자연어: 사람들이 일상언어로 쓰는 언어

인공어: 사람이 이해할 수 있지만, 엄격한 문법과 모호성이 없는 형태의 전달방식(프로그래밍 언어 – 우리가 배워야 한다.)

# NLP는!

- Discrete Value를 다룬다(이산적은 값을 다룬다). – 단어, 문장

- 분류 문제로 접근할 수 있다. (Text Classification)

- 샘플의 확률 값을 구할 수 있다. (좀더 자연스러운(확률이 높은) 단어가 뭐니?)

- 문장생성 (자연어 생성): auto-regressive 속성을 가진다. (이전에 생성한 단어에 따라 다음 단어가 예측된다. One – to – many 방식)

- GAN적용 불가!

- 처음 잘못 예측하면 계속 잘못하게 된다.

- NLP는 데이터를 구하기 쉬운 반면, 전처리가 매우 어렵다. 전처리는 Task에 따라 다르다. 따라서 경험이 매우 중요하다!

- Domain Knowledge로 언어적 지식이 필요하다! 한국어는 어떠한 언어적 특성을 가지는지?

- NLP뿐만 아니라 모든 딥러닝은 end-to-end방식을 지양한다.

텍스트, 도표, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

# 한국어의 NLP는 왜 어려운가?

**Ambiguity** 애매성, 모호성: 모호성을 극대화하는 문장들! 여러가지 기계번역기가 있지만 번역하는게 다르다.

->주변 단어를 보고 해당 단어의 중의성을 해소할 수 있다(RNN).

1. 동음 이의어에 따른 어려움이 존재한다. -> 단어의 중의성

ex) 차를 마시러 공원에 가던 차 안에서 나는 그녀에게 차였다.

1. 생략으로 인해, 문장 내 정보의 부족으로 인한 모호성이 증가한다.

ex) 나는 철수를 안 때렸다.

해석이 여러가지로 될 수 있다. 철수는 맞았지만 때린사람이 난 아니다, 나는 누군가를 때렸지만 그게 철수는 아니다, 나는 누군가를 때린적도 없고, 철수도 맞은적이 없다.

1. 문장 내 정보부족으로 야기한 구조 해석의 문제

ex) 선생님은 울면서 돌아오는 우리를 위로했다.

선생님이 우는건지? 돌아오는 우리가 우는건지?

언어는 마치 생명체와 같이 진화하며, 특히 효율성을 극대화하는 방향으로 진화한다.

최대한 짧은 문장 내에서 많은 정보를 담고자 한다. 정보량이 낮은 내용(context)이다. 따라서, 모호함(ambiguity)가 발생한다. 사람은 생략된 context를 효율적으로 채울 수 있지만, 기계는 이런 task에 힘들다.

**Paraphrase** 한글의 경우에는 같은정보로 굉장히 다양한 문장 표현이 가능하다. 문장의 표현 형식은 다양하고 비슷한 의미의 단어들이 존재하기 때문에 컴퓨터 입장에선 다양한 표현을 배워야한다. 이것은 컴퓨터 입장에서 매우 어렵다.

One-hot 인코딩으로 표현된 값은 유사도나 모호성을 표현할 수 없다. 서로 다른 one-hot 벡터끼리의 유사도나 거리는 모두 동일하다. 수많은 벡터중 1이 하나기 때문에 sparse하게 된다. 이것은 유사도를 분류하는데 불리하다.

* 딥러닝에선 word embedding을 통해 Sparse한 vector를 Dense하게 표현하게 됨으로써, 유사도, 모호성 문제를 해결한다.

**Discrete, not Continuous** 우리는 단어들을 Continuous하게 생각하는데, 실제로는 Discrete한 개념으로 존재하다 보니, 단어 사이의 관계들을 컴퓨터는 잘 반영하지 못한다.

**Agglutinative language(교착어)** 교착어에는 여러가지 특징이 있는데, NLP에 적합하지 않다.

1. 접사 추가에 따른 의미 파생: 어간+ 접사구조 이므로, 의미가 많아지게 되므로, 문법적으로 수가 너무 많아진다. 즉, 조합이 너무 많아진다. -> 한 단어를 여러가지로 표현하게 된다.

ex) 잡다는 잡히다/잡히셨다/잡았다/잡겠다/잡더라/잡혔다/잡혔겠다/잡았겠다 …. 등등 엄청 많다. 하지만 영어는 단지 catch 하나로 표현 가능하다.

만약 위에 단어를 다르게 취급하면 단어수가 엄청 많아진다. 그에 따라서 하나로 처리해야 한다. 따라서 형태소 분석기를 통해서 분절(segmentation)을 해준다. 어간을 추출해 하나의 형태로 뜻을 이해할 수 있게 된다.

2. 유연한 단어 규칙: 어순이 뒤바뀌어도 의미가 같다! 접사만 잘 붙어있으면 된다. 따라서 같은 뜻이지만, 많은 표현이 가능하게 컴퓨터가 이해하기 어렵다. 반면 영어나 고립어는 어순이 중요하다. -> 같은뜻을 더 많은 표현으로 나타낸다.

3. 모호한 띄어쓰기: 원래 동양권 언어에는 띄어쓰기가 존재하지 않았다. 서양권에서만 존재했다. 영어는 띄어쓰기가 명확하지만, 한글은 그렇지 않다. 띄어쓰기가 모호하기 때문에 인터넷에서 데이터 크롤링 할 경우 데이터가 중구 난방이다.

4. 평서문과 의문문 차이 부재: 평서문과 의문문의 차이가 없는 경우가 있다. 물음표나 억양 외에는 차이가 없을 수 있다.

ex)

|  |  |  |
| --- | --- | --- |
| 언어 | 평서문 | 의문문 |
| 영어 | I ate dinner, | Did you have dinner? |
| 한국어 | 점심 먹었어. | 점심 먹었니? |

심지어 주어 생략도 있다. 영어 입장에서는 동사와 목적어밖에 없다.

5. 주어부재: 한국어는 주어에 문장이 없더라도 대화가 가능하다. 영어에서 한국어로 번역을 할때는 주어가 없어도 되지만, 한국어에서 영어로 번역될때는 주억가 필수다. 예시는 4번 과 같다.

6. 한자 기반의 언어: 표의 문자인 한자는 표음 문자인 한글로 wrapping했다. Wrapping 과정에서 정보의 손실이 발생한다.

ex) 車타는 차, 茶마시는 차: 한글에서는 같은 발음이지만, 한자에선 다른 표기법.

7. 단어 중의성으로 인한 문제 발생: 마시는 ‘차’, 타는 ‘차’ 단어의 중의성으로 인해 모호성이 발생한다. 이것을 문맥으로 파악해야한다.

위와 같은 이유로 한자는 매우 어렵다. 효율이 극대화되었기 때문이다. 한국어는 코퍼스, 전처리, 텍스트 분류등 자연어 생성에서는 한국어의 특성이 훨씬 부각된다.

+ 전처리가 어렵다

텍스트, 스크린샷, 폰트, 일렉트릭 블루이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

- 데이터 코퍼스 수집에 있어서 주의해야 할점들은 성격이 모두 다르다.

- 정제에서는 Task에 맞는 각종 노이즈들을 제거해야하고, 인코딩 변환을 한다. 보통 UFT-8을 사용한다.

- Labeling에서는 데이터에 맞는 레이블을 부여한다. 레이블링이 되어있는경우 생략하기도 한다.

- Tokenization은 교착어에 속하기 때문에 형태소 분석기를 통해서 접사를 분리한다. 한국어 형태소 분석기로는 mecab, kkoma(kkma), Hannanum, Komoran, Okt가 있다.

- Subword segmentation(optinal) 추가적인 분절을 BPE알고리즘을 통해 수행한다. 딥러닝에서 생긴 기법으로, vocabulary숫자를 compact하게 가져간다. 또하나의 모델을 갖는 과정으로, OoV(Out of Vocabulary)문제를 해결하는데 도움을 준다.

- Batchify는 mini batch를 만드는 과정이다. 이것은 torch text로 해결할 수 있다. 사전순으로 index mapping을 해서, integer값으로 변환해준다.

말뭉치란? 자연어처리를 위한 문장들로 구성된 데이터 셋이며, 복수표현은 Corpora이다.

다음은 NLP의 Service Pipe Line이다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

Detokenization은 원래 사람이 띄어쓰기 한 것처럼 복원해주는 것이다.

**In Zoom Data Crawling: 데이터 수집**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

우리는 모두 같은 최신 알고리즘을 이용할것이다. 그렇기 떄문에 중요한 것은 데이터의 양과 품질이다.

데이터는 구입 또는 외주를 맞길수 있지만, 양이 제한적이라는 한계가 있다.

Craling을 통해 무한한 양의 콧퍼스를 수집 가능하지만, 품질이 천차만별이며, 정제 과정에 많은 노력이 필요하다.

무료공개 사이트도 많지만, 게다가 한국어 코퍼스는 흔하지 않다. 한국어 무료 코퍼스로는 AI-hurb이 있다.

크롤링에 있어서 중요한건 아직 외색지대기 때문에 저작권이 매우 중요하다. 이것에 유의하여 데이터를 크롤링하자.

**In Zoom Data Cleaning: 데이터 정제**

우리는 기본적으로 기계적인 노이즈 제거가 필요하다. 무조건적으로, 전각문자를 반각문자로 바꿔줘야 한다.

Interactive한 노이즈 제거도 필요한데, 코퍼스의 특성에 따라서, 작업자가 상황을 봐가며 task에 알맞게 수행해야 한다. 규칙에 의해 노이즈를 제거하기 때문에, 노이즈 전부를 제거하는 것은 어렵다. 따라서 반복적인 규칙 생성 및 적용 과정이 필요하다. 노력과 품질 사이의 trade off다.

각 언어, 도메인, 코퍼스 별 특성이 다르므로, Task에 따라서 전처리(preprocessing) 전략이 다르기 때문에 신중한 접근이 필요하다.

우리는 정규식(regular expression)을 활용해서 정제할 수 있으며, 코딩없이 단순히 텍스트 에디터(VSCode)로도 제거할 수 있다.

전용 모듈을 통해 제거하는 방법은 python등을 활용해 모듈을 만들고, regex 리스트를 파일로받아서 처리한다. regex 재활용 가능하고 한 번에 모든 regex를 적용한다 하지만 중간 결과가 확인 불가능하다.

그다음 우리는 text editor를 활용할 수 있다. 파일을 열어 적용 과정을 보면서 바로바로 결과를확인할 수 있지만, log가 남지 않아서 재활용이 불가능하다.

따라서 우리는 필수적으로 해야하는 것을 정규식을 통해 제거하고 task에 따라서 text editor를 통

해 제거하는 것이 좋다.

**\* 정규식**

|  |  |  |  |
| --- | --- | --- | --- |
| [] |  | {n, m} |  |
| [-] |  | . |  |
| [^] |  | ^$ |  |
| () |  | \t |  |
| | |  | \s |  |
| ? |  | \S |  |
| + |  | \w |  |
| \* |  | \W |  |
| {n} |  | \d |  |
| {n,} |  | \D |  |

꿀팁! ([a-z]bc[a-z]) -> \1\2: 가운데 bc를 중심으로 양쪽이 a-z으로 되어있으면 bc제거

abcd -> ad, 0bc1 -> 0bc1

Ex) ^(positive|negative)\t[,.0-9a-zA-Z\-=;]+$

^문장열 시작, positive 또는 negative / \t은 tab / ,.숫자 소문자영어 대문자영어 \ - = | ; 값 제거 / + 앞의 패턴 하나이상 $ 문자열 끝(+가 없으면 문자가 딱 한번만 등장 한다는 뜻)

**In Zoom Data Labeling: 데이터 레이블링**

Label

|  |  |  |
| --- | --- | --- |
| Task | Input | Output |
| Text Classification | Sentence | Class |
| Token Classification | Sentence | Tag for each token(sentence) |
| Sequence-to-Sequnce | Sentence | Sentence |

Label Example

* Sentence -> Class

TSV 형태의 하나의 파일.

TSV형태의 하나의 파일이며, 각 ROW가 문장과 대응되는 레이블이다. 문장 column과 레이블 column으로 구성되어 있다.

* Sentence -> Sentence (Sequence)

- TSV형태의 하나의 파일인경우, 각 ROW가 대응되는 문장쌍이며, 각 문장 별로 column을 구성한다.

- 두개 이상의 파일로 구성된 경우, 같은 순서의 row가 대응되는 문장 쌍이며, 한 문장당 여러 레이블이 존재할 수 있다.

레이블링은 직접 진행하는게 좋다. Prototyping시 굉장히 강력한 도구다. 성능이 너무 뛰어나다.

효과적인 레이블링 도구인 엑셀을 사용하자

\* 파일형식

|  |  |  |
| --- | --- | --- |
| TSV (Tab-Separated Values) | 데이터를 탭(tab)으로 구분하여 저장하는 형식 | column1 column2  value1 value2 |
| TXT (Plain Text): | TXT 파일은 어떤 특별한 형식이나 구분자를 사용하지 않고 일반적인 텍스트 데이터를 포함한다 |  |
| CSV (Comma-Separated Values) | 데이터를 쉼표(,)로 구분하여 저장하는 형식입니다. | column1, column2, column3  value1, value2, value3 |

따라서 우리는 tab으로 나뉘어져 있는 tsv 파일형식을 사용해야 한다. 콤마(,)가 있으면 label을 구별하기 위한 콤마(,)인지 문장내에 있는 콤마(,)인지 구별하기가 어렵다.

**In Zoom Tokenization: 분할**

보통 훈련할 때 우리가 원하는 입력 데이터는 1sentence/line이다. 그러나, 우리가 수집한 corpus는 한 라인에 여러 문장이 들어있거나, 한 문장이 여러 라인에 들어있다. 우리는 Sentence Segmentation을 통해서 원하는 형태로 변환할 수 있다. nltk, konlpy, mecab을 통해서 할수있다. Nltk는 한국어에서도 애매하지만 어느정도 되는편이다. 그냥 mecab을 쓰자

Tokenization 역할

두개 이상의 다른 token들의 결합으로 일어진 단어를 쪼개어, vocabulary숫자를 줄이고, 희소성(sparseness)를 낮추기 위함이다.

한국어에서 더 중요한데, 교착어의 특성 때문이다. 어근에 접사가 붙어 다양한 단어가 파생되고, 띄어쓰기의 통일성이 필요하기 때문이다. **이것은 어근에 접사를 분절해주는 역할을 하며, 희소성에 대한 문제를 해결해준다. 또한 정착되지 않은 띄어쓰기를 해결하는데 도와준다.**

형태소 분석 및 품사태깅 (Part of Speech Tagging, POS)

형태소 분석: 형태소를 비롯하여, 어근, 접두사/접미사, 품사(POS)등 다양한 언어적 속성의 구조를 파악하는 것.

품사태깅: 형태소의 뜻과 문맥을 고려하여 그것을 마크업(지정 표시) 하는 일

\* 토큰 평균 길이에 따른 성격과 특징

- 짧을수록(더 잘게 분절): Vocabulary크기가 감소하여 희소성 문제가 감소한다. OoV(Out of Vocabulary)가 줄어들고, Sequence가 길어짐(time step의 길이가 길어짐)으로써 모델의 부담이 증가한다. 극단적인 예시로 Character 단위의 분절이 있다.

- 길수록: Vocabulary크기가 증가하여 희소성 문제가 증대한다. Sequence의 길이가 짧아짐(time step의 수가 줄어듬)으로써 모델의 부담이 감소한다.

\* OoV(Out of Vocabulary)가 중요한 이유

우리는 학습할 때 단어를 INDEX로 MAPPING을 한다. 즉, 어휘사전을 만들고, index값으로 순서대로 mapping한다. test단계에서 train때 보지못한 OoV가 발생하면 어느 index로 mapping을 해야할지 알 수 없다. 그러면 UNK(unknown)으로 처리한다. 이 unknown은 자연어 생성에 있어서 치명적이다.

\*정보량에 따른 이상적인 형태

빈도가 높을수록 하나의 token으로 나타내고, 빈도가 낮을 경우 더 잘게 쪼개어, 각각 빈도가 높은 token으로 구성한다.

(기존의 띄어쓰기가 \_로 post tokenization을 통해 대체한다.)

**In Zoom Subword Segmentation: 세부분할**

단어보다 더 작은 의미 단위를 sub-word라고 표현한다.

예를들면 conference를 con(=together) + contr(=center) + ate(=make)로 쪼갠다. 이렇게 쪼개기 위해서 필요한게 subword다.

* Byte Pair Encoding(BPE) 알고리즘

- 압축 알고리즘을 통하여 subword segmentation을 수행한다.

- 학습 코퍼스를 활용하여 BPE모델을 학습 후, 학습 테스트 코퍼스에 적용한다.

- 희소성을 통계에 기반하여 효과적으로 낮춘다.

- 언어별 특성에 대한 정보 없이, 더 작은 의미 단위로 분절함으로써, OoV를 없앨 수 있다.

- 학습 데이터 별로 BPE모델도 생성된다. 따라서 Deep Neural Network모델 파라미터를 그대로 가지고 있어야 한다는 단점이 있다.

영어는 바로 BPE를 해도 되지만, 한글은 띄어쓰기가 애매하기 때문에 위험하다. 따라서 형태소 분석을 한 이후에 BPE알고리즘을 적용하는 것을 권장한다.

**In Zoom Detokenization**

Tokenization을 수행하면, 기존 띄어쓰기와 구분을 위해 \_가 생긴다. (기존의 띄어쓰기가 \_로 post tokenization을 통해 대체한다.)

Subword segmentation을 수행하면 또 공백 구분을 위한 \_ 이 삽입된다. (Tokenization으로 생긴 띄어쓰기가 \_ 로 대체된다.)

* 따라서 \_가 두개가 생긴다!

그렇다면 subword segmentation의 빈공간(space bar)’, ‘tokenization을 대체한 \_’, ‘기존의 띄어쓰기를 대체한\_’ 총 3개를 제거해야한다.

실수하지마라. 훈련 돌리고 detokenization을 해라.

결국에는 오픈소스 문화의 확신으로 데이터가 큰 자산이다.

# TorchText

우린 모델에 넣기 전에 마지막으로 해야할 일이 있다!

텍스트, 스크린샷, 폰트, 로고이(가) 표시된 사진

자동 생성된 설명

데이터를 받아서 스플릿하고,

* Read Text & Build Deictionaly

- text를 불러오고 사전을 만든다. 빈도순으로 사전을 정렬하되, 불용어는 제외하고 만들어야 한다.

-필요에 따라 min\_count보다 작은 빈도를 갖는 어휘를 제외하고, (최소 몇번 이상)

- max\_vocab에 따라 빈도순으로 어휘를 제외하기도 한다. (불용어)

- 필요에 따라 특수 토큰도 어휘 사전에 포함한다. (<EOS>,<BOS>,<UNK>,<PAD>)

ex) 불용어(a, the…, 은, 는, 이, 가, 을, 를)

* Chunking and Padding

- 미니배치 형태는 (batch\_size, length, |V|)

- One-hot 벡터를 모두 저장할 필요가 없다. (batch\_size, length, 1) = (batch\_size, length) – 을 가진 index 값만 알면 된다

- Sequnce차원의 크기는 미니배치 내의 가장 긴 문장에 의해 결정된다. 각 샘플별 모자라는 부분은 padding으로 대체된다. 따라서 <PAD> 토큰이 필요하다. Pytorch의 Packed Sequnce를 활용할 경우 PAD생략 가능하다. (자연어 생성에서)

문제점은? 첫번째 문장은 100단어, 두번째 문장은 2단어인경우 98 time step은 버려진다. -> sort를 해야하지 않을까?

* Increase Training Efficiency

- Sort by sequence length

- get chunk with similar length of sequences

미니배치를 랜덤하게 구성하지 않고, sort하여 길이에 따라서 구성한다. 짧은 것은 짧은 것끼리, 긴 것은 긴 것끼리 길이에 따라 구성하고 chunking한 후에 mini batch를 만들고 shffling한다.

이것들을 torchtext가 대신 해줄것이다. Torchtext는 dataloader에서 사용한다.

**데이터 로딩, 전처리, 토큰화, 단어집합 생성, 배치처리, 사정 정의된 데이터셋 및 임베딩**

주피어테서 할수 있는 것은 데이터로더, 미니배치처리, vocab 단어사진 만들기, 빈도수 확인, 텐서로부터 단어 복구