FP-Tree를 이용한 문서 분류 방법
(Text Document Categorization using FP-Tree)

박 용 기 † 김 황 수 ††
(Yongki Park) (Hwangsoo Kim)

요 약 전자 문서의 급속한 증가로 인하여 자동 문서 분류의 필요성도 증가하고 있다. 기존의 문서 분류 방법들은 대개 문서를 단어의 집합으로 간주하여 기계 학습의 방법을 그대로 적용하거나 약간의 변형을 가한 방법들이 대부분이다. 본 논문에서는 데이타 마이닝 분야에서 사용되는 FP-Tree 구조를 이용하여 문서내의 문장들의 패턴을 저장하고 이를 사용하여 문서를 분류하는 방법(FPTC)을 제시한다.

또한 FP-Tree를 이용한 방법에 상호 정보량과 문장별 엔트로피를 적용하여 분류 정확도를 높이는 방법 그리고 각각의 실험 결과와 함께 다른 문서 분류 알고리즘과 비교 분석한 결과를 살펴보기로 한다.

키워드 : 문서자동분류, FP-Tree, 상호정보량, 엔트로피

Abstract As the amount of electronic documents increases explosively, automatic text categorization methods are needed to identify those of interest. Most methods use machine learning techniques based on a word set. This paper introduces a new method, called FPTC (FP-Tree based Text Classifier). FP-Tree is a data structure used in data-mining. In this paper, a method of storing text sentence patterns in the FP-Tree structure and classifying text using the patterns is presented.

In the experiments conducted, we use our algorithm with a 'Mutual Information and Entropy' approach to improve performance. We also present an analysis of the algorithm via an ordinary differential categorization method.

Key words : Text Document Categorization, FP-Tree, Mutual Information, Entropy

1. 서 론

인터넷의 급속한 발전으로 문서의 수, 특히 웹 문서의 수는 폭발적으로 늘어나고 있다. 이러한 상황에서 짧은 시간에 정확한 분류가 가능한 자동화된 문서 분류방법이 요구된다.

본 논문에서는 빠르고 정확하게 분류하기 위하여 텍스트를 단어들의 집합으로 간주한 채 분류하는 기존의 기계 학습적인 방법(TF·IDF, Bayes[1], Decision Tree, k-NN[2], AdaBoost[3], SVM[4] 등)과 달리, 문서를 문장들의 집합으로 바라보고 데이타 마이닝에서 빈번한 패턴을 찾는 데 사용되는 FP-Tree라는 방법을 사용하여 문서를 분류하는 방법을 제시한다.

기존의 방법들과는 달리 단어가 아닌 문장을 분류의 단위로 선택한 것은 단어의 정보량이 각 음절의 정보량의 합보다 크듯, 문장의 정보량이 각 단어의 정보량의 합보다 클 것이라는 가정 아래, 문장을 수치적으로 비교 가능하도록 패턴화한다면 더욱 정확한 분류가 이루어질 것이라 예상하였기 때문이다.

논문의 구성은 2절에서 FP-Tree를 소개하고, 3절에서 이를 이용하는 문서 분류 방법, 그리고 더 나은 결과를 위하여 상호 정보량과 문장별 엔트로피를 이용하는 방법을 알아보고, 4절에서 reuters-21578 문서 집합을 통해 FP-Tree를 이용한 문서 분류의 정확도와 이를 여러 가지 다른 알고리즘과 비교한 결과를 살펴본다.

2. 관련 연구

기계 학습 관점에서는 다양한 방법이 문서 분류에 사용되었는데, 대표적인 것이 k-NN[2], SVM(support vector machine)[4], Bayes[1] 등이다. 최근에는 여러 개의 분류 방법을 묶어서 추정을 하고자 하는 시도가 이루어지고 있는데, 이에는 Committee Machine, Boosting, Bagging 등이 있다. Schapire 등은 AdaBoost와 Rocchio를 REUTER-21578과 TREC-3에 대해서 비교하고, LF1측정 방법에 대해서 두 방법이 뛰어남을 보였다[3].

† 학생회원 : 경북대학교 컴퓨터과학과
ykpark@cs.knu.ac.kr
†† 종신회원 : 경북대학교 컴퓨터과학과 교수
hsk@knu.ac.kr
논문접수 : 2006년 1월 31일
심사완료 : 2007년 8월 21일

3. FP-Tree (Frequent Pattern - Tree)

FP-Tree란 빈번히 발생하는 패턴의 중요한 정보를 효율적으로 압축, 저장하는 자료구조이다. 예를 들어, 마트에서 한 사람이 한 번에 구매한 물품의 목록이 저장된 데이타베이스를 자세히 살펴보면 아마도 '누군가 빵을 산 경우에는 우유를 같이 사는 경향이 많다.'와 같은 구매 패턴을 발견할 수 있을 것이다. FP-Tree는 바로 이와 같은 빈번히 발생하는 패턴을 쉽게 찾아내는 방법이다. FP-Tree는 J. Han 등이 효율적인 데이타 마이닝을 위해 제안하였으며[5] Prefix-tree를 확장한 형태이다.

3.1 FP-Tree 생성하기

그림 1은 트랜잭션 데이타베이스의 한 예이다. 'TID'는 트랜잭션의 ID이고, 'Items bought'는 한 번에 구입한 물품의 목록이고, '(ordered) frequent items'는 트랜잭션 데이타베이스의 모든 item들을 많이 나온 순으로 정렬한 다음 모든 트랜잭션에 50% 이상 나타나는 item들로만 나타낸 것이다.

이렇게 만든 frequent items로 그림 2의 FP-Tree를 구성하게 된다. FP-Tree를 구성하는 방법은 간단하다. 우선 데이타베이스 내의 frequent items를 많이 나타난 순으로 정렬하여 'Header Table'을 만들고, 각 트랜잭션별 frequent items를 그 순서를 유지한 채 트리 구조에 넣으면 된다.

완성된 FP-Tree로부터 Frequent Pattern을 찾는 방법은 다음과 같다. 예를 들어 item 'b'가 들어간 Frequent Pattern을 찾는다면 그림 2 'Header Table'의 item 'b'의 link를 따라 가면서 각각의 부모 노드를 살펴보면 된다. 그 결과는 { (b, a, c, f : 1), (b, f : 1), (b, c : 1) } 이 나오게 되고, 이 결과를 종합하면 b는 item 'f'와 2번, item 'c'와 2번, item 'a'와 1번 함께 나타나는 것을 알 수 있다.

TID	*Items bought*	*(ordered) frequent items*
100	{*f, a, c, d, g, i, m, p*}	{*f, c, a, m, p*}
200	{*a, b, c, f, l, m, o*}	{*f, c, a, b, m*}
300	{*b, f, h, j, o*}	{*f, b*}
400	{*b, c, k, s, p*}	{*c, b, p*}
500	{*a, f, c, e, l, p, m, n*}	{*f, c, a, m, p*}

그림 1 쇼핑몰 트랜잭션 데이타베이스 예제

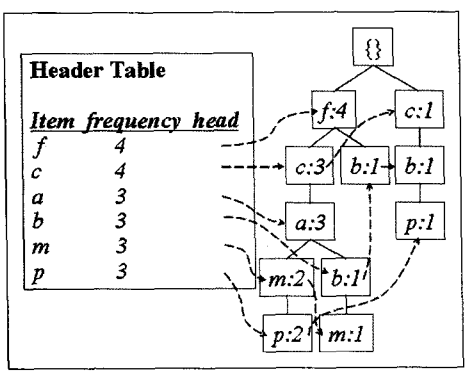


그림 2 FP-Tree (그림 1 예제로 부터)

4. FP-Tree를 이용한 문서 분류 방법(FPTC : FP-Tree based Classifier)

분류 대상이 쇼핑 상품이 아니라 문서일 경우에는 2절의 예제에서 물품 하나를 한 단어로, 트랜잭션 하나를 하나의 문장으로 보고 생성하면 된다. 즉, 하나의 카테고리에 속하는 모든 학습 문서들이 그 카테고리에 대한 FP-Tree 하나를 만들게 된다. 그리고 각 테스트 문서들을 각각의 FP-Tree와 비교하여 점수를 얻고 그 테스트 문서의 카테고리는 이 점수가 가장 큰 것이 된다.

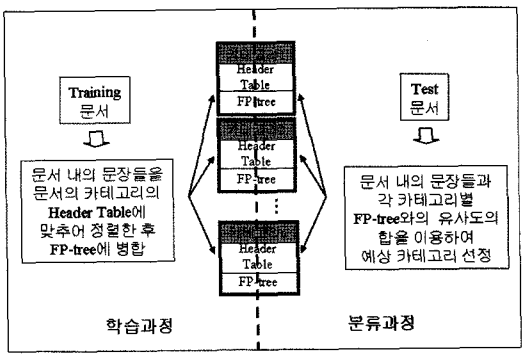


그림 3 문서의 학습과 분류 과정

4.1 전처리

학습 문서를 FP-Tree에 저장하거나, 테스트 문서를 FP-Tree와 비교하기 전에 거쳐야 할 작업이 전처리 과정이다. 전처리 과정을 거치지 않으면 문서는 캐릭터들의 의미 없는 나열에 불과하기 때문이다.

우선 하나의 문서가 들어오면 문장들로 나눈다. 그 다음 문장 내의 단어들을 추출한다. 추출된 단어에서 Stopword[6]는 제외한다.

특히, 영어 단어의 경우에는 의미는 같지만 형태가 다른 경우가 있다. 이러한 단어는 다른 단어로 구분하는 것보다 단어의 원형으로 통일하여 하나의 단어로 구분하는 것이 더욱 효율적이며 정확한 방법일 것이다. 따라서 본 논문에서는 Princeton University의 Cognitive Science 연구실에서 개발한 영문 어휘 사전인 Wordnet [7] (v 2.0)을 이용하여 단어의 원형을 얻었다.

4.2 문서들을 각 카테고리 FP-Tree에 넣기

2절에서 살펴 본 FP-Tree 알고리즘에서는 트리를 생성하기 전에 데이타베이스를 한번 읽어 들여서 모든 물

품들의 빈도수를 구하여 이를 바탕으로 'Header Table'을 만들었다.

트랜잭션 데이타베이스의 경우에는 모든 트랜잭션들이 하나의 데이타베이스에 있기 때문에 이 방법도 큰 무리가 없지만, 문서의 경우에는 수많은 문서가 있고, 또 이 수많은 문서 각각에 수많은 문장들이 존재하기 때문에 'Header Table'을 처음부터 완성하는 것은 가능은 하겠지만 그에 따르는 비용이 너무 크다.

따라서, 본 논문에서는 기존의 방법에서 약간 변형된 incremental FP-Tree 생성법(그림 4)을 사용한다. 새로운 방법에서는 첫 학습 문서에 나타난 단어들을 빈도순으로 정렬하여 'Header Table'을 만들고 그 순서에 맞게 문서 내 각 문장의 단어들을 정렬하여 FP-Tree를 생성한다. 이후의 각 학습 문서는 이미 만들어진 'Header Table'에 있는 단어는 그 빈도수를 더하여 주고, 새로운 단어이면 새로운 단어들의 빈도순으로 기존 'Header Table'의 제일 밑에 덧붙여 준다. 그리고 이 순서로 각 문장의 단어들을 정렬하여 기존의 FP-Tree에 덧붙여 준다. 단, 이때 각 문서 내에서 2% 이상 나타난 단어만을 사용하였다.

이렇게 함으로써 첫 문서에서의 문장이나, 나중에 쓰인 문서에서의 문장이나 단어들의 순서는 일관성 있게 유지된다. 결과적으로 문장의 패턴은 그대로 유지한 채 모든 학습 문서들을 검색하여 'Header Table'을 생성하는 과정이 생략되어 속도의 향상을 가져올 수 있다.

다만 이 경우에는 'Header Table'의 단어들이 빈도순으로 정확히 정렬되어 있지 않기 때문에 생성된 FP-Tree의 크기는 기존의 FP-Tree에 비해 같거나 커진다.

간단히 말해서 변형된 FP-Tree 생성법은 'Header Table'을 동적으로 그때그때 덧붙여 가며 만드는 방법이며, 따라서 자연히 '(ordered) frequent items'에서의 단어(즉, 물품) 순서가 기존의 FP-Tree 알고리즘과 다르게 된다.

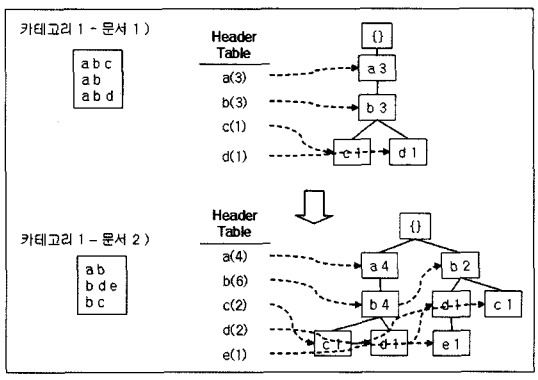


그림 4 변경된 FP-Tree 생성법

```
점수 = 0;
가중치 = W;
단어 = 문장의 마지막 단어;
while(문장의 첫 단어 전까지)
{
  노드 = Header Table[단어]->link;
  빈도 = 노드.빈도 * 0.01;

  while(노드 != null)
  {
      현 노드로부터 FP-Tree의 root까지의
      모든 상위 노드를 구한다.
      구한 모든 노드의 각 단어들과
      입력받은 문장 중에서 현재 단어부터
      첫 단어까지의 공통된 단어 수를 구한다.

      점수 = 점수 + ( 빈도 * (공통된 단어 수^가중치) );
      노드 = 노드->link;
   }
   단어 = 문장에서 지금 단어의 이전 단어;
}
점수 = 점수 +1;
// 점수가 0이 되지 않도록
점수 = log2(점수);
return 점수;
```

그림 5 문장, FP-Tree 간의 유사도 알고리즘

4.3 문서와 FP-Tree 와의 유사도 구하기

학습 문서가 저장된 FP-Tree는 그 구조상 학습 문서들이 가지고 있던 문장들의 패턴을 정보성의 측면에서 보았을 때 거의 완전하게 가지고 있다. 따라서 문서와 FP-Tree 와의 유사도를 구하려면 문서의 각 문장의 패턴이 FP-Tree에 얼마나 자주 나타나는지와 얼마나 유사한 패턴으로 나타나는지를 조사하면 된다.

본 논문에서는 얼마나 자주 나타나는지의 정도는 해당 패턴의 빈도로써 수치화하였고, 얼마나 유사한 패턴으로 나타나는지의 정도는 패턴의 길이, 즉 일치하는 단어의 수로 수치화하였다.

하나의 문장과 FP-Tree 간의 유사도를 구하는 알고리즘은 아래와 같다.

위 알고리즘을 통하여 우리는 테스트 문서의 하나의 문장과 FP-Tree 간의 유사도를 구할 수 있다. 특히, FPTC에서의 유사도란 현재 주어진 문장의 패턴이 기존에 학습된 문장들의 정보가 저장된 FP-Tree에 얼마나 자주 출현하는지, 또 자주 출현한다면 얼마나 많은 단어가 함께 출현하는지를 측정한 값을 의미한다.

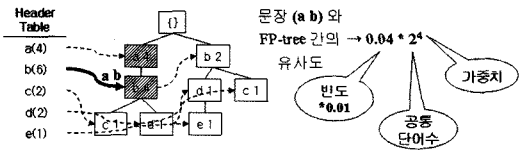
예를 들어, 그림 4와 같은 두 학습 문서 {(a b c), (a b), (a b d)}, {(a b), (b d e), (b c)}로부터 생성된 FP-Tree가 있고, 이와 하나의 테스트 문서 {(b, a, f), (b, g, d, e)} 사이의 유사도를 구하면 다음과 같다.

우선 테스트 문서 각 문장의 단어들을 'Header Table'의 순서 {a, b, c, d, e}에 맞게 {(a, b), (b, d, e)}로 바꾸어 준다. 이 때 당연히 'Header Table'에 없는 단어는 쓰이지 않는다. 그 뒤 각 문장과 FP-Tree와의 유사도

를 구하여 합한 값이 테스트 문서와 FP-Tree 사이의 유사도가 된다.

위 알고리즘에 따라 테스트 문서 중에서 문장 (a, b)와 FP-Tree 사이의 유사도를 구하는 순서는 다음과 같다. 참고로 가중치 W는 4로 두었다.

1.

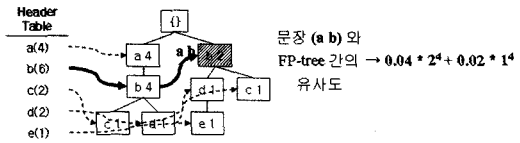


문장 (a, b)중 마지막 단어 b의 'Header Table'에서의 연결 링크를 따라 가면 FP-Tree의 좌측 가지의 중간 노드 (b:4)를 만난다.

(b:4)와 이 노드의 모든 상위 노드를 구하면 {(b:4), (a:4)}, 단어 'b'에서 문장의 첫 단어 'a'까지 모든 단어는 (b, a). 둘 사이의 공통 단어 수는 2이고, 빈도는 (b:4)의 빈도수 4*0.01=0.04가 되어 현재 점수는

$$0 + (0.04 * (2^4)) = 0.64$$

2.



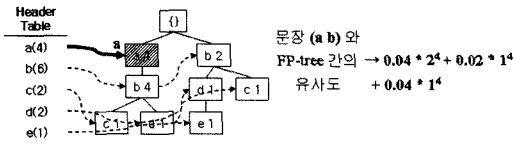
(b:4)에서 다음 연결 링크를 따라 가면 FP-Tree 우측 가지 상단 노드 (b:2)를 만난다.

(b:2)는 제일 상위 노드, 단어 'b'에서 문장의 첫 단어 'a'까지 모든 단어는 (b, a). 둘 사이의 공통 단어 수는 1이고 빈도는 (b:2)의 빈도수 2*0.01=0.02가 되어 현재 점수는

$$0.64 + (0.02 * (1^4)) = 0.66$$

3. 더 이상 연결 링크가 없기 때문에 다음 단어 'a'로 넘어간다.

4.



단어 a의 'Header Table'에서의 연결 링크를 따라 가면 FP-Tree의 좌측 가지의 상위 노드 (a:4)를 만난다. (a:4)는 제일 상위 노드, 단어 'a'는 문장의 첫 단어. 둘 사이의 공통 단어 수는 1이고 빈도는 (a:4)의 빈도수 4*0.01=0.04가 되어 현재 점수는

$$0.66 + (0.04 * (1^4)) = 0.7$$

같은 방식으로 테스트 문서의 두 번째 문장 (b, d, e)의 유사도를 구하면 1.13이 나오고 이 값을 첫 문장의 유사도와 더하면 테스트 문서와 FP-Tree 사이의 유사도 1.88이 나온다.

위와 같은 방법으로 테스트 문서와 각각의 카테고리 FP-Tree 사이의 유사도를 구하여 가장 높은 유사도를 가진 카테고리를 테스트 문서의 예상 카테고리로 분류한다.

여기에서 log2를 취한 이유는, log2를 취하지 않을 경우 각 단어의 빈도가 실제 유사도에 너무 크게 작용하여 결과에 악영향을 미칠 수 있기 때문이다. 예를 들어, 20 Newsgroups 문서 집합에서 Computer 관련 5개의 그룹에서는 당연히 'computer'라는 단어가 많이 나오기 마련인데 이러한 단어가 나온 문장의 점수가 다른 문장의 점수보다 월등히 높아져서 실제 분류에 부정적으로 작용하는 경우가 많았다.

4.4 상호 정보량의 이용

위 4.3절에서는 가장 기본적이고 간단한 방법으로 문서와 각 카테고리 별 FP-Tree 사이의 비교 점수를 구하는 방법을 제시하였다. 이 절에서는 문서 분류의 정확도를 향상시키기 위하여 상호 정보량(Mutual Information)[9]을 이용하는 방법을 제시한다.

상호 정보량은 두 랜덤 변수가 있을 때, 두 랜덤 변수가 서로 얼마나 밀접한가를 수치화하여 보여준다. 만약, 두 랜덤 변수가 상호 독립이라면 자연히 상호 정보량의 값은 0이 될 것이고, 두 랜덤 변수가 일기 예보와 실제로 비가 올 확률처럼 상관이 큰 경우에는 상호 정보량의 값은 커질 것이다. 두 랜덤 변수 X, Y간의 상호 정보량은 다음과 같은 수식으로 계산된다.

$$I(X;Y) = \sum_{x,y} P(X=x, Y=y) \log_2 \frac{P(X=x, Y=y)}{P(X=x)P(Y=y)} \quad (1)$$

위 수식에서 두 랜덤 변수 X, Y를 단어와 카테고리로 치환하여 계산하면 학습 문서에 나타난 단어들과 카테고리들 간의 상호 정보량을 구할 수 있다. 즉, 단어 집합과 카테고리 집합 사이의 상관도를 수치적으로 알 수 있다.

본 논문에서 필요한 수치는 단어 집합과 카테고리 집합 사이의 상관도가 아니라 하나의 단어와 하나의 카테고리 간의 상관도이다.

Fano에 의하면 하나의 단어와 하나의 카테고리 간의 상호 정보량[9]은 두 변수(단어 w, 카테고리 c)가 확률값 P(w), P(c)를 가질 때 다음 수식으로 정의할 수 있다.

$$I(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)} \quad (w:\text{한 단어}, c:\text{한 카테고리}) \quad (2)$$

실제 사용에 있어서는 식 (2)는 아래식과 같이 근사화하여 사용하였다.

$$I(w,c) \approx \log_2 \frac{A^*N}{A^*C} \quad (3)$$

A	w 이고 c 인 문서수
B	w 인 문서수
C	c 인 문서수
N	전체 문서수

식 (2)에서 각 확률 P(w)는 (단어 w가 나타난 문서수/전체 문서 수), P(c)는 (카테고리 c에 속한 문서 수/전체 문서 수), P(w,c)는 (카테고리 c에 속한 문서 중 단어 w가 나타난 문서 수/전체 문서 수)로 근사화 하였고, 이는 식 (3)으로 유도 된다.

실제로 상호 정보량과 FP-Tree를 이용하여 문서를 분류하기 위해서는 우선 각 카테고리별 'Header Table'에 들어있는 각 단어들과 소속 카테고리간의 상호 정보량을 식 (3)의 근사식을 이용하여 구한 후, 이를 'Header Table'에 저장해 두어야 한다.

저장된 상호 정보량을 FP-Tree 내에서 단어별로 사용하는 방법은 다음과 같다. 위 4.3절의 알고리즘에서

```
점수 = 점수 + ( 빈도 * (공통된 단어 수^가중치) )
```
(4)

부분의 '공통된 단어 수'는 각 단어의 단위가 모두 1이라고 볼 수 있다. 따라서 각 단어의 단위를 상호 정보량으로 치환하여 다음과 같은 식으로 수정하여 사용하였다.

```
점수 = 점수 + ( 빈도 * (공통된 단어들의 상호
정보량의 합^가중치) )
```
(5)

4.5 문장 별 엔트로피의 적용

문서를 분류함에 있어서 분류의 척도가 되는 문장, 분류에 기여하는 문장은 분명 문서의 모든 문장은 아닐 것이다. 만약, 분류에 순영향을 미치는 문장들만으로 자동 분류를 한다면 분류의 정확도는 더욱 높아질 것으로 예상된다.

따라서, 본 논문에서는 문서 내 문장 각각의 엔트로피[8]를 구하고, 이 값을 각 문장이 분류에 미치는 영향의 정도로 판단하여, 이 값을 토대로 해당 문장을 분류를 하는데, 즉 FP-Tree와의 유사도를 구하는데 사용할 것인가를 결정할 것이다.

엔트로피는 '물질계의 열적 상태를 나타내는 물리량'을 의미한다. 이러한 엔트로피는 Shannon(1948)에 의해 정보량을 나타내는데 쓰이게 되었다. 엔트로피 값을 문서 분류에 적용하는 과정은 다음과 같다.

예를 들어, 한 테스트 문서에서 문장1과 5개의 카테고리별 FP-Tree와의 유사도를 각각 10, 1, 2, 1, 1이라고 하고, 문장2와 5개의 카테고리별 FP-Tree와의 유사도를 각각 3, 2, 4, 3, 1이라고 하자.

이때 두 문장의 엔트로피를 각각 구해보자. 우선 랜덤변수 X의 엔트로피의 정의는 다음과 같다.

$$H(X) = \sum_x P(X=x)\log_2 \frac{1}{P(X=x)} \quad (6)$$

본 논문에서는 문장 S의 엔트로피는 문장 S와 각 카테고리 c간의 유사도를 이용하여 구한다. 우선 위 식 (6)의 랜덤 변수 X를 문장 S로 치환하고, P(X=x)를 문장 X가 특정 문장 X=x가 될 확률을 계산하는 대신 하나의 문장 S와 하나의 카테고리 c 간의 유사도의 비율, 즉 문장 S와 카테고리 c 와의 유사도를 문장 S와 모든 카테고리와의 유사도들을 합한 값을 나눈 값으로 치환하여 사용하였다.

$$H(S) = \sum_c \frac{SIM(S,c)}{\sum_{ci} SIM(S,ci)} \log_2 \frac{\sum_{ci} SIM(S,ci)}{SIM(S,c)} \quad (7)$$

※ SIM(S,c) : 문장 S와 카테고리 c 간의 유사도. 본 논문에서는 3.3절의 알고리즘을 이용하여 구하는 방법, 3.4절의 FPTC에 상호 정보량을 이용하여 구하는 방법이 각각 쓰인다.

위 식 (7)은 식 (6)의 변형된 형태로 본 논문에서 실제 사용된 수식이다.

문장1의 유사도의 총 합은 15, 문장2의 유사도의 총 합은 13으로 하였을 때, 위 식 (7)을 사용하여 두 문장의 엔트로피를 각각 구하면,

문장 1의 엔트로피 =

$$\frac{10}{15}\log_2\frac{15}{10} + \frac{1}{15}\log_2\frac{15}{1} + \frac{2}{15}\log_2\frac{15}{2} + \frac{1}{15}\log_2\frac{15}{1} + \frac{1}{15}\log_2\frac{15}{1} = 1.56$$

문장 2의 엔트로피 =

$$\frac{3}{13}\log_2\frac{13}{3} + \frac{2}{13}\log_2\frac{13}{2} + \frac{4}{13}\log_2\frac{13}{4} + \frac{3}{13}\log_2\frac{13}{3} + \frac{1}{13}\log_2\frac{13}{1} = 2.20$$

문장1의 엔트로피가 문장2의 엔트로피보다 더 낮음을 알 수 있다. 이는 다시 말해서 문장1이 실제 분류에 기여하는 정도가 문장2가 실제 분류에 기여하는 정도보다 더 크다는 의미이다.

이렇게 한 문장이 각각의 카테고리 별 FP-Tree와 가지는 유사도를 이용하여 그 문장의 엔트로피를 구한다면, 그 엔트로피가 낮을수록 그 문장의 분류 기여도는 더 크다고 볼 수 있다. 이 점을 이용하여 한 문서에서 임의의 비율을 정하여, 각 문장들의 엔트로피를 낮은 순으로 정렬하여 앞에서 정한 임의의 비율 밖의 문장들은 분류에 사용하지 않았다.

5. 실험 및 고찰

이 절에서는 앞에서 살펴본 FP-Tree를 이용한 문서 분류 방법, FPTC와 상호 정보량을 이용한 방법, FPTC와 문장별 엔트로피를 이용한 방법 각각의 실험 결과와 함께 다른 문서 분류 알고리즘과 비교 분석한 결과를 살펴본다.

참고로 실험은 펜티엄 4 2.0GHz, 512MB, 윈도우 XP 상에서 Program Language C#을 통해 이루어졌다.

본 논문에서는, FP-Tree를 이용한 문서 분류의 성능을 알아보기 위해 실험용 문서 데이타로 reuters-21578 문서 집합[10]을 사용하였다. reuters-21578 문서 집합은 로이터 통신의 기사 21578개를 모은 것으로 David Lewis 등이 총 135개의 중복이 허용되는 Topic으로 분류한 문서 집합으로, 문서 분류에서 가장 널리 쓰이는 문서 데이타이다. 본 논문에서는 이중 상위 10개의 Topic만을 사용하였다.

표 1은 reuters-21578 문서 집합의 상위 10개 토픽의 분류 정확도를 여러 가지 다른 알고리즘과 비교[11]한 값이다.

FPTC를 이용한 reuters-21578 문서 집합 분류의 경우 대체로 기존의 알고리즘이 보여준 분류 정확도와 비슷하거나 상회하는 것으로 나타났다.

6. 결 론

FPTC의 경우 여타의 알고리즘과는 달리 분류를 위한 단위로 문장 간의 유사도를 사용한다. 따라서 문장의 패턴에 따라 기존의 알고리즘과는 다른 결과를 보여준다. 특히, 기존 알고리즘 중 가장 높은 정확도를 보여준 Linear SVM과 단순한 FPTC의 분류 정확도를 살펴보면, trade 토픽의 경우 FPTC가 10%가량 정확도가 높고, corn 토픽의 경우 10% 가량 정확도가 낮은 결과를 보여준다. 이는 corn 토픽의 경우 grain, wheat 토픽과 유사한 문장 패턴이 많기 때문으로 풀이 된다.

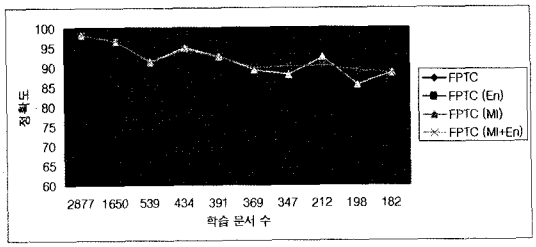


그림 6 reuters-21578 학습 문서 수와 정확도

그러나, 문장의 패턴이 다른 토픽와의 많은 차이를 보이지 않더라도 상호정보량과 문장별 엔트로피와 같은 보조 수단을 통해 손실을 거의 보상할 수 있다.

한 가지 흥미있는 사실은 그림 6에서 볼 수 있듯이, FPTC의 각 토픽 별 정확도의 순서가 거의 학습 문서 수의 순서와 일치한다는 사실이다. 이는 FPTC가 대용량의 문서 분류에 더욱 강한 문서 분류 방법이라는 기대를 가지게 한다.

참 고 문 헌

[1] D.D.Lewis, An evaluation of phrasal and clustered representations on a text categorization task, In Proceedings of SIGIR-92, pages 37-50, 1992.

[2] W.Lam, C.Y.Ho, Using a generalized instance set for automatic text categorization, In Proceedings of SIGIR-98, pages 81-89, 1998.

[3] R.E.Schapire, Y.Singer, BoosTexter: a boosting-based system for text categorization, Mach. Learn. 39 2000.

[4] T.Joachims, Text categorization with support vector machines: learning with many relevant features, In Proceedings of ECML-98, pages 137-142, 1998.

[5] Jiawei Han, Jian Pet, Yiwen Yin Runying Mao, Mining Frequent Patterns without Candidate Generation, Data Mining and Knowledge Discovery 2004.

[6] Gerard Salton, Chris Buckley, 571 stopword list

표 1 reuters-21578 상위 10개 토픽의 분류 정확도 (FPTC와 그 외 방법들 비교)

	FPTC	FPTC (En)	FPTC (MI)	FPTC (MI+En)	FindSim	NBayes	C4.5	Linear SVM
acq	96.2	96.3	96.6	96.8	64.7	87.8	89.7	93.6
crude	89.1	89.1	92.7	92.6	70.1	79.5	85.0	88.9
earn	97.3	97.4	98.2	98.2	92.9	95.9	97.8	98.0
grain	92.2	92.6	94.9	94.5	67.5	78.8	85.0	94.6
interest	83.9	83.9	88.1	90.2	63.4	64.9	67.1	77.7
money-fx	87.6	88.3	91.4	91.3	46.7	56.6	66.2	74.5
trade	85.4	85.8	89.2	89.9	65.1	63.9	72.5	75.9
corn	78.3	78.3	88.5	88.4	48.2	65.3	91.8	90.3
ship	81.9	81.9	85.5	89.2	49.2	85.4	74.2	85.6
wheat	84.5	83.8	92.5	90.4	68.9	69.7	92.5	91.8
Total	93.3	93.5	95.3	95.5	64.6	81.5	88.4	92.0

for the experimental SMART information retrieval system at Cornell University http://www.lextek.com/manuals/onix/stopwords2.html.
[7] G.A. Miller, WordNet: A Dictionary Browser, 1st Int'l Conf. Information in data 1985.
[8] David J.C. Mackay, Information Theory, Inference, and Learning Algorithm. Cambridge University Press 2003.
[9] Yiming Yang and J. O. Pedersen, A Comparative Study on Feature Selection in Text Categorization, Proceedings of the 14th International Conference on Machine Learning pages 412-420 1997.
[10] D.D.Lewis 'Reuters-21578' http://www.research.att.com/~lewis.
[11] S. T. Dumais, J. Platt, D. Heckerman, M. Sahami, Inductive learning algorithms and representations for text categorization. Proceedings of ACM-CIKM98 pages 148-155, 1998.

박 용 기
1999년 경북대학교 자연대학 컴퓨터과학과 학사. 2004년 경북대학교 자연대학 컴퓨터과학과 석사

김 황 수
1975년 서울대학교 공과대학 전기공학과 학사. 1988년 미시간 대학 석·박사. 1989~현재 경북대학교 전자전기컴퓨터학부 교수