FP-Tree를 이용한 문서 분류

박용기° 김황수
경북대학교 컴퓨터과학과 계산지능연구실
ykpark@cs.knu.ac.kr° hsk@knu.ac.kr

Text Document Categorization using FP-Tree

Yongki Park° Hwangsoo Kim
CIMV Lab., Dept. of Computer Science, KyungPook University

요 약

기존의 문서 분류 방법들은 대게 기존의 기계 학습의 방법을 그대로 가져오거나 문서라는 데이터에 맞춰 약간의 변형을 가한 방법들이 대부분이다. 본 논문에서는 기존의 방법에서 벗어나 데이터 마이닝 분야에서 쓰이는 FP-Tree 방법을 이용하여 문서내의 문장들의 패턴을 저장하고 이를 사용하여 문서 분류를 하는 방법을 소개한다.

1. 서 론

인터넷의 급속한 발전으로 문서의 수, 특히 웹 문서의 수는 기하급수적으로 늘어났고, 또 지금도 늘어나고 있다. 이러한 상황에서 좋은 문서 분류 방법은 무엇일까? 당연한 이야기이지만 짧은 시간에 정확한 분류가 가능한 방법이 가장 좋은 방법일 것이다.

본 논문에서는 빠르고 정확한 분류를 해내기 위하여 기존의 기계 학습적인 방법(Bayes, Decision Tree, SVM 등)이 아니라 FP-tree라는 데이터 마이닝에서 Frequent Pattern을 찾는 데 사용되는 방법을 사용해 문서를 분류하는 방법을 소개할 것이다.

FP-tree를 이용한 문서 분류 방법(FPTC : FP-tree based Classifier)에서는 각 문서내의 하나 하나의 문장이 비교에 있어서 기본 단위로 사용된다. 즉, 각 문서의 문장들을 카테고리 별로 FP-Tree 형태로 저장하고, 테스트 문서들의 문장 각각을 카테고리 별 FP-Tree 와 비교하여 유사도를 측정하게 된다.

앞으로 FP-Tree 의 소개 (2장)와 함께 이를 이용하여 어떻게 문서를 분류하는지(3장), 그 분류 결과는 어떠한지(4장), 마지막으로 앞으로 할 일(5장)을 살펴볼 것이다.

2.FP-Tree

FP-Tree 알고리즘[1][2]은 방대한 데이터베이스를 FP-tree (Frequent-Pattern tree) 구조로 압축 변형하여, 데이터베이스 상에서 자주 나오는 패턴을 쉽고 빠르게 찾아내는 알고리즘이다.

예를 들면, 쇼핑몰에서 한 사람 한 사람이 한 번에 산 물품의 목록을 조사하여 '빵과 우유는 함께 사는 경향이 많다.' 등의 구매 패턴을 찾아내는 것이다.

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}

Figure 2.1 트렌젝션 데이터베이스 예제

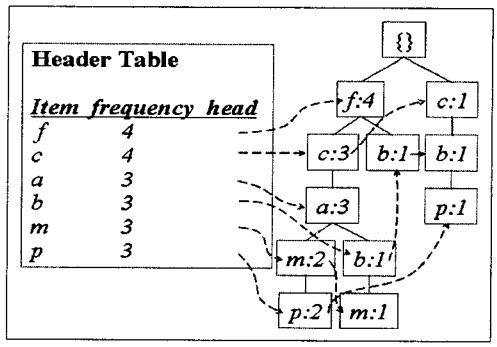

Figure 2.2 FP-tree (Figure 2.1 예제)

실제로 FP-tree가 어떻게 동작하여 Frequent Pattern을 찾아내는지 알아보자.

Figure 2.1은 transaction database의 한 예이다. TID는 한 번에 구입한 transaction의 ID이고, Items bought는 한 번에 구입한 물품의 목록이고, (ordered) frequent items는 transaction database의 모든 item들을 많이 나온 순으로 정렬한 다음 모든 transaction에 50% 이상 나타나는 item들로만 다시 나타낸 것이다.

이렇게 만들어진 frequent items로 Figure 2.2의 FP-tree를 구성하게 된다. FP-tree를 구성하는 방법은 간단하다. 우선 database내의 frequent items를 많이 나타난 순으로 정렬하여 Header Table을 만들고, 각

transaction 별 frequent items를 그 순서를 유지한 채 tree 구조에 넣어주면 된다.

완성된 FP-tree로부터 Frequent Pattern을 찾는 건 쉬운 일이다. 만약, item 'b'가 들어간 Frequent Pattern을 찾는다면 Figure 2.2 Header Table의 item 'b'의 link를 따라 가면서 각각의 부모 노드를 살펴보면 된다. 실제로 살펴보면 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.

3.FP-tree를 이용한 문서 분류 방법(FPTC:FP-tree based Classifier)

실제 문서를 FP-tree에 넣는 방법은 간단하다. item을 한 단어로 보고 transaction을 하나의 문장으로 보고 구성하면 된다. 즉, 하나의 카테고리 내의 모든 문서의 모든 문장들이 그 카테고리의 FP-tree를 만들게 되는 것이다. 그리고 각 테스트 문서들을 각각의 FP-tree와 비교하여 점수를 얻고 이 점수가 가장 큰 카테고리로 그 테스트 문서를 예측하는 것이다.

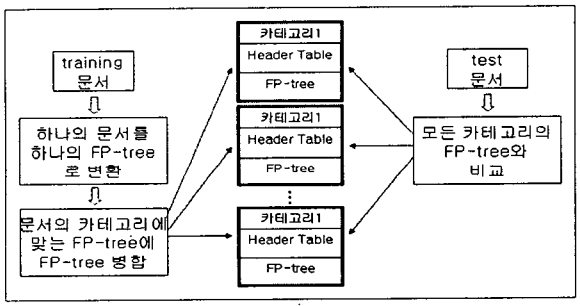

Figure 3.1 문서 분류 과정

다만, 여기서는 frequent items를 만들 때 50 이상 나타난 item이 아닌 모든 item, 즉 Stopword[3]를 제외한 모든 단어를 사용하였고, 각 단어는 wordnet[4]을 이용해 그 원형으로 사용하였다.

3.1.문서들을 각 카테고리 FP-tree에 넣기

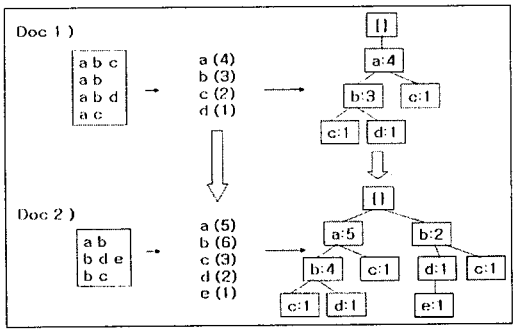

Figure 3.2 개선된 FP-tree

3.2.새 문서와 각 카테고리 FP-tree와의 비교 점수 구하기

Figure 3.2 와 같은 두 문서의 집합으로 된 카테고리 FP-tree가 있고, 이와 새로운 문서와의 비교 점수를 구하는 방법은 이러하다.

만약, 새로운 문서 a가 (b, a), (b, d, e) 이렇게 두 문장으로 이루어진 문서라고 하면, 우선 문서 a를 카테고리 Header table의 순서와 맞게 (a, b), (b, d, e)로 바꾸어 준다.

그 뒤 각 문장별로 점수를 구해 각 점수들의 평균을 문서의 점수로 한다. 각 문장별 점수를 구하는 알고리즘은 아래와 같다.

```
점수 = 0;
가중치 = W;
단어 = 문장의 마지막 단어;
while(문장의 첫 단어 전까지)
{
  노드 = Header Table[단어]->link;
  빈도 = 노드.빈도;
  while(노드 != null)
  {
      현 노드로부터 FP-tree의 root까지의 모든 노드를 구한다.
      구한 모든 노드의 각 단어들과 입력된 문장의 단어들 사이의
      공통된 단어의 수를 구한다.
      if(공통된 단어 수 >= 2)
         점수 = 점수 + (가중치^공통된 단어 수);
      노드 = 노드->link;
  }
  단어 = 문장에서 지금 단어의 이전 단어;
}
return 점수;
```

Algorithm 3.1 각 문장의 점수 구하는 알고리즘

위 알고리즘으로 각 문장의 점수를 구하면 (a, b) 문장의 점수는 4, (b, d, e) 문장의 점수는 5가 되어 두 점수의 평균인 4.5점이 된다.

이러한 방법으로 각 테스트 문서를 각각의 카테고리 FP-tree와 점수를 구하여 가장 높은 점수를 얻은 카테고리를 테스트 문서의 예상 카테고리로 분류하게 된다.

3.3.Mutual Information의 이용

위 3.2.장에서는 기본적이고 단순한 방법으로 문서와 각 카테고리 별 FP-tree 사이의 비교 점수를 구하는 방법을 알아보았다.

여기서는 문서 분류의 정확도를 좀 더 향상시키기 위해 Mutual Information을 이용하려 한다. 우선 각 카테고리 별 Header Table에 들어있는 단어들의 각각의 Mutual Information 값

$$I(w, c) = \log \frac{P(t, c)}{P(w) \times P(c)} \quad (w\ word,\ c\ category)$$[5]

에 우선 1을 더하고 1보다 작은 값을 1로 저장하여둔다.

그 뒤 문서와 각 카테고리 별 FP-tree 사이의 비교 점

수를 구하는 방법은 3.2.장과 동일하다. 다만 Algorithm 3.1에서 점수를 구할 때 모든 공통 단어들의 Mutual Information 값들을 곱하여 주었다.

4.실험

FPTC의 성능을 알아보기 위해 본 논문에서는 문서 데이터로 reuters-21578 문서 집합[6]을 사용하였다. reuters -21578 문서 집합은 로이터 통신의 기사 21578개를 모은 것으로 David Lewis 등이 총 135개의 중복이 허용되는 Topic으로 분류하였다. 본 논문에서는 이중 상위 10개의 Topic 만을 사용하였다.

Topic	training num	test num
acq	1650	647
crude	391	163
earn	2877	1045
grain	434	134
interest	347	104
money-fx	539	143
trade	369	112
corn	182	48
ship	198	85
wheat	212	66
Total	6494	2548

Table 4.1 reuters-21578 문서의 상위 10개 Topic의 갯수 (※복수 Topic 허용)

실험은 우선 각 Topic에 해당하는 학습 문서들로 카테고리별, 즉 Topic별 FP-tree를 만들고, test 문서 각각을 각 FP-tree들과 비교하여 가장 큰 점수를 얻은 Topic을 test 문서의 예상 Topic으로 하여, precision 값과 recall 값의 평균인 break-even point[7]를 계산하여 타 알고리즘의 결과들과 비교하였다.

Table 4.2가 바로 그 결과이다. FPTC의 결과는 점수 계산식을 $100^{(n-2)}$*frequency로 했을 때의 결과이다. 결과에서도 알 수 있듯이 FPTC의 분류 결과는 기존의 결과 중 최고인 Linear SVM에는 못 미치지만 두 번째인 Decision Tree에 근접한 결과를 보여주었다.

하지만 FPTC의 특성상 특별한 특징 추출 과정이나 학습의 과정 없이 Topic 별 FP-tree와의 단순 비교만으로 이러한 정확도를 보여준 것은 의미 있는 결과라 할 수 있다.

반면 FPTC(with Mutual Information)의 결과는 기존 결과 중 최고인 Linear SVM과 비슷한 수준의 높은 정확도를 보여준다. 타 알고리즘들이 단어들의 Feature로 각 단어의 Mutual Information을 이용하였다는 점을 감안하면 타 알고리즘에 비해 단순하지만 좋은 결과를 보여주는 FPTC가 매우 뛰어난 알고리즘임을 알 수 있다.

	FPTC	FPTC (MI)	Findsim	NBayes	C4.5	Linear SVM
acq	87.9	93.3	64.7	87.8	89.7	93.6
crude	80.1	89.4	70.1	79.5	85.0	88.9
earn	94.1	96.1	92.9	95.9	97.8	98.0
grain	84.0	91.4	67.5	78.8	85.0	94.6
interest	76.3	83.4	63.4	64.9	67.1	77.7
money-fx	78.0	85.4	46.7	56.6	66.2	74.5
trade	73.3	82.6	65.1	63.9	72.5	75.9
corn	55.1	82.6	48.2	65.3	91.8	90.3
ship	68.2	76.9	49.2	85.4	74.2	85.6
wheat	74.1	85.9	68.9	69.7	92.5	91.8
Total	86.5	91.8	64.6	81.5	88.4	92.0

Table 4.2 Reuters 상위 10개 Topic의 Break-even point (FPTC와 그 외 방법들 비교)[8]

재미있는 사실은 FPTC의 각 Topic별 break-even point의 수치의 순서가 거의 training 문서 수의 순서와 일치한다는 사실이다. 이는 FPTC가 training 한 문서 수가 많을수록 정확도가 올라갈 것이라는 기대를 가지게 한다.

5.결론

FPTC를 이용하면 이미 분류된 웹페이지들을 토대로 새로운 웹페이지를 빠르고 정확하게 자동 분류하는 시스템의 설계가 가능할 것이다.

궁극적으로 개인이 열어본 페이지들을 자동 분류하여 FP-tree로 만들어 놓고 웹 검색 시 이를 이용하여 각 사용자에게 더 잘 맞는 웹 페이지를 화면 상위에 보여주는 검색 시스템을 개발할 예정이다.

6.참 조

[1] Ian H. Witten, Eibe Frank. 'Data Mining : Practical Machine Tools and Techniques with Java Implementations' 2000

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

[3] Gerard Salton, Chris Buckley '571 stopword list for the experimental SMART information retrieval system at Cornell University'

[4] G.A. Miller 'WordNet: A Dictionary Browser' *1st Int'l Conf. Information in data* 1985

[5] Yiming Yang and J. O. Pedersen. 'A Comparative Study on Feature Selection in Text Categorization.' *Proceedings of the 14th International Conference on Machine Learning*, 1997

[6] D.D.Lewis 'Reuters-21578' http://www.research.att.com/~lewis

[7] R. Bekkerman, R. El-Yaniv, N. Tishby, and Y. Winter. 'On feature distributional clustering for text categorization' *Proceedings of SIGIR-01*, 2001

[8] S. T. Dumais, J. Platt, D. Heckerman and M. Sahami 'Inductive learning algorithms and representations for text categorization' *Proceedings of ACM-CIKM98* 1998