언어모델 기반 단어 클러스터링 알고리즘의 효율성 향상 기법

박상우○, 김영태, 강동민, 나동열
연세대학교 컴퓨터정보통신공학부
jsys3@naver.com

An Improving Method of Efficiency for Word Clustering Based on Language Model

Sang-Woo Park○, Youngtae Kim, Dong-Min Kang, Dongyul Ra
Computer & Telecommunications Eng. Div. Yonsei University

요 약

단어 클러스터링 (word clustering) 또는 군집화는 자연어처리에서 데이터 부족 문제로 인하여 단어 간의 의미관계와 관련된 정보를 사용하기 어렵게 만드는 문제에 대처할 수 있는 중요한 기술이다. 단어 클러스터링과 관련하여 알려진 가장 대표적인 기법으로는 클래스-기반 n-gram 언어모델의 개발을 위하여 제안된 Brown 단어 클러스터링 기법이다[1]. 그러나 Brown 클러스터링 기법을 이용하는데 있어서 부딪치는 가장 큰 문제점은 시간과 공간적인 면에서 자원 소요량이 너무 방대하다는 점이다. 본 연구는 이 클러스터링 기법의 효율성을 개선하는 실험을 수행하였다. 실험 결과 가장 단순한(naive) 접근에 비하여 약 7.9 배 이상의 속도 향상을 이룰 수 있음을 관찰하였다.

주제어: 클러스터링, 단어 클러스터링, 자연어처리, n-gram, 언어모델

1. 서론

자연어처리 소프트웨어 개발에서 많이 부딪치는 가장 큰 문제 중의 하나는 데이터 부족 문제이다. 이와 관련된 한 예로 자연어 파싱을 들 수 있다. 파서가 판단해야 할 큰 결정 사항 중의 하나는 두 단어 사이의 의존관계가 가능한 지에 대한 것이다. 이 결정을 위한 핵심적인 정보로는 두 단어의 어휘(lexeme)까지 포함하는 자질 정보이다. 그러나 구문구조부착 말뭉치의 크기가 방대하지 않은 이상 이러한 정보를 학습할 수 없게 되는 데이터부족 문제가 일어난다. 이러한 문제에 대한 대처로 단어 대신 단어들의 클래스(집단)를 이용하는 방법이 제안되었으며 상당히 효과가 있음이 알려졌다[2]. 이와 같이 자연어처리에서 필연적으로 부딪치는 데이터 부족 문제를 해결하기 위해서 클러스터링 기술을 이용하고자 하는 것은 일반적이 접근 방법이다[3,4,5]. 즉 단어 클러스터링은 여러 자연어처리 분야에서 데이터 부족 문제를 극복하기 위하여 사용하는 매우 중요한 요소 기술이 되었다[6].

단어 클러스터링과 관련된 지금까지의 연구는 크게 2 가지 방향에서 시도되고 있다. 하나는 어떠한 단어들을 동일 클래스로 포함시킬 것인가를 결정하는데 있어서 사람이 일일이 단어의 의미를 고려하여 거의 수작업을 위주로 진행하는 방법이다. 이러한 접근법의 대표적인 예로는 wordNet, 단어의미망 연구 등을 들 수 있다[8,9]. 그러나 이러한 접근 방법은 크게 두 가지 문제점을 가지고 있다. 하나는 막대한 시간과 노력을 필요로 한다는 점이다. 대규모의 클러스터링 개발 작업은 여러 명의 전문가가 다년간의 작업을 하여야 가능한 대단히 큰 작업이다. 또 다른 보다 심각한 문제로는 작업에 참여하는 사람 간에 그리고 같은 사람의 작업이더라도 작업 시점 간에 클러스터링 관련 결정을 내리는 데 있어서 일관성을 유지하기 어렵다는 점이다.

이러한 점에서 단어 클러스터링 작업의 자동화와 관련된 기술은 매우 중요하다고 볼 수 있다[1,3,6]. 단어들의 자동 클러스터링 기술에 대한 가장 중요한 연구로는 Brown 단어 클러스터링 기술을 들 수 있다[1]. 이 기법은 원래 언어모델(language model)의 개발을 위해서 고안한 방법으로 n-gram 언어모델을 개발하는데 있어서 데이터 부족 문제를 해소하기 위해서 클래스에 기반한 n-gram 모델로 달성하자는 방법론이다. 이를 위해서 단어 클래스를 생성하는 클러스터링 기법이 제안되었다(이를 본 논문에서는 Brown 클러스터링이라 부르자).

대상 언어의 모든 가능한 단어의 수 (vocabulary size)를 V 라 가정하자. Brown 클러스터링의 기본적인 방법은 소위 계층적 클러스터링 기법을 따르는데 처음에는 각 클래스 마다 1 개의 단어를 가진 V 개의 클래스를 가진 상태에서 출발한다. 모든 클래스 쌍 중에서 합병(merge) 기준 (criteria)에 가장 부합하는 쌍을 하나로 합병한다. 이와 같이 합병 작업을 계속 수행하여 결국 단 하나의 클래스만이 남을 때까지 이 작업을 계속한다. 이렇게 하여 모든 단어를 리프노드로 가지는 완전한 클러스터 계층 구조를 생성하는 것이다(이를 Brown-1 기법이라 하자)[1]. 그러나 위 기법은 너무 계산량이 크기 때문에 실제로는 사용하기 어렵다. 따라서 계산량을 줄이고자 제안된 축소된 기법은 초기 클래스의 수를 k ($k < V$) 로 지정하고 계속 k 개의 클래스 수를 유지하면서 작업을 진행하는 기법이 제안되었다(Brown-2 기법)[1]. 이 기법은 클러스터링의 질적인 면에서는 전자보다 못하다고 생각되지만 계산량을 줄여서 실용적인 기법이 되게 하는데 그 목적이 있다.

그러나 이러한 Brown-2 기법도 실제로는 계산량이 매우 커서 이용하기 쉽지 않은 것으로 관찰되었다. 본 논문은 Brwon-2 클러스터링을 수행하는 데 있어 보다

계산량을 경감시키는 기법을 제안하고자 한다. 속도향상을 위해 합병기준으로 상호정보 감소량의 이용 그리고 단어쌍 공기정보를 메모리에 저장하는 방법을 이용하는 것을 제안한다. 우리가 실시한 실험 결과 약 7.9배 이상의 속도 향상을 얻게 되었다.

2. 합병기준

Brown 클러스터링 연구의 출발점은 n-gram 언어모델의 구현이다. 그러나 데이터부족 문제로 인하여 클래스 기반 모델로 근사화(approximation) 하기 위해서 클러스터링 작업이 필요하게 된 것이다.

즉 n-gram 언어모델에서는 (1)과 같은 $P(w_n|w_1^{n-1})$ 과 같은 확률이 필요하다(기호 $C(x)$ 는 스트링 x의 출현횟수를 나타냄):

$$P(w_n|w_1^{n-1}) = C(w_1^{n-1}w^n)/\sum_w C(w_1^{n-1}w) \qquad (1)$$

그러나 여기에 필요한 스트링w_1^{n-1} 또는 w_1^n이 학습말뭉치에서 관찰되지 않는 데이터 부족 문제가 나타나는 경우가 많다. 이를 극복하기 위해서 (2)와 같이 클래스 스트링을 이용하여 이에 근사한 확률을 계산하고자 하는 것이다:

$$P(w_n|w_1^{n-1}) = P(w_n|c_n)P(c_n|c_1^{n-1}) \qquad (2)$$

여기에서 클래스 c_k는 단어 w_k가 속하는 클래스를 지칭한다. 본 논문에서 우리는 n=2 인 바이그램 언어모델에 대해서만 생각한다.

합병의 기준은 훈련말뭉치 t_1^T 의 perplexity 가 크게 되도록 하는 클러스터링 π 가 되게 하는 것으로 이는 t_1^T 의 average log-likelihood L(π)가 가능하면 크게 되도록 하는 것과 같다:

$$L(\pi) = (T-1)^{-1}\log P(t_1^T|t_1) \qquad (3)$$

위의 L(π) 에 대한 식은 (4) 로 표시할 있게 되는데 (자세한 유도는 [1] 참조):

$$L(\pi) = -H(w) + I(c_1, c_2) \qquad (4)$$

$$H(w) = -\sum_w P(w)\log P(w) \qquad (5)$$

$$I(c_1,c_2) = \sum_{c_1c_2} P(c_1c_2)\log\frac{P(c_1c_2)}{P(c_1)P(c_2)} \qquad (6)$$

여기에서 $H(w)$ 는 클러스터링 π 에 대하여 상수이므로 단지 $I(c_1,c_2)$ 를 최대화하는 π 가 되도록 하는 것이 클러스터링의 합병 기준이 된다. $I(c_1,c_2)$ 는 "인접 클래스간 평균 상호정보"를 나타내는 값이 된다.

3. 단어 클러스터링 알고리즘 개발

3.1. Brown 2 알고리즘 및 속도 향상 기법

Brown-2 클러스터링 기법의 대략적인 구성은 그림 1과 같다. 여기서 알고리즘 시작 전에 변수 V'과 k 를 제공하여야 한다. 그러면 이 알고리즘은 출현 빈도가 높은 V' 개의 단어들만을 대상으로 하여 클러스터링을 수행한다 ($V' = V$ 로 하면 전체 단어에 대한 클러스터링이 됨). 그리고 k 는 최종 클러스터가 가질 클래스 수를 나타낸다.

(1) 모든 단어를 훈련말뭉치에서의 출현빈도순으로 내림차순 정렬하여 단어리스트 WL 에 넣는다.
(2) /* 초기 클러스터 준비작업 단계 */
 · 클래스 $c_1, ..., c_{k+1}$ 각각을 WL 에서 첫 $k+1$ 개의 단어를 꺼내어 1 단어씩 가지도록 초기화한다.
 · 처리 단어수 W_{cnt} = $k+1$ 로 놓는다.
 · $c_1, ..., c_{k+1}$ 클래스 중 합병기준에 가장 적합한 쌍 i, j 을 선택한다.
(3) /* 합병 단계 */
 · 앞 단계에서 정해준 클래스 i 와 j 를 합병한다.
 (클래스 수는 k 개가 된다.)
(4) If (W_{cnt} == V') 이면 클러스터링 알고리즘을 종료한다.
(5) /* 다음 단어를 새 클래스로 추가하는 단계 */
 · WL 의 다음 단어를 꺼내어 새 클래스c_{k+1} 를 생성한다.
 · 처리 단어수 W_{cnt} 를 1 만큼 증가한다.
 · $c_1, ..., c_{k+1}$ 클래스 중 합병기준에 가장 적합한 쌍 i, j 을 선택한다.
(6) 단계 3 으로 간다.

그림 1. Brown-2 클러스터링 알고리즘

이 알고리즘의 핵심은 항상 클래스 수를 k 또는 k+1 로 유지하는 것이다. 단계 3에서 클래스 수가 k 인 상황에서(각 클래스를 1에서 k 까지의 번호로 나타냄) 단계 5에서는 다음 처리할 단어 하나만을 가진 번호 k+1 인 클래스를 생성한 후 k+1개의 클래스 중에서 합병기준에 가장 부합하는 두 개의 클래스를 선정한다. 여기서 조사하여야 할 쌍의 수는 조합의 수 $\binom{k+1}{2}$로서 대략 k^2 로 볼 수 있다. 이 알고리즘은 단계 3~6의 루프를 $V'-k$ 번 반복한다.

합병할 클래스 쌍의 선정에 이용할 합병기준은 앞의 식 (6)에서 소개한 클러스터 평균상호정보 $I(c_1,c_2)$이다. 이것이 최대화 되는 쌍을 선정하여야 한다. 결국 이 결정을 위해 모든 가능한 쌍마다 이 값을 계산해 보아야

한다. 그러나 Brown 에서는 각 쌍마다 I를 계산하는 대신에 상호정보 감소량 L을 이용하는 것을 제안하였다. L이 최소화되는 쌍은 바로 I를 최대화하는 쌍과 동일하기 때문이다. 그런데 중요한 발견은 I의 계산에는 O(k^2), L의 계산량은 O(k)이므로 L을 합병기준으로 이용할 경우에는 상당한 속도 향상이 가능하다는 점이다. 그러나 우리는 L의 계산에 있어서 Brown에서 제시한 식이 올바름을 증명할 수 없었고 따라서 더 직관적으로 쉬운 다른 식을 이용한다. 하지만 우리의 식 역시 O(k) 만큼의 시간만 필요하므로 역시 속도 향상을 얻을 수 있다.

3.2. 파라메타의 종류

위 알고리즘은 클러스터와 관련하여 다음과 같은 종류의 파라메타를 필요로 한다:

- $P^1(l)$: 클래스 l 에 속하는 단어의 출현확률.
- $P^2(l,m)$: 클래스 l, m에 속하는 단어가 이어서 나올 확률.
- $Q(l,m)$: 가중치를 곱한 클래스 l, m 의 상호 정보로서
$$Q(l,m) = P^2(l,m)\log\frac{P^2(l,m)}{P^1(l)P^1(m)}.$$
- $s(l)$: 클래스 l 과 관련된 모든 q 의 합.
- $I(i,j)$: 클래스 i, j 를 합병한 후의 상호정보.
- $L(i,j)$: 클래스 i, j 를 합병한 후의 상호정보 감소량으로 (합병이전의 I 값) − (합병 후 I 값)과 같다.

클러스터링 과정에서 π 가 가진 클래스의 수는 k 인 경우와 k+1 인 경우의 2가지만 가능하다. 이 둘 중에 어느 경우인지를 나타내기 위해 위 파라메타의 아래첨자에 k 또는 k+1 을 표시하여 이를 나타내도록 한다.

3.3. 단계 2: 초기 클러스터의 준비작업

이 단계에서 각 파라메타의 준비는 다음과 같다. 아래에서 C(w) 는 단어 w의 출현 횟수, T 는 훈련말뭉치의 크기를 나타낸다.

$$P^1_{k+1}(l) = \frac{C(w)}{T}$$

$$P^2_{k+1}(l,m) = \frac{C(w_1,w_2)}{T}$$

$$Q_{k+1}(\ell,m) = P^2_{k+1}(\ell,m)\cdot\log P^2_{k+1}\frac{P^2_{k+1}(\ell,m)}{P^1_{k+1}(\ell)+P^1_{k+1}(m)}$$

$$S_{k+1}(i) = \sum_m Q_{k+1}(m,i)+\sum_m Q_{k+1}(i,m)-Q_{k+1}(i,i)$$

위에서 l, m은 1에서 k+1 사이의 모든 정수를 말한다. 우리는 이 클러스터에서 모든 클래스 쌍 i, j 를 합병할 경우를 고려하여야 하며 가능한 쌍의 수는 k^2 에 비례한다. 클래스 i와 j를 합병한 후의 상호정보는

$$I(i,j) = \sum_{l,m} Q_{k+1}(\ell,m)$$

위 $I(i,j)$의 계산량은 모든 가능한 l, m 에 대한 합을 구하므로 O(k^2) 이 된다. 그러나 그 대신에 다음과 같이 상호정보감소량 L을 구하는 작업은 O(k) 이므로 상당한 계산량 감소 효과를 얻는다.

$$\begin{aligned}&L_{k+1}(i,j)\\&= S_{k+1}(i)+S_{k+1}(j)-Q_{k+1}(i,j)-Q_{k+1}(j,i)\\&-[Q_{k+1}(i+j,i+j)+\sum_l Q_{k+1}(l,i+j)+\sum_m Q_{k+1}(i+j,m)]\end{aligned}$$

위에서 l,m은 i도 j도 아니면 1에서 k+1 사이의 모든 정수를 말한다. 위식에서 Q 의 계산은 다음과 같다.

$$\begin{aligned}&Q_{k+1}(i+j,i+j)\\&= P^2_{k+1}(i+j,i+j)\cdot\log\frac{P^2_{k+1}(i+j,i+j)}{P^1_{k+1}(i+j)\cdot P^1_{k+1}(i+j)}\end{aligned}$$

여기에서 이용한 P 의 계산은 다음과 같다:

$$\begin{aligned}&P^2_{k+1}(i+j,i+j)\\&= P^2_{k+1}(i,i)+P^2_{k+1}(j,j)+P^2_{k+1}(i,j)+P^2_{k+1}(j,i)\end{aligned}$$

결과적으로 모든 가능한 L_{k+1} 을 구하는 계산량은 $O(k^3)$ 을 가진다.

이러한 모든 $L_{k+1}(i,j)$ 중에서 가장 작은 값을 주는 i, j 쌍을 구한다. 이것은 $L_{k+1}(i,j)$ 하나하나마다 그 값이 구해질 때마다 min과 비교하면서 min을 갱신하면서 구하게 된다. 따라서 가장 작은 L_{k+1} 을 주는 i, j 를 구하는데 필요한 계산량은 역시 $O(k^3)$ 과 같다.

3.4. 단계 3: 두 클래스를 합병한 후의 작업

이 단계에서는 앞 단계에서 결정한 클래스 i 와 j (단, i <j)를 합병한 후의 클러스터에 대한 파라메타를 준비한다.

$P^1_k(l) = P^1_{k+1}(l)$ (l 은 i도 j도 아닌 $1 \sim k$ 사이의 모든 정수)

$P_k^1(i) = P_k^1(i) + P_k^1(j)$

if $j < (k+1)$, $P_k^1(j) = P_k^1(k+1)$.

다음에서 l, m 은 i도 j도 아닌 $1 \sim k$ 사이의 모든 정수를 말한다.

$P_k^2(l,m) = P_{k+1}^2(l,m)$
$P_k^2(m,i) = P_{k+1}^2(m,i) + P_{k+1}^2(m,j)$
$P_k^2(i,m) = P_{k+1}^2(i,m) + P_{k+1}^2(j,m)$
$P_k^2(i,i) = P_{k+1}^2(i,i) + P_{k+1}^2(i,j) + P_{k+1}^2(j,i) + P_{k+1}^2(j,j)$

위의 계산량은 $O(k^2)$ 이다.

다음은 j 가 k+1보다 작은 경우에 계산한다 (m 은 i도 j도 아닌 $1 \sim k$ 사이의 모든 정수).

$P_k^2(m,j) = P_{k+1}^2(m,k+1)$
$P_k^2(j,m) = P_{k+1}^2(k+1,m)$

$P_k^2(i,j) = P_{k+1}^2(i,k+1) + P_{k+1}^2(j,k+1)$
$P_k^2(j,i) = P_{k+1}^2(k+1,i) + P_{k+1}^2(k+1,j)$
$P_k^2(j,j) = P_{k+1}^2(k+1,k+1)$

모든 P_k^2 의 계산마다 대응하는 Q_k 를 다음처럼 계산해 준다 :

$$Q_k(i,j) = P_k^2(i,j) \cdot \log \frac{P_k^2(i,j)}{P_k^1(i) \cdot P_k^1(j)}$$

i도 j도 아닌 $1 \sim k$ 사이의 모든 l 마다 다음을 계산한다.

$S_k(l) = S_{k+1(l)} - Q_{k+1}(l,i) - Q_{k+1}(i,l)$
$- Q_{k+1}(l,j) - Q_{k+1}(j,l) + Q_k(l,i) + Q_k(i,l)$

$s_k(i) = \sum_l Q_k(l,i) + \sum_l Q_k(i,l) - Q_k(i,i)$ (l 은$1 \sim k$ 사이의 모든 정수이다. 결국 계산량은 $O(k)$ 이다).

if $j < (k+1)$,
$s_k(j) = s_{k+1}(k+1) - Q_{k+1}(i,k+1) - Q_{k+1}(k+1,i)$
$- Q_{k+1}(j,k+1) - Q_{k+1}(k+1,j)$
$+ Q_k(j,i) + Q_k(i,j)$

다음에서 l, m 은 i도 j도 아닌 $1 \sim k$ 사이의 모든 정수를 말한다 (단, $l < m$ 보다 작은 경우에만 계산한다).

$L_k(l,m) = L_{k+1}(l,m)$
$+ [Q_k(l,i) + Q_k(m,i) + Q_k(i,l) + Q_k(i,m)]$
$- [Q_{k+1}(l,i) + Q_{k+1}(i,l) + Q_{k+1}(m,i) + Q_{k+1}(i,m)$
$+ Q_{k+1}(l,j) + Q_{k+1}(j,l) + Q_{k+1}(m,j) + Q_{k+1}(j,m)]$
$- [Q_k(l+m,i) + Q_k(i,l+m)]$
$+ [Q_{k+1}(l+m,i) + Q_{k+1}(i,l+m)$
$+ Q_{k+1}(l+m,j) + Q_{k+1}(j,l+m)]$

위 식 자체의 계산량은 O(1) 이며 모든 l, m 에 대해서는 $O(k^2)$ 이 된다. 이 점이 전체 알고리즘의 계산량을 줄이는 핵심적인 역할을 한다. 만약 상호정보 I를 합병기준으로 이용한다면 $I(l,m)$ 하나의 계산에 O(k) 가 소요된다(여기서 O(k^2) 이 아닌 이유는 변경된 항들만을 갱신하여 주기 때문이다). 모든 l, m 에 대해서는 O(k^3) 이 소요된다. 결국 I 대신 L을 합병기준으로 이용하면 O(k) 만큼의 속도향상을 얻을 수 있다.

다음에서 l 은 i 가 아니며 $1 \sim k$ 사이의 모든 정수에 대해서 계산한다. m 은 i 도 l 도 아닌 $1 \sim k$ 사이의 모든 정수에 대해서 계산한다.

$L_k(l,i) = [S_k(l) + S_k(i) - Q_k(l,i) - Q_k(i,l)]$
$- [\sum_m Q_k(m,l+i) + \sum_m Q_k(l+i,m) + Q_k(l+i,l+i)]$

결국 모든 l 에 대한 위의 계산량은 $O(k^2)$ 이 된다.

만약 $j < k+1$ 이면, 다음을 계산한다. 여기에서 l 은 i도 j도 아닌 $1 \sim k$ 사이의 모든 정수를 말한다. 따라서 아래 모든 L_k 의 계산량은 $O(k)$ 가 된다.

$L_k(l,j) = L_{k+1}(l,k+1)$
$+ [Q_k(l,i) + Q_k(j,i) + Q_k(i,l) + Q_k(i,j)]$
$- [Q_k(l+j,i) + Q_k(i,l+j)]$
$+ [Q_{k+1}(l+(k+1),i) + Q_{k+1}(i,l+(k+1))$
$Q_{k+1}(l+(k+1),j) + Q_{k+1}(j,l+(k+1))]$
$+ [Q_{k+1}(l,i) + Q_{k+1}(i,l) + Q_{k+1}((k+1),i)$
$+ Q_{k+1}(i,(k+1)) + Q_{k+1}(l,j) + Q_{k+1}(j,l)$
$+ Q_{k+1}((k+1),j) + Q_{k+1}(j,(k+1))]$

위에서

$Q_{k+1}(l+(k+1),i)$

$$= P_{k+1}^2(l+(k+1),i) \cdot \log \frac{P_{k+1}^2(l+(k+1),i)}{P_{k+1}^1(l+(k+1)) \cdot P_{k+1}^1(i)}$$

위 식에서

$P_{k+1}^2(l+(k+1),i) = P_{k+1}^2(l,i) + P_{k+1}^2((k+1),i)$
$P_{k+1}^1(l+(k+1)) = P_{k+1}^1(l) + P_{k+1}^1(k+1)$

결국 이 단계의 전체적인 계산량은 $O(k^2)$ 이 된다.

3.5. 단계 5: 클래스의 추가 작업

다음은 다음 단어 w 하나만을 가지는 클래스 하나를 추가한 후의 클러스터에 대한 파라메타 준비 작업에 대한 것이다. 아래에서 l, m 은 $1 \sim k$ 사이의 모든 정수에 대해서 계산한다.

$$P_{k+1}^1(m) = P_k^1(m)$$
$$P_{k+1}^1(k+1) = \frac{C(w)}{T}$$
$$P_{k+1}^2(l,m) = P_{k+1}^2(l,m)$$
$$P_{k+1}^2(l,k+1) = \sum_{w \in c_l} \frac{C(w',w)}{T}$$
$$P_{k+1}^2(k+1,m) = \sum_{w' \in c_m} \frac{C(w,w')}{T}$$
$$P_{k+1}^2(k+1,k+1) = \frac{C(w,w)}{T}$$

위에서 각 P_{k+1}^2 마다 해당 Q_{k+1}도 그때 같이 계산해 준다. 결국 위 계산량은 $O(k^2)$ 이 된다.

$$S_{k+1}(l) = S_k(l) + Q_{k+1}(l,k+1) + Q_{k+1}(k+1,l)$$
$$S_{k+1}(k+1) = \sum_m Q_{k+1}(m,k+1) + \sum_m Q_{k+1}(k+1,m) + Q_{k+1}(k+1,k+1)$$

$$L_{k+1}(l,m) = L_k(l,m) + [Q_{k+1}(l,k+1) + Q_{k+1}(k+1,l) + Q_{k+1}(m,k+1) + Q_{k+1}(k+1,m)] - [Q_{k+1}(l+m,k+1) + Q_{k+1}(k+1,l+m)]$$

위에서 l, m은 1에서 k 까지의 모든 정수를 말한다. 결국 모든 L_{k+1} 을 구하는 데 필요한 계산량은 $O(k^2)$ 이다.

$$L_{k+1}(l,k+1) = [S_{k+1}(l) + S_{k+1}(k+1) - Q_{k+1}(l,k+1) - Q_{k+1}(k+1,l)] - [\sum_m Q_{k+1}(m,l+(k+1)) + \sum_m Q_{k+1}(l+(k+1),m) + Q_{k+1}(l+(k+1),l+(k+1))]$$

위에서 m 은 l 이 아닌 1에서 k 사이의 모든 정수를 말한다. 결국 위 L_{k+1} 을 구하는 계산량은 $O(k^2)$ 이다. 위 두 식에서 구한 L_{k+1} 을 최소로 하는 i, j 를 찾는 방법은 앞 절과 같다. 결국 단계 4는 $O(k^2)$ 의 계산량을 가진다.

결과적으로 각 단계에 대한 계산량을 기반으로 하여 **전체 알고리즘**의 계산량을 구하면 $O(k^3 + V'k^2)$ 이 된다. 이것이 우리가 제안하는 효율적인 클러스터링 알고리즘의 성능이 된다.

4. 성능 측정 실험

우리가 제안하는 알고리즘은 다음과 같은 2 가지 관점에서 효율성의 향상을 시도하였다. 먼저 클러스터 안의 모든 클래스 쌍의 합병으로 인한 상호정보의 계산 작업에 대한 것이다. 가능한 쌍의 수는 k^2 개 이다. 만약 상호정보 자체를 계산하는 방식을 취하면 각 $I(i,j)$의 계산에 $O(k)$ 만큼의 시간이 필요하므로 모든 쌍에 대한 작업은 $O(k^3)$이 된다. 결국 전체 알고리즘의 계산량은 $O(k^3 + V'k^3)$이 된다. 그러나 상호정보의 감소량 $L(i,j)$를 구하는 방식을 취하면 하나의 $L(i,j)$ 의 계산에 $O(1)$의 시간이 소요되므로 바로 앞 절에서 언급한 대로 전체 알고리즘의 성능은 $O(k^3 + V'k^2)$이 된다. 이러한 점에서 우리의 기법은 계산량의 감소를 얻는다.

두 번째로 제안하는 성능향상에 대한 기법은 $C(w_1,w_2)$ 정보를 하드디스크에 관리하는 대신 메모리에 넣고 이용할 수 있도록 하는 것이다. 모든 가능한 단어 쌍의 수는 약 V^2으로서 대단히 큰 수가 된다 (예를 들어 V=200,000 이라면 40,000,000,000). 따라서 고속처리를 위하여 2차원 테이블을 이용할 경우 320GB 의 저장 공간이 필요하다. 결국 이러한 방식은 메모리에는 구현이 어렵게 된다. 그러나 우리는 속도 향상을 위하여 2차원 테이블을 이용하는 대신에 각 단어쌍에 고유키를 부여하고 실제 출현한 단어쌍만을 저장하는 방식을 취함으로서 메모리에 이 정보를 저장할 수 있도록 하였다. 그 결과 알고리즘의 속도 향상을 얻을 수 있게 되었다.

우리는 2천만 어절의 품사부착 세종말뭉치를 기반으로 하는 클러스터링 실험을 수행하였다. 여기에서 얻은 전체 단어의 수 V는 약 27만 어절이다. 표 1은 실험 결과 얻은 여러 경우에 대한 Brown 2 클러스터링 알고리즘의 소요 시간 측정치를 보여 준다. Loss은 본 논문에서 제안하는 상호정보 감소량 L을 합병기준으로 이용하는 방법을 나타내며, MI는 상호정보 I를 합병기준으로 이용하는 방법을 나타낸다. (아래 표에서 V'=100,000 에 대한 MI 의 실험은 너무 많은 시간이 소요되어 시간을 측정

하지 못하였다.) 표에서 Memory 는 $C(w_1, w_2)$의 정보를 메모리에 넣고 이용하는 방법을, Disk 는 하드디스크에 넣고 이용하는 방법을 나타낸다.

표 1. 성능 측정 실험 결과

	Memory		Disk	
	k/V'	Time	k/V'	Time
Loss (L)	100/10000	1.58m	100/10000	8.34m
	100/50000	28.44m	100/50000	5h12m
	100/100000	1h11m	100/100000	23h1m
	200/10000	2.56m	200/10000	12.31m
	200/50000	33.63m	200/50000	6h15m
	500/10000	8.95m	500/10000	24.54m
	500/50000	1h17m	500/50000	6h13m
MI (I)	100/10000	4.85m	100/10000	12.52m
	100/50000	1h8m	100/50000	7h54m
	200/10000	20.53m	200/10000	46.34m
	200/50000	2h53m	200/50000	26h15m
	500/10000	4h5m	500/10000	9h50m
	500/50000	36h5m	500/50000	122h49m

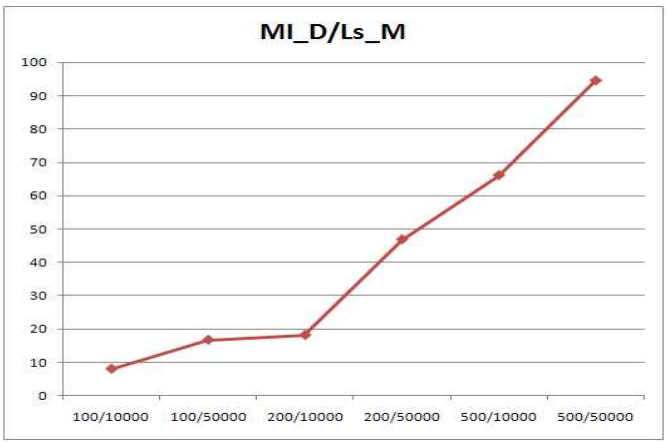

그림 2. 속도 향상 효과

우리는 그림 2에 위 표의 4 가지 방법들 중에서 가장 느린 방법 MI_D(MI, Disk)과 가장 빠른 방법 Ls_M(Loss, Memory) 사이의 소요시간의 비(ratio)를 나타내었다. 이를 보면 작업량이 많아지면 속도향상의 효과가 커지는 현상을 관찰할 수 있다.

5. 결론

언어 모델 기반의 단어 클러스터링 작업은 많은 양의 계산을 필요로 하므로 실용화에 어려움이 많다는 문제점이 있다. 이에 본 논문에서는 이러한 문제를 개선하기 위한 알고리즘 효율성 향상 기법을 연구하였다. 성능 개선의 대상이 되는 클러스터링 알고리즘으로는 가장 널리 알려진 Brown의 클래스 기반 n-gram 모델에 기반한 단어 클러스터링 알고리즘을 택하였다. 속도향상을 위해 합병기준으로 상호정보 감소량의 이용 그리고 단어쌍 공기정보를 메모리에 저장하는 방법을 이용하는 것을 제안한다. 실험 결과 가장 단순한 방법보다 약 7.9배 이상의 속도 향상을 얻을 수 있음을 관찰하였다.

참고문헌

[1] P.F. Brown, V.J. Della Pietra, P. V. DeSouza, J.C. Lai, and R.L. Mercer, "Class-Based n-gram Models of Natural Language," Computational Linguistics, V. 18, No. 4, pp. 283-298, 1992.

[2] T. Koo, X. Carreras and M. Collins, "Simple Semi-supervised Dependency Parsing," Proceedings of ACL 2008, 2008.

[3] H. Li and N. Abe, "Word Clustering and Disambiguation Based on Co-occurrence Data," Proceeding of COLING '98, pp. 749-755, 1998.

[4] F. Pereira, N. Tishby and L. Lee, "Distributional Clustering of English Words," Proceedings of ACL '93, pp. 183-190, 1993.

[5] S.K. Saha, P. Mitra and S. Sarkar, "Word Clustering and Word Selection based Feature Reduction for MaxEnt based Hindi NER," Proceedings of ACL '08, pp. 488-495, 2008.

[6] S. Mori and M. Nagao, "A stochastic language model using dependency and its improvement by word clustering," Proceedings of Coling '98, pp. 898-904, 1998.

[7] L.D. Baker and A.K. McCallum, "Distributional Clustering of Words for Text Classification," Proceedings of SIGIR '98, 1998.

[8] G.A. Miller, "WordNet: a lexical database for English," Communications of the ACM, V. 38, Issue 11, 1995.

[9] 최호섭, 옥철영, "한국어 의미망 구축과 활용," 한국어학 제17집, 한국어학회, pp.301-329, 2002.