

KUBIG 2023-1

겨울방학 자연어처리(NLP) 분반

WEEK 2

주차별 커리큘럼

커리큘럼

주차	복습과제	진도	예습과제
1주차 1/12	<ul style="list-style-type: none">- 지난 주차시에 배운 내용 관련해서 교재 내용 or 논문 or 강의 자료 자유롭게 복습 정리- 2~3회 정도 추가 코딩 구현 과제 나갈 예정 (선택)- 수업 시작 전 복습 과제 발표 (1~2명)- 마감기한 : 세션 시작 전까지	스터디 OT 딥러닝 복습	<ul style="list-style-type: none">- 다음 주차시에 배울 내용과 관련해서 실습 코드 제공- 그대로 클론 코딩 해봐도 좋고 실습 파일에 주어진 데이터 이외에의 데이터를 갖고와 다양한 시도를 해봐도 좋습니다- 마감기한 : 수요일 11시 59분까지
2주차 1/19		텍스트 전처리 카운트 기반 전처리	
3주차 1/26		언어모델 , 워드 임베딩 : Word2Vec GloVe ELMo	
4주차 2/2		순환 신경망 : RNN LSTM	
5주차 2/9		Attention Transformer	
6주차 2/16		BERT BART	
7주차 2/23		GPT	
쿠빅 콘테스트 3/2			



Part 1

복습과제 리뷰

- Pytorch tutorial
- Ch1. 신경망 복습

pytorch

◦ 텐서와 벡터의 차원 handling

```

start
token_ids ==> tensor([[ 2, 9050, 739, ..., 3, 3, 3],
                    [ 2, 9501, 7623, ..., 3, 3, 3],
                    [ 2, 739, 5, ..., 3, 3, 3],
                    ...,
                    [ 2, 9912, 11732, ..., 3, 3, 3],
                    [ 2, 9669, 7055, ..., 3, 3, 3],
                    [ 2, 739, 7569, ..., 3, 3, 3]])
mask ==> tensor([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]])
label ==> tensor([[9, 9, 9, ..., 3, 3, 3],
                 [9, 9, 9, ..., 3, 3, 3],
                 [9, 9, 9, ..., 3, 3, 3],
                 ...,
                 [9, 9, 9, ..., 3, 3, 3],
                 [9, 9, 9, ..., 3, 3, 3],
                 [9, 9, 9, ..., 3, 3, 3]])
token_ids ==> tensor([[ 2, 10607, 23971, ..., 3, 3, 3],
                    [ 2, 9050, 8125, ..., 3, 3, 3],
                    [ 2, 739, 7088, ..., 3, 3, 3],

```

◦ 직접 neural network 쌓아보기

```

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120) # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

```

NN 구현

forward_net.ipynb

show_spiral_dataset.ipynb

train_custom_loop.ipynb

two_layer_net.ipynb

two_layer_net.py

taeyoung kim

```

import numpy as np

class Sigmoid :
    def __init__(self) :
        self.params = []

    def forward(self, x) :
        return 1/(1+np.exp(-x))

class Affine :
    def __init__(self, W, b):
        self.params = [W,b]

    def forward(self, x) :
        W, b = self.params
        out = np.matmul(x,W) + b
        return out

```

```

class TwoLayerNet :
    def __init__(self, input_size, hidden_size, output_size):
        I, H, O = input_size, hidden_size, output_size


        # 가중치, 편향 초기화
        W1 = np.random.randn(I,H)
        b1 = np.random.randn(H)
        W2 = np.random.randn(H,O)
        b2 = np.random.randn(O)

        # 계층 생성
        self.layers = [
            Affine(W1, b1),
            Sigmoid(),
            Affine(W2, b2)
        ]

        # 가중치 모으기
        self.params = []
        for layer in self.layers:
            self.params += layer.params

    def predict(self, x) :
        for layer in self.layers :
            x = layer.forward(x)
        return x

```



Part 2

2주차진도 (1)

텍스트전처리

전처리 흐름

| 텍스트 전처리

: 비정형 데이터인 텍스트 데이터를 일정하게 정형화된 feautre의 형태로 바꿔주는 작업

| 전처리 순서

Cleansing

-정규표현식

Normalization

-대소문자 통합
-어간/표제어 추출

Tokenization

-불용어 제거
-토큰화

Encoding

-정수 인코딩
-패딩
-원핫 인코딩

| 주요 자연어 처리 패키지

영어 : NLTK

```
import nltk
nltk.download("book", quiet=True)
from nltk.book import *

*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
```

한국어 : konlpy

API

- [konlpy Package](#)
 - [Subpackages](#)
 - [tag Package](#)
 - [Hannanum Class](#)
 - [Kkma Class](#)
 - [Komoran Class](#)
 - [Mecab Class](#)
 - [Okt Class](#)
 - [corpus Package](#)

Cleansing

I 정규표현식

: 규칙이 있는 데이터에 대해서 규칙을 선언해 정제할 수 있는 방법

ex) 크롤링해서 얻은 데이터에 html 태그가 있을 때, 주어진 데이터가 json 파일일 때

◦ 특수문자

특수 문자	설명	특수 문자	설명
.	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 \n는 제외)	{숫자}	숫자만큼 반복합니다.
?	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)	{숫자1, 숫자2}	숫자1 이상 숫자2 이하만큼 반복합니다. ?, *, +를 이것으로 대체할 수 있습니다.
*	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)	{숫자,}	숫자 이상만큼 반복합니다.
+	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)	[]	대괄호 안의 문자들 중 한 개의 문자와 매치합니다. [amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재하면 매치를 의미합니다. [a-z]와 같이 범위를 지정할 수도 있습니다. [a-zA-Z]는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미합니다.
^	뒤의 문자열로 문자열이 시작됩니다.	[^문자]	해당 문자를 제외한 문자를 매치합니다.
\$	앞의 문자열로 문자열이 끝납니다.		A B와 같이 쓰이며 A 또는 B의 의미를 가집니다.

₩n 흠.. 근처 갈일 있어서 옛 기억에 다시 찾아가봤다.여기보다 ...

₩n 평냉은 빙산의 일각, 본체는 만두와 제육.₩ 냉면 (13,00...

₩n 드디어 진미평양냉면에 방문했다. 평양냉면 치곤 간이 기대했던 ...

₩n 인기 있는 어복쟁반과 냉면냉면 육수를 내고 난 사태를 썰어 쟁...

₩n 특색을 모르겠다₩n

```
{
  "PORT-speech-00011",
  "REPORT",
  "speech",
  "속기자로 10쪽 (20).hwp",
  1,
  null,
  "doc_origin": "문화체육관광부 정책브리핑 포털"
},
{
  "Meta(Refine)": {
    "passage_id": "REPORT-speech-00011-00004",
    "passage": "다음은 건설산업기본법 시행령 일부개정 관련",
    "passage_Cnt": 794
  },
  "Annotation": {
    "summary1": "기술능력 기술인은 상시 근무하는 사람만 인",
    "summary2": null,
    "summary3": "건설업을 하려는 자는 기술능력.자본금.시설",
    "summary_3_cnt": 176
  }
}
```

ex) 주민등록번호 뒷자리

[1-4][0-9]{6}

내가 찾고 싶은 문자열이

숫자 : [0-9]

한글 모두 : [ㄱ-ㅎ가-힣]

내가 찾고 싶은 문자열이

문자 또는 숫자가 아닌 문자일 때

: [^a-zA-Z0-9]

Cleansing

I 정규표현식

◦ ~로 시작/끝/포함 하는 문자열

정규표현식	패턴
^The	The로 시작하는 문자열
of despair\$	of despair로 끝나는 문자열
^abc\$	abc로 시작하고 abc로 끝나는 문자열 (abc 라는 문자열도 해당됨)
notice	notice가 들어 있는 문자열

◦ () : 문자열 묶음 처리, | : 또는

a(bc)*	a 다음에 bc가 0개 이상 (묶음 처리)
a(bc){1,5}	a 다음에 bc가 1개에서 5개 사이
hi hello	hi나 hello가 들어 있는 문자열
(b cd)ef	bef 또는 cdef
(a b)*c	a와 b가 섞여서 여러번 나타나고 그뒤에 c가 붙어있는 패턴

◦ ~ 다음의 문자열

정규표현식	패턴
ab*	a 다음에 b가 0개 이상 (a, ab, abbb 등등)
ab+	a 다음에 b가 1개 이상 (ab, abbb 등등)
ab?	a 다음에 b가 있거나 없거나 (ab 또는 a)
ab{2}	a 다음에 b가 2개 있는 문자열 (abb)
ab{2,}	a 다음에 b가 2개 이상 (abb, abbbb 등등)
ab{3,5}	a 다음에 b가 3개에서 5개 사이 (abbb, abbbb, 또는 abbbbbb)

Cleansing

I 정규표현식 모듈 함수

re.compile()	정규표현식을 컴파일하는 함수입니다. 다시 말해, 파이썬에게 전해주는 역할을 합니다. 찾고자 하는 패턴이 빈번한 경우에는 미리 컴파일해놓고 사용하면 속도와 편의성면에서 유리합니다.
re.search()	문자열 전체에 대해서 정규표현식과 매치되는지를 검색합니다.
re.match()	문자열의 처음이 정규표현식과 매치되는지를 검색합니다.
re.split()	정규 표현식을 기준으로 문자열을 분리하여 리스트로 리턴합니다.
re.findall()	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열을 찾아서 리스트로 리턴합니다. 만약, 매치되는 문자열이 없다면 빈 리스트가 리턴됩니다.
re.finditer()	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열에 대한 이터레이터 객체를 리턴합니다.
re.sub()	문자열에서 정규 표현식과 일치하는 부분에 대해서 다른 문자열로 대체합니다.

```
import re
p = re.compile('[a-z]+')
m = p.match("hi, 안녕하세요")
print(m)
```

```
<re.Match object; span=(0, 2), match='hi'>
```

```
result = p.findall("life is too short")
print(result)
```

```
['life', 'is', 'too', 'short']
```

```
text = "I like apple And apple"
text_mod = re.sub('apple|apple',"apple",text)

print (text_mod)
```

```
I like apple And apple
```

Cleansing

유용한 정규표현식

- 전자우편 주소: `/^[a-z0-9_+.-]+@([a-z0-9-]+\.)+[a-z0-9]{2,4}$/`
- URL: `/^(file|gopher|news|nntp|telnet|https?|ftp|sftp):\\\/([a-z0-9-]+\.)+[a-z0-9]{2,4}.*$/`
- HTML 태그: `/<\/?[^\>]+\>/`
- 전화 번호: `/(\d{3}).(\d{3}).(\d{4})/`
예) 123-123-2344 또는 123-1234-1234
- 날짜: `/^\d{1,2}\/\d{1,2}\/\d{2,4}$/`
예) 3/28/2007 또는 3/28/07
- jpg, gif 또는 png 확장자를 가진 그림 파일명: `/([^\s]+(?:\.jpg|gif|png))\.\\2/`
- 적어도 소문자 하나, 대문자 하나, 숫자 하나가 포함되어 있는 문자열(8글자 이상 15글자 이하) → 많은 사이트에서 올바른 암호형식인지 확인하기 위한 용도: `/(?=.\d)(?=.*[a-z])(?=.*[A-Z]).{8,15}/`
- 주민번호 (-까지 포함): `/^(?:[0-9]{2}(?:0[1-9]|1[0-2])(?:0[1-9]|[1,2][0-9]|3[0,1]))-[1-4][0-9]{6}$/`

```
[ ] ## 부호를 제거해주는 함수 > 한글, 영어, 숫자, 한자, 공백만 남김
def alpha_num(text):
    return re.sub(r'[^가-힣ㄱ-ㅎㅏ-ㅣa-zA-Z0-9- -]', '', text)
```

Nomalization

| 대소문자 통합

```
[13] s3 = 'abCDef'.upper()
      print('s3 : ' + s3)

s3 : ABCDEF
```

```
[14] s3 = 'WE HAVE TO CHANGE IT INTO LOWER CASE'.lower()

      print('s3 : ' + s3)

s3 : we have to change it into lower case
```

△주의

The New York Times, Jay, USA, Korea Repulic,

| 어간 추출 (stemmatization)

단어의 어간 추출 -> 복수형, 진행형 제거하고 어간만을 추출

```
# PorterStemmer
stemmer = nltk.stem.PorterStemmer()
print(stemmer.stem('messaging'))
print(stemmer.stem('runs'))
print(stemmer.stem('watched'))
print(stemmer.stem('has'))
```

```
messag
run
watch
ha
```

| 표제어 추출 (lemmatization)

품사 정보가 보존된 형태의 기본형으로 변환하는 작업

```
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

print(wordnet_lemmatizer.lemmatize('meessages'))
print(wordnet_lemmatizer.lemmatize('watched'))
print(wordnet_lemmatizer.lemmatize('watching'))
```

```
meessages
watched
watching
```

Tokenization

I 토큰화

문서나 문장을 분석하기 좋도록 의미있는 작은 단위로 분리하는 작업 (=형태소)

영어 : 띄어쓰기(whitespace)를 기준으로 잘라냄

```
▶ text = "Courage is very importatn when it comes to anything"
print('원문 : ',text)
print('단어 토큰화 : ',WordPunctTokenizer().tokenize(text))
```

```
☞ 원문 : Courage is very importatn when it comes to anything
단어 토큰화 : ['Courage', 'is', 'very', 'importatn', 'when', 'it', 'comes', 'to', 'anything']
```

△주의 : ' (Apostrophe)

```
▶ print('단어 토큰화1 : ',word_tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a pastry shop."))
print('단어 토큰화2 : ',WordPunctTokenizer().tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a pastry shop."))
print('단어 토큰화3 : ',text_to_word_sequence("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a pastry shop."))
```

```
☞ 단어 토큰화1 : ['Do', "n't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr.', 'Jone', "'s", 'Orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop', '.']
단어 토큰화2 : ['Don', "'", 't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr', '.', 'Jone', "'", 's', 'Orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop', '.']
단어 토큰화3 : ['don't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', 'mr', "jone's", 'orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop']
```

+) NLTK – SpaceTokenizer, NLTK – TweetTokenizer

Tokenization

| 토큰화

문서나 문장을 분석하기 좋도록 의미있는 작은 단위로 분리하는 작업 (=형태소)

한국어에서 형태소 : 일정한 의미가 있는 가장 작은 말의 단위
띄어쓰기로 정의 x

이 편지는 영국에서 최초로 시작되어 일년에 한바퀴를 돌면서 받는 사람에게 행운을 주었고 지금은 당신에게로 옮겨진 이 편지는 4일 안에 당신 곁을 떠나야 합니다. 이 편지를 포함해서 7통을 행운이 필요한 사람에게 보내 주셔야 합니다. 복사를 해도 좋습니다. 혹 미신이라 하실지 모르지만 사실입니다. 영국에서 HGXWCH이라는 사람은 1930년에 이 편지를 받았습니다. 그는 비서에게 복사해서 보내라고 했습니다. 며칠 뒤에 복권이 당첨되어 20억을 받았습니다. 어떤 이는 이 편지를 받았으나 96시간 이내 자신의 손에서 떠나야 한다는 사실을 잊었습니다. 그는 곧 사직되었습니다. 나중에야 이 사실을 알고 7통의 편지를 보냈는데 다시 좋은 직장을 얻었습니다. 미국의 케네디 대통령은 이 편지를 받았지만 그냥 버렸습니다. 결국 9일 후 그는 암살당했습니다. 기억해 주세요. 이 편지를 보내면 7년의 행운이 있을 것이고 그렇지 않으면 3년의 불행이 있을 것입니다. 그리고 이 편지를 버리거나 낙서를 해서는 절대로 안됩니다. 7통입니다. 이 편지를 받은 사람은 행운이 깃들것입니다. 힘들겠지만 좋은게 좋다고 생각하세요. 7년의 행운을 빌면서...

**기계가 이해할 수 있는 유의미한 최소한의 단위로
 나누어주는 것이 필요!**



Tokenization

I 토큰화

문서나 문장을 분석하기 좋도록 의미있는 작은 단위로 분리하는 작업 (=형태소)

한국어에서 형태소 : 일정한 의미가 있는 가장 작은 말의 단위
띄어쓰기로 정의 x

예 : 공간적인 범위, 이유 등 꽤 다양한 의미로 쓰이는 처격조사

어제는 종일 집에 있었다.(공간적 범위)

나팔꽃은 아침에 피었다가 저녁에 진다.(시간적 범위)

이 책은 어디에 보낼 거니?(지향점)

올림픽은 4년에 한 번씩 열린다.(단위)

그 글은 내 마음에 들이 않는다.(추상적 공간)

천둥소리에 모두들 깜짝 놀랐다.(이유)

생각난 김에, 순식간에, 에 대하여...(관용구)

'제성이가 늦은 밤까지 유튜브를 봤다'
 -> ['제성이가', '늦은', '밤까지', '유튜브를', '봤다']

자립 형태소 : 제성, 밤, 유튜브

의존 형태소 : -이가, 늦-, -은, -까지, 를, 보-, -았-, -다

실질 형태소 : 제성, 늦-, 밤, 유튜브, 보-

형식 형태소 : -이가, -은, -까지, 를, -았-, -다

Tokenization

| 토큰화

문서나 문장을 분석하기 좋도록 의미있는 작은 단위로 분리하는 작업 (=형태소)

한국어 토큰화 - 형태소 분석 / 품사 태깅 (POS tagging)

```
[46] from konlpy.tag import Kkma  
     kkma = Kkma()
```

```
▶ text = "호의가 계속되면 그게 권리인 줄 알아."  
  print(" 형태소 단위 : ",kkma.morphs(text))  
  print(" 명사 단위 : ",kkma.nouns(text))  
  print(" 품사 단위 : ",kkma.pos(text))
```

- 형태소 단위 : ['호의', '가', '계속', '되', '면', '그', '게', '권리', '이', 'ㄴ', '줄', '알', '아', '.']
- 명사 단위 : ['호의', '계속', '권리', '줄']
- 품사 단위 : [('호의', 'NNG'), ('가', 'JKS'), ('계속', 'NNG'), ('되', 'XSV'), ('면', 'ECE'), ('그', 'VV'), ('게', 'ECD'), ('권리', 'NNG'), ('이', 'VCP'), ('ㄴ', 'ETD'), ('줄', 'NNB'), ('알', 'VV'), ('아', 'ECD'), ('.', 'SF')]

+) Okt(), Mecab(), Pororo()

Tokenization

I 불용어(Stopword) 제거

분석에 큰 의미가 없는 단어 (관사, 일반대명사..)

영어 : NLTK 패키지에서 미리 정의된 불용어들과 매칭해서 제거

```
[4] stop_words_list = stopwords.words('english')
    print('불용어 개수 :', len(stop_words_list))
    print('불용어 10개 출력 :', stop_words_list[:10])
```

불용어 개수 : 179

불용어 10개 출력 : ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]

한국어 : 사용자가 직접 불용어를 사전 정의 or '한국어 불용어 리스트' 파일을 다운받아서 이용

한국어불용어100.txt

이	VCP	0.018279601
있	VA	0.011699049
하	VV	0.009773659
것	NNB	0.00973315
들	XSN	0.00689824
그	MM	0.005327252
되	VV	0.00361335
수	NNB	0.003473622
이	NP	0.003361203
보	VX	0.003310379
않	VX	0.0029757
았	VA	0.002920476
나	NP	0.002690479
사람	NNG	0.002073917
주	VV	0.001884579
아니	VCN	0.001870774
등	NNB	0.001822074
할	VA	0.001724522
우리	NP	0.001714509
때	NNG	0.001685684

```
[ ] 1 #konlpy로 명사만 추출하는 토큰화를 진행
    2 okt = Okt()
    3 token = []
    4 for sentence in df['text']:
    5     temp_X = []
    6     temp_X = okt.nouns(sentence) # 명사 토큰화
    7     temp_X = [word for word in temp_X if not word in stopwords] # 불용어 제거
    8     token.append(temp_X)
    9 df['token'] = token
    10 tokenizer = Tokenizer()
    11 tokenizer.fit_on_texts(token)
```

4 # 불용어 설정

```
5 stopwords = ['않다', '에서', '있다', '없다', '그렇다', '아니다', '것', '이다', '의', '가', '이', '은', '들', '주',
6              '는', '좀', '잘', '강', '과', '도', '을', '를', '으로', '자', '에', '와', '한', '하다', '휴', '수', '일']
7
```

Encoding

I 정수 인코딩

텍스트 데이터를 기계가 인식할 수 있는 숫자로 변환

◦ 빈도수 기반 Dictionary

raw_text = "A barber is a person. a barber is good person. a barber is huge person. he Knew A Secret! The Secret He Kept is huge secret. Huge secret. His barber kept his word. a barber kept his word. His barber kept his secret. But keeping and keeping such a huge secret to himself was driving the barber crazy. the barber went up a huge mountain."

barber : secret : huge :

◦ Counter 기반 Dictionary

```
from collections import Counter
word_list = ['재벌집 막내 아들', '더 글로리', '수리남', '재벌집 막내 아들', '더 글로리', '일타 스캔들', '더 글로리', '더 글로리', '일타 스캔들']
vocab = Counter(word_list)
print(vocab)
```

Counter({'더 글로리': 4, '재벌집 막내 아들': 2, '일타 스캔들': 2, '수리남': 1})

Encoding

| 원핫 인코딩

['더 글로리':1, '재벌집 막내아들':2, '일타 스캔들':3, '수리남':4]
1 -> [1,0,0,0], 2->[0,1,0,0],,,

●한계점

1) 비효율적인 메모리 사용량

: 표현 단어의 개수가 늘어나고 벡터를 저장하기 위한 공간이 방대하게 증가

2) 단어의 유사도 표현 불가

: 단어들 간의 의미 차이, 의미 유사함을 표현하기가 어려움

Encoding

I 패딩 Padding

: 문자열 간의 길이가 다를 경우, 임의의 길이 만큼으로 통일시키고 나머지 부분은 0으로 패딩

"A barber is a person. a barber is good person. a barber is huge person. he Knew A Secret! The Secret He Kept is huge secret. Huge secret. His barber kept his word. a barber kept his word. His barber kept his secret. But keeping and keeping such a huge secret to himself was driving the barber crazy. the barber went up a huge mountain."



Counter({'barber': 8, 'secret': 6, 'huge': 5, 'kept': 4, 'person': 3, 'word': 2, 'keeping': 2, 'good': 1, 'knew': 1, 'driving': 1, 'crazy': 1, 'went': 1, 'mountain': 1})



[[1, 5], [1, 8, 5], [1, 3, 5], [9, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [7, 7, 3, 2, 10, 1, 11], [1, 12, 3, 13]]

Encoding

| 패딩


: 문자열 간의 길이가 다를 경우, 임의의 길이 만큼으로 통일시키고 나머지 부분은 0으로 패딩



[[1, 5], [1, 8, 5], [1, 3, 5], [9, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [7, 7, 3, 2, 10, 1, 11], [1, 12, 3, 13]]



([[1, 5, 0, 0, 0, 0, 0],
[1, 8, 5, 0, 0, 0, 0],
[1, 3, 5, 0, 0, 0, 0],
[9, 2, 0, 0, 0, 0, 0],
[2, 4, 3, 2, 0, 0, 0],
[3, 2, 0, 0, 0, 0, 0],
[1, 4, 6, 0, 0, 0, 0],
[1, 4, 6, 0, 0, 0, 0],
[1, 4, 2, 0, 0, 0, 0],
[7, 7, 3, 2, 10, 1, 11],
[1, 12, 3, 13, 0, 0, 0]])



Part 3

2주차진도 (2)

카운트기반 단어 표현

Bag of Words

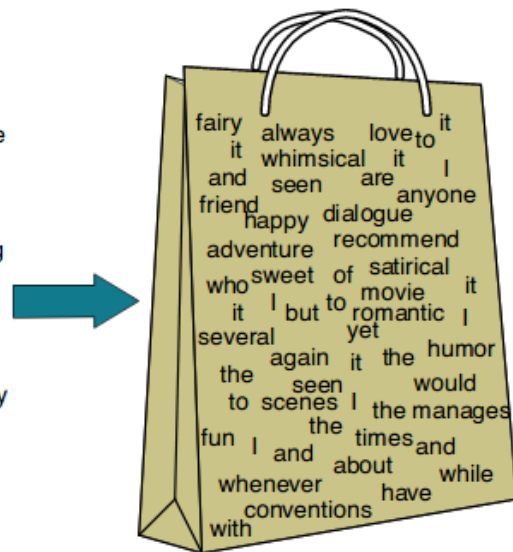
| 단순 단어의 빈도 순으로 고유 인덱스를 부여 (≡ 정수 인코딩)

text : 정부가 발표하는 물가상승률과 소비자가 느끼는 물가상승률은 다르다.

인코딩 :

Bag of words vector :

I love this movie! It's sweet,
but with satirical humor. The
dialogue is great and the
adventure scenes are fun...
It manages to be whimsical
and romantic while laughing
at the conventions of the
fairy tale genre. I would
recommend it to just about
anyone. I've seen it several
times, and I'm always happy
to see it again whenever I
have a friend who hasn't
seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

DTM

| 문서 단어 행렬 (Document-Term Matrix)

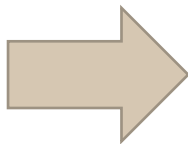
: 다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것

문서1 : 먹고 싶은 사과

문서2 : 먹고 싶은 바나나

문서3 : 길고 노란 바나나 바나나

문서4 : 저는 과일이 좋아요



띄어쓰기 단위 토큰화

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

●한계점

1) 희소 표현 (Sparse representation)

: 단어 집합 크기를 감당하기 위한 공간적 낭비와 계산 리소스 증가

2) 단순 빈도 수 기반 접근

: 단어의 중요도 or 유사도는 표현하기가 어려움

TF-IDF

I 단어 빈도 - 역 문서 빈도

: 단어의 중요도를 고려하여 가중치를 부여

1) TF(Term Frequency) : 특정 문서 d에서 특정 단어 t의 **출현 빈도**

문서3에서 '바나나'의 TF =

2) DF(Document Frequency) : 전체 문서 D에서 특정 단어 t가 등장한 **문서의 개수**

df('바나나') =

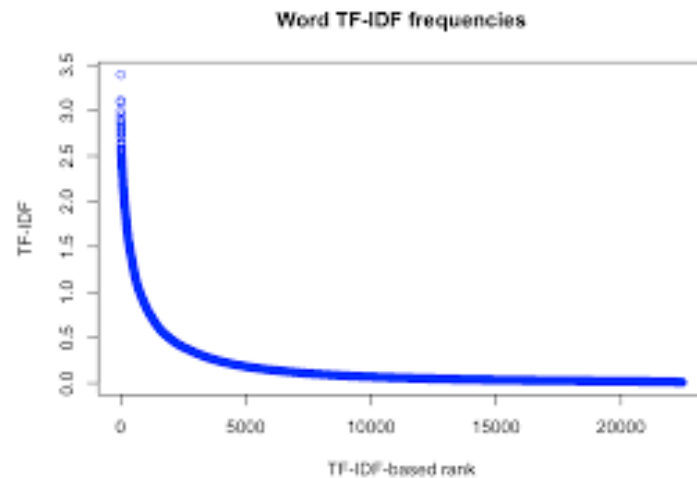
3) IDF(Inverse Document Frequency) : df의 역수 값 + 일부 scale

$$idf(d, t) = \ln \left(\frac{D}{1 + df(t)} \right)$$

수식값이 기하급수적으로 증가하는 것 방지
+ 분모가 0이 되는 상황 방지

$$TF-IDF = TF * IDF$$

	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1



코드

```
[53] import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
```

```
[54] documents = [
    "먹고 싶은 사과",
    "먹고 싶은 바나나",
    "길고 노란 바나나 바나나",
    "저는 과일이 좋아요"
]
```

```
# Document Term Matrix
dtm = vectorizer.fit_transform(documents)

# Term Frequency
tf = pd.DataFrame(dtm.toarray(), columns = vectorizer.get_feature_names())
```

과일이 길고 노란 먹고 바나나 사과 싶은 저는 좋아요

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
0	0	0	0	1	0	1	1	0	0
1	0	0	0	1	1	0	1	0	0
2	0	1	1	0	2	0	0	0	0
3	1	0	0	0	0	0	0	1	1

```
[60] # Document Frequency
df = tf.astype(bool).sum(axis = 0)
df
```

```
과일이    1
길고      1
노란      1
먹고      2
바나나    2
사과      1
싶은      2
저는      1
좋아요    1
dtype: int64
```

```
[61] # 문서 개수
D = len(tf)

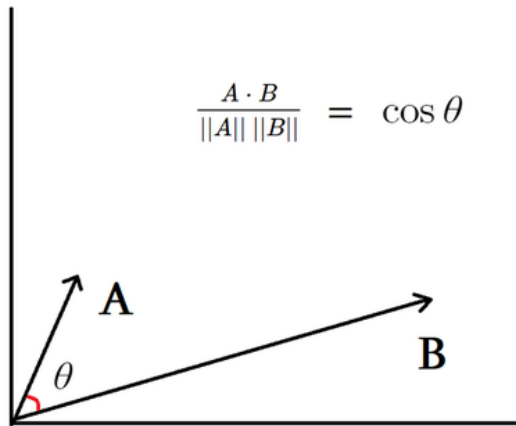
# Inverse Document Frequency
idf = np.log((D+1) / (df+1)) + 1
idf
```

```
과일이    1.916291
길고      1.916291
노란      1.916291
먹고      1.510826
바나나    1.510826
사과      1.916291
싶은      1.510826
저는      1.916291
좋아요    1.916291
dtype: float64
```

```
# TF-IDF
tfidf = tf * idf
tfidf = tfidf / np.linalg.norm(tfidf, axis = 1, keepdims = True)
tfidf
```

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
0	0.00000	0.00000	0.00000	0.526405	0.00000	0.667679	0.526405	0.00000	0.00000
1	0.00000	0.00000	0.00000	0.577350	0.57735	0.000000	0.577350	0.00000	0.00000
2	0.00000	0.47212	0.47212	0.000000	0.74445	0.000000	0.000000	0.00000	0.00000
3	0.57735	0.00000	0.00000	0.000000	0.00000	0.000000	0.000000	0.57735	0.57735

응용



```
[75] import numpy as np
      from numpy import dot
      from numpy.linalg import norm

      def cos_sim(A, B):
          return dot(A, B)/(norm(A)*norm(B))

      doc1 = np.array([0,1,1,1])
      doc2 = np.array([1,0,1,1])
      doc3 = np.array([2,0,2,2])

      print('문서 1과 문서2의 유사도 :',cos_sim(doc1, doc2))
```

문서 1과 문서2의 유사도 : 0.6666666666666667

응용

| 유사도 기반 추천 시스템

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overview	...	release_date	revenue	runtime	spoken_languages	status	tagline	title	video	vote_ave
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...}	30000000	{'id': 16, 'name': 'Animation'}, {'id': 35, 'name': 'Adventure'}	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody, Andy's toys live happily in his room until Mr. Potato Head's face falls off and rolls down the road. A flying saucer crashes down, and out of it comes a mysterious alien space ship. Now it's just a matter of time before the bad guys show up and take over the world. But Woody and his friends are ready to fight back.	...	1995-10-30	373554033.0	81.0	{'iso_639_1': 'en', 'name': 'English'}	Released	NaN	Toy Story	False	
1	False	NaN	65000000	{'id': 12, 'name': 'Adventure'}, {'id': 14, 'name': 'Fantasy'}	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an enchanted board game that opens the door to a magical world of adventure, they discover an enchanted board game that opens the door to a magical world of adventure.	...	1995-12-15	262797249.0	104.0	{'iso_639_1': 'en', 'name': 'English'}, {'iso_639_1': 'en', 'name': 'English'}	Released	Roll the dice and unleash the excitement!	Jumanji	False	

```
data['overview'] = data['overview'].fillna('')
```

```
[8] tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(data['overview'])
print('TF-IDF 행렬의 크기(shape) :', tfidf_matrix.shape)
```

TF-IDF 행렬의 크기(shape) : (20000, 47487)

```
tfidf_matrix[4].todense()
matrix([[0., 0., 0., ..., 0., 0., 0.]])
```

```
[ ] cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

```
[ ] print('코사인 유사도 연산 결과 :', cosine_sim.shape)
```

코사인 유사도 연산 결과 : (20000, 20000)

응용

| 유사도 기반 추천 시스템

```
▶ title_to_index = dict(zip(data['title'], data.index))
```

```
# 영화 제목 Father of the Bride Part II의 인덱스를 리턴
idx = title_to_index['Father of the Bride Part II']
print(idx)
```

```
↳ 4
```

```
[ ] def get_recommendations(title, cosine_sim=cosine_sim):
    # 선택한 영화의 타이틀로부터 해당 영화의 인덱스를 받아온다.
    idx = title_to_index[title]

    # 해당 영화와 모든 영화와의 유사도를 가져온다.
    sim_scores = list(enumerate(cosine_sim[idx]))

    # 유사도에 따라 영화들을 정렬한다.
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # 가장 유사한 10개의 영화를 받아온다.
    sim_scores = sim_scores[1:11]

    # 가장 유사한 10개의 영화의 인덱스를 얻는다.
    movie_indices = [idx[0] for idx in sim_scores]

    # 가장 유사한 10개의 영화의 제목을 리턴한다.
    return data['title'].iloc[movie_indices]
```

```
▶ get_recommendations('The Dark Knight Rises')
```

```
12481      The Dark Knight
150        Batman Forever
1328        Batman Returns
15511      Batman: Under the Red Hood
585        Batman
9230      Batman Beyond: Return of the Joker
18035      Batman: Year One
19792      Batman: The Dark Knight Returns, Part 1
3095      Batman: Mask of the Phantasm
10122      Batman Begins
Name: title, dtype: object
```

```
[15] get_recommendations('The Shawshank Redemption')
```

```
16947      They Made Me a Fugitive
6548      Civil Brand
11327      Brute Force
17446      Girls in Prison
9225      Female Prisoner Scorpion: Jailhouse 41
9391      In Hell
8307      Sherlock, Jr.
19035      The Peach Thief
11767      Island of Fire
15130      Chained Heat
Name: title, dtype: object
```



Part 4

예습과제리뷰

- IMDB

1. BeautifulSoup을 통해 HTML 태그를 제거
2. 정규표현식으로 알파벳 이외의 문자를 공백으로 치환
3. 토큰화
4. NLTK 데이터를 사용해 Stopword를 제거
5. 어간추출(Stemming)과 음소표기법(Lemmatizing)의 개념을 이해하고 SnowballStemmer를 통해 어간을 추출

전처리 순서에 정답은 없습니다! 매 상황마다 데이터를 미리 파악하시고 상황에 맞게 필요한 전처리 기법을 활용하시는 걸 추천드려요

▼ 문자열 처리

- 위에서 간략하게 살펴본 내용을 바탕으로 함수를 만들어 문자열을 처리해 본다.

```
[ ] def review_to_words(raw_review):
    # 1. HTML 제거
    review_text = BeautifulSoup(raw_review, 'html.parser').get_text()
    # 2. 영문자가 아닌 문자는 공백으로 변환
    letters_only = re.sub('[^a-zA-Z]', ' ', review_text)
    # 3. 소문자 변환
    words = letters_only.lower().split()
    # 4. Stopwords를 세트로 변환
    # 파이썬에서는 리스트보다 세트로 찾는게 훨씬 빠르다.
    stops = set(stopwords.words('english'))
    # 5. Stopwords 제거
    meaningful_words = [w for w in words if not w in stops]
    # 6. 어간추출
    stemming_words = [stemmer.stem(w) for w in meaningful_words]
    # 7. 공백으로 구분된 문자열로 결합하여 결과를 반환
    return(' '.join(stemming_words))
```

서너 가지 샘플로 필요한 전처리 함수를 찾아보시고
마지막에 모든 데이터에 대해 적용할 땐
하나의 함수를 정의해서 전처리를 진행해보기!

폰트 설치용

```
!sudo apt-get install -y fonts-nanum
```

```
!sudo fc-cache -fv
```

```
!rm ~/.cache/matplotlib -rf
```

```
import matplotlib.pyplot as plt
```

```
plt.rc('font', family='NanumBarunGothic')
```

```
import matplotlib
```

```
get_ipython().run_line_magic('config', "InlineBackend.figure_format='retina' #화질 좋게 해주기")
```

이후에 런타임 다시 시작을 해주시면 폰트가 나옵니다!

코랩에서 matplotlib 한글 폰트 자꾸 깨질때

☆ Scikit-Learn의 CountVectorizer를 통해 feature 생성

- 정규표현식을 사용해 토큰을 추출한다.
- 모두 소문자로 변환시키기 때문에 good, Good, gOod이 모두 같은 특성이 된다.
- 의미없는 특성을 많이 생성하기 때문에 적어도 두 개의 문서에 나타난 토큰만을 사용한다.
- min_df로 토큰이 나타날 최소 문서 개수를 지정할 수 있다.

☆ CountVectorizer의 인수

- stop_words : 문자열 {'english'}, 리스트 또는 None (디폴트)
 - stop words 목록 'english'이면 영어용 스탑 워드 사용.
- analyzer : 문자열 {'word', 'char', 'char_wb'} 또는 함수
 - 단어 n-그램, 문자 n-그램, 단어 내의 문자 n-그램

- token_pattern : string
 - 토큰 정의용 정규 표현식
- tokenizer : 함수 또는 None (디폴트)
 - 토큰 생성 함수 .
- ngram_range : (min_n, max_n) 튜플
 - n-그램 범위
- max_df : 정수 또는 [0.0, 1.0] 사이의 실수. 디폴트 1
 - 단어장에 포함되기 위한 최대 빈도
- min_df : 정수 또는 [0.0, 1.0] 사이의 실수. 디폴트 1
 - 단어장에 포함되기 위한 최소 빈도

영노 님

느낀점 :

- 빈도수 기준 감성리뷰는 한계가 있음. 맥락이나 전체적인 주제, 중요도를 고려하는 것도 필요해보임. 예를들어, 처음부터 마지막까지 안좋은 얘기를 하다가, 마지막에 '그래도 볼만 하다' 라고 얘기하면, 부정적인 리뷰긴 하지만 그래도 좋은 평가를 남긴 것인데, 빈도수 기준 모델은 이런 맥락이나 중요도를 고려하지 못할수 있단 생각이 막연하게 들었음.
- 리뷰 길이 긴 텍스트의 경우 감성이 명확하게 나뉘겠지만, 짧은 텍스트이 경우 성능이 낮아질수도 있다는 생각이 듦.
- 긍정/부정 단어를 구분하는 기준도 더 정확하게 살펴봐야 할듯. 어떤 기준으로 구분하는거지?

규빈 님

```
▶ import nltk
nltk.download('wordnet')
```

```
ⓘ [nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
True
```

```
[ ] nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Package omw-1.4 is already up-to-date!
True
```

```
import nltk
nltk.download('all')
```

❖ 오류(해결)

아래 part에서 문제 발생 error code : omw-1.4 resource error

git hub 해결책 <https://github.com/nltk/nltk/issues/3024>

-> nltk.download('omw-1.4') 를 입력해 코드 추가하여 별도 다운받을 것

(아마 버전 오류인듯함)

```
[ ] from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
```

희준 님

```
# 트리 알고리즘 3개를 사용하고, 3개를 스택킹
rf = RandomForestClassifier(n_estimators = 200,
                           n_jobs = -1,
                           random_state=42,
                           max_depth=20)

xgb = XGBClassifier(n_estimators=200,
                   max_depth=10,
                   learning_rate=0.05,
                   objective='binary:logistic')

lgbm = LGBMClassifier(n_estimators=200,
                     max_depth=10,
                     metric='binary_logloss')

estimators = [('rf', rf),
              ('xgb', xgb),
              ('lgbm', lgbm)]

clf = StackingClassifier(estimators=estimators,
                        final_estimator=LogisticRegression())
```

```
print('Random Forest AUC Score :', roc_auc_score(y_val, y_pred_rf[:,1]))
print('XGBoost AUC Score :', roc_auc_score(y_val, y_pred_xgb[:,1]))
print('LGBM AUC Score :', roc_auc_score(y_val, y_pred_lgbm[:,1]))
print('Stacking Classifier AUC Score :', roc_auc_score(y_val, y_pred_clf[:,1]))
```

```
Random Forest AUC Score : 0.9076715878903935
XGBoost AUC Score : 0.9163296321043216
LGBM AUC Score : 0.9260239891523155
Stacking Classifier AUC Score : 0.925895703394225
```

```
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation, GRU, Flatten
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model, Sequential
import keras
import numpy as np
```

```
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
max_features = 5000
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(train['clean_reviews'])
list_tokenized_train = tokenizer.texts_to_sequences(train['clean_reviews'])
```

```
maxlen = 130
x_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
y = train['sentiment']
```

```
embed_size = 128
model = Sequential()
model.add(Embedding(max_features, embed_size))
model.add(Bidirectional(LSTM(32, return_sequences = True)))
model.add(GlobalMaxPool1D())
model.add(Dense(20, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(1, activation="sigmoid"))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
callback = keras.callbacks.EarlyStopping(monitor="val_loss", patience=3)
```

```
[ ] test['review'] = test.review.apply(lambda x: review_to_words(x))
```

```
[ ] test['sentiment'] = test['id'].map(lambda x: 1 if int(x.strip('').split('_')[1]) >= 5 else 0)
y_test = test['sentiment']
list_sentences_test = test['review']
list_tokenized_test = tokenizer.texts_to_sequences(list_sentences_test)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
prediction = model.predict(X_te)
y_pred = (prediction > 0.5)
from sklearn.metrics import f1_score, confusion_matrix
print('F1-score: {0}'.format(f1_score(y_pred, y_test)))
print('Confusion matrix:')
confusion_matrix(y_pred, y_test)
```

```
782/782 [=====] - 19s 23ms/step
F1-score: 0.8484603245255534
Confusion matrix:
array([[10771, 2016],
       [ 1729, 10484]])
```

복습과제

- (1) 전처리 전후 모델결과 비교
- (2) 유사도 기반 음식점 추천 시스템

다음주 진도

- <딥러닝을 이용한 자연어처리 입문>
Ch3. 언어 모델
- <밑바닥부터 시작하는 딥러닝2>
Ch3. word2vec

예습과제

- 네이버 영화 리뷰 Word2Vec



수고하셨습니다!

Contact

15기 김제성

✉ rlawptjd1409@korea.ac.kr

☎ 010-2609-5046