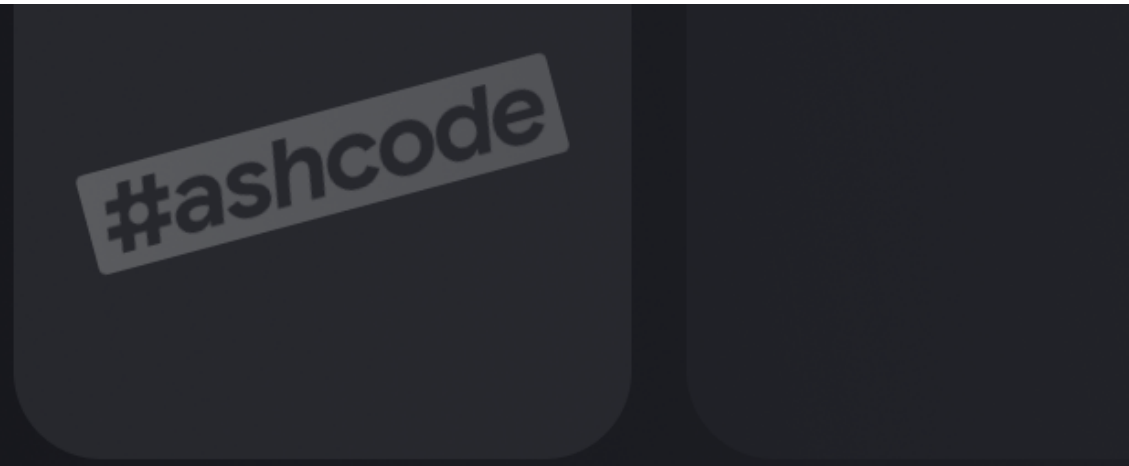
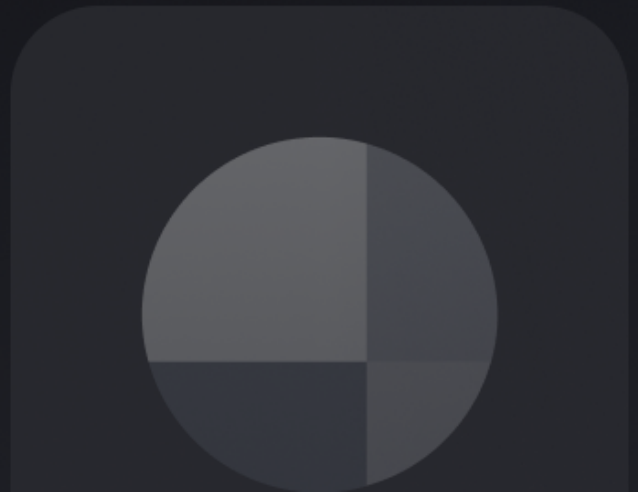
텍스트 전처리와 언어모델

Grepp
Zayden





`딥러닝을 이용한 자연어 처리 입문` 도서를 바탕으로 진행 (https://wikidocs.net/book/2155)

목차

1장. 자연어 처리 준비하기

자연어 처리란?

자연어는 우리가 일상에서 사용하는 언어를 말한다. 자연어 처리는 자연어의 의미를 분석하여 컴퓨터가 처리하도록 하는 일.

필요한 프레임워크와 라이브러리

Tensorflow & Keras

Tensorflow를 백엔드로 동작시키는 Keras를 사용할 수 있음.

> 독립적으로 설치된 `keras`와 tensorflow에 내장된 `tf.keras`를 이용할 수 있음 → 케라스 개발자 François Chollet가 `tf.keras`를 사용하기를 권장하고 있음

Gensim

머신러닝을 사용하여 토픽 모델링과 자연어 처리 등을 수행하는 오픈소스 라이브러리.

NLTK

자연어 처리를 위한 파이썬 패키지. NLTK를 잘 이용하기 위해서는 NLTK Data라는 여러 데이터를 추가 다운로드 해주는것이 좋음. (링크에서 수동으로 다운로드 가능)

```
import nltk

nltk.download()
```

KoNLPy

한국어 자연어 처리를 위한 형태소 분석기 패키지.

기타

- Scikit-learn
- jupyter notebook (lab)
- pandas
- numpy
- matplotlib

2장. 텍스트 전처리

말뭉치(코퍼스, Corpus)

자연어 상태로 존재하는 데이터. 말뭉치가 여러개 있는경우 copora라고 부른다. (참고)

```
영화가 참 재미있습니다, 1
스토리가 지루했어요, 0
```

토큰화 (Tokenization)

주어진 코퍼스에서 토큰이라 불리는 단위로 나누는 작업. 토큰의 단위는 상황에 따라 다르고 의미있는 단위로 토큰을 정의한다.

단어 토큰화 (word tokenization)

토큰의 기준을 단어로 하는 경우 단어 토큰화(word tokenization)라고 한다. 단어(word)의 단위는 단어 외에도 단어구, 의미를 가지는 문자열로 간주되기도 한다.

[구두점 제거 & 토큰화]

```
Time is an illusion. Lunchtime double so!
→ [“Time”, “is”, “an”, “illustion”, “Lunchtime”, “double”, “so”]
```

> 앞서 언급한 `NLTK`를 사용하면 영어 코퍼스를 토큰화할 수 있다. 토그나이저별로 구두점 제거 방식이나 토큰화 단위가 다 다르다.

표준 토큰화 (Penn Treebank Tokenization)

1. 하이푼으로 구성된 단어는 하나로 유지한다.
2. doesn’t와 같이 아포스트로피로 `접어` 가 함께하는 단어는 분리해준다.

문장 토큰화 (sentence tokenization)

토큰의 단위가 문장인 경우 사용하는 토큰화. 문장 분류 (sentence segmentation) 같은 곳에서 활용할 수 있다.

```
from nltk.tokenize import sent_tokenize

text = "I am actively looking for Ph.D. students. and you are a Ph.D student."
print('문장 토큰화:',sent_tokenize(text))
```

```
문장 토큰화: ['I am actively looking for Ph.D. students.', \
                                        'and you are a Ph. D student.']
```

> 한국어의 경우에는 박상길님이 개발한 KSS(Korean Sentence Splitter)를 추천한다고 한다.

한국어의 토큰화

한국어에서는 띄어쓰기 단위를 어절이라고 하며 한국어 자연어처리에서는 지양되고 있다.
한국어는 영어와 달리 교착어(조사, 어미 등을 붙여서 말을 만드는 언어)를 사용하기 때문에 고려해야 할 사항이 많다.

→ 형태소(morpheme, 뜻을 가진 가장 작은 말의 단위)단위의 토큰화가 필요하고 개념에 대한 이해가 있어야함

- 자립 형태소: 명사, 대명사, 수사, 관형사, 부사, 감탄사 등
- 의존 형태소: 접사, 어미, 조사, 어간

에디가 책을 읽었다 라는 문장이 주어질때,

띄어쓰기 단위 토큰화

→ [’에디가’, ‘책을’, ‘읽었다’]

형태소 단위 분해

→ 자립 형태소 [에디, 책], 의존 형태소 [-가, -을, 읽-, -었, -다]

한국어에서는 영어의 단어 토큰화와 유사한 결과를 얻으려면 형태소 단위의 토큰화를 진행해야 한다.

품사 태깅 (Part-of-speech tagging)

표기는 같지만 품사에 따라서 단어의 뜻이 달라지는 경우.

```
fly → 날다(동사), 파리 (명사)
못 → 할수 없는(부사), 목재등을 고정하는 물건 (명사)
```

토큰화 과정에서 각 단어가 어떠한 품사로 쓰였는지 태깅이 필요하다.

> NLTK와 KoNLPy를 통해서 영어, 한국어 토큰화, 품사 태깅이 가능하다.

NLTK

```
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag

text = "I am actively looking for Ph.D. students. and you are a Ph.D. student."
tokenized_sentence = word_tokenize(text)

print('단어 토큰화:', tokenized_sentence)
print('품사 태깅:', pos_tag(tokenized_sentence))
```

```
단어 토큰화: ['I', 'am', 'actively', 'looking', 'for', 'Ph.D.', 'students', '. ', 'and', 'you', 'are', 'a', 'Ph.D.', 'student', '.']
품사 태깅: [('I', 'PRP'), ('am', 'VBP'), ('actively', 'RB'), ('looking', 'VBG' ), ('for', 'IN'), ('Ph.D.', 'NNP'), ('students', 'NNS'), ('.', '.'), ('and','CC'), ('you', 'PRP'), ('are', 'VBP'), ('a', 'DT'), ('Ph.D.', 'NNP'), ('student', 'NN'), ('.', '.')]
```

KoNLPy

형태소 분석기로 Okt(Open Korea Text), 메캅 (Mecab), 코모란 (Komoran), 한나눔 (Hannanum), 꼬꼬마 (Kkma)를 가지고 있음.

```
from konlpy.tag import Okt
from konlpy.tag import Kkma

okt = Okt()
print('OKT 형태소 분석 :',okt.morphs("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
print('OKT 품사 태깅 :',okt.pos("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))

kkma = Kkma()
print('꼬꼬마 형태소 분석 :',kkma.morphs("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
print('꼬꼬마 품사 태깅 :',kkma.pos("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
```

```
OKT 형태소 분석 :['열심히','코딩','한','당신',',','연휴','에는','여행', '을', '가봐요']
OKT 품 사 태 깅 : [('열심히', 'Adverb'), ('코딩', 'Noun'), ('한', 'Josa'), ('당신', 'Noun'), (',', 'Punctuation'), ('연휴', 'Noun'), ('에는', 'Josa'), ('여행', 'Noun'), ('을', 'Josa'), ('가봐요', 'Verb')]
꼬꼬마 형태소 분석 :['열심히','코딩','하','ㄴ','당신',',','연휴','에', '는', '여행', '을', '가보', '아요']
꼬꼬마 품사 태깅 :[('열심히','MAG'),('코딩','NNG'),('하','XSV'),('ㄴ',' ETD'), ('당신', 'NP'), (',', 'SP'), ('연휴', 'NNG'), ('에', 'JKM'), ('는', ' JX'), ('여행', 'NNG'), ('을', 'JKO'), ('가보', 'VV'), ('아요', 'EFN')]
```

정제(Cleaning)와 정규화(Normalization)

정제

갖고 있는 코퍼스로부터 노이즈 데이터를 제거하는 일

정규화

표현 방법이 다른 단어들을 통합시켜서 같은 단어로 만드는 일

→ 코퍼스의 복잡성을 줄이는것이 목적

1. 규칙에 기반한 표기가 다른 단어 통합
    - 어간 추출(steamming), 표제어 추출(lemmatization)
2. 대, 소문자 통합
3. 불필요한 단어 제거
    - 등장빈도가 적거나 단어의 길이가 짧은 단어

> 정규 표현식을 활용하자!

표제어 추출 (Lemmatization)

표제어란 기본 사전형 단어 라는 의미. 표제어를 추출한다는것은 단어들이 다른 형태를 가지더라도 그 뿌리 단어를 찾아서 같은 의미인지 확인.

```
am, are, is → be
```

표제어를 추출하는 좋은 방법은 형태학적(형태소로부터 단어를 만드는것) 파싱을 진행하는것.

형태학적 파싱

어간과 접사 요소를 분리하는 작업

- 어간: 단어의 의미를 담고 있는 부분
- 접사: 단어에 추가적인 의미를 주는 부분

```
cats → cat(어간) + -s(접사)
```

어간 추출 (Stemming)

어간을 추출하는 작업. 명확하기보다는 알고리즘을 통해서 어간을 추출하는 작업이여서 추출후 단어가 사전에 존재하지 않는 단어일 수도 있음.

- 포터(Porter) 알고리즘

  ```
  from nltk.stem import PorterStemmer
  ```

- 랭커스터 스태머(Lancaster Stemmer) 알고리즘

  ```
  from nltk.stem import LancasterStemmer
  ```

표제어 추출과 어간 추출의 비교

	표제어 추출(Lemmatization)	어간 추출(Stemming)
am	be	am
the going	the going	the go
having	have	hav

한국어의 어간 추출

한국어는 아래와 같이 5언 9품사의 구조를 갖고있다.
용언에 해당하는 동사와 형용사는 어간과 어미의 결합으로 구성된다.

언	품사
체언	명사, 대명사, 수사
수식언	관형사, 부사
관계언	조사
독립언	감탄사
용언	동사, 형용사

활용 (conjugation)

1. **규칙활용**

   어간이 어미를 취할때 어간의 모습이 일정한 경우

   ```
   잡다 → 잡 (어간) + 다(어미)
   ```

2. **불규칙 활용**

   어간이 어미를 취할때 어간의 모습이 바뀌거나 취하는 어미가 특수한 경우

   ```
   곱/ 고우 → 곱다, 고운
   이르 + 아/어 → 이르러
   ```

불용어란

큰의미가 없는 단어로 데이터에서 유의미한 단어 토큰만을 선별하기 위해 의미없는 단어 토큰을 제거하는 작업이 필요하다.

```
I, my, me
```

한국어에서는 토큰화 후에 조사, 접속사 등을 제거하는 방법, 일부 명사, 형용사를 제거하는 방법, 사용자가 직접 불용어 사전을 만드는 방법이 있다.

정수 인코딩

텍스트를 컴퓨팅으로 처리하기 위해서 숫자로 맵핑하는 전처리

대표적인 방법은 단어를 빈도수로 정리하여 빈도수가 높은 순서부터 index를 부여하는 방법

인코딩을 진행하면서 빈도수가 낮은 단어는 제거

→ 추후 인코딩이 불가능한 새로운 단어가 나오는 경우 발생

→ OOV(Out of vocabulary) 토큰을 추가한다.

패딩

문장의 길이가 서로 다른 데이터들을 병렬처리를 위해서 길이를 임의로 동일하게 맞춰주는 작업

가장 길이가 긴 단어에 맞추거나 적절한 최대 크기를 설정한다.

원-핫 인코딩

앞서 정수 인코딩된 토큰들을 백터형태로 변환하는일

단어 집합 (vocabulary)

서로 다른 단어들의 집합으로 같은 단어의 변형 형태도 다른 단어로 간주한다.

단어 집합에 있는 단어들을 가지고 문자를 숫자로, 이 숫자를 벡터로 바꾸는 여러 전처리 방법이 필요하다.

원핫 벡터의 한계

- 단어 사이의 유사도를 표현하지 못한다.
- vocabulary가 커지는 경우 벡터의 크기가 커지게 된다.

→ 다른 방식의 벡터화 기법도 도입

1. 카운트 기반의 벡터화: LSA, HAL
2. 예측 기반의 벡터화: NNLM, RNNLM, Word2Vec, FastText

한국어 전처리 패키지

- PyKoSpacing

  띄어쓰기가 되어있지 않은 문장을 띄어쓰기한 문장으로 변환해주는 패키지

- Py-Hanspell

  네이버 한글 맞춤법 검사기를 바탕으로 만들어진 전처리 패키지

- SOYNLP

  품사 태깅, 단어 토큰화 등을 지원하는 단어 토크나이저

3장. 언어 모델

언어 모델

단어 시퀀스에 확률을 할당하는 일을 하는 모델. 가장 보편적으로 사용하는 방법은 언어 모델이 이전 단어들이 주어졌을때 다음 단어를 예측하도록 하는 것

언어 모델링

주어진 단어들로부터 아직 모르는 단어를 예측하는 작업

단어 시퀀스의 확률 할당

- **기계 번역**
  P(나는 버스를 탔다) >P(나는 버스를 태운다)
- **오타 교정**
  선생님이 교실로 부리나케 (P(달려갔다) > P(잘려갔다))
- **음성 인식**
  P(나는 메롱을 먹는다) > P(나는 메론을 먹는다)

이처럼 확률을 통해서 보다 적절한 문장을 판단.

조건부 확률

	남학생	여학생	계
중학생	100	60	160
고등학생	80	120	200
계	180	180	360

```
A: 학생이 남학생인 사건
B: 학생이 여학생인 사건
C: 학생이 중학생인 사건
D: 학생이 고등학생인 사건
```

- 학생을 뽑았을 때 남학생일 확률
  $P(A)$ = 180/360 = 0.5
- 학생을 뽑았을 때 고등학생이면서 남학생일 확률
  $P(A \cap B)$ = 80/360 = 0.223
- 고등학생을 뽑았을 때 남학생일 확률
  $P(A \mid D)$ = 80 / 200 = $P(A \cap D)/P(D)$ = (80/360)/(200/360) = 80/200

조건부 확률의 연쇄 법칙

4개의 확률이 조건부 확률의 관계를 가진다면 다음과 같이 표현 가능하다.

$$P(A, B, C, D) = P(A)P(B \mid A)P(C \mid A, B)P(D \mid A, B, C)$$

이를 n개로 일반화 하면 다음과 같다.

$$P(x_1, x_2, x_3, ...x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2)..P(x_n \mid x_1...x_{n-1})$$

주어진 이전 단어들로 부터 다음 단어를 예측하기

단어 시퀀스에 확률을 할당하기 위해서 가장 보편적으로 사용하는 방법은 이전단어들이 주어졌을 때 다음 단어를 예측하도록 하는 것.

단어 시퀀스의 확률

하나의 단어를 w, 단어 시퀀스를 W 라고 한다면 n개의 단어가 등장하는 단어 시퀀스 W의 확률은 다음과 같다.

$$P(W) = P(w_1, w_2, w_3, w_4 ..., w_n)$$

> 여기서 $w_1, w_2, w_3 ..., w_n$은 문장 $w_1\ w_2\ w_3\ ..w_n$을 의미한다.
> 예를들어서 P(I like you) 는 P(I, like, you) 가 된다.

다음 단어의 등장 확률

n-1 개의 단어가 나열된 상태에서 n번째 단어의 확률은 다음과 같다.

$$P(w_n \mid w_1, ..., w_n)$$

전체 단어 시퀀스 W의 확률은 모든 단어가 예측되고 나서 알 수 있으므로 다음과 같다.

$$P(W) = P(w_1, w_2, w_3, w_4..., w_n) = \prod_{i=1}^{n} P(w_i \mid w_1, ..., w_{i-1})$$

카운트 기반의 접근

앞서 I like you 문장을 이용해서 I like 가 나왔을때 you가 나올 확률을 카운트 기반으로 계산하면 다음과 같다.

$$P(\text{you} \mid \text{I like}) = \frac{\textbf{count}(\text{I like you})}{\textbf{count}(\text{I like})}$$

→ 만약 문장이 길어지는 경우, 다음에 나올 단어가 나오는 횟수가 데이터에 적을 경우 해당 확률이 0이 되기 때문에 방대한 양의 데이터가 필요하다. (희소 문제, Sparsity Problem)

N-gram 언어 모델

카운트 기반 접근법의 희소 문제를 줄이기 위해서 n-1 개의 단어만 고려

다음과 같은 영어 문장이 주어질때,

Explain the sparsity problem that can occur if given a long sentence.

카운트 기반

Explain the sparsity problem that can occur if given a long 이라는 문장이 나올 확률이 극히 적다.

$$P(\text{sentence} \mid \text{Explain the sparsity problem that can occur if given a long}) = \frac{\textbf{count}(\text{Explain the sparsity problem that can occur if given a long sentence})}{\textbf{count}(\text{Explain the sparsity problem that can occur if given a long})}$$

N(4)-gram 모델

given a long sentence 라는 문장은 비교적 발생하기 쉽다.

$$P(\text{sentence} \mid \text{given a long}) = \frac{\textbf{count}(\text{given a long sentence})}{\textbf{count}(\text{given a long})}$$

N-gram 언어 모델의 한계

- 여전히 희소 문제는 발생한다.
- n의 길이를 선택함에 따라서 희소 문제와 실제 성능 사이의 trade-off 관계가 생긴다.
  - n이 길어지면 희소 문제 발생
  - n이 짧아지면 근사 정확도가 현실의 확률분포와 멀어짐

한국어의 언어 모델

한국어는 아래와 같은 문제로 인해서 타 언어에 비해서 자연어 처리가 까다로움

1. 한국어는 어순이 중요하지 않다.
2. 한국어는 교착어이다.
3. 한국어는 띄어쓰기가 제대로 지켜지지 않는다.

펄플렉서티 (Perplexity, PPL)

언어모델의 평가 지표중 하나이며 다음과 같이 정의된다.

$$PPL(W) = P(w_1, w_2, w_3, ...w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{\prod_{i=1}^{n} P(w_i \mid w_1, ..., w_{i-1})}}$$

이때, n-gram을 적용할 수도 있다.

분기 계수

PPL의 의미를 살펴보면 선택할 수 있는 경우의 수를 의미하는 분기 계수이다.

식의 의미가 평균 확률의 역수, 즉 발생할 수 있는 확률의 가지 수를 의미한다. PPL이 10이라면 다음 단어를 예측할때 평균 10개의 단어를 가지고 분기가 가능하다는것을 의미한다.

- PPL이 더 낮다고 현실에서 더 좋은 효과를 보이는것은 아니다.
- 정량적으로 양이 많고 도메인에 알맞은 데이터를 이용해서 두 모델을 비교하는것이 신뢰도가 높다.