음절 정보만 이용한 한국어 복합 명사 분해*

서울대학교 컴퓨터공학부 바이오지능연구실
박 성 배 · 장 병 탁

Korean Compound Noun Decomposition Only Using Syllabic Information

Seong-Bae Park, Byoung-Tak Zhang
Biointelligence Lab., School of Computer Science and Engineering, Seoul National University, Seoul, Korea

요 약

한국어에서는 복합 명사 생성이 매우 자유스럽다. 즉, 독립된 명사를 연속으로 붙여 쓰는 것이 가능하다. 하지만, 기계번역이나 정보 검색과 같이 복합 명사를 처리하는 시스템에서 정확한 분석을 위해서는 복합 명사를 다시 단일 명사들로 분해하는 과정이 필요하다. 본 논문에서는 한국어 복합 명사 분해를 위해 GECORAM (GEneralized COmbination of Rule-based learning And Memory-based learning) 알고리듬을 제시한다. 규칙 학습 알고리듬의 장점은 생성된 학습 결과를 사람이 쉽게 이해할 수 있다는 점이지만, 다른 지도학습 알고리듬에 비해 성능이 떨어진다는 단점이 있다. 본 논문에서는 이를 위해 규칙 학습 알고리듬과 기억기반 학습을 결합하는 방법을 제시한다. 실험 결과, GECORAM 알고리듬은 규칙 기반 학습이나 기억 기반 학습을 단독으로 쓰는 경우보다 높은 정확도를 보였다.

서 론(1장)

한국어에서는 연속해서 나타나는 명사들을 붙여씀으로써 하나의 복합 명사를 만드는 것이 가능하다. 이런 복합 명사는 단일 명사에 비해 문맥 정보를 더 많이 가지며,[1] 자연언어처리에서 매우 중요한 역할을 한다. 하지만, 이런 복합 명사를 처리하는데 있어 문제는 존재할 수 있는 복합 명사의 수가 무한하기 때문에 모든 복합 명사를 모두 사전에 등록할 수 없다는 점이다. 따라서, 복합 명사를 처리하기 위해서는 주어진 복합 명사를 다시 그 복합 명사를 구성하는 단일 명사들로 분리할 필요가 있다.

n개의 음절로 된 복합 명사에 대해서 분해할 수 있는 경우의 수는 2^{n-1}개이다. 따라서, 복합 명사를 분해하는 가장 쉬운 방법은 이 2^{n-1}개 중에서 가장 가능성이 높은 것을 취하는 것이다. 이런 방식으로 복합 명사를 분해하는 많은 선행 연구가 있었다. 심광섭은 110만 어절의 말뭉치로부터 학습된 상호정보를 이용하여 복합 명사를 분해하였다.[1] Lee et al.은 복합 명사 분해를 품사 태깅과 같은 문제로 보고 마코프 모델(Markov model)을 적용하였다.[2]

이와 같은 통계기반 방법의 가장 큰 문제는 학습된 결과를 사람이 이해하기 힘들다는 점이다. 이에 비해, 규칙을 사용하면 그 동작 과정을 이해하기 쉽다. 따라서, 복합 명사 분해에 규칙을 사용하려는 연구 결과도 있었다. 예를 들어, 강승식은 형태소 분석결과로 추정되는 복합 명사를 네 개의 분해 규칙과 두 가지 예외 규칙을 사용하여 가능한 분해 후보를 생성하고, 후보들에 대하여 가중치를 부여함으로써 최적 후보를 선택하는 알고리즘을 제시하였다.[3] 그리고, 윤보현은 통계 정보와 우선 적용 규칙을 사용하고, 미등록어를 포함한 복합명사는 휴리스틱을 이용하여 분할하여 약 96%의 분할 성능을 보였다.[4]

하지만, 규칙을 사용하기 위해서는 정확한 규칙을 작성해 줄 전문가가 필요하다. 규칙 기반 방법의 성능은 만들어진 규칙의 질에 의해 결정되므로, 이 전문가는 풀고자 하는 문제에 대해 매우 깊은 지식을 가지고 있어야 한다. 하지만, 이런 전문가를 활용하는 방법은 비용이 많이 드는

*이 논문은 과기부 NRL, BrainTech 프로그램과 교육부 BK 21 사업에 의하여 지원되었음.
E-mail : sbpark@bi.snu.ac.kr
E-mail : btzhang@bi.snu.ac.kr

작업이다.

따라서, 기계학습 분야에서는 이러한 규칙을 자질-값의 벡터로 표현된 데이터로부터 자동으로 학습하는 여러가지 방법을 제시하였다. Clark과 Niblett는 일반적인 것으로부터 구체적인 것을 검색하는 방법을 통해 규칙을 생성해 내는 CN2 알고리듬을 제시하였고,[5] Furnkranz와 Widmar는 IREP을 제안하였다.[6] Cohen은 IREP을 개선하여 RIPPER 알고리듬을 제안하였으며,[7] Cohen과 Singer는 RIPPER에 부스팅 개념을 도입하여 SLIPPER 알고리듬을 발표하였다.[8]

자동으로 학습된 규칙 기반 방법의 문제는 성능이 다른 지도 학습 알고리듬(supervised learning algorithm)에 비해 낮다는 점이다. 본 논문에서는 규칙 기반 학습의 이런 문제를 해결하기 위하여, 규칙 기반 학습과 기억 기반 학습을 결합하는 방법을 제시한다. 우리는 이전 연구에서 규칙과 기억 기반 학습을 함께 사용하면 높은 성능을 얻을 수 있음을 보였다.[12] 본 논문에서는 미리 만들어진 규칙이 없는 문제에서도 이 방법을 적용하기 위해서, 규칙 기반 학습과 기억 기반 학습의 일반화된 결합 방법인 GECORAM (GEneralized Combination of Rule-based learning And Memory-based learning) 알고리듬을 제시한다. 이런 규칙 기반 학습은 ILP(Inductive Logic Programming)의 기본이 되기도 하므로 응용 문제에서 뿐만 아니라 방법론 그 자체로서도 중요한 의미를 지닌다.

본 논문의 나머지 부분은 다음과 같이 구성되어 진다. 2장에서는 규칙 기반 학습에 대한 사전 연구를 설명하고, 3장에서 GECORAM 알고리듬을 제시한다. 4장에서 한국어 복합 명사 분리에 대한 실험 결과를 보이고, 마지막으로 5장에서 결론을 맺는다.

```
function IREP(data)
begin
    RuleSet ← φ
    while ∃ positive examples ∈ data do
      Split data into grow and prune.
      rule ← GrowRule(grow)
      rule ← PruneRule(rule, prune)
      Add rule to RuleSet.
      Remove examples covered by rule
from data.
      if Accuracy(rule) ≤ P/(P+N) then
  return RuleSet
      endif
    endwhile
    return RuleSet
end
```

Fig. 1. IREP 알고리듬. P는 data 내의 양의 예제의 수이고, N은 음의 예제의 수이다.

규칙 기반 학습 알고리듬(2장)

1. IREP 알고리듬

GECORAM 알고리듬은 기본적으로 IREP(Incremental Reduced Error Pruning) 알고리듬에 기초하고 있으므로, GECORAM을 이해하기 위해서는 우선 IREP 알고리듬을 설명할 필요가 있다. Fig. 1은 IREP 알고리듬을 간단하게 보이고 있다. 이 알고리듬은 크게 봐서 두 단계로 된 욕심쟁이 알고리듬(greedy algorithm)이다. 즉, 이 알고리듬은 한번에 하나의 규칙을 생성하고, 주어진 데이터에서 이 예제가 적용된 예제를 제거하는 방식이다. 규칙을 생성할 때 사용되는 원칙은 가능한 많은 수의 양의 예제와 가능한 적은 수의 음의 예제가 처리되도록 한다는 것이다.

이를 위해, 우선 주어진 data[1]를 grow와 prune의 두 집합으로 나눈다. 일반적으로 grow가 data의 2/3을, prune이 1/3을 차지한다. grow는 규칙을 증가시킬 때 사용되고, prune은 규칙을 감소시킬 때 사용된다. 규칙을 증가시키는 단계(Fig. 1의 GrowRule 함수)는 조건이 없는 상태에서 조건을 하나씩 추가하면서 규칙을 생성해 낸다. 조건이 없는 규칙을 r_0라고 하자. 매 단계 i마다, r_i에 조건이 하나씩 붙어서 더 복잡한 규칙 r_{i+1}을 만든다. r_{i+1}를 만들 때 추가된 조건은 r_i에 대한 정보이득(information gain)[9]이 가장 큰 것이다. 즉, T_i^+를 i번째 단계에서 규칙 r_i에 의해 처리되는 양의 예제의 수, T_i^-를 음의 예제의 수라고 했을 때, 정보 이득은 다음과 같이 계산된다.

$$Gain(r_{i+1}, r_i) = T_{i+1}^+ \cdot \left(-\log \frac{T_i^+}{T_i^+ + T_i^-} + \log \frac{T_{i+1}^+}{T_{i+1}^+ + T_{i+1}^-} \right)$$

이와 같은 방식으로 정보이득이 0이 될 때까지 규칙에 조건을 계속 추가한다.

두 번째 단계에서는 GrowRule에서 만들어진 규칙의 조건을 하나씩 제거해 나감으로써 규칙을 간결하게 한다. 즉, PruneRule 함수에서는 아래의 함수

$$f(r_{i+1}) = \frac{T_{i+1}^+ - T_{i+1}^-}{T_{i+1}^+ + T_{i+1}^-}$$

를 최대로 만드는 조건을 차례로 제거해 나간다. 규칙을 간략화한 다음, 이 규칙을 RuleSet에 추가하고, data에서 이 규칙이 처리하는 예제들을 모두 제거한다.

규칙을 만들 때 정보이득을 사용하였는데, 정보이득은 항

1 본 논문에서는 data가 벡터 형태로 표현된다고 가정한다.

상 0보다 크거나 같다. 따라서, 만들어지는 모든 규칙은 항상 data 내의 몇 개의 양의 규칙을 처리하게 된다. 그러므로, 나중에 만들어지는 규칙은 너무 구체적으로 될 가능성이 높고 이런 규칙은 노이즈가 있는 데이터에 대해서 부정적인 영향을 미칠 것이다. IREP에서는 이런 규칙이 RuleSet에 포함되는 것을 막기 위해서, P를 data 내의 양의 예제의 수, N을 음의 예제의 수라고 했을 때, 만들어진 규칙의 정확도가 $P/(P+N)$보다 낮으면 학습을 중단한다.

2. RIPPER 알고리듬

RIPPER(Repeated Incremental Pruning to Produce Error Reduction) 알고리듬은 IREP에 기초하여 성능을 개선시킨 모델이다. Fig. 2는 RIPPER 알고리듬을 간략하게 설명한 의사 코드이다.

우선, RIPPER는 IREP 알고리듬으로 만든 규칙 집합 RuleSet의 규칙 수를 줄이고 성능을 높이기 위해서 최적화시킨다. 함수 Optimize에서 각 규칙은 만들어진 순서대로 하나씩 검사를 받는다. 이 함수에서는 각 규칙 c에 대해, c'과 c''의 두 가지 대응 규칙을 생성한다. c'를 간략화할 때, c를 c'로 바꾼 후 전체 규칙 집합 RuleSet의 성능이 prune에 대해 좋아지도록 간략화한다. c''은 GrowRule을 호출할 때, 빈 규칙에서 시작하는 것이 아니라 c로부터 시작해서 만들어진다. 최종적으로 RuleSet에는 c, c', c'' 중에서 가장 좋은 것이 포함된다.

```
function  Optimize(RuleSet, data)
begin
    for each rule c ∈RuleSet do
        Split data into grow and prune.
        c' ← GrowRule(grow)
        c' ← PruneRule(c', prune) with
             RuleSet – c + c'
        c'' ← GrowRule(c, grow)
        c'' ← PruneRule(c'', prune) with
             RuleSet – c + c''
        Replace c in RuleSet with best c, c', c''.
    endfor
    return RuleSet
end

function RIPPER(data)
begin
    RuleSet ← IREP(data)
    repeat twice:
        RuleSet  ← Optimize(RuleSet, data)
        UncovData ← examples ∈ data not
covered by the rules in RuleSet
        RuleSet ← RuleSet + IREP(UncovData)
    endrepeat
end
```

Fig. 2. RIPPER 알고리듬.

그러면, 이 세 규칙 중에서 어느 것이 가장 좋은 것일까? 이것을 결정하기 위해서, RIPPER는 MDL(Minimum Description Length)를 사용한다. 즉, DL(Description Length)이 가장 짧은 규칙이 가장 좋은 원칙이다. RIPPER에서는 규칙의 DL을 측정하는 방법으로 Quinlan이 제시한 방법[10]을 사용한다.

최적화를 하고 나면, RuleSet이 원래보다 더 적은 수의 양의 예제를 처리할 가능성이 높기 때문에 RuleSet이 처리하지 못한 데이터 UncovData에 대해 다시 IREP을 실행시켜 새로운 규칙을 포함시킨다. 이 과정을 2번 하는 이유는 실험적으로 이렇게 했을 때 성능이 좋아졌기 때문이다.

GECORAM 알고리듬(3장)

1. 기본 아이디어

한국어 복합 명사 분해 문제는 분류 문제로 생각되어질 수 있다. 즉, 복합 명사 분해 문제는 주어진 음절 w_i에 대해서, 여기를 띄울 것이냐를 결정하는 이진 분류 문제(binary classification problem)이다. 이것을 결정하기 위해서, w_i의 문맥 정보 h_i를 사용한다. 본 논문에서는 문맥 정보로 좌우 n개의 음절을 사용한다.

GECORAM 알고리듬은 분류 문제를 풀기 위하여 규칙 기반 학습과 기억 기반 학습을 효과적으로 결합하는 일반적인 방법이다. 이 알고리듬의 기본적인 아이디어는 Fig. 3처럼 표현될 수 있다. w_i에서 분리할 것인지를 우선 학습된 규칙을 사용하여 결정한 후, 기억 기반 학습에 현재 문맥 h_i가 규칙의 예외 상황인지를 물어본다. 만약 h_i가 규칙의 예외 상황이면 규칙이 잘못 분류할 가능성이 높으므로, 규칙이 결정한 결과를 버리고 기억 기반 학습이 결정한 바를 따른다. 그러므로, 기억 기반 학습은 규칙의 예외만 특별히 처리하는 요소이다.

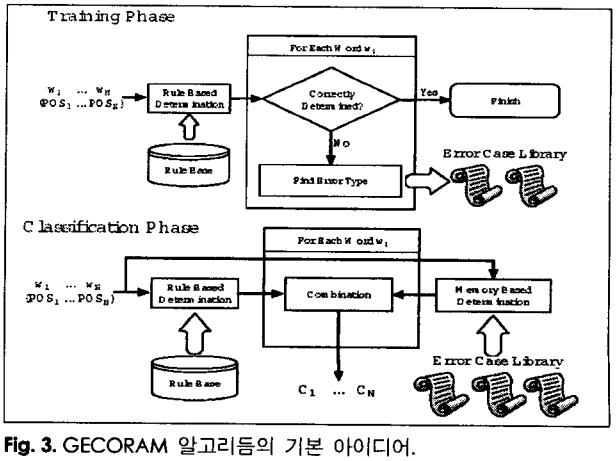

Fig. 3. GECORAM 알고리듬의 기본 아이디어.

이를 위해, 학습 단계에서는 주어진 학습데이타에 대해 규칙을 학습하고, 학습된 규칙으로 복합 명사 분할 문제를 풀어본다. 규칙으로 잘못 분류되는 학습 예제들은 따로 모아서 오류 라이브러리(error case library)에 저장한다. 이 오류 라이브러리에는 규칙의 예외만 저장되므로, 규칙이 충분히 정확하다면 저장된 예제의 수는 많지 않을 것이다. 기억 기반 학습은 이 오류 라이브러리만으로 학습된다.

분류 단계에서는 학습된 규칙과 기억 기반 학습 모두를 이용해서 학습 단계에서 경험하지 못한 예제들을 분류한다. 우선 규칙이 w_i와 h_i가 주어지면 여기서 분해를 할 것인지 말 것인지를 결정한다. 그리고, h_i가 규칙의 예외 상황인지 다시 한번 더 조사한다. 만약 h_i가 규칙의 예외 상황이면, 앞에서 설명한 바와 같이 규칙으로 결정된 결정을 취소하고 기억 기반 학습의 분류 결과를 따른다.

2. GECORAM 알고리듬

Fig. 4는 GECORAM 알고리듬의 학습 단계가 어떻게 동작하는지 간략하게 의사코드로 보여준다. GECORAM의 첫 번째 단계는 주어진 학습 데이터 집합 data로부터 규칙을 학습하는 것이다. data에서 규칙을 학습하기 위해 변형된 IREP 알고리듬인 MODIFIED_IREP을 사용한다. MODIFIED_IREP과 IREP의 차이는 MODIFIED_IREP에는 prune 모드가 없다는 점이다. 즉, MODIFIED_IREP에서는 규칙이 커지기만하고 간결화하지는 않는다. PruneRule 함수의 역할은 뒤에서 설명할 기억 기반 학습이 대신한다.

다음 단계에서는 MODIFIED_IREP에서 학습되지 못한 예제들을 모아 기억 기반 학습으로 학습시킨다. 기억 기반 학습은 k-nearest neighbor(k-NN)[11]의 직접적인 후계 알고리듬이다. 자연언어처리 문제들이 대부분 수많은 데이터를 다루고 중요도가 서로 다른 여러 가지 속성들이 분류에 관여하기 때문에, 일반적으로 기억 기반 학습은 k-NN보다 복잡한 자료 구조와 속도 개선 방법을 가지고 있다.

기억 기반 학습에서의 학습은 예제를 메모리에 저장하는 것이다. 그리고, 한 예제 x와 메모리 내에 있는 모든 예제 y 사이의 유사도는 거리 단위, △(x, y)로 계산된다. 예제 x의 클래스는 메모리 내에서 x와 가장 비슷한 k개의 예제들 사이의 가장 빈도수가 높은 클래스로 결정된다. 그리고, x와 y 사이의 거리, △(x, y)는 다음 수식에 의해 계산된다.

$$\Delta(\mathbf{x},\mathbf{y}) \equiv \sum_{j=1}^{m} \alpha_j \delta(x_j, y_j)$$

여기서, m은 사용된 자질의 수이고, α_j는 j번째 자질의 중요도이며, $\delta(x_j, y_j)$는 다음과 같이 계산된다.

$$\delta(x_j, y_j) = \begin{cases} 0 & \text{if } x_j = y_j \\ 1 & \text{if } x_j \neq y_j \end{cases}$$

α_j가 정보 이득에 의해 결정되면 이 단위를 쓰는 k-NN 알고리듬을 IB1-IG[13]라고 불린다. 본 논문의 모든 기억 기반 학습 알고리듬으로 IB1-IG가 사용되었다.

GECORAM에서는 규칙 기반 학습과 기억 기반 학습이 모두 사용되기 때문에, 언제 규칙을 사용하고 언제 기억 기반 학습을 사용할 것인지를 결정하는 것이 중요한 문제이다. 이 문제를 해결하기 위해서, GECORAM은 θ라는 임계값을 둔다. 최적의 θ값은 다음과 같은 방법으로 찾는다. 학습 데이터인 data와는 독립인 held-out 데이터 집합 HeldOutData를 가지고, Fig. 5의 GECORAM 분류 알고리듬을 다양한 θ값에 대해 적용하여 가장 좋은 성능을 내는 θ값을 찾는다.

주어진 x와 메모리 내에서 x와 가장 비슷한 예제 y 사이의 거리, △(x, y)가 매우 가까우면 x는 규칙의 예외일 가능성이 매우 높다. 메모리 내에는 규칙이 잘못 적용된

```
function  Support(RuleSet, data)
begin
  Err ← φ
    for each (<wi, hi>, ci) ∈ data do
        if  RuleSet(wi, hi) ≠ ci  then
            Add(<wi, hi>, ci) into Err.
        endif
    endfor
    MBL ← Memory-Based-Learning(Err)
    return MBL
end

function Training-GECORAM(data)
begin
    RuleSet ← MODIFIED_IREP(data)
    MBL ← Support(RuleSet, data)
  θ  ← GetThresholdMBL(RuleSet, MBL,
HeldOutData)
    return RuleSet + MBL + θ
end
```

Fig. 4. GECORAM 학습 알고리듬.

```
function Classify-GECORAM(x, θ, RuleSet, MBL)
begin
    c ← RuleSet(x)
    y ← the nearest instance of x in Err
  if  Δ(x, y) ≥ θ  then
    c ← MBL(x)
    endif
    return  c
end
```

Fig. 5. GECORAM 분류 알고리듬.

경우만 저장되어 있으므로, x가 y와 가깝다는 것은 x도 규칙의 예외 상황일 가능성이 높다는 것을 의미한다. 따라서, 이 거리가 임계값 θ보다 크면 규칙을 적용하지 않아야 한다.

θ가 $\triangle(x, y)$에 대한 임계값이기 때문에,

$$\beta \equiv \sum_{j=1}^{m} \alpha_j$$

라고 할 때, $0 \le \theta \le \beta$를 만족한다. $\theta=0$이면 항상 규칙을 따르지 않게 되고, $\theta=\beta$이면 항상 규칙을 따르게 된다. 그러므로, $\theta=\beta$이면 규칙 기반 학습만 사용하는 것과 같게 된다.

θ값이 정해지면, Fig. 5의 GECORAM 분류 알고리듬으로 학습 시 경험하지 못한 데이터에 대해 분류하게 된다. 분류에서는 $\triangle(x, y) \ge \theta$이면, 규칙을 적용하지 않고, 기억 기반 학습의 분류 결과를 따른다. 따라서, Fig. 5는 Fig. 3의 분류 단계(classification phase)를 의사 코드로 표현한 것이다.

2. 다른 알고리듬과 비교

이 절에서는 본 논문에서 제시한 GECORAM 알고리듬을 기존의 RIPPER 및 TBL(Transformation-Based Learning),[14] AdaBoost[15]와 비교한다.

GECORAM 알고리듬과 RIPPER 알고리듬은 IREP을 기반으로 하고 있다는 점에서 비슷하다. RIPPER가 순수히 ILP를 위한 규칙 기반 학습인데 비하여, GECORAM 알고리듬은 규칙 기반 학습과 기억 기반 학습을 결합한 일종의 혼합 모델(mixture model)이다. 따라서, RIPPER는 너무 세세한 규칙이 만들어 지는 것을 막기 위해서 Optimize 함수를 두었지만, GECORAM에서는 이 역할을 기억 기반 학습이 한다. 기억 기반 학습이 일종의 lazy learning이기 때문에, GECORAM에서는 IREP에 있는 PruneRule 함수도 사용하지 않는다.

또한, GECORAM 알고리듬은 TBL 및 AdaBoost와 여러 가지 면에서 비슷한 성질을 가지고 있다. TBL과 AdaBoost 모두 간단한 규칙을 결합하여 분류기를 만든다는 점에서 비슷하다. 하지만, AdaBoost가 이론적으로 견고한데 비하여, TBL은 직관적이다. 그리고, 자연언어처리의 여러 문제에서 AdaBoost가 TBL보다 좋은 성능을 보였다.[16]

TBL의 단점은 세 가지로 요약할 수 있다. 첫째, TBL은 학습 데이터에 대해 과도한 학습을 하기 쉽다. 둘째, TBL은 노이즈에 민감하다. 마지막으로 셋째는 위에서 언급한 바와 같이 추가된 각 변형(transformation)이 응용 문제의 성능을 높이는 쪽으로 나아가는지에 대한 보장이 전혀 없다.

AdaBoost는 현재 분류기의 오류에 의해 학습이 진행된다는 점에서 TBL과 비슷하다. 하지만, AdaBoost는 TBL과는 달리 이론적으로 견고한 알고리듬이다. ε_t를 AdaBoost에서 사용된 t번째 약학습자의 오류율이라고 하고, $\gamma_t=1/2-\varepsilon_t$라고 하자. $\varepsilon_t \le 1/2$라면, AdaBoost의 최종 분류기 h_{fin}은 다음과 같은 오차한계치(error bound)를 갖는다.

$$\frac{1}{N}\left|\left\{i : h_{fin}(\mathbf{x}_i) \neq c_i\right\}\right| \le \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$$

여기서, N은 전체 학습 데이터의 수이고, T는 AdaBoost 내에 있는 약 학습자의 수이다. 따라서, $\varepsilon_t \le 1/2$라면 AdaBoost는 성능이 나빠지지 않는다.

AdaBoost와 GECORAM의 가장 큰 차이는 AdaBoost가 같은 방식으로 동작하는 여러 개의 약 학습자를 갖는다는 점이다. AdaBoost의 수렴 속도는 사용된 약 학습자의 성능에 의해 결정된다. 약 학습자의 정확도가 1/2보다 크면 수렴하겠지만, 이 성능이 낮으면 T가 증가하게 된다. 만약 약 학습자의 성능이 어떤 사전 지식을 사용하여 높아진다면, AdaBoost는 매우 빠르게 수렴할 것이다. GECORAM은 이런 면에서 (T-1)개의 약 학습자가 하나의 규칙 기반 학습, 그리고 마지막 T 번째 약 학습자가 기억 기반 학습으로 된 AdaBoost로 생각되어 질 수 있다. 그리고, θ는 약 학습자의 중요도(weight)로 생각될 수 있다. 하지만, GECORAM은 AdaBoost가 T단계를 거치는 반면에, 단 두 단계만 거치므로 학습 속도가 훨씬 빠르고 계산량도 적다.

실 험(4장)

1. 데이터 집합

본 논문에서 사용된 데이터집합은 두 가지이다. 첫번째 데이터 집합(Shim)은 [1]에서 사용된 것이고, 두번째 데이터 집합(Yoon)은 [4]에서 사용된 것이다. 이 두 데이터 집합에 대한 통계정보는 Table 1에 있다. GECORAM은 학습 데이터 집합, 검증 데이터 집합 외에 held-out 집합을 필요하므로, 전체 데이터 중 80%를 학습 데이터로, 10%를 held-out으로, 나머지 10%를 검증 데이터로 사용하였다. 두 데이터 집합 모두 데이터의 수가 많지 않으므로, 10회 교차 검증(10-fold cross validation)을 실행하였다.

Table 1. 한국어 복합 명사 분해를 데이터의 통계 정보

데이터 집합	Shim	Yoon
예제의 수	9,863	15,096
음절의 수	562	557
복합 명사의 평균 길이	7.26	4.92

2. 실험 결과

GECORAM 알고리듬의 성능을 평가하기 위해서, GECORAM을 RIPPER,[7] SLIPPER,[8] C4.5,[9] TiMBL[13]과 비교하였다. 앞의 세 알고리듬은 규칙 기반 학습 알고리듬으로 볼 수 있고, TiMBL은 기억 기반 학습 방법이다.

Fig. 6과 Fig. 7은 각각 문맥의 길이에 대한 정확도의 변화를 보이고 있다. 두 데이터 집합 모두에 대해서, GECORAM이 다른 알고리듬보다 높은 정확도를 보인다. TiMBL이 RIPPER나 SLIPPER와 같은 규칙 기반 학습 알고리듬보다 성능이 높다. 일반적으로, 규칙 기반 학습 알고리듬은 성능보다 학습 결과의 이해력에 중점을 두므로, 정확도가 높지 않은 경향이 있다.

문맥의 길이가 0이면, 문맥을 전혀 보지 않고 한 음절만 보고 분해하는 것을 의미한다. 이 경우에도, 모든 알고리듬이 70% 이상의 정확도를 보인다. 이는 음절이 복합 명사를 분해하는데 매우 중요한 정보임을 의미한다. Shim 데이터 집합의 평균 음절 수가 7.26개이므로, 양쪽 음절을 3개씩 보면 7개의 음절을 보게 된다. 따라서, 문맥 길이를 3으로 했을 때 가장 좋은 성능이 나왔고, 더 많은 문맥 정보는 노이즈로 작용한다. Yoon 데이터 집합의 평균 음절 수가 4.92개이므로, 문맥 길이가 2일 때 가장 좋은 성능을 보일 것으로 예상되나 실제로는 4일 때 가장 좋은 성능을 보였다. 하지만, 이 때의 성능도 2일 때와 큰 차이는 없었다.

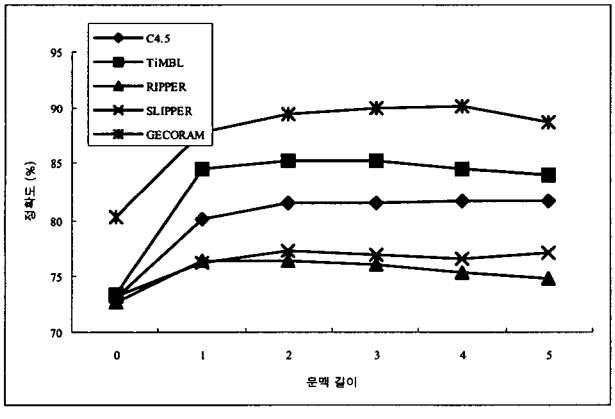

Fig. 6. Shim 데이터 집합의 문맥의 길이에 대한 정확도.

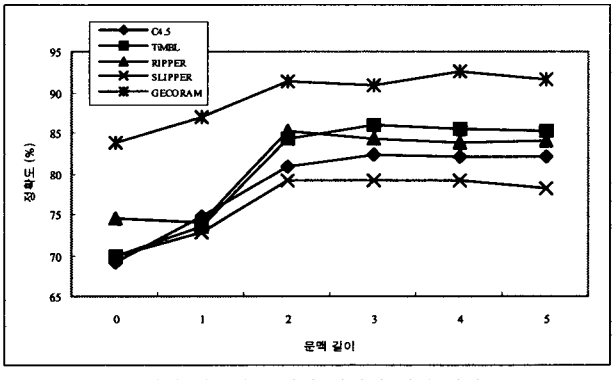

Fig. 7. Yoon 데이터 집합의 문맥의 길이에 대한 정확도.

Fig. 8과 Fig. 9는 규칙 기반 학습 알고리듬들이 생성한 규칙의 수를 각각 보이고 있다. C4.5가 가장 많은 규칙을 생성하고, RIPPER와 SLIPPER가 100개 미만의 규칙을 생성한다. 이에 비해, GECORAM은 20개 미만의 규칙만 생성한다. 이는 GECORAM에서 규칙의 예외를 기억 기반 학습이 처리하므로, RIPPER에 있는 optimize 과정을 거치지 않았기 때문이다.

마지막으로, Table 2는 각 알고리듬이 가장 좋은 성능을 보였을 때의 정확도이다. GECORAM 알고리듬은 Shim 데이터 집합에 대해 90.13%, Yoon 데이터 집합에 대해 92.57%의 정확도를 보였다. 이는 평균적으로 RIPPER보다 10.5%, TiMBL보다 5.8% 정도 높은 정확도이다. 따라

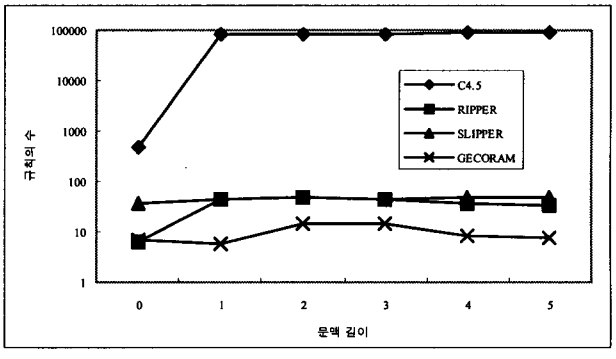

Fig. 8. Shim 데이터 집합의 문맥의 길이에 대한 생성된 규칙의 수.

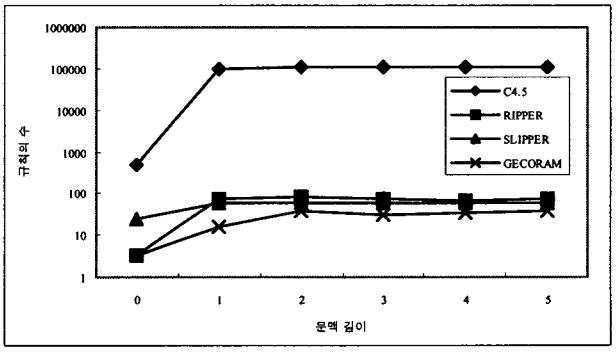

Fig. 9. Yoon 데이터 집합의 문맥의 길이에 대한 생성된 규칙의 수.

Table 2. 각 학습 방법의 최고 정확도

방 법	Shim	Yoon
C4.5	81.67%	82.37%
TiMBL	85.24%	85.89%
RIPPER	76.36%	85.27%
SLIPPER	77.22%	79.23%
GECORAM	**90.13%**	**92.57%**

서, GECORAM은 규칙 기반 학습과 기억 기반 학습 중 하나만 사용하는 것보다 높은 정확도를 보였다.

결론 및 향후 과제(5장)

본 논문은 규칙 기반 학습과 기억 기반 학습을 결합하여 한국어 복합 명사를 분해하는 새로운 알고리듬을 제시하였다. 제시된 GECORAM 알고리듬은 규칙 기반 학습인 RIPPER보다 10.5%, 기억 기반 학습인 TiMBL보다 5.8% 높은 정확도를 보여, 이 두 방법을 결합한 알고리듬이 한국어 복합 명사 분해에 효율적임을 보였다.

하지만, 제시된 방법은 오직 음절 정보만 사용하였으로, 복합 명사 분리에 대한 기존 연구보다 전체 성능이 낮다.[1-4] 향후, 성능을 더 높이기 위해서 다른 연구에서처럼 사전이나 시소러스와 같은 추가적인 정보를 활용할 계획이다. 이 경우 가장 좋은 분해 순서를 찾기 위해 Viterbi 알고리듬을 사용하는 것이 일반적이다. 하지만, GECORAM 알고리듬은 Markov 가정을 따르지 않고 또 확률 모델이 아니므로, 어떻게 가장 좋은 분해 순서를 찾을지를 연구할 계획이다.

REFERENCES

1) 심광섭(1997) : "합성된 상호 정보를 이용한 복합 명사 분리", 한국정보과학회 논문지(*B*), *24*권, *11*호, *pp1307-1317*
2) Lee JW, Zhang BT, Kim YT(1999) : *"Compound Noun Decomposition using a Markov Model," In Proceedings of MT Summit VII, pp427-431*
3) 강승식(1998) : "한국어 복합명사 분해 알고리즘", 정보과학회 논문지(*B*), *25*권, *1*호, *pp172-182*
4) 윤보현, 조민정, 임해창(1997) : "통계 정보와 선호 규칙을 이용한 한국어 복합 명사의 분해", 정보과학회논문지(*B*), *24*권, *8*호, *pp900-909*
5) Clark P, Niblett T(1989) : *"The CN2 Induction Algorithm," Machine Learning, Vol. 3, No. 1, pp261-284*
6) Furnkranz J, Widmar G(1994) : *"Incremental Reduced Error Pruning," In Proceedings of ICML-94, pp70-77*
7) Cohen W(1995) : *"Fast Effective Rule Induction," In Proceedings of ICML-95, pp115-123*
8) Cohen W, Singer Y(1999) : *"A Simple, Fast, and Effective Rule Learner," In Proceedings of AAAI-99, pp335-342*
9) Quinlan R(1993) : *C4.5 : Programs for Machine Learning, Morgan Kaufmann Publisher*
10) Quinlan R(1995) : *"MDL and Categorical Theories (continued)," In ICML-95, pp464-470*
11) Cover T, Hart P(1967) : *"Nearest Neighbor Pattern Classification," IEEE Transactions on Information Theory, Vol. 13, pp21-27*
12) Park SB, Zhang BT(2003) : *"Text Chunking by Combining Hand-Crafted Rules and Memory-Based Learning," In Proceedings of ACL-03, pp497-504*
13) Daelemans W, Zavrel J, Sloot K, Bosch A(2001) : *TiMBL : Tilburg Memory Based Learner, version 4.1, Reference Guide, ILK 01-04, Tilburg University*
14) Brill E(1995) : *"Transformation-Based Error-Driven Learning and Natural Language Processing : A Case Study in Part of Speech Tagging," Computational Linguistics, Vol. 21, No. 4, pp543-566*
15) Freund Y, Schapire R(1996) : *"Experiments with a New Boosting Algorithm," In Proceedings of ICML-96, pp148-156*
16) Abney S, Schapire R, Singer Y(1999) : *"Boosting Applied to Tagging and PP Attachment," In Proceedings of EMNLP-VLC-99, pp38-45*