Sent2Vec 문장 임베딩을 통한 한국어 유사 문장 판별 구현

박상길, 신명철
Sang-Kil Park, MyeongCheol Shin
카카오
Kakao Corp.
{kaon.park, index.shin}@kakaocorp.com

요약—본 논문에서는 Sent2Vec을 이용한 문장 임베딩으로 구현한 유사 문장 판별 시스템을 제안한다. 또한 한글 특성에 맞게 모델을 개선하여 성능을 향상시키는 방법을 소개한다. 고성능 라이브러리 구현과 제품화 가능한 수준의 완성도 높은 구현을 보였으며, 자체 구축한 평가셋으로 한글 특성을 반영한 모델에 대한 P@1 평가 결과 Word2Vec CBOW에 비해 9.25%, Sent2Vec에 비해 1.93% 더 높은 성능을 보였다.

1. 서론

유사 문장 판별은 챗봇을 구현하는 핵심 기술로 그 중요성이 나날이 높아지고 있다. 최근 딥러닝을 이용한 다양한 접근 방법[1]이 좋은 성과를 내면서 주목을 받고 있으나 대부분이 지도 학습에 의존해 대량의 레이블링된 데이터가 필요하다는 한계가 존재한다. 이에 반해 Word2Vec은 비지도 학습 방식[2]이면서도 인상적인 모습을 보여주었고, 최근 NLP 어플리케이션에서 Word2Vec을 이용하는 방식은 사실상 표준으로 자리잡았다.[3]

Word2Vec을 이용하면 단어에 대해서는 매우 유용한 수준의 의미론적 표현이 가능하지만 이를 문장, 단락을 너머 문서 단위의 긴 텍스트에 적용하는 일은 쉽지 않은 일이다. 또한 비지도 학습으로 유용한 방식을 찾아내는 것 또한 매우 도전적인 과제다.

본 논문에서는 Word2Vec의 CBOW 모델을 확장한 Sent2Vec[3]을 통해 문장 단위의 임베딩을 성공적으로 구현한 결과를 보고한다. 또한 문장 임베딩 간의 유사도 판별을 통해 유사 문장 판별 기능을 고성능 라이브러리로 구현하고, 파이썬 바인딩을 이용한 서비스 연동을 통해 곧바로 제품화 가능한 수준(Production-Ready)의 완성도 높은 구현을 보였다. 이에 더해 한글 특성에 맞게 모델을 개선하여 성능을 향상시키는 방법을 소개한다.

2. 관련 연구

Word2Vec을 문장 단위로 확장한 ParagraphVector DBOW 모델[4]은 문장 및 단어 임베딩을 함께 학습한 다음, 소프트맥스 분포를 이용해 문장 벡터에 해당하는 문장에 포함 된 단어를 예측하도록 훈련 된 로그 선형 모델로 Doc2Vec으로도 불린다.[3] 문장 ID를 하나의 단어로 간주해 학습 데이터로 추가하고 Word2Vec과 동일한 방법으로 단어와 레이블에 대한 표현을 동시에 학습하여 문장의 의미가 임베딩 될 수 있게 했다.

Word Mover's Distance[5]는 문장의 단어들이 Word2Vec 공간에 내장된 두 문장이 주어졌을때 문장 1의 모든 단어가 문장 2의 단어들과 정확히 일치하도록 이동해야 하는 최소 누적 거리로 유사도를 판별한다.

Latent Semantic Indexing[6]은 구조화되지 않은 텍스트 모음에 포함 된 용어와 개념 간의 관계에서 패턴을 식별하기 위해 SVD(Singular Value Decomposition)를 사용하는 색인 및 검색 방법이다.[7]

3. 모델

본 논문에서는 유사 문장을 판별하기 위한 모델로 Sent2Vec을 이용한다. Sent2Vec은 범용적인 문장 임베딩을 목표로 하는 비지도 학습 모델로, Word2Vec의 CBOW 모델을 문장 단위로 확장한 모델이다. 문장 임베딩은 아래의 수식으로 정의한다.[3]

$$\boldsymbol{v}_S := \frac{1}{|R(S)|}\boldsymbol{V}\iota_{R(S)} = \frac{1}{|R(S)|}\sum_{w\in R(S)}\boldsymbol{v}_w$$

여기서 $R(S)$는 문장 S에 존재하는 유니그램을 포함한 모든 n-그램 목록이며, 수식의 계산 결과로 문장에 존재하는 모든 요소의 벡터 합에 평균을 취한 값을 문장 임베딩으로 갖는다.

Sent2Vec은 Word2Vec의 CBOW 모델을 기반으로 한다. 따라서 Sent2Vec을 자세히 살펴보기에 앞서 먼저 Word2Vec CBOW 모델의 주요 특징을 살펴보도록 한다.

3.1. CBOW

CBOW는 컨텍스트(주변)가 타겟(중앙) 벡터를 갖도록 학습한다. 단어 위치를 보지 않는 Bag-of-Words 형태로, 학습 속도가 빠른 반면 일반적으로 Skip-gram에 비해 성능은 떨어지는 것으로 알려져 있다.[2] Sent2Vec의 기본적인 학습 방식은 Word2Vec과 동일하며 문장의 네거티브 샘플링을 통해 컨텍스트 전체의 loss를 최소화 하는 형태로 학습한다.[3]

$$\min_{\boldsymbol{U},\boldsymbol{V}}\sum_{S\in\mathcal{C}}\sum_{w_t\in S}\left(\ell\left(\boldsymbol{u}_{w_t}^{\top}\boldsymbol{v}_{S\setminus\{w_t\}}\right)+\sum_{w'\in N_{w_t}}\ell\left(-\boldsymbol{u}_{w'}^{\top}\boldsymbol{v}_{S\setminus\{w_t\}}\right)\right) \tag{1}$$

S는 주어진 문장이며, N_{wt}는 단어 $w_t \in S$에 네거티브 샘플링된 단어 집합이다. 네거티브는 각 단어 w가 확률 $q_n(w) := \frac{\sqrt{f_w}}{(\sum_{w_i \in v} \sqrt{f_{w_i}})}$과 연관된 다항 분포에 따라 샘플링된다. 여기서 f_w는 코퍼스내 w의 출현 빈도를 정규화한 값이다.

3.1.1. 네거티브 샘플링. 소프트맥스 함수를 사용한 기본적인 CBOW의 학습 수식은 다음과 같다.[2]

$$p(w_O|w_I) = \frac{({v'_{w_O}}^\top v_{w_I})}{\sum_{w=1}^{W}({v'_w}^\top v_{w_I})}$$

여기서 v_w와 v'_w은 각각 w의 입력과 출력 벡터다. W는 전체 단어의 수로, 따라서 이 방식은 연산 비용이 매우 높으며 비효율적이다. 이 문제를 해결하기 위해 본 논문에서는 수식 (1)과 같은 네거티브 샘플링을 이용한다.[2]

네거티브 샘플의 수는 데이터셋이 작을 경우 5-20, 대형 코퍼스에서는 2-5 정도의 값을 사용한다.[2] 코퍼스가 클 수록 이 값은 더 낮출 수 있으며, 네거티브 샘플과 수치 확률이 모두 필요한 NCE(Noise Contrastive Estimation)[8]와 달리 네거티브 샘플링은 샘플만을 필요로 한다.

3.1.2. 서브샘플링. 매우 빈번하게 등장하는 단어의 경우 학습과정 중에 지나치게 많은 영향을 끼쳐 추론시 강한 편향을 가져올 수 있다.[3] 따라서 빈도 수가 높은 단어는 확률적으로 제외하여 불용어 제거와 유사한 효과를 갖도록 하며 또한 학습 속도를 높여준다. 초기 Word2Vec 논문[2]에는 $P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$를 사용하며 이후 fastText에 이르러 현재 구현은 $P(w_i) = \sqrt{\frac{t}{f(w_i)}} + \frac{t}{f(w_i)}$를 사용하며 이에 따라 $1 - P(w_i)$의 확률로 학습에서 배제된다.

여기서 $f(w_i)$는 단어 w_i의 출현 빈도, t는 임계값이며 Word2Vec 논문에서는 10^{-5}을 권장하며 fastText 구현 또한 동일한 10^{-5}를 디폴트로 사용한다.[9]

3.1.3. 다이나믹 컨텍스트 윈도우. Word2Vec은 컨텍스트 윈도우의 사이즈가 학습 도중 다이나믹하게 변한다. 사용자가 설정한 최대 윈도우 값을 기준으로 균등하게 샘플링된 윈도우 사이즈로 랜덤하게 정해지는데, 이는 타겟 w에서 윈도우 크기로 나눈 거리만큼 가중치를 부여하는 효과를 갖는다.[10]

3.1.4. 저빈도 단어 삭제. 출현 빈도가 낮은 단어는 학습에서 배제한다. 학습 배제는 성능에 영향을 끼치지 않으면서도[10] 모델 압축에 탁월한 효과를 보인다. 실험 결과 100만 단어의 코퍼스에서 저빈도 설정을 통해 학습에 참여하는 단어의 수를 20만으로 줄였고, 그 결과 모델 크기를 5.4G에서 1.2G로 줄이면서도 동일한 성능을 유지할 수 있었다.

모델의 크기가 줄면 로딩 시간이 줄어들며 추론 또한 보다 짧은 시간에 가능하다. 뿐만 아니라 빠른 추론을 위해 모델을 모두 메모리에 올려야 하는 본 구현의 특성상 모델 크기는 매우 중요하다. 본 논문 구현에 사용한 주 언어인 파이썬은 시스템 리소스를 최대한으로 활용하기 위해 일반적으로 멀티 프로세스로 구동한다. 이 경우 각각의 프로세스가 각각의 메모리를 별도로 점유하는 형태로 구동되는데, 모델 크기에 프로세스의 수 만큼의 메모리가 필요하게 되며 따라서 모델 크기를 줄여나가는 일은 무척 중요하다.

3.1.5. 문자 n-그램. fastText의 CBOW 구현에는 중요한 개선점이 한 가지 있다. 문자 n-그램의 지원 여부로 fastText는 문자 단위로 서브워드를 추출하여 학습에 참여하는 기능이 있다. 동사의 변형을 매우 잘 유추할 수 있으나[11] 모든 서브워드가 학습에 참여하기 때문에 컨텍스트 윈도우 대비 수 배 가까이 학습해야할 단어의 수가 늘어나기도 한다. 영어에서는 다양한 변형을 학습할 수 있는 장점이 있으나 한글의 경우 조합 가능한 문자의 수가 많기 때문에 이 기법이 유용한 결과를 보이긴 어렵다.

3.2. Sent2Vec

Sent2Vec은 문장 임베딩을 위한 특성을 지니기 위해 CBOW 모델에 비해 다음과 같은 차이점이 있다.

3.2.1. 서브샘플링 비활성화. Sent2Vec은 서브샘플링을 사용하지 않는다. 서브샘플링은 n-그램 생성을 가로막고 문장에서 중요한 부분을 앗아갈 수 있다. 또한 서브샘플링된 단어만 남게되면 단어간 거리를 단축시켜 컨텍스트 윈도우의 크기를 암묵적으로 증가시키는 부작용을 낳는다.[3]

3.2.2. 다이나믹 컨텍스트 윈도우 비활성화. Sent2Vec은 문장 전체의 의미를 살리기 위해 문장의 모든 n-그램을 조합하여 학습 하기 때문에 다이나믹 컨텍스트 윈도우는 사용하지 않는다. Sent2Vec의 컨텍스트 윈도우 크기는 문장의 전체 길이로 고정한다.

3.2.3. 단어 n-그램. fastText CBOW 구현의 중요한 개선점 중 하나인 문자 n-그램은, 한글의 기하급수적으로 늘어나는 조합 가능한 문자 수의 특성상 본 구현에서는 적용하지 않는다. Sent2Vec에서는 단어 단위의 n-그램을 적용하는데, 여기서는 일반적으로 전산 언어학에서 얘기하는 단어를 구성하는 n-그램의 의미가 아니라 바이그램의 최대 거리를 뜻한다. 즉, 문장이 (A,B,C,D,E)로 구성되어 있을때,

- 단어 n-그램=3: (A), (A,B), (A,C)
- 단어 n-그램=4: (A), (A,B), (A,C), (A,D)

조합을 컨텍스트로 하여 학습한다. n-그램 확장은 뒤로 하되 이전 단어는 포함하지 않는다. 아울러 드롭아웃을 설정하여 랜덤하게 확장에서 배제하여 과적합을 방지한다.

Sent2Vec 구현에서는 각각의 n-그램에 대한 단어 ID 조합의 해시에 버킷 사이즈를 모듈라 연산 한 결과를 키 값으로 한 새로운 단어 벡터를 생성한다.

새로운 단어 벡터는 마찬가지로 타겟(중앙) 벡터를 갖도록 학습을 진행한다.

Table 1: 단어 n-그램=3, 단어 ID 조합과 새로운 단어 ID

타입	단어	단어 ID
단어	카드결제	14
	알림	17
	서비스	26
	계좌변경은	1538
	어떻게	2
	하나요	6
	단어 ID 조합	**새로운 단어 ID**
단어 n-그램=3	14 17	692956
	14 26	1780052
	17 26	841078
	17 1538	711288
	26 1538	1285391
	26 2	888413
	1538 2	1747
	1538 6	100493
	2 6	758302

3.3. 한글 특성

본 구현에서는 한글의 특성을 고려한 몇 가지 기법을 추가로 적용한다.

3.3.1. 주어부 가중치. 우리 말은 술어부 보다 주어부에서 중요한 단어가 등장한다는 가정하에 일정 길이 이상의 단어를 지닌 문장에서 절반 지점 상위를 주어부, 하위를 술어부로 정하고 주어부 단어 벡터에 α 배율로 가중치를 적용한다. 여기서 α는 1.2-1.5 사이일때 가장 좋은 결과를 보였다.

3.3.2. 가중치 감소. 주어부에 가중치를 부여했다면 코퍼스 출현 빈도에 따라 빈도가 높은 단어에 대해서는 가중치를 감소시킨다. 이는 불용어 제거와 유사한 효과를 갖는다.[10]

$$w = \frac{(t+1)\cdot k}{t+k}$$

여기서 k는 코퍼스 내 단어의 출현 빈도 순 순위이며, 가장 빈도가 높은 단어는 1이 된다. t는 임계값으로 여기서는 1000을 설정하여 각 단어 벡터에 대한 $\frac{w}{t}$ 배율로 가중치를 적용한다.

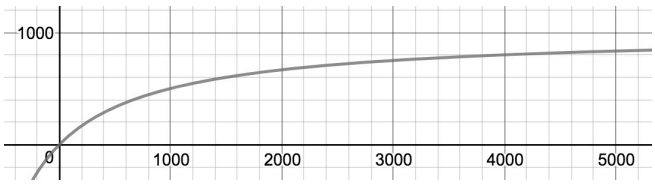

Figure 1: $t = 1000$, 단어 출현 빈도 순위 k에 대한 가중치 w

4. 실험 및 평가

4.1. 연동

Sent2Vec의 구현은 fastText를 fork 한 형태로 구현되어 있고, C++로 작성된 라이브러리를 파이썬과 연동하여 서비스로 구현한다. 현재 fastText의 파이썬 바인딩은 총 4가지다. 애당초 SWIG로 각 언어별 바인딩을 지원한 텐서플로와 달리 fastText의 바인딩은 뒤늦게 추가되었고, 공식 바인딩이 지원되기 이전에 3rd-Party 바인딩이 혼재하여 다양한 구현이 존재한다.

4.1.1. 파이썬 바인딩. fastText.py[1]는 가장 먼저 출시된 파이썬 바인딩으로 인터페이스를 C++로 따로 작성한 뒤 Cython C로 연동하여 제공한다. 초기 pypi에서 제공하는 버전이기도 하며, 현재는 페이스북에서 직접 제작한 버전으로 대체되어 관리되고 있다. fastText 공식 바인딩[2]은 페이스북에서 공식적으로 제공하는 버전이며, 앞서 언급한대로 현재 pypi에서 관리되는 버전으로 pybind11로 구현되어 있다. gensim은 자사 라이브러리에 fastText를 통채로 탑재[3]하여 제공하고 있으며, fastText C++ 구현을 파이썬으로 변환하여 탑재했고 이를 다시 Cython C로 최적화하여 제품화 가능한 수준(Production-Ready)으로 제공하고 있다. pyfasttext[4]는 또 다른(yet another) 구현으로 fastText.py와 유사하다. 인터페이스를 C++로 구현하고 이를 Cython C로 연동했다.

4.1.2. 인터페이스 구현. 본 논문 구현에서는 공식 바인딩을 제외한 대부분의 바인딩 구현 방식이기도 한 Cython으로 인터페이스를 작성하여 fastText와 연동했다.

- `load_model()`: 최초 실행시 모델을 메모리에 적재할때 사용한다.
- `embed_sentence()`: 단일 문장을 처리할때 사용한다.
- `embed_sentences()`: 다수의 문장을 한꺼번에 처리할때 사용한다. 둘 중 택일하여 파이썬에서 자유롭게 사용할 수 있으며, 다수의 문장을 처리할때는 쓰레드 갯수를 설정할 수 있는 `num_threads` 옵션을 두어 기존 파이썬으로는 처리 하기 힘든, 멀티 쓰레드로 문장 임베딩을 계산하도록 C++로 구현 하여 고성능으로 계산할 수 있도록 했다.

4.2. 학습 데이터

학습 데이터는 2013년 부터 2015년 까지의 모든 언론사의 뉴스를 형태소 분석한 결과로 4.5억개의 문장, 85G 크기의 파일로 구성되어 있다. Sent2Vec은 비지도 학습 모델이므로 문장에 대한 별도의 레이블링은 진행하지 않는다.

4.3. 실험 방법

4.3.1. 코사인 유사도. 문장의 유사도를 판별하는 방식은 유입 문장에 대한 임베딩 계산 후 비교 대상이 되는 모든 문장의 임베딩 벡터에 대해 코사인 유사도[12]를 계산하여 가장 유사한 벡터에 해당하는 문장을 정답으로 간주하고 거리순으로 결과를 제시한다.

1. https://github.com/salestock/fastText.py
2. https://github.com/facebookresearch/fastText
3. https://radimrehurek.com/gensim/models/fasttext.html
4. https://github.com/vrasneur/pyfasttext

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

여기서 A_i와 B_i는 각각 벡터 A와 B의 구성요소다.

4.3.2. 평가. 모델의 성능을 정량적 평가로 진행할 수 있도록 5,000여개의 문장으로 구성된 테스트셋을 구축하고 이를 통해 모델의 성능을 기계적으로 측정할 수 있게 했다. 사람이 선택한 정답 문장이 각각 1위, 3위, 5위 내에 존재하는지 여부로 Precision at k(P@k)를 정하고 이 점수로 모델 성능을 평가했다.

Table 2: 각 모델에 따른 성능 평가 결과

모델	P@1	P@3	P@5
Word2Vec CBOW	0.7574	0.8618	0.8927
Sent2Vec uni. (baseline)	0.7854	0.8997	0.9271
Sent2Vec bi.	0.8306	0.9296	0.9539
Sent2Vec bi. + 주어부 가중치 1.2	0.8369	0.9312	0.9544
Sent2Vec bi. + 주어부 가중치 1.2 + 가중치 감소*	**0.8499**	**0.9374**	**0.9597**

5. 결론

본 논문은 Sent2Vec 모델을 통해 문장의 유사도를 판별하는 내용을 담고 있다. 이 방식은 비지도 학습 으로 대용량 코퍼스에 대해 별도의 레이블링 없이 학습 모델을 구축할 수 있으며, 한글 특성에 맞게 모델을 개선하여 성능을 향상시키는 방법을 살펴 보았다. RNN 등의 깊은 신경망에 비해 얕은 행렬 연산만으로[3] 훨씬 빠른 시간내에 추론을 수행할 수 있으며, C++로 개발된 fastText 구현을 통해 멀티 쓰레드로 고성능 추론을 수행할 수 있는 제품화 가능한 수준임을 살펴 보았다.

향후 연구로는 문장 벡터 생성 시 한국어의 의미가 보다 잘 임베딩 될 수 있도록 품사 태그 정보를 포함한 다양한 학습 데이터를 추가하여 실험해 볼 것이며, GloVe를 비롯한 각기 다른 임베딩 기법을 혼합하여 사용하는 앙상블도 시도해 볼 것이다. 다양한 방식의 임베딩을 통해 압축 표현된 정보의 의미를 보다 효율적으로 재구성한다면 유사 문장 판별의 성능을 더욱 높일 수 있을 것으로 기대한다.

참고 문헌

[1] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in Proceedings of the thirtieth aaai conference on artificial intelligence, 2016, pp. 2786–2792.

[2] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," CoRR, vol. abs/1310.4546, 2013.

[3] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised learning of sentence embeddings using compositional n-gram features," CoRR, vol. abs/1703.02507, 2017.

[4] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," CoRR, vol. abs/1405.4053, 2014.

[5] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, "From word embeddings to document distances," in Proceedings of the 32Nd international conference on international conference on machine learning - volume 37, 2015, pp. 957–966.

[6] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, vol. 41, no. 6, pp. 391–407, 1990.

[7] Michael Hardy et al., "Latent semantic indexing — Wikipedia, the free encyclopedia." 2018.

[8] C. Dyer, "Notes on noise contrastive estimation and negative sampling," CoRR, vol. abs/1410.8251, 2014.

[9] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," Transactions of the Association for Computational Linguistics, vol. 5, pp. 135–146, 2017.

[10] O. Levy, Y. Goldberg, and I. Dagan, "Improving distributional similarity with lessons learned from word embeddings," Transactions of the Association for Computational Linguistics, vol. 3, pp. 211–225, 2015.

[11] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in Proceedings of the 15th conference of the european chapter of the association for computational linguistics: Volume 2, short papers, 2017, pp. 427–431.

[12] Danko Georgiev et al., "Cosine similarity — Wikipedia, the free encyclopedia." 2018.