자기 조직화 *n*-gram모델을 이용한 자동 띄어쓰기*

태윤식 박성배 이상조 박세영
경북대학교 컴퓨터공학과
{ystae,sbpark,sjlee,sypark}@sejong.knu.ac.kr

Self-Organizing *n*-gram Model for Automatic Word Spacing

Yoon-Shik Tae Seong-Bae Park Sang-Jo Lee Se-Young Park
Dept. of Computer Engineering, Kyungpook National University

요 약

한국어의 자연어처리 및 정보검색분야에서 자동 띄어쓰기는 매우 중요한 문제이다. 신문기사에서조차 잘못된 띄어쓰기를 발견할 수 있을 정도로 띄어쓰기가 어려운 경우가 많다. 본 논문에서는 자기 조직화 *n*-gram모델을 이용해 자동 띄어쓰기의 정확도를 높이는 방법을 제안한다. 본 논문에서 제안하는 방법은 문맥의 길이를 바꿀 수 있는 가변길이 *n*-gram모델을 기본으로 하여 모델이 자동으로 문맥의 길이를 결정하도록 한 것으로, 일반적인 *n*-gram모델에 비해 더욱 높은 성능을 얻을 수 있다. 자기조직화 *n*-gram모델은 최적의 문맥의 길이를 찾기 위해 문맥의 길이를 늘였을 때 나타나는 확률분포와 문맥의 길이를 늘이지 않았을 때의 확률분포를 비교하여 그 차이가 크다면 문맥의 길이를 늘이고, 그렇지 않다면 문맥의 길이를 자동으로 줄인다. 즉, 더 많은 정보가 필요한 경우는 데이터의 차원을 높여 정확도를 올리며, 이로 인해 증가된 계산량은 필요 없는 데이터의 양을 줄임으로써 줄일 수 있다. 본 논문에서는 실험을 통해 *n*-gram모델의 자기 조직화 구조가 기본적인 모델보다 성능이 뛰어나다는 것을 확인하였다.

1. 서론

한국어의 기본적인 띄어쓰기 규칙은 매우 간단하지만, 자동 띄어쓰기는 실제로는 간단한 문제가 아니다. 기본적인 규칙은 모든 개념적인 단위 사이에 모두 띄어쓰기를 하는 것이지만, 불완전 명사에서 조사를 구분하기가 쉽지 않은 것처럼 조사 및 어미와 관련하여 수많은 예외가 있다. 이러한 예외는 띄어쓰기에서 많은 오류를 발생시키며, 심지어 신문기사에서도 이러한 오류들을 종종 볼 수 있다.

잘못된 띄어쓰기는 잘못된 형태소 분석 결과를 발생시키므로 자연어 처리 및 정보검색에서 치명적인 문제가 될 수 있다. 예를 들어 "아버지가방에들어가셨다."라는 문장이 주어졌을 때 실제 올바른 띄어쓰기는 "아버지가 방에 들어가셨다."이지만, 만약 "아버지 가방에 들어가셨다."와 같이 띄어쓰기가 된다면, 원래 의미와는 전혀 다른 형태소 분석 결과가 나올 것이다. 형태소 분석은 거의 모든 자연어처리분야에서 가장 첫 단계가 이므로 만일 잘못된 띄어쓰기가 있으면 반드시

형태소 분석이 되기 전에 수정되어야 한다. 또한 정보검색에 있어서 잘못된 띄어쓰기는 각 단어에 대해 잘못된 색인어를 생성하는 결과를 가져온다.

자동 띄어쓰기에 있어서 가장 간단하면서도 강력한 모델 중 하나는 *n*-gram 모델이다. *n*-gram 모델은 많은 장점을 갖고 있지만 높은 성능을 얻기에는 한계가 있다. *n*-gram모델의 가장 큰 문제는 고정된 윈도우 크기인데, 만약 *n*이 작다면 고려되는 문맥의 크기가 작으므로 대개 좋은 성능을 보여주지 못하고, 더 나은 성능을 위해서 *n*의 크기를 늘리면 데이터 부족의 문제가 발생하게 된다. 즉, 말뭉치의 크기는 물리적으로 제한되어있으므로, 많은 *n*-gram모델에서 실제로는 나타나는 데이터이지만 말뭉치에는 나타나지 않는 데이터가 있을 가능성이 매우 높다.

본 논문에서는 *n*-gram모델을 이용한 새로운 자동 띄어쓰기 처리 방법을 제안하며, 이 방법은 각각의 입력

* 본 논문은 정통부 및 정보통신연구진흥원의 정보통신선도기반기술개발사업의 연구결과로 수행되었음

데이터에 대해 자동으로 윈도우의 크기 n을 조절하도록 되어있다. 즉, 문맥의 크기가 bigram에서 시작하여 데이터 부족의 문제가 발생하면 unigram까지 축소되기도 하며, 띄어 쓸 것인가 말 것인가를 결정하기 위해 더욱 자세한 정보가 필요한 경우는 trigram, fourgram 또는 그 이상으로 커지기도 한다. 다시 말하면, 본 논문에서 제안한 방법에서는 윈도우의 크기 n을 각각의 입력 데이터에 대해 온라인으로 결정하여 데이터 부족 문제와 정보 결핍 문제를 제거함으로써 높은 정확도를 얻도록 하였다.

본 논문은 다음과 같이 구성되어 있다. 2절에서는 이전에 이미 연구된 자동 띄어쓰기 및 n-gram모델에서의 평탄화 방법들에 대해 살펴 보며, 3절에서는 기본적인 n-gram모델을 이용한 자동 띄어쓰기에 대해 다룬다. 4절에서는 일반적인 n-gram모델의 약점을 보완할 수 있는 자기 조직화 n-gram모델을 제안하며, 5절에서는 이와 관련된 실험결과를 보여준다. 마지막으로 6절은 결론으로 본 논문에서 제안한 방법의 특징과 그 가능성에 대해 언급한다.

2. 관련 연구

자동 띄어쓰기에 관한 연구들은 이미 많이 이루어져 있으며 그 결과들도 널리 알려져 있다. 이러한 연구들에서 제안된 방법들은 크게 분석적인 방법과 통계적인 방법 두 가지로 분류할 수 있다. 분석적인 접근방법은 형태소 분석에 그 기초를 두고 있다[1,2]. 하지만 이러한 형태소 분석기술을 이용한 접근방법에는 두 가지 단점이 있는데, 하나는 형태소를 분석하는 절차가 매우 복잡하다는 것이며, 다른 하나는 분석한 결과를 유지하고 이를 응용하는데 비용이 많이 든다는 것이다.

반면에 통계적인 접근방법은 두 음절 사이를 띄울 것인가 말 것인가를 확률적으로 계산하며, 그 확률을 말뭉치로부터 얻는다. 이러한 방법은 모델을 디자인하는 사람이 필요한 정보를 직접 지정해주는 것이 아니라 모델이 자동으로 얻어 사용하며, 그러므로 언어학적인 지식이 비교적 많이 필요로 하지 않는다. 따라서 이러한 지식을 체계화시키고 유지하는데 드는 비용이 적게 들게 된다. 또한 이러한 접근방법에서는 형태소 분석기를 사용하지 않으므로, 새로운 단어가 입력되더라도 안정적인 성능을 보인다.

이전의 말뭉치를 이용한 자동 띄어쓰기 관련연구들에서는 주로 bigram을 이용하였으며, 현대의 한국어에서의 음절의 수가 약 10^4개이므로, 이에 따른 bigram의 수는 10^8개이다[3]. 따라서, 모든 가능한 bigram을 고려하였을 때 안정적인 통계를 얻기 위해서는 매우 큰 크기의 말뭉치가 필요하다. 만일 띄어쓰기의 높은 정확도를 위해서 더 큰 크기의 윈도우를 사용하였다면 필요한 말뭉치의 크기는 지수적으로 증가할 것이다.

그렇기 때문에 말뭉치의 크기가 아무리 크다 해도 빈도가 0인 n-gram은 나타나므로 데이터의 부족문제를 항상 겪을 수 있으며, 이는 n-gram모델의 가장 큰 문제이다.

이러한 문제점을 해결하기 위해서 n-gram모델에 적용할 수 있는 많은 평탄화 방법들이 연구되어 왔으며[4] 이러한 방법들은 크게 두 가지로 나눌 수 있다. 첫 번째는 각각의 n-gram이 모두 말뭉치상에 적어도 한 번 이상 나타나는 것처럼 만드는 방법들이며[5] 두 번째는 더 낮은 차원의 데이터를 n-gram에 삽입하는 방법들이다[6,7]. 이러한 방법들은 원래 말뭉치상에서의 분포를 강제적으로 바꾸게 되므로, n-gram을 이용한 학습에 사용된 확률들은 결국 평탄화에 의해 왜곡된 확률이다.

이 외에도 데이터의 부족 때문에 확률이 0이 되는 것을 피하기 위해 사용될 수 있는 또 다른 방법으로 최대엔트로피 모델이 있다[8]. 각각의 모델을 나누어 구성하는 것이 아니라 이들을 하나로 구성함으로써 가지고 있는 모든 정보를 모델을 학습시키는데 활용하는 것이다. 최대엔트로피 모델은 단순하고, 일반적인 특징 때문에 많은 문제에 적용할 수 있으며 좋은 성능을 보여준다는 장점이 있지만, 이에 반해 계산비용이 매우 크며, 또한 모델의 성능이 사전지식(prior knowledge)에 좌우되므로 어떠한 사전지식을 활용할 것인지도 매우 중요한 문제가 된다[9]. 그러므로, 사전지식이 명확하지 않고, 계산비용이 한정된 문제에서는 반드시 최대엔트로피모델 보다는 n-gram모델이 더욱 적합하다고 할 수 있다.

언어를 모델링하는데 있어서 큰 이슈 중 하나는 문맥과 같은 자질들을 어떻게 효율적으로 모델에 적용하는가 하는 것이다[10]. 문제에 따라서는 문맥을 자질로 사용할 때 가까이 있는 문맥만이 아닌 비교적 멀리 떨어져 있는 문맥을 자질로 사용해야 하는 경우도 있다. 이러한 문제를 해결하기 위해서 trigger를 이용하기도 하며[8], particle filter를 이용하기도 한다[11]. 그러나 이러한 방법들은 특정한 언어모델에만 국한되어 적용된다는 단점이 있다.

이와는 달리 멀리 떨어져있는 문맥을 사용하는 대신 지역적으로 문맥을 확장시키는데 중점을 둔 연구들도 있다[12]. 이를 위해 가변 메모리 마코프모델이 제안되기도 하였으며, POS tagging을 위한 선택적인 지역문맥 확장방법도 제안되었다[13,14]. 또한 문법적인 주석(syntactic annotation)을 다는데 있어서 오류를 찾기 위해 확장문맥 n-gram모델을 제안하기도 하였다[15].

이러한 방법들은 바로 옆의 문맥을 확장함으로써 더 많은 정보를 활용하게 되고, 이로 인해 더 나은 성능을 얻을 수 있다.

3. n-gram 모델을 이용한 자동 띄어쓰기

자동 띄어쓰기 문제는 주어진 문장의 각 음절에 대해 띄어 쓸 것인지 혹은 아닌지를 구분하는 분류문제로 볼 수 있다. 주어진 문장을 $S = w_1 w_2 ... w_N$ 라고 했을 때, 모든 데이터들은 같은 분포로부터 독립적으로 샘플링 되었다고 가정한다. 이 문장으로부터 주어지는 데이

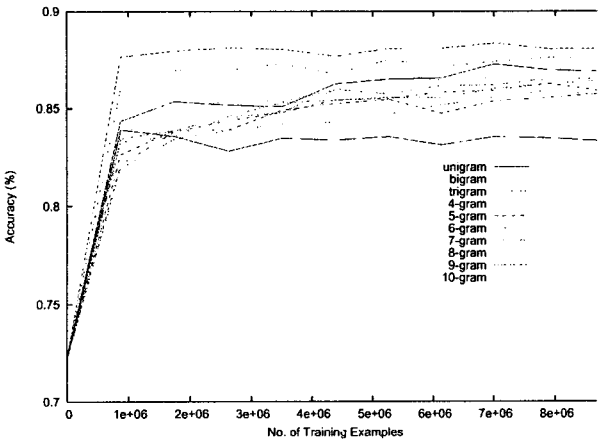

그림 1. *n*-gram모델을 이용한 자동 띄어쓰기에서 문맥의 크기에 따른 성능의 차이.

터는 아래와 같다

$$D = \langle (\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n) \rangle$$
$$\mathbf{x}_i \in \mathbf{R}^n, \quad y_i \in \{true, false\}$$

$\mathbf{x}_i$ 는 음절 w_i의 문맥을 나타내며, w_i 뒤에 띄어 쓰기를 한다면 y_i는 *true*가 되며, 아닐 경우 y_i는 *false*가 된다.

위의 정의를 이용해 자동 띄어 쓰기 문제를 함수 $f : \mathbf{R}^n \rightarrow \{true, false\}$를 구하는 문제로 표현할 수 있다. 즉, 자동 띄어 쓰기 문제는 문맥의 벡터 $\mathbf{x}_i$로 표현된 어떤 음절 w_i가 주어졌을 때 그 뒤에 공백을 두어야 하는지 아닌지를 결정하는 것이다.

여기서 함수 f를 추정하는데 가장 좋은 방법 중 하나는 확률적인 방법으로 각 음절 w_i에 대하여 다음과 같이 나타낼 수 있다.

$$f(\mathbf{x}_i) = \arg\max_{y_i \in \{true, false\}} P(y_i \mid \mathbf{x}_i),$$

여기서 $P(y_i \mid \mathbf{x}_i)$는 베이즈규칙에 의해서 다음과 같이 나타낼 수 있다.

$$P(y_i \mid \mathbf{x}_i) = \frac{P(\mathbf{x}_i \mid y_i) P(y_i)}{P(\mathbf{x}_i)}$$

위의 식에서 $P(\mathbf{x}_i)$는 벡터 $\mathbf{x}_i$의 클래스를 결정하는 문제와는 독립이어서 생략이 가능하므로 함수 $f(\mathbf{x}_i)$는 결국 다음과 같이 나타낼 수 있다.

$$f(\mathbf{x}_i) = \arg\max_{y_i \in \{true, false\}} P(\mathbf{x}_i \mid y_i) P(y_i)$$

n-gram모델에서 $\mathbf{x}_i$ 는 *n*개의 이웃하는 음절들을 나타내며, 보통 bigram이나 trigram을 많이 사용하므로, 대개 *n*은 2 또는 3이 된다. n = 2인 경우 $\mathbf{x}_i$는 $w_{i-1}w_i$가 되며, 같은 방법으로 n = 3인 경우는 $w_{i-2}w_{i-1}w_i$가 된다. 여기서 확률 $P(y_i \mid \mathbf{x}_i)$를 계산하는데 가장 많이 쓰이는 방법은 해당 음절이 나왔을 때의 빈도를 이용하는 것이다. 예를 들어, n = 2라고 하면 $P(y_i \mid \mathbf{x}_i)$는 $P(w_{i-1}w_i \mid y_i)$ 이므로 확률계산은 다음과 같이 할 수 있다.

$$P(w_{i-1}w_i \mid y_i) = \frac{P(w_{i-1}w_i \,\&\, y_i)}{P(y_i)} = \frac{C(w_{i-1}w_i \,\&\, y_i)}{C(y_i)}, \quad (1)$$

C는 빈도를 전체 데이터 집합에서의 빈도를 반환하는 함수이다.

문맥의 크기 *n*을 결정하는 것은 *n*-gram모델의 성능과 계산비용을 좌우하는 가장 중요한 문제이며 이는 말뭉치의 크기에 따라 좌우된다. 문맥의 크기가 너무 크다면 데이터의 부족으로 인해 빈도를 이용한 확률의 계산이 불가능해진다. 반대로 문맥의 크기가 너무 작다면 데이터가 부족한 문제는 없으나 문맥의 정보가 부족하므로 언어의 특징을 잘 반영하지 못하게 되어 정확도가 떨어지게 된다.

대부분의 이전연구들에서는 주로 n이 2 또는 3일 때 가장 좋은 성능을 보여 주었다[16]. 그림 1은 *n*-gram 모델에서 문맥의 크기를 다양하게 적용하여 실험한 결과로 n의 크기에 따른 성능의 차이를 확실하게 보여준다. 그림에 나타난 것과 같이 bigram과 trigram모델이 unigram이나 fourgram등의 모델보다 훨씬 나은 성능을 보였다. 그 중 bigram이 가장 좋은 성능을 보여주었으며, trigram이 그 다음으로 좋은 성능을 보였다. 또한 자동 띄어쓰기를 위한 정보가 가장 부족한 unigram의 성능이 가장 낮게 나타났다.

이 실험을 통해 bigram이 자동 띄어쓰기에서 가장 좋은 성능을 보였으므로, 본 논문에서 제안한 자기조직화 *n*-gram모델에서는 기본적인 문맥의 크기를 2로 설정하였다.

4. 자기 조직화 *n*-gram모델

이 절에서는 문맥의 크기가 고정되어 변하지 않는 기본적인 *n*-gram 모델의 문제점들을 보완하기 위한 자기조직화 *n*-gram 모델의 아이디어와 알고리즘을 제안한다.

4.1. *n*-gram의 확장

일반적으로 *n*-gram 모델과 (*n*+1)-gram 모델의 성능

Function *HowLargeExpand*($\mathbf{x}_n$)

Input: $\mathbf{x}_n$: n-gram

Output: an integer for expanding size

1. Retrieve $(n+1)$-grams $\mathbf{x}_{n+1}$ for $\mathbf{x}_n$.
2. Compute
   $D = KL(P(y|\mathbf{x}_n) \| P(y|\mathbf{x}_{n+1}))$.
3. **If** $D > \theta_{EXP}$ **Then return** 0.
4. **return** *HowLargeExpand*($\mathbf{x}_{n+1}$)+1.

그림 2. 문맥이 확장될 크기를 결정하는 알고리즘

을 비교하면 n-gram에 비해 더욱 많은 정보를 가지고 있는 $(n+1)$-gram 모델이 더욱 나은 성능을 보인다[17]. 하지만, 많은 정보를 보는 만큼 더욱 큰 계산 비용이 소요되므로 $(n+1)$-gram을 사용하였을 때 더 좋은 성능을 기대할 수 있을 때만 $(n+1)$-gram 을 사용하는 것이 당연하다. 즉, $(n+1)$-gram이 n-gram과는 달라야만 $(n+1)$-gram 을 사용할 필요가 있다. 그렇지 않다면, 계산비용만 높아질 뿐 성능은 더 나아지지 않게 된다.

본 논문에서는 n-gram과 $(n+1)$-gram을 비교하기 위해 확률적인 방법을 사용하였다. 즉, 조건부 확률분포 $P(y_i|\mathbf{x}_{ni})$ 와 $P(y_i|\mathbf{x}_{(n+1)i})$ 를 문맥이 주어졌을 때 띄어쓰기에 대한 조건부 확률이라 했을 때 이 두 분포를 비교하여 그 차이가 임계값을 넘어야만 $(n+1)$-gram으로 확장을 한다.

본 논문에서는 두 확률분포 사이의 차이 $D(n, n+1)$를 측정하는 척도로 아래와 같이 Kullback-Leibler divergence를 이용하며, 결과는 식 (2).와 같이 계산하여 얻을 수 있다.

$$D(n,n+1) = KL(P(y_i|\mathbf{x}_{ni}) \| P(y_i|\mathbf{x}_{(n+1)i}))$$
$$= \sum_{z\in\{true,\ false\}} p(z|\mathbf{x}_{ni}) \log \frac{p(z|\mathbf{x}_{ni})}{p(z|\mathbf{x}_{(n+1)i})} \quad (2)$$

만일 위의 식을 이용하여 구한 $D(n, n+1)$의 값이 미리 정의된 임계 값인 θ_{EXP}보다 크다면 이는 $P(y_i|\mathbf{x}_{(n+1)i})$ 가 $P(y_i|\mathbf{x}_{ni})$와는 전혀 다른 분포임을 나타내는 것이다. 즉, 추가된 문맥이 띄어쓰기의 여부를 결정하는데 역할을 하는 문맥이라는 것이다. 그러므로 이런 경우 모델은 n-gram 대신 $(n+1)$-gram을 사용한다.

그림 2는 문맥의 크기를 얼마나 확장해야 하는지를 결정하는 알고리즘을 나타낸다. 예를 들어, 처음에 bigram으로부터 시작한다고 할 때, 다시 말하면 현재의 음절과 그 바로 앞의 음절만을 바탕으로 띄어쓰기를 결정한다고 하면, 먼저 bigram과 trigram의 차이를 위의 식 (2)를 이용해 구하게 되고, 만일 그 차이가 임계값 θ_{EXP} 보다 작을 경우, 다시 trigram과 fourgram의 차이를 구하게 된다. 여기서 그 차이가 임계값 보다 크다고 하면, 함수는 0을 반환하게 되고, 최종적으로 확장할 문맥의 크기는 1이 된다. 즉, bigram대신 trigram을 사용하여 띄어쓰기 여부를 결정하게 된다. 만일 trigram과 fourgram의 차이 역시 θ_{EXP} 보다 큰 경우는 다시 fivegram까지 고려한다.

Function *HowSmallShrink*($\mathbf{x}_n$)

Input: $\mathbf{x}_n$: n-grams

Output: an integer for shrinking size

1. **If** $n < 1$ **Then return** 0.
2. Retrieve $(n-1)$-grams $\mathbf{x}_{n-1}$ for $\mathbf{x}_n$.
3. Compute
   $D = KL(P(y|\mathbf{x}_n) \| P(y|\mathbf{x}_{n-1}))$.
4. **If** $D < \theta_{SHR}$ **Then** return 0.
5. **return** *HowSmallShrink*($\mathbf{x}_{n-1}$)−1.

그림 3. 문맥이 축소될 크기를 결정하는 알고리즘

4.2. *n*-gram의 축소

문맥의 축소는 문맥의 확장과는 완전히 반대의 과정을 거쳐 이루어진다. 만일 n-gram과 $(n-1)$-gram이 차이가 없다면 굳이 n-gram을 사용할 필요가 없으므로 문맥의 크기를 줄인다. 즉, n-gram과 $(n-1)$-gram 의 두 분포의 차이 $D(n, n-1)$의 값이 미리 정의된 임계값 θ_{SHR} 보다 크다면 $(n-1)$-gram이 n-gram대신 사용된다.

문맥의 축소에서도 역시 확장과 마찬가지로 두 분포 사이의 차이를 측정하는 척도로써 아래와 같이 Kullback-Leibler divergence를 사용한다.

$$D(n,n-1) = KL(P(y_i|\mathbf{x}_{ni}) \| P(y_i|\mathbf{x}_{(n-1)i}))$$

그림 3은 문맥의 크기를 얼마나 축소해야 하는지를 결정하는 알고리즘을 나타낸다. 알고리즘의 동작과정은 문맥의 확장 알고리즘과 거의 같으며, 단, 최대 문맥의 크기가 정해져 있지 않은 확장 알고리즘과는 달리 축소 알고리즘에서는 unigram이 최소의 문맥 크기가 된다.

문맥의 축소를 고려해야 하는 이유는 문맥의 크기를 줄이는 것이 단지 계산 비용을 줄이는 역할을 하는 것이 아니라 모델의 성능을 높이는 데에도 큰 영향을 미치기 때문이다.

즉, 필요 없는 정보는 고려하지 않음으로써 입력 데이터의 차원을 줄일 수가 있으며, 입력 데이터의 차원을 줄이게 되면 이에 따라 필요한 데이터의 밀도가 급감하기 때문에 음절이 나타나는 빈도를 이용해 확률을 구하

Function *ChangingWindowSize*($\mathbf{x}_n$)
Input: $\mathbf{x}_n$: n-grams
Output: an integer for changing window size
1. Set exp := *HowLargeExpand*($\mathbf{x}_n$).
2. If exp > 0 Then return exp.
3. Set shr := *HowSmallShrink*($\mathbf{x}_n$).
4. If shr < 0 Then return shr.
5. return 0.

그림 4. 문맥의 크기를 반환하는 함수

는 n-gram 모델에서 말뭉치의 크기를 늘리지 않으면서도 비교적 안정적인 확률을 구할 수 있게 된다.

4.3. 자기조직화 n-gram 모델의 구조

자기조직화 n-gram모델은 문맥의 크기가 변하지 않거나 확장되거나 축소되는 세가지 경우로 나누어 볼 수 있다.

첫 번째, 문맥의 크기가 변하지 않는 경우는 아래의 조건을 만족 할 때 뿐이다.

$$D(n,n+1) \le \theta_{\mathrm{EXP}} \quad \text{and} \quad D(n,n-1) \ge \theta_{\mathrm{SHR}}$$

즉, 확장을 하기에는 현재의 확률분포와 확장했을 때의 확률분포와의 차이가 크지 않으며, 축소하기에는 현재의 확률분포와 축소했을 때의 확률분포와의 차이가 큰 경우이다.

나머지는 문맥의 크기가 반드시 확장되거나 줄어드는 경우이다. 이런 경우에는 확장과 축소 중 어떤 것을 먼저 적용해야 할지를 반드시 고려해야 한다. 예를 들어 분포를 비교했을 때 아래와 같은 경우는 확장이 될 수도 있고 축소가 될 수도 있다.

$$D(n,n+1) > \theta_{\mathrm{EXP}} \quad \text{and} \quad D(n,n-1) < \theta_{\mathrm{SHR}}$$

확장만 되거나, 축소만 되는 경우는 아무 문제가 없지만, 위와 같은 경우에는 확장을 먼저 고려할 것인지, 축소를 먼저 고려할 것인지에 따라 띄어쓰기의 정확도에 차이를 보인다.

본 논문에서는 그림 4와 같이 축소보다는 확장을 먼저 적용하였다. 그림 4의 *ChangeWindowSize*함수에서 먼저 *HowLargeExpand*를 호출함으로써 문맥을 확장할 크기를 받아온 그 값이 0보다 크다면 확장을 위해 그 값을 반환하고, 확장을 하지 않는다면, 즉, exp의 값이 0이라면, 다시 축소를 할것인지를 결정하기 위해 *HowSmallShrink* 함수를 호출하게 된다.

문맥을 확장하게 되면 확장하지 않은 것 보다 더 많은 정보를 가지게 되고, 일반적으로, 더 많은 정보를 가진 모델은 그렇지 않은 모델보다 더욱 높은 정확도를 보인다. 그러므로 본 논문에서는 축소보다는 확장을 먼저 고려하도록 하였다. 또한 확장을 먼저 고려함으로써 늘어나는 계산량은 다른 데이터에서 일어나는 축소연산으로 인해 보완될 수 있다.

4.4. POS Tagging

자연어 문장의 연속적인 특징 때문에 띄어쓰기 문제는 POS 태깅의 문제로 볼 수도 있다[18]. 즉, POS 태깅에서 주로 사용되는 HMM (Hidden Markov Model)을 이용해 띄어쓰기를 풀 수 있다.

HMM를 이용해 구할 수 있는 가장 좋은 띄어쓰기 시퀀스는 아래와 같이 정의되며, N은 문장의 길이이다.

$$\begin{aligned} y^*_{1,N} &= \arg\max_{y_{1,N}} P(y_{1,N} \mid \mathbf{x}_{1,N}) \\ &= \arg\max_{y_{1,N}} \frac{P(\mathbf{x}_{1,N} \mid y_{1,N}) P(y_{1,N})}{P(\mathbf{x}_{1,N})} \\ &= \arg\max_{y_{1,N}} P(\mathbf{x}_{1,N} \mid y_{1,N}) P(y_{1,N}) \end{aligned}$$

각 음절이 모두 각각 독립이라면 $P(\mathbf{x}_{1,N} \mid y_{1,N})$ 는 다음과 같이 나타낼 수 있다.

$$P(\mathbf{x}_{1,N} \mid y_{1,N}) = \prod_{i=1}^{N} P(\mathbf{x}_i \mid y_i)$$

위의 식은 식 (1)을 이용해 계산할 수 있으며 Markov 가정에 따라 현재의 태그 y_i는 앞의 k개의 태그에 만 영향을 받으며, 이를 식으로 표현하면 아래와 같다.

$$P(y_{1,N}) = \prod_{i=1}^{N} P(y_i \mid y_{i-k,i-1})$$

그러므로, 가장 좋은 띄어쓰기의 시퀀스는 다음의 식 (3)과 같이 나타낼 수 있으며, Viterbi알고리즘에 의해 쉽게 계산이 가능하다.

$$y^*_{1,N} = \arg\max_{y_{1,N}} \prod_{i=1}^{N} P(\mathbf{x}_i \mid y_i) \cdot P(y_i \mid y_{i-k,i-1}) \qquad (3)$$

5. 실험

자기조직화 n-gram모델을 평가하기 위한 실험에서 말뭉치는 KISTI에서 배포한 HANTEC 코퍼스 버전 2.0 중에서 다른 문서들보다 비교적 띄어쓰기가 잘 되어있는 데이터를 사용하기 위해 한국일보의 1994년의 신문기사를 모아놓은 HKIB94를 사용하였다.

표 1. 여러가지 모델들의 자동 띄어쓰기 성능

모델	정확도(%)
baseline	72.19
bigram	88.34
trigram	87.59
자기조직화 bigram	91.31
decision tree	88.68
support vector machine	89.10

표 2. 확장과 축소의 적용순서에 따른 오류의 수

적용순서	오류의 수
확장한 다음 축소	108,831
축소한 다음 확장	114,343

HKIB94는 22,000개의 신문 기사로 총 12,523,688개의 음절로 구성되어 있으며, 그 중 중복되는 음절과 특수 기호, 숫자와 영어 알파벳을 빼면 2,037개의 음절로 구성되어 있다.

실험을 위해 이 말뭉치를 학습 데이터(70%)와 테스트 데이터(10%) 그리고 held-out 데이터(20%)의 세 부분으로 나누었으며, 이 중 held-out 데이터는 임계값인 θ_{EXP}와 θ_{SHR}를 구하기 위해 사용되었다.

각 데이터 집합의 인스턴스의 수는 학습데이터의 경우 8,766,578개, held-out데이터의 경우 2,504,739개 이며, 테스트 데이터의 경우는 1,252,371개이다. 테스트 데이터의 인스턴스들 중 네가티브 인스턴스가 904,093개로 약 72%를 차지하였으므로, 모델이 모든 테스트 데이터 인스턴스를 네거티브로 판단하는 경우의 성능인 72%의 정확도를 자동띄어쓰기 모델의 base-line으로 정하였다.

5.1. 실험 결과

실험은 본 논문에서 제안한 방법 성능을 다른 기계학습 방법과 비교하기 위해 decision tree와 support vector machine을 이용한 띄어쓰기 실험도 같이 이루어졌다. decision tree를 이용한 실험에서는 *C*4.5[19]를사용하였고 SVM의 실험에서는 SVM^{light}[20]를 사용하였으며, 앞의 3절에서 *n*-gram모델에서 bigram이 제일 나은 성능을 보였으므로 위의 두 기계학습 방법에서 사용된문맥의 크기는 2로 정하였다

여러 가지 방법들을 이용해 실험한 결과가 표 1에 나타나있다. 일반적인 *n*-gram모델에서는 bigram이 88.34%로 가장 좋은 성능을 보였으며, 이보다는 decision tree와 support vector machine이 더욱 나은 성능을 보였다. 하지만 본 논문에서 제안한 문맥의 크기를 자동으로 조직화하는 '자기조직화 bigram' 이 91.31%의 가장 높은 정확도를 보였다. 이는 baseline보다 약 19%, 일반적인 bigram모델보다는 약 3%, 다른 기계학습방법들보다 약 2% 높은 수치이다.

앞서 언급했듯이 확장과 축소의 적용순서에 따라 모델의 성능은 차이를 보인다. 표 2에 적용순서에 따른 성능의 차이가 나타나 있다. 확장보다 축소를 먼저 고려했을 경우 약 5000개의 오류가 더 많이 발생하였으며, 이는 본 논문에서 제시하였듯이 확장을 먼저 고려하여 더 많은 정보를 사용할 수 있도록 했을 때 훨씬 좋은 성능을 보인다는 것을 확인할 수 있다. 확장을 해서 더욱 많은 정보를 보더라도 오류는 발생하게 되는데 그 원인을 두 가지로 설명할 수 있다.

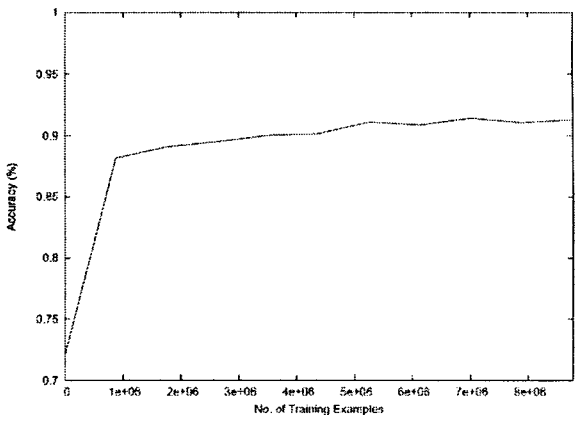

그림 5. 학습 데이터의 수에 따른 자기조직화 n-gram 모델의 성능의 변화

첫 번째는 한국어 자체의 특징 때문이다. 한국어는 부분적으로 단어의 순서가 자유롭고, 생략이 매우 많다. 그러므로 어느 정도까지는 문맥의 크기를 확장하더라도 역시 음절 주위의 정보만을 보기 때문에 띄어쓰기를 하는데 한계가 있다.

두 번째는 데이터 부족문제이다. 즉, 학습 데이터가 부족함으로써 빈도를 이용한 확률계산에서 문맥을 확장한 분포와 그렇지 않은 분포가 완전히 달라질 수 있기 때문이다. 이러한 경우 확장을 하지 않아야 하지만 두 분포의 차이만을 이용해 확장여부를 결정하므로 오류가 발생할 수 있다.

그러나, 실험 결과 데이터 부족으로 인한 오류는 전체 오류에서 큰 부분을 차지하지는 않았다. 그림 5는 학습 데이터 수가 *n*-gram모델의 확률을 계산하는데 큰 영향을 미치지 않는 것을 보여준다. 학습 데이터의 인스턴스 수가 900,000개 이상이 되면 정확도가 증가하기는 하지만 더 이상의 데이터가 성능에 중요한 영향을 끼치지는 않는다. 즉, 본 논문에서 제안한 모델에서 에러를 발생시키는 가장 큰 원인은 학습 데이터의 부족이 아니라 한국어의 특징을 충분히 표현하지 못하는 모델의 문제이다.

5.2. 오른쪽 문맥의 고려

지금까지 본 논문에서 제시한 방법은 현재 음절의 앞

표 3. 문맥의 방향에 따른 효과

문맥	정확도(%)
왼쪽 문맥만 고려	91.31
오른쪽 문맥만 고려	88.26
양쪽 문맥 모두 고려	**92.54**

쪽의 음절의 정보만을 이용하는 것이었다. 하지만, 앞뿐만이 아니라 뒤의 음절의 정보까지도 이용한다면 더 높은 성능을 얻을 수 있다. 이전의 연구에서 두 음절 w_i와 w_{i+1}사이에 띄어쓰기를 정하는 확률 $P(w_i, w_{i+1})$는 다음과 같이 정의되었다[3].

$$\begin{aligned} P(w_i, w_{i+1}) = & 0.25 \times P_L(w_{i-1}, w_i) \\ & + 0.5 \times P_M(w_i, w_{i+1}) \\ & + 0.25 \times P_R(w_{i+1}, w_{i+2}) \end{aligned} \quad (4)$$

P_L, P_M, P_R 은 각각 왼쪽 문맥, 가운데, 오른쪽 문맥의 확률이다. 이 확률들은 각각의 빈도를 이용해 계산한다. 이를 본 논문에서 제시한 모델에 적용하여 식(1)의 $P(w_{i-1}\, w_i | y_i)$ 를 아래의 식과 같이 변경하였다.

$$\begin{aligned} \tilde{P}(w_{i-1}, w_i \mid y_i) = & \alpha_1 \times P(w_{i-2}, w_{i-1} \mid y_i) \\ & + \alpha_2 \times P(w_{i-1}, w_i \mid y_i) \\ & + \alpha_3 \times P(w_i, w_{i+1} \mid y_i) \end{aligned}$$

각 항의 계수 α_1, α_2과 α_3는 held-out데이터를 이용하여 경험적으로 구하였으며, 가장 좋은 성능은 $\alpha_1 = 0.5$, $\alpha_2 = 0.3, \alpha_3 = 0.2$일 때 얻을 수 있었다.

양 방향의 문맥을 모두 고려하였을 때의 성능의 변화는 표 3에 나타난 것과 같이 오른쪽 문맥만을 고려하였을 때 성능이 88.26%로 가장 낮았으며, 양쪽 방향을 모두 고려하였을 때 92.54%로 가장 좋은 성능을 보여주었다. 즉, 오른쪽 문맥정보까지 고려함으로써 1.23% 성능이 향상되었다.

5.3. 띄어쓰기 시퀀스의 고려

지금까지 연구된 결과에 의하면 자동 띄어쓰기 문제에서 가장 좋은 성능을 보인 것은 hidden Markov model로 두 개의 태그 정보와 두 개의 음절을 보았을 때 가장 좋은 성능을 보였다[17].

표 4에 태그 정보를 고려하였을 때의 실험결과가 나타나 있다. 식 (3)의 k의 값은 실험을 간단히 하기 위해 1로 정하였다. 일반적인 HMM모델에서는 고정된 크기의 음절 정보를 보지만, 본 논문에서 제안한 모델에서는 자동으로 문맥의 크기를 가변적으로 정하므로 '자기조직화 HMM모델'이라 하였다. 자기조직화 HMM모델의 성능은 94.71%로 일반적인 HMM 모델보다 2.34% 높은 성능을 보였다. 즉, 본 논문에서 제안하는 태그정보를 고려한 자기조직화 HMM은 지금까지 알려진 다른 어떤 방법보다 좋은 성능을 보였다.

표 4. 확장과 축소의 적용순서에 따른 오류의 수

방법	정확도(%)
HMM	92.37
자기조직화 HMM	**94.71**

6. 결론

본 논문에서는 문맥의 크기를 모델이 스스로 결정하여 띄어쓰기를 하도록 하는 자기조직화 *n*-gram모델을 제안하였다. 이 방법은 단순한 *n*-gram모델을 기반으로 하고 있지만 윈도우의 크기가 사용자에 의해 일정하게 정해지는 것이 아니라 더 많은 정보가 필요한 경우는 크기를 늘림으로써 정확도를 향상시키며, 반대의 경우라면 오히려 윈도우의 크기를 줄임으로써 계산 비용을 줄일 수 있도록 하였다.

현재의 윈도우의 크기를 늘였을 때 확률분포와 늘이지 않았을 때의 확률분포의 차이가 크다면 이는 정보가 더욱 필요하다는 것이며, 모델은 자동으로 앞의 음절을 하나 더 보게 된다. 반대로 윈도우의 크기를 줄였을 때 확률분포와 현재의 확률분포의 차이가 크지 않다면 이는 필요 없는 정보를 가지고 있는 것이므로 윈도우의 크기를 자동으로 줄이게 된다.

본 논문에서의 HANTEC의 말뭉치를 이용한 실험을 통해 자기 조직화 *n*-gram모델의 성능을 평가한 결과 일반적인 trigram을 이용한 모델보다 본 논문에서 제안한 모델이 3.72% 더 높은 정확도를 보였으며, 이 외에도 잘 알려진 기계학습모델들과 비교했을 때에는 Decision Tree보다 2.63% 높고, Support Vector Machine 보다는 2.21% 높은 정확도를 보였다.

또한 자기조직화 *n*-gram모델의 성능을 더욱 높일 수 있는 두 가지 방법을 제안하였는데, 그 중 하나는 앞의 문맥정보만을 추가하는 것이 아니라 뒤의 문맥까지 고려하는 방법이고, 다른 하나는 음절정보만을 고려하는 것이 아니라 띄어쓰기정보까지 같이 고려하는 방법이다. 뒤의 문맥정보를 같이 고려한 경우 그렇지 않았을 때보다 1.23%높은 정확도를 보였으며, 문맥의 띄어쓰기 정보까지 고려하였을 경우는 2.34% 더 높은 정확도를 보였다.

n-gram 모델은 일반적으로 자연어처리나 정보검색분야에서 가장 널리 사용되는 방법으로 본 논문에서 제안한 방법은 자동 띄어쓰기뿐 만 아니라 *n*-gram 모델을 적용할 수 있는 문제라면 모두 적용이 가능하다.

참고 문헌

[1] 강승식, "한글 문장의 자동 띄어쓰기를 위한 어절 블록 양방향 알고리즘", *한국정보과학회 논문지*, Vol. 27, No. 4, pp. 441–447, 2000.

[2] 김계성, 이현주, 이상조, "연속음절 문장에 대한 3단계 한국어 띄어쓰기 시스템", *한국정보과학회 논문지*, Vol. 25, No.12, pp.1838– 1844, 1998.

[3] 강승식, "음절 바이그램 단순화 기법에 의한 한국어 자동 띄어쓰기 시스템의 성능 개선", 제15회 한글 및 한국어 정보처리 학술발표 논문집, pp.227–231, 2004.

[4] S. Chen and J. Goodman. 1996. An Empirical Study of Smoothing Techniques for Language Modeling. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pp. 310–318.

[5] T. Mitchell. 1997. *Machine Learning*. McGraw Hill.

[6] F. Jelinek and R. Mercer. 1980. Interpolated Estimation of Markov Source Parameters from Sparse Data. In *Proceedings of the Workshop on Pattern Recognition in Practice*.

[7] S. Katz. 1987. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*. Vol. 35, No. 3, pp. 400–401.

[8] R. Rosenfeld. 1996. A Maximum Entropy Approach to Adaptive Statistical Language Modeling. *Computer,Speech and Language*, Vol. 10, pp. 187– 228.

[9] S.-B. Park and B.-T. Zhang. 2002. A Boosted Maximum Entropy Model for Learning Text Chunking. In *Proceedings of the 19th International Conference on Machine Learning*, pp. 482–489.

[10] M. Siu and M. Ostendorf. 2000. Variable N-Grams and Extensions for Conversational Speech Language Modeling. *IEEE Transactions on Speech and Audio Processing*, Vol. 8, No. 1, pp. 63–75.

[11] D. Mochihashi and Y. Matsumoto. 2006. Context as Filtering. *Advances in Neural Information Processing Systems 18*, pp. 907–914.

[12] H. Sch¨utze and Y. Singer. 1994. Part-of-Speech Tagging Using a Variable Memory Markov Model. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pp. 181–187.

[13] D. Ron, Y. Singer, and N. Tishby. 1996. The Power of Amnesia: Learning Proba- bilistic Automata with Variable Memory Length. *Machine Learning*, Vol. 25, No. 2, pp. 117–149.

[14] J.-D. Kim, H.-C. Rim, and J. Tsujii. 2003. Self-Organizing Markov Models and Their Application to Part-of-Speech Tagging. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pp. 296–302.

[15] M. Dickinson and W. Meurers. 2005. Detecting Errors in Discontinuous Structural Annotation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pp. 322–329.

[16] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, Vol. 3, pp.1137–1155.

[17] E. Charniak. 1993. *Statistical Language Learning*. MIT Press.

[18] D.-G. Lee, S.-Z. Lee, H.-C. Rim, and H.-S. Lim, 2002. Automatic Word Spacing Using Hidden Markov Model for Refining Korean Text Corpora. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization*, pp.51–57.

[19] R. Quinlan. 1993. *C4.5: Program for Machine Learning*. Morgan Kaufmann Publishers.

[20] T. Joachims. 1998. *Making Large-Scale SVM Learning Practical*. LS8, Universit_t Dortmund.