한 문서 분류 연구가 활발히 진행되고 있다[2-5, 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, 7], 다른 하나는 문서 분류의 속도를 높이는 기술에 관한 연구이다[3, 13]. 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. 학습단계는 미리 정해진 분류 집합과 각 분류별로 전문가에 의하여 정확히 분류된 학습 문서 집합을 입력으로 받아 학습하는 과정이다. 이 단계의 결과는 알고리즘별로 차이는 있으나 궁극적으로 분류단계에서 최적의 성능을 얻을 수 있도록 입력 데이터인 학습 문서를 가공하는 것이다. 분류단계는 이러한 가공된 데이터와 새로운 문서를 입력 받아 입력된 문서가 어느 분류에 속하는지를 판단하는 단계이다. 따라서 학습단계는 대부분의 경우 한번만 실행되고, 분류단계는 새로운 문서가 발생할 때 마다 실행되어야 한다. 이러한 관계로 최근 뉴스 문서, 전자 메일 문서 관리 시스템 등에서는 분류단계를 실시간으로 처리되

※ 본 논문은 2003학년도 한성대학교 교내연구비 지원과제임.
† 정 회 원 : 한성대학교 컴퓨터공학부
논문접수 : 2003년 10월 12일, 심사완료 : 2004년 2월 6일

도록 요구하고 있다[12, 13]. 본 연구는 이러한 경향에 적합하도록 NaiveBayes 방법의 정확도에 대한 손실없이 기존의 방법보다 효율적으로 동작하는 NaiveBayes 방법의 분류단계를 제안한다.

2장에서는 NaiveBayes에 대한 전반적인 소개와 실행 속도에 대하여 시간 복잡도를 계산하며, 3장에서는 행열 전치를 통한 개선된 알고리즘을 제안하며, 제안된 방법에 대하여 시간 복잡도를 계산한다. 4장에서는 기존의 NaiveBayes와 개선된 NaiveBayes사이의 성능 비교를 하며, 5장에서 결론을 논한다.

2. NaiveBayes 문서 분류

2.1 NaiveBayes 개념

NaiveBayes 방법은 확률에 근거한 문서 분류 방법이다. 이는 임의의 문서가 특정 분류에 속할 확률을 계산하여 계산된 확률 중 가장 높은 확률을 가지는 분류를 선택하는 것을 말한다. 이때 확률은 다음 식에 의하여 계산된다.

$$P(c_j \mid d_i)=\frac{P(c_j)P(d_i \mid c_j)}{P(d_i)} \quad (1)$$

상기식의 의미는 문서 집합 $d_1, d_2, \cdots, d_{|D|}$중에서 문서 d_i가 분류 집합 $c_1, c_2, \cdots, c_{|C|}$중에서 c_j에 속할 확률을 의미한다. 여기서 $P(d_i)$는 문서 집합에서 임의로 추출한 문서가 d_i일 확률이고, $P(c_j)$는 문서 집합에서 임의로 추출한 문서가 분류 c_j에 속할 확률이다. $P(d_i \mid c_j)$는 분류 c_j에 속하는 문서집합에서 임의로 추출한 문서가 d_i가 될 확률이다. 따라서 NaiveBayes 방법은 주어진 문서 d_i에 대하여 $P(d_i \mid c_1), P(d_i \mid c_2), \cdots, P(d_i \mid c_{|C|})$의 확률을 구하여 가장 높은 확률을 가지는 분류를 선택하는 것이다. 이것을 일반적으로 다음과 같이 표시한다.

$$C_{best}=\underset{c_i \in C}{ArgMax}\left[\frac{P(c_j)P(d_i \mid c_j)}{P(d_i)}\right] \quad (2)$$

C_{best}를 구하는 과정에서 $P(d_i)$는 모든 c_j에 대하여 항상 일정한 값을 가지므로 생략할 수 있다. 그러나 $P(d_i \mid c_j)$의 단어간 의존성을 고려하여야 하기 때문에 쉬운 일이 아니다. 따라서 대부분의 경우 계산을 효율적으로 하기 위하여 하나의 문서에서 단어간 의존성은 없다고 가정하여 계산한다[9, 11]. 이러한 가정 하에 문서 d_i가 $(t_{i1}, t_{i2}, \cdots, t_{in})$로 표현된다고 하자. 여기서 t_{ik}는 문서 d_i가 포함하고 있는 용어이고 n은 포함하고 있는 용어의 개수이다. 이 경우 $P(d_i \mid c_j)$는 $\prod_{k=1}^{n} P(t_{ik} \mid c_j)$로 표현되고, $\prod_{k=1}^{n} P(t_{ik} \mid c_j)$에서 $P(t_{ik} \mid c_j)$가 0인 경우를 피하기 위하여 라플라스 스무더(Laplace Smoother)를 적용하여 이를 $\frac{1+TF(t_k, c_j)}{|T|+\sum_{s=1}^{N_{cj}} TF(t_s, c_j)}$로 표현한다[9, 11]. 여기서 $TF(t, x)$는 x에서 용어 t의 가중치를 의미한다. 또한 $\sum_{s=1}^{N_{cj}} TF(t_s, c_j)$는 분류 c_j에 나타나는 모든 용어에 대한 가중치의 합이고 $|T|$는 학습 문서 집합에 나타나는 전체 용어의 집합의 크기이다. $P(t_{ik} \mid c_j)$가 항상 1보다 작은 값을 가지는 확률이므로 $\prod_{k=1}^{n} P(t_{ik} \mid c_j)$는 너무 작은 값을 가진다. 이를 피하기 위하여 대부분의 경우 식 (2)에 log를 적용하여 계산한다. 또한 시험 문서 d에 나타나는 용어의 가중치를 적용하기 위하여 아래 식 (3)과 같이 $TF(t_k, d)$를 적용한다[10].

$$C_{best}=\underset{c_i \in C}{ArgMax}\left[\log P(c_j)+\sum_{k=1}^{n} \left\{\log\left(\frac{1+TF(t_k, c_j)}{S(c_j)}\right)\times f(TF(t_k, d))\right\}\right] \quad (3)$$

상기 식에서 $P(c_j)$는 $\frac{DF(c_j)}{|D|}$이고, $S(c_j)$는 $|T|+\sum_{s=1}^{N_{cj}} TF(t_s, c_j)$이다. 여기서 $|D|$는 학습 문서의 수이며, $DF(c_j)$는 분류 c_j에 속하는 학습 문서의 수이다. $f(x)$는 x의 값을 변환하는 함수로써 대부분의 경우 $f(TF(t_k, d))=1$ 또는 $f(TF(t_k, d))=TF(t_k, d)$를 사용한다. 전자의 경우 시험 문서 d에 나타나는 용어의 가중치를 전혀 반영하지 않은 경우이며[11], 후자의 경우 가중치에 대한 특별한 변환 없이 반영하는 경우이다. 본 본문에서는 후자의 경우라 가정한다.

2.2 NaiveBayes 알고리즘

문서 분류에서 기계학습을 사용하는 모든 방법처럼 NaiveBayes 방법도 학습단계와 분류단계로 나뉘어 동작한다. NaiveBayes 방법의 학습단계 알고리즘은 분류단계에서 계산량을 최소화 할 수 있도록 식 (3)에서 $P(c_j)$, $S(c_j)$ 및 $TF(t_k, c_j)$를 계산하여 저장하는 것이다. NB_Train NaiveBayes 방법에 대한 학습단계 알고리즘이다. 이 알고리즘은 분류 집합 C와 학습 문서 집합 D를 입력으로 받고, $P(c_j)$, $S(c_j)$, $TF(t_k, c_j)$를 출력한다.

NB_Train 알고리즘의 스텝 4.1.3.2에서 c_j.vector$\oplus\{\langle t_k, w_k\rangle\}$는 집합 c_j.vector에 t_k를 포함 하는 요소가 없으면 단

순히 c_j.vector Y{$\langle t_k, w_k \rangle$}이 되고, 만약 존재한다면(c_j.vector - {$\langle t_k, w_x \rangle$}) Y {$\langle t_k, w_x + w_k \rangle$}의 연산을 하는 것이다. 스텝 6은 실제적으로 출력될 $P(c_j)$ $S(c_j)$ 및 $TF(t_k, c_j)$를 구하는 과정이다. sum_weight(c_j.vector)는 c_j.vector의 요소 $\langle t_k, w_k \rangle$에서 모든 w_k값을 합하여 출력하는 함수이다.

NB_Train(입력 : 분류집합 C[], 학습문서집합 D[],
출력 : P_c[], S_c[], $TF_{t|c}$[])

스텝 1 : foreach $c_j \in C$ {
스텝 1.1 : c_j.vector = { } ; c_j.count = 0 ;
스텝 2 : }
스텝 3 : T = { } ;
스텝 4 : foreach $d_i \in D$ {
스텝 4.1 : foreach $c_j \in C$ {
스텝 4.1.1 : if(d_i is not classified in c_j) continue ;
스텝 4.1.2 : c_j.count++ ;
스텝 4.1.3 : foreach $\langle t_k, w_k \rangle \in d_i$ {
스텝 4.1.3.1 : $T = T\ Y\ \{t_k\}$;
스텝 4.1.3.2 : c_j.vector = c_j.vector$\oplus$ {$\langle t_k, w_k \rangle$} ;
스텝 4.1.4 : }
스텝 4.2 : }
스텝 5 : }
스텝 6 : foreach $c_j \in C$ {
스텝 6.1 : $P_c[c_j] = c_j$.count / |D| ;
스텝 6.2 : $S_c[c_j]$ = | T | + sum_weight(c_j.vector) ;
스텝 6.3 : $TF_{t|c}[c_j] = c_j$.vector ;
스텝 7 : }

〈표 1〉 분류 집합

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

〈표 2〉 학습 문서 집합 및 시험 문서

종 류	문서	용 어 벡 터
학습문서	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, 10>}
시험문서	d	{<B, 20><C, 10><H, 10>}

<표 1>과 <표 2>를 이용하여 상기 알고리즘의 동작을 설명하도록 한다. <표 1>은 분류 집합에 대한 정보이고, <표 2>는 학습 문서에 대한 정보로 이들은 상기 알고리즘의 입력이라고 할 수 있다. 문서 d_1에서 {<C, 30><D, 10><E, 30><H, 30>}의 의미는 문서 d_1이 용어 C, D, E, H로 구성되어 있고, 각각의 가중치는 30, 10, 30, 30이라는 것을 의미한다. 먼저 스텝 1~5까지 실행결과는 <표 3>에 나타나 있다.

〈표 3〉 스텝 5까지 실행 결과

분류	c_j.vector	c_j.count
c_1	<C, 30><D, 10>,<E, 40><G, 10><H, 30>	2
c_2	<A, 30><B, 10><C, 50><D, 10><E, 50><F, 30><H, 70>	3
c_3	<A, 20><E, 30><G, 10>	2
c_4	<A, 20><B, 20><C, 30><E, 10><F, 30><G, 10><H, 40>	3
T	{A, B, C, D, E, F, G. H}	

스텝 6~7까지의 결과는 <표 4>에 나타나 있다. Naive-Bayes 방법에서 흥미로운 것은 학습 문서의 수를 증가하여 입력 데이터가 많아지더라도 출력 데이터는 거의 일정하다는 것이다. 이것은 학습 문서의 수가 증가하더라도 분류의 수는 일정하기 때문이다.

〈표 4〉 최종 출력 P_c[], S_c[], $TF_{t|c}$[]

C	P_c	S_c	$TF_{t|c}$: < 용어, 가중치 >
c_1	2/5	8+120	<C,30><D,10><E,40><G,10><H,30>
c_2	3/5	8+250	<A,30><B,10><C,50><D,10><E,50><F,30><H,70>
c_3	2/5	8+60	<A,20><E,30><G,100>
c_4	3/5	8+160	<A,20><B,20><C,30><E,10><F,30><G,10><H,40>

NB_Categorizers는 임의의 시험 문서 d를 분류하는 분류단계에 대한 알고리즘이다. 분류단계의 알고리즘은 학습단계의 출력과 시험 문서를 입력으로 하고, 이를 바탕으로 등급화된 분류 집합을 출력하는 것으로 한다.

NB_Categorizer(입력 : C[], P_c[], S_c[], $TF_{t|c}$[], 시험문서 d,
출력 : 등급화된 C_{rank}[ ])

스텝 1 : $C_{rank}[c_j] = \log(P_c[c_j])$foreach $c_j \in C$;
스텝 2 : foreach $c_j \in C$ {
스텝 2.1 : foreach $\langle t_k, w_k \rangle \in d$ {
스텝 2.1.1 : $w = (\langle t_k, any \rangle \in TF_{t|c}[c_j])?any+1 : 1$;
스텝 2.1.2 : $C_{rank}[c_j] += w_k \times \log(w / S_c[c_j])$;
스텝 2.2 : }
스텝 3 : }

상기 알고리즘의 입력에서 P_c[], S_c[], $TF_{t|c}$[]는 학습단

계의 출력을 입력으로 받은 것이다. 스텝 2.1.1에서 any는 $TF_{t|c}[c_j]$에서 용어 t_k에 대한 가중치이다. 이 스텝에서 w는 용어 t_k가 분류 c_j에 존재하면 $any+1$이 되며, 만약 존재하지 않으면 1이 된다. 이것은 존재하지 않은 용어에 대하여 라플라스 스무더를 적용하는 것을 의미한다.

다시 앞의 예제의 결과를 NB_Categorizer에 적용해 보기로 하자. 시험 문서 d는 <표 2>에서와 같이 {<B, 20><C, 10><H, 10>}으로 구성되어 있다고 한다.

〈표 5〉 등급화된 분류, C_{rank}

C	C_{rank}
c_1	log(2/5)+20×log(1/128)+10×log(31/128)+10×log(31/128)
c_2	log(3/5)+20×log(11/258)+10×log(51/258)+10×log(71/258)
c_3	log(2/5)+20×log(1/68)+10×log(1/68)+10×log(1/68)
c_4	log(3/5)+20×log(21/168)+10×log(31/168)+10×log(41/168)

이를 이용하여 NB_Categorizer을 실행한 결과는 <표 5>와 같다. $C_{rank}[c_3]$의 경우 시험 문서 d가 포함하는 어떠한 용어도 포함하지 않기 때문에 스텝 2.1.1에서 w의 모든 값이 1로 계산된다. C_{best}를 찾는 것은 <표 5>에서 가장 큰 값을 가지는 c_j를 선택하는 것이다.

2.3 성능 분석

이 장에서는 NaiveBayes 방법의 효율성에 대하여 시간 복잡도를 계산하기로 한다. 대부분의 경우 학습단계보다는 분류단계의 효율성이 훨씬 중요하기 때문에 학습단계의 효율성을 크게 고려하지 않고, 분류단계의 효율성에 집중하기로 한다. 성능 분석에서 분류 집합은 고정되고, 그 종류도 매우 제한적이어서 분류 $c_1, c_2, \cdots, c_n$은 $0, 1, \cdots, n-1$로 색인화 되어 $X[c_j]$의 비용은 배열의 액세스 비용과 동일하다고 가정한다. 또한 임의의 집합 Y에 대하여 $|Y|$는 집합 Y에서 요소의 개수를 표시한다고 정의한다.

알고리즘 NB_Categorizer의 효율성을 시간 복잡도 측면에서 분석하기로 하자. 스텝 1의 비용은 $|C|$개의 배열을 초기화 하는 과정이다. 따라서 $const_1 \times |C|$의 비용이 된다. 스텝 2~3는 $|C|$번, 스텝 2.1~2.2은 $|d|$번 반복하게 되고, 스텝 2.1.1은 배열 $TF_{t|c}$에서 t_k의 탐색하는 연산을 포함하고 있다. 이 비용은 해쉬를 사용하는 경우 $h(|TF^{avg}_{t|c}|)$이 된다. 여기서 $|TF^{avg}_{t|c}|$는 모든 분류에 대한 $TF_{t\ c}$의 평균 요소의 수를 의미하며, 대부분의 경우 $h(|TF^{avg}_{t|d}|)$는 상수 값이다. 또한 스텝 2.1.2는 단순한 치환 연산이므로 $const_2$라 할 수 있다. 따라서 스텝 2.1~2.2의 비용은 $(h(|TF^{avg}_{t|c}|)+const_2) \times |d|$이 되고, 스텝 2~3의 비용은 $(h(|TF^{avg}_{t|c}|)+const_2) \times |d|) \times |C|$가 된다. 따라서 전체 비용은 다음과 같이 표현된다.

$$T_{NB\_categorizer} = (const_1 + h(|TF^{avg}_{t|c}|) + const_2) \times |d|) \times |C| \quad (4)$$

$$T_{NB\_categorizer} = O((h(|TF^{avg}_{t|c}|) \times |d|) \times |C| \quad (5)$$

식 (4), 식 (5)에서 $|d|$는 분류될 문서에서 용어의 수이며, $|C|$는 전체 분류의 수이다. 이 식으로부터 알 수 있듯이 NaiveBayes 방법의 분류단계는 학습 문서의 수, 즉 $|D|$와는 전혀 관계가 없다.

3. 효율적인 NaiveBayes 알고리즘

3.1 행렬 전치를 이용한 알고리즘

NaiveBayes 방법은 kNN, SVM 등 다른 알고리즘에 비해 비교적 실행 속도가 빠른 방법이나 문서 분류 태스크에서 분류되어야 하는 문서가 급속히 증가하고 있으며, 또한 실시간으로 처리되어야 하는 환경으로 변하고 있기 때문에 효율적인 방법이 필요하다.기존 NaiveBayes 방법의 단점은 분류 단계에서 분류에 속한 모든 용어에 대하여 시험 문서에 속한 용어를 탐색하여 확률을 계산하기 때문에 비효율적이다. 이것은 극단적인 경우 시험 문서 d에 있는 어떠한 용어도 포함하지 않는 분류에 대해서도 탐색을 해야 하는 단점이 있다. 예를 들어 상기 <표 3>에서 c_1의 경우 용어 B를 포함하지 않는다는 사실을 사전에 알 수 있다면 NB_Categorizer의 스텝 2.1.1에서 B를 탐색 비용을 줄일 수 있을 것이다. 더욱이 c_3의 경우 시험 문서 d에 포함된 어떠한 용어도 포함하지 않는다는 사실을 알 수 있다면 단순히 $C_{rank}[c_3] = \log(2/5) + 40 \times \log(1/60)$만 계산함으로써 스텝 2.1~2.2를 보다 효율적으로 실행할 수 있을 것이다.

본 연구에서는 시험 문서 d와 상관없는 분류 또는 용어들은 전혀 탐색하지 않도록 하여 실행속도를 개선하는 새로운 알고리즘을 제안하기로 한다. <표 6>은 <표 4>에서 $TF(t_k, c_j)$ 부분만 전치한 것이다. 즉, <표 4>는 분류별 <용어, 가중치>의 목록이나 <표 6>는 용어별 <분류, 가중치>의 목록이다. <표 6>과 문서 d를 이용하여 <표 5>와 같은결과를 구할 수 있음을 설명함으로써 제안하는 방법을 설명하기로 한다. 시험 문서 d는 <표 2>에서와 같이 {<B,

〈표 6〉 〈표 4〉의 행렬에 대한 전치

용 어	분류 벡터<분류, 가중치>
A	$< c_2, 30 >< c_3, 20 >< c_4, 20 >$
B	$< c_2, 10 >< c_4, 20 >$
C	$< c_1, 30 >< c_2, 50 >< c_4, 30 >$
D	$< c_1, 10 >< c_2, 10 >$
E	$< c_1, 40 >< c_2, 50 >< c_3, 30 >< c_4, 10 >$
F	$< c_2, 30 >< c_4, 30 >$
G	$< c_1, 10 >< c_3, 10 >< c_4, 10 >$
H	$< c_1, 30 >< c_2, 70 >< c_4, 40 >$

20><C, 10><H, 10>}으로 구성되어 있다. 따라서 확률의 계산에 필요한 용어는 <표 6>에서 단지 B, C, H이다. 즉, 용어 A, D, E, F, G에 대해서는 어떠한 처리도 할 필요가 없다. 따라서 <표 6>에서 B, C, H와 관련된 벡터들만 연산을 하면 된다. <표 7>은 이 두 가지를 이용한 연산 결과를 보이고 있다.

〈표 7〉 용어 벡터를 이용한 확률 계산

용 어	c_1	c_2	c_3	c_4
B		20×log(11/258)		20×log(21/168)
C	10×log(31/128)	10×log(51/258)		10×log(31/168)
H	10×log(31/128)	10×log(71/258)		10×log(41/168)

<표 7>을 구하는 과정은 단순하다. 시험 문서 d의 요소 B, C, H를 순차적으로 읽으면서 이 용어에 대응하는 벡터를 <표 6>에서 찾는다. 찾아진 벡터의 각 요소 $\langle c_j, w_j \rangle$에 대하여 $\log((w_j+1)/S_c[c_j])$를 구한 후 d의 해당 용어의 가중치를 곱함으로써 구해진다. 예를 들어 d의 요소 <B, 20>에 대하여 <표 6>에서 {$< c_2$, 10>$< c_4$, 20>} 벡터를 찾고, 벡터의 요소 $< c_2$, 10>에 대하여 20×log(11/258)을 계산하여 c_2에 저장하고 $< c_4$, 20>에 대하여 20×log(21/168)를 계산하여 c_4에 저장하면 된다. 하지만 <표 7>은 <표 5>와 비교하여 두 가지가 부족하다. 첫 번째 확률 $P_c(c_j)$에 대한 값이 없다. 이것을 위하여 각 분류에 대하여 초기에 $\log P_c(c_j)$값을 더해 주어야 한다. 다른 하나는 <표 7>에서 보이듯이 공란(예 [B, c_1])에 대하여 라플라스 스무드 값으로 채워 넣어야 한다. 예를 들어 <표 7>에서 [B, c_1]에 대하여 20×log(1/128)을 넣어야 한다. 두 가지를 보충하면 최종적으로 <표 8>과 같은 결과를 얻을 수 있다.

〈표 8〉 용어 벡터를 이용한 최종 결과

	c_1	c_2	c_3	c_4
$P(c_j)$	log(2/5)	log(3/5)	log(2/5)	log(3/5)
B		20×log(11/258)		20×log(21/168)
C	10×log(31/128)	10×log(51/258)		10×log(31/168)
H	10×log(31/128)	10×log(71/258)		10×log(41/168)
라플라스 스무드	20×log(1/128)		40×log(1/68)	

<표 8>에서 각 분류별로 그 값을 합하면 그 결과는 <표 5>의 결과와 정확히 일치 한다. 다음은 상기 예제에서 설명한 내용에 기초한 행렬 전치를 이용하는 새로운 알고리즘이다. 이것을 FastNB_Categorizer라 부르기로 한다.

FastNB_Categorizer(입력 : $C[]$, $P_c[]$, $TF_{c|t}[]$, $S_c[]$, d,
출력 : 등급화된 $C_{rank}[\,]$)

스텝 1 : $C_{rank}[c_j] = \log(P_c[c_j])$foreach $c_j \in C$;

스텝 2 : $w_{total} = 0$;

스텝 3 : foreach $\langle t_k, w_k \rangle \in d$ {

스텝 3.1 : foreach $\langle c_j, w_j \rangle \in TF_{c|t}[t_k]$ {

스텝 3.1.1: $C_{rank}[c_j] += w_k \times \log(w_j+1)$;

스텝 3.2 : }

스텝 3.3 : $w_{total} += w_k$;

스텝 4 : }

스텝 5 : $C_{rank}[c_j] += w_{total} \times \log(1/S_c[c_j])$ foreach $c_j \in C$;

상기 알고리즘은 입력으로 $C[]$, $P_c[]$, $S_c[]$, $TF_{c|t}[]$ 및 d를 받는다. 여기서 $TF_{c|t}$는 <표 6>과 같이 용어별 <분류, 가중치>의 집합으로 구성되어 있다고 가정한다. 이것은 학습과정에서 단순히 행렬 전치만 더 실행하면 된다. 따라서 행렬 전치를 위하여 분류단계에서 어떠한 처리도 하지 않는다고 가정한다. 상기 알고리즘은 앞의 예제와 약간의 차이가 있다. 먼저 스텝 1은 모든 분류에 대하여 $P(c_j)$ 값으로 초기화하는 과정이다. 스텝 3~4는 시험 문서 d에 있는 모든 용어에 대하여 분류별 확률에 가중치를 곱하는 과정이다. 스텝 3.1.1에서 $w_k \times \log((w_j+1)/S_c[c_j])$이 아니라 $w_k \times \log(w_j+1)$로 하고, 스텝 5에서 $w_{total} \times \log(1/S_c[c_j])$를 더해 준 것은 프로그램 실행을 효율적으로 하기 위함이다. <표 9>는 FastNB_Categorizer에 대한 <표 6>의 실행

결과이다. <표 8>를 관찰하면 이것은 쉽게 <표 9>와 같이 고쳐질 수 있음을 알 수 있다.

〈표 9〉 수정된 용어 벡터를 이용한 최종 결과

구분	용어	c_1	c_2	c_3	c_4
스텝1	$\log P(c_j)$	log(2/5)	log(3/5)	log(2/5)	log(3/5)
스텝3,4	B		20×log(11)		20×log(21)
스텝3,4	C	10×log(31)	10×log(51)		10×log(31)
스텝3,4	H	10×log(31)	10×log(71)		10×log(41)
스텝5		40×log(1/128)	40×log(1/258)	40×log(1/78)	40×log(1/168)

여기서 40은 시험 문서 d에서 모든 가중치의 합으로 알고리즘에서 w_{total}과 같다.

3.2 성능 분석

이 장에서는 FastNB_categorizer의 효율성을 시간 복잡도 측면에서 분석하기로 한다. 스텝 1의 비용은 앞의 NB_Categorizer와 동일하며 따라서 $const_1 \times |C|$의 비용이 된다. 스텝 2는 단순한 상수 $const_2$이고, 스텝 3~4는 $|d|$번 반복하게 되고, 스텝 3.1~3.2는 $|TF_{c|t}^{avg}|$번 반복하게 된다. 여기서 $|TF_{c|t}^{avg}|$는 $TF_{c|t}$의 평균 요소의 수이다. 스텝 3.1의 $TF_{c|t}[t_k]$는 학습 문서 집합에 나타나는 전체 용어에서 t_k를 찾아야 하므로 해쉬를 이용하는 경우 $h(|TF_{c|t}|)$의 비용이 된다. $|TF_{c|t}|$는 $TF_{c|t}$에서 용어의 수이다. 스텝 3.1.1은 단순 상수 $const_3$의 비용이라 할 수 있고, 스텝 3.3도 상수 $const_4$의 비용이라 할 수 있다. 따라서 스텝 3~4의 비용은 $(h(|TF_{c|t}|) + const_3 \times |TF_{c|t}^{avg}| + const_4) \times |d|$가 된다. 스텝 5의 비용은 스텝 1의 비용과 동일하므로 $const_5 \times |C|$라 할 수 있다. 따라서 전체 비용은 다음과 같이 표현된다.

$$T_{FastNB\_categorizer} = (const_1 + const_5) \times |C| + (h(|TF_{c|t}|) + const_3 \times |TF_{c|t}^{avg}| + const_4) \times |d| \quad (6)$$

$$T_{FastNB\_categorizer} = O(|TF_{c|t}^{avg}|) \times |d| \quad (7)$$

상기 식 (7)은 $h(|TF_{c|t}|)$의 비용이 $const_3 \times |TF_{c|t}^{avg}|$보다 작다는 가정을 포함하고 있다.

4. 성능비교

4.1 시간 복잡도 비교

시간 복잡도를 이용하여 기존의 NaiveBayes와 제안하는 방법을 정확히 비교하는 것은 쉽지 않다. 따라서 본 절에서는 식 (4)와 식 (6)을 비교하는 것이 아니라 식 (5)와 식 (7)을 비교하는 것으로 한다. 또한 비교를 간단히 하기 위하여 해쉬를 사용하는 탐색 비용은 O(1)이라고 가정한다.

식 (5)와 식 (7)을 비교해보면 어느 것이 우수한지 쉽게 알 수 없다. 그러나 <표 4>에서 $TF_{t|c}$ 컬럼과 <표 6>를 비교하면 두 표에서 요소의 수는 동일하다는 것을 알 수 있다. 왜냐하면 <표 6>는 단지 <표 4>의 $TF_{t|c}$를 전치한 것이기 때문이다. 따라서 $|TF_{t|c}| \times |TF_{t|c}^{avg}| = |TF_{c|t}| \times |TF_{c|t}^{avg}|$의 관계가 성립한다. 또한 $|TF_{t|c}|$는 분류 집합의 수인 $|C|$와 같기 때문에 $|TF_{c|t}^{avg}| = |C| \times |TF_{t|c}^{avg}| / |TF_{c|t}|$이 성립하고, 이것을 이용하면 $T_{FastNB_categorizer}$를 다음과 같이 표현할 수 있다.

$$T_{FastNB\_categorizer} = O(|C| \times |TF_{t|c}^{avg}| / |TF_{c|t}| \times |d|) = O(|TF_{t|c}^{avg}| / |TF_{c|t}|) \times T_{NB\_categorizer} \quad (8)$$

식 (8)에서 $|TF_{t|c}^{avg}|$는 각 분류별 포함하는 용어 개수의 평균이고, $|TF_{c|t}|$는 모든 분류에 나타나는 용어의 개수이다. 따라서 $|TF_{t|c}^{avg}| \le |TF_{c|t}|$관계는 항상 성립한다. 이것의 의미는 상기의 가정하에서 제안하는 방법이 기존의 방법보다 효율적임을 의미한다.

4.2 실험적 비교

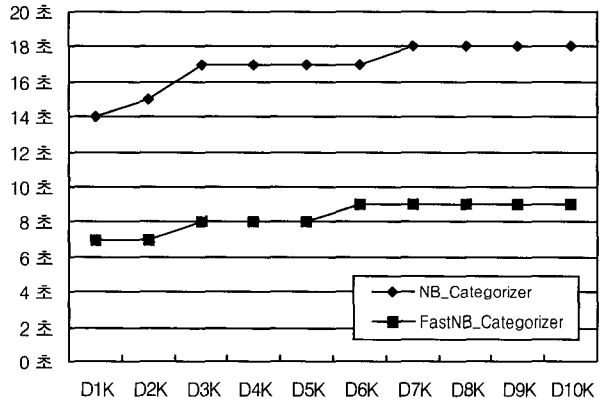
다음은 실험을 통하여 기존의 방법과 제안한 방법의 효율성을 비교해 보기로 하자. 실험을 위하여 사용된 데이터는 로이터-21578 ApteMod 버전[8]이다. 이것은 약 40M 바이트의 데이터로 10,788 뉴스 문서로 구성되어 있다. 전체 분류 집합에서 분류의 수는 90이다. NaiveBayes의 실행 시간을 측정하기 위해서 ApteMod는 하나의 시험 문서 집합과 다수의 학습 문서 집합으로 나누어 졌다. 시험 문서로는 ApteMod로부터 788개의 문서가 임의적으로 선택되었으며, 나머지 10,000개의 문서는 학습 문서로써 사용되었다. 학습 문서는 포함된 문서의 수에 따라 10 종류로 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 객체지향 문서 분류 프레임웍인 AI::Categorizer[10]을 이용하였다.

<표 10>은 학습 문서의 종류에 따른 분류 집합의 수($|C|$), 각 분류별 평균 용어의 수($|TF_{t|c}^{avg}|$) 및 전체 용어의

수($|TF_{c|t}|$)에 대한 통계정보 이다. 학습 문서가 증가함에 따라 세 가지 모두 증가하고 있으나 그 증가율은 점차적으로 감소하고 있음을 알 수 있다. 이것으로부터 학습 문서의 수가 증가함에 따라 실행 시간이 증가할 것이나, 점차적으로 증가 속도가 감소 할 것임을 예측할 수 있다.

〈표 10〉 학습 문서 집합에 따른 $|C|$, $|TF_{t|c}^{avg}|$, $|TF_{c|t}|$의 통계

실험데이터	D1K	D2K	D3K	D4K	D5K	D6K	D7K	D8K	D9K	D10K
$\|C\|$	68	75	83	86	86	86	90	90	90	90
$\|TF_{t\|c}^{avg}\|$	243	377	462	534	599	691	724	777	839	890
$\|TF_{c\|t}\|$	1293	1953	2470	2851	3213	3532	3830	4088	4321	4563



(그림 1) NB_Categorizer 및 FastNB_Categorizer의 실행 시간

(그림 1)은 실행 결과이다. x축은 학습 문서 집합을 나타내며, y축은 실행 시간을 나타내고 있다. 앞에서 예측한 것과 같이 학습 문서가 증가함에 따라 실행 속도가 약간씩 증가한다. 이는 학습 문서가 증가함에 따라 분류 집합의 수($|C|$)도 증가하지만, 분류 집합에 속하는 용어의 수도 증가하여 해쉬 비용이 작은 값이지만 증가하기 때문이다. (그림 1)로 부터 대부분의 경우 제안된 방법이 기존의 방법보다 약 2배 정도 우수한 성능을 보이고 있음을 알 수 있다. 그러나 식 (8)과 <표 10>에 의한 $|TF_{c|t}|/|TF_{t|c}^{avg}|$배 즉, 약 5배 정도의 성능 향상은 아니다. 이것은 (그림 1)에서 측정된 시간은 시험 문서를 디스크로부터 읽고 그리고 벡터화 등 모든 부수적인 시간을 포함하고 있기 때문이다. 상기 10가지 실험에서 두 방법의 정확도에 대하여 정밀도(Precision), 회귀도(Recall), F_1[9]를 각각 측정하였고, 이들이 모두 동일함을 알 수 있었다. 이것은 제안하는 방법이 기존의 방법과 동일한 정확도를 가진다는 것을 의미한다.

5. 결 론

이 논문에서 NaiveBayes 정확도의 손실 없이 효율적으로 동작하는 NaiveBayes에 대한 새로운 알고리즘을 제안하였다. 제안된 방법은 분류 벡터에 대한 행렬 전치를 사용하여 NaiveBayes에서 확률 계산을 최소화함으로써 효율적으로 동작하도록 하는 것이다.분류 벡터에 대한 행렬 전치는 학습단계에서 실행되도록 함으로써 분류단계에서 특별한 부가 작업이 없도록 제안하였다. 제안된 방법은 문서 분류 프레임웍인 AI :: Categorizer 상에서 구현되었으며, 잘 알려진 로이터-21578 데이터를 사용하여 기존의 NaiveBayes과 비교 되었다. 성능 비교의 결과로부터 제안된 휴리스틱을 적용한 방법이 기존의 NaiveBayes보다 실행 속도측면에서 약 2배 정도의 성능 개선 효과가 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, 4(5), 2000.

[5] Calvo R. A., "*Classifying financial news with neural networks*," In 6th Australian Document Symposium, page 6, Dec., 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., Dec., 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.

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)

1992~1994년 한국통신 연구개발단 선임연구원

1994~현재 한성대학교 컴퓨터공학부 부교수

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