Deep Learning for NLP
Text Classification (English)

Prof. Joongheon Kim
https://sites.google.com/site/joongheonkim/
joongheon@gmail.com

서론

- 분류 대상 데이터
  - 데이터 이름: Bag of Words Meets Bags of Popcorn
  - 데이터 용도: 텍스트 분류의 학습을 목적으로 사용한다.
  - 데이터 출처: https://www.Kaggle.com/c/word2vec-nlp-tutorial/data
- 데이터 기초 설명
  - 인터넷 영화 데이터베이스(IMDB)에서 나온 영화 평점 데이터
  - 각 데이터는 영화 리뷰 텍스트와 평점에 따른 감정 값(긍정 vs 부정)
  - 감정분석(Sentiment Analysis)에 주로 사용됨
- SW구조
  - 1단계: 데이터를 불러옴
  - 2단계: 데이터 전처리 및 데이터 분석
  - 3단계: 알고리즘 모델링

Deep Learning for NLP
Text Classification (English)

Prof. Joongheon Kim
https://sites.google.com/site/joongheonkim/
joongheon@gmail.com

1단계: 데이터 불러옴(Step 1: Text Loading)

- **1단계: 데이터를 불러옴**
  - https://www.Kaggle.com/c/word2vec-nlp-tutorial 에서 직접 데이터를 받아올 수 있음
    - sampleSubmission.csv
    - unlabeledTrainData.tsv.zip
    - testData.tsv.zip
    - LabeledTrainData.tsv.zip
  - 위의 네 파일 중에서 3개는 zip이므로 압축 푸는 과정 필요(import zipfile)
- 2단계: 데이터 전처리 및 데이터 분석
- 3단계: 알고리즘 모델링

Step 1: Text Loading (Part 1)

```
1  import zipfile
2  DATA_IN_PATH = './data_in/'
3  file_list = ['labeledTrainData.tsv.zip', 'unlabeledTrainData.tsv.zip', 'testData.tsv.zip']
4
5  for file in file_list:
6      zipRef = zipfile.ZipFile(DATA_IN_PATH + file, 'r')
7      zipRef.extractall(DATA_IN_PATH)
8      zipRef.close()
9
10 import numpy as np
11 import pandas as pd
12 import os
13 import matplotlib.pyplot as plt
14 import seaborn as sns
15
16 print("파일 크기 : ")
17 for file in os.listdir(DATA_IN_PATH):
18     if 'tsv' in file and 'zip' not in file:
19         print(file.ljust(30) + str(round(os.path.getsize(DATA_IN_PATH + file) / 1000000, 2)) + 'MB')
20
21 train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)
22 train_data.head()
23
24 print('전체 학습데이터의 개수: {}'.format(len(train_data)))
25
26 train_length = train_data['review'].apply(len)
27 train_length.head()
```

압축을 푸는 과정(대상파일위치: ./data_in/ [2번째 줄, DATA_IN_PATH])

```
파일 크기 :
testData.tsv                   32.72MB
labeledTrainData.tsv           33.56MB
unlabeledTrainData.tsv         67.28MB
```

위의 셋 압축이 풀어진 tsv파일들의 사이즈를 MB단위로 읽어 옴

Step 1: Text Loading (Part 1)

```
1  import zipfile
2  DATA_IN_PATH = './data_in/'
3  file_list = ['labeledTrainData.tsv.zip', 'unlabeledTrainData.tsv.zip', 'testData.tsv.zip']
4
5  for file in file_list:
6      zipRef = zipfile.ZipFile(DATA_IN_PATH + file, 'r')
7      zipRef.extractall(DATA_IN_PATH)
8      zipRef.close()
9
10 import numpy as np
11 import pandas as pd
12 import os
13 import matplotlib.pyplot as plt
14 import seaborn as sns
15
16 print("파일 크기 : ")
17 for file in os.listdir(DATA_IN_PATH):
18     if 'tsv' in file and 'zip' not in file:
19         print(file.ljust(30) + str(round(os.path.getsize(DATA_IN_PATH + file) / 1000000, 2)) + 'MB')
20
21 train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)
22 train_data.head()
23
24 print('전체 학습데이터의 개수: {}'.format(len(train_da
25
26 train_length = train_data['review'].apply(len)
27 train_length.head()
```

read_csv함수: 데이터를 불러올 때에 사용하는 함수

- 첫번째 인자: tsv파일의 경로
- 두번째 인자: Header: 각 데이터에 항목명이 있으므로 Header는 0으로 설정
- 세번째 인자: 현재 사용할 데이터는 탭(\t)으로 구별되어 있음
- 네번째 인자: 쌍따옴표를 무시하기 위하여 3으로 세팅

Step 1: Text Loading (Part 1)

```
1  import zipfile
2  DATA_IN_PATH = './data_in/'
3  file_list = ['labeledTrainData.tsv.zip', 'unlabeledTrainData.tsv.zip', 'testData.tsv.zip']
4
5  for file in file_list:
6      zipRef = zipfile.ZipFile(DATA_IN_PATH + file, 'r')
7      zipRef.extractall(DATA_IN_PATH)
8      zipRef.close()
9
10 import numpy as np
11 import pandas as pd
12 import os
13 import matplotlib.pyplot as plt
14 import seaborn as sns
15
16 print("파일 크기 : ")
17 for file in os.listdir(DATA_IN_PATH)
18     if 'tsv' in file and 'zip' not i   ile:
19         print(file.ljust(30) + str(r    d(os.path.getsize(DATA_IN_PATH + file) / 1000000, 2)) + 'MB')
20
21 train_data = pd.read_csv( DATA_IN_PA    + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)
22 train_data.head()
23
24 print('전체 학습데이터의 개수: {}'.format(len(train_data)))
25
26 train_length = train_data['review'].apply(len)
27 train_length.head()
```

	id	sentiment	review
0	"5814_8"	1	"With all this stuff going down at the moment ...
1	"2381_9"	1	"\"The Classic War of the Worlds\" by Timothy ...
2	"7759_3"	0	"The film starts with a manager (Nicholas Bell...
3	"3630_4"	0	"It must be assumed that those who praised thi...
4	"9495_8"	1	"Superbly trashy and wondrously unpretentious ...

Step 1: Text Loading (Part 1)

```
1  import zipfile
2  DATA_IN_PATH = './data_in/'
3  file_list = ['labeledTrainData.tsv.zip', 'unlabeledTrainData.tsv.zip', 'testData.tsv.zip']
4
5  for file in file_list:
6      zipRef = zipfile.ZipFile(DATA_IN_PATH + file, 'r')
7      zipRef.extractall(DATA_IN_PATH)
8      zipRef.close()
9
10 import numpy as np
11 import pandas as pd
12 import os
13 import matplotlib.pyplot as plt
14 import seaborn as sns
15
16 print("파일 크기 : ")
17 for file in os.listdir(DATA_IN_PATH):
18     if 'tsv' in file and 'zip' not in file:
19         print(file.ljust(30) + str(round(os.path.getsize(DATA_IN_PATH + file) / 1000000, 2)) + 'MB')
20
21 train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)
22 train_data.head()
23
24 print('전체 학습데이터의 개수: {}'.format(len(train_data)))
25
26 train_length = train_data['review'].apply(len)
27 train_length.head()
```

```
전체 학습데이터의 개수: 25000
```

```
0    2304
1     948
2    2451
3    2247
4    2233
Name: review, dtype: int64
```

각 데이터의 문자열의 길이를 계산 함

Step 1: Text Loading (Part 2)

```
리뷰 길이 최대 값: 13710
리뷰 길이 최소 값: 54
리뷰 길이 평균 값: 1329.71
리뷰 길이 표준편차: 1005.22
리뷰 길이 중간 값: 983.0
리뷰 길이 제 1 사분위: 705.0
리뷰 길이 제 3 사분위: 1619.0

리뷰 단어 개수 최대 값: 2470
리뷰 단어 개수 최소 값: 10
리뷰 단어 개수 평균 값: 233.79
리뷰 단어 개수 표준편차: 173.74
리뷰 단어 개수 중간 값: 174.0
리뷰 단어 개수 제 1 사분위: 127.0
리뷰 단어 개수 제 3 사분위: 284.0

물음표가있는 질문: 29.55%
마침표가 있는 질문: 99.69%
첫 글자가 대문자 인 질문: 0.00%
대문자가있는 질문: 99.59%
숫자가있는 질문: 56.66%
```

```
29  print("")
30  print('리뷰 길이 최대 값: {}'.format(np.max(train_length)))
31  print('리뷰 길이 최소 값: {}'.format(np.min(train_length)))
32  print('리뷰 길이 평균 값: {:.2f}'.format(np.mean(train_length)))
33  print('리뷰 길이 표준편차: {:.2f}'.format(np.std(train_length)))
34  print('리뷰 길이 중간 값: {}'.format(np.median(train_length)))
35  print('리뷰 길이 제 1 사분위: {}'.format(np.percentile(train_length, 25)))
36  print('리뷰 길이 제 3 사분위: {}'.format(np.percentile(train_length, 75)))
37
38  print("")
39  train_word_counts = train_data['review'].apply(lambda x:len(x.split(' ')))
40  print('리뷰 단어 개수 최대 값: {}'.format(np.max(train_word_counts)))
41  print('리뷰 단어 개수 최소 값: {}'.format(np.min(train_word_counts)))
42  print('리뷰 단어 개수 평균 값: {:.2f}'.format(np.mean(train_word_counts)))
43  print('리뷰 단어 개수 표준편차: {:.2f}'.format(np.std(train_word_counts)))
44  print('리뷰 단어 개수 중간 값: {}'.format(np.median(train_word_counts)))
45  print('리뷰 단어 개수 제 1 사분위: {}'.format(np.percentile(train_word_counts, 25)))
46  print('리뷰 단어 개수 제 3 사분위: {}'.format(np.percentile(train_word_counts, 75)))
47
48
49  print("")
50  qmarks = np.mean(train_data['review'].apply(lambda x: '?' in x)) # 물음표가 구두점으로 쓰임
51  fullstop = np.mean(train_data['review'].apply(lambda x: '.' in x)) # 마침표
52  capital_first = np.mean(train_data['review'].apply(lambda x: x[0].isupper())) #  첫번째 대문자
53  capitals = np.mean(train_data['review'].apply(lambda x: max([y.isupper() for y in x]))) # 대문자가 몇개
54  numbers = np.mean(train_data['review'].apply(lambda x: max([y.isdigit() for y in x]))) # 숫자가 몇개
55  print('물음표가있는 질문: {:.2f}%'.format(qmarks * 100))
56  print('마침표가 있는 질문: {:.2f}%'.format(fullstop * 100))
57  print('첫 글자가 대문자 인 질문: {:.2f}%'.format(capital_first * 100))
58  print('대문자가있는 질문: {:.2f}%'.format(capitals * 100))
59  print('숫자가있는 질문: {:.2f}%'.format(numbers * 100))
```

2단계: 데이터 전처리 및 데이터 분석

- 1단계: 데이터를 불러옴
- **2단계: 데이터 전처리 및 데이터 분석**
- 3단계: 알고리즘 모델링

Step 2: Preprocessing (Part 1)

```
import re
import json
import pandas as pd
import numpy as np
import nltk
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
from tensorflow.python.keras.preprocessing.text import Tokenizer

DATA_IN_PATH = './data_in/'
train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)

review = train_data['review'][0] # 첫 번째 리뷰를 가져옴
review_text = BeautifulSoup(review,"html5lib").get_text() # HTML 태그 제거
review_text = re.sub("[^a-zA-Z]", " ", review_text) # 영어 문자를 제외한 나머지는 모두 공백으로 바꿈
print(review_text)

nltk.download('stopwords')
stop_words = set(stopwords.words('english')) # 영어 불용어 집합 구성

review_text = review_text.lower()
words = review_text.split() # 소문자 변환 후 단어마다 나눠서 단어 리스트로 만듦
words = [w for w in words if not w in stop_words] # 불용어를 제거한 리스트를 구성함
print(words)
clean_review = ' '.join(words) # 단어 리스트들을 다시 하나의 글로 합침
print(clean_review)
```

Step 2: Preprocessing (Part 1)

라이브러리

```
import re
import json
import pandas as pd
import numpy as np
import nltk
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
from tensorflow.python.keras.preprocessing.text import Tokenizer

DATA_IN_PATH = './data_in/'
train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)

review = train_data['review'][0] # 첫 번째 리뷰를 가져옴
review_text = BeautifulSoup(review,"html5lib").get_text() # HTML 태그 제거
review_text = re.sub("[^a-zA-Z]", " ", review_text) # 영어 문자를 제외한 나머지는 모두 공백으로 바꿈
print(review_text)

nltk.download('stopwords')
stop_words = set(stopwords.words('english')) # 영어 불용어 집합 구성

review_text = review_text.lower()
words = review_text.split() # 소문자 변환 후 단어마다 나눠서 단어 리스트로 만듦
words = [w for w in words if not w in stop_words] # 불용어를 제거한 리스트를 구성함
print(words)
clean_review = ' '.join(words) # 단어 리스트들을 다시 하나의 글로 합침
print(clean_review)
```

- 데이터 정제 → re, Beautiful Soup
- 불용어 제거 → NLTK 라이브러리의 stopwords 모듈

Step 2: Preprocessing (Part 1)

```
1  import re
2  import json
3  import pandas as pd
4  import numpy as np
5  import nltk
6  from bs4 import BeautifulSoup
7  from nltk.corpus import stopwords
8  from tensorflow.python.keras.preprocessing.sequence import pad_sequences
9  from tensorflow.python.keras.preprocessing.text import Tokenizer
10
11 DATA_IN_PATH = './data_in/'
12 train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)
13
14 review = train_data['review'][0] # 첫 번째 리뷰를 가져옴
15 review_text = BeautifulSoup(review,"html5lib").get_text() # HTML 태그 제거
16 review_text = re.sub("[^a-zA-Z]", " ", review_text) # 영어 문자를 제외한 나머지는 모두 공백으로 바꿈
17 print(review_text)
18
19 nltk.download('stopwords')
20 stop_words = set(stopwords.words('english')) # 영어 불용어 집합 구성
21
22 review_text = review_text.lower()
23 words = review_text.split() # 소문자 변환 후 단어마다 나눠서 단어 리스트로 만듦
24 words = [w for w in words if not w in stop_words] # 불용어를 제거한 리스트를 구성함
25 print(words)
26 clean_review = ' '.join(words) # 단어 리스트들을 다시 하나의 글로 합침
27 print(clean_review)
```

데이터 정제

- Beautiful Soup: get_text()함수를 사용하여 HTML 태그를 모두 제외
- re.sub: 영어 알파벳을 제외한 모든 문자(즉, 숫자/특수기호)를 공백으로 대체

Step 2: Preprocessing (Part 1)

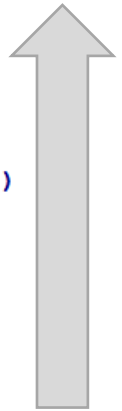
```
1  import re
2  import json
3  import pandas as pd
4  import numpy as np
5  import nltk
6  from bs4 import BeautifulSoup
7  from nltk.corpus import stopwords
8  from tensorflow.python.keras.preprocessing.sequence import pad_sequences
9  from tensorflow.python.keras.preprocessing.text import Tokenizer
10
11 DATA_IN_PATH = './data_in/'
12 train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)
13
14 review = train_data['review'][0] # 첫 번째 리뷰를 가져옴
15 review_text = BeautifulSoup(review,"html5lib").get_text() # HTML 태그 제거
16 review_text = re.sub("[^a-zA-Z]", " ", review_text) # 영어 문자를 제외한 나머지는 모두 공백으로 바꿈
17 print(review_text)
18
19 nltk.download('stopwords')
20 stop_words = set(stopwords.words('english')) # 영어 불용어 집합 구성
21
22 review_text = review_text.lower()
23 words = review_text.split() # 소문자 변환 후 단어마다 나눠서 단어 리스트로 만듦
24 words = [w for w in words if not w in stop_words] # 불용어를 제거한 리스트를 구성함
25 print(words)
26 clean_review = ' '.join(words) # 단어 리스트들을 다시 하나의 글로 합침
27 print(clean_review)
```

NLTK의 영어 불용어 사전 이용

Step 2: Preprocessing (Part 1)

```
1  import re
2  import json
3  import pandas as pd
4  import numpy as np
5  import nltk
6  from bs4 import BeautifulSoup
7  from nltk.corpus import stopwords
8  from tensorflow.python.keras.preprocessing.sequence import pad_sequences
9  from tensorflow.python.keras.preprocessing.text import Tokenizer
10
11 DATA_IN_PATH = './data_in/'
12 train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)
13
14 review = train_data['review'][0] # 첫 번째 리뷰를 가져옴
15 review_text = BeautifulSoup(review,"html5lib").get_text() # HTML 태그 제거
16 review_text = re.sub("[^a-zA-Z]", " ", review_text) # 영어 문자를 제외한 나머지는 모두 공백으로 바꿈
17 print(review_text)
18
19 nltk.download('stopwords')
20 stop_words = set(stopwords.words('english')) # 영어 불용어 집합 구성
21
22 review_text = review_text.lower()
23 words = review_text.split() # 소문자 변환 후 단어마다 나눠서 단어 리스트로 만듦
24 words = [w for w in words if not w in stop_words] # 불용어를 제거한 리스트를 구성함
25 print(words)
26 clean_review = ' '.join(words) # 단어 리스트들을 다시 하나의 글로 합침
27 print(clean_review)
```

- [라인22] 모든 알파벳을 소문자로 바꿈(불용어 사전 적용을 위하여)
- [라인23] 단어마다 나누어 단어 리스트로 만듦
- [라인24] 불용어가 아닌 단어들만 가지고 다시 단어 집합 구성
- [라인26] 단어들을 합하여 다시 문장으로 만듦



불용어 제거

Step 2: Preprocessing (Part 2)

```
29  def preprocessing(review, remove_stopwords = False):
30      # 불용어 제거는 옵션으로 선택 가능(remove_stopwords)
31      # 1. HTML 태그 제거
32      review_text = BeautifulSoup(review, "html5lib").get_text()
33      # 2. 영어가 아닌 특수문자들을 공백(" ")으로 바꾸기
34      review_text = re.sub("[^a-zA-Z]", " ", review_text)
35      # 3. 대문자들을 소문자로 바꾸고 공백단위로 텍스트들 나눠서 리스트로 만든다.
36      words = review_text.lower().split()
37
38      if remove_stopwords:
39          # 4. 불용어들을 제거
40          # 영어에 관련된 불용어 불러오기
41          stops = set(stopwords.words("english"))
42          # 불용어가 아닌 단어들로 이루어진 새로운 리스트 생성
43          words = [w for w in words if not w in stops]
44          # 5. 단어 리스트를 공백을 넣어서 하나의 글로 합침
45          clean_review = ' '.join(words)
46      else: # 불용어들을 제거하지 않는 경우
47          clean_review = ' '.join(words)
48
49      return clean_review
50
51  clean_train_reviews = []
52  for review in train_data['review']:
53      clean_train_reviews.append(preprocessing(review, remove_stopwords = True))
54  clean_train_df = pd.DataFrame({'review': clean_train_reviews, 'sentiment': train_data['sentiment']})
```

이 전 장의 “Step 2: Preprocessing (Part 1)”의 내용을 함수로 구성함

Step 2: Preprocessing (Part 3)

```
56  tokenizer = Tokenizer()
57  tokenizer.fit_on_texts(clean_train_reviews)
58  text_sequences = tokenizer.texts_to_sequences(clean_train_reviews)
59  print(text_sequences[0])
60
61  word_vocab = tokenizer.word_index
62  print(word_vocab)
63
64  print("전체 단어 개수: ", len(word_vocab) + 1)
65
66  data_configs = {}
67  data_configs['vocab'] = word_vocab
68  data_configs['vocab_size'] = len(word_vocab)
69
70  MAX_SEQUENCE_LENGTH = 174
71
72  train_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
73  print('Shape of train data: ', train_inputs.shape)
74
75  train_labels = np.array(train_data['sentiment'])
76  print('Shape of label tensor:', train_labels.shape)
```

Step 2: Preprocessing (Part 3)

```
56  tokenizer = Tokenizer()
57  tokenizer.fit_on_texts(clean_train_reviews)
58  text_sequences = tokenizer.texts_to_sequences(clean_train_reviews)
59  print(text_sequences[0])
60
61  word_vocab = tokenizer.word_index
62  print(word_vocab)
63
64  print("전체 단어 개수: ", len(word_vocab) + 1)
65
66  data_configs = {}
67  data_configs['vocab'] = word_vocab
68  data_configs['vocab_size'] = len(word_vocab)
69
70  MAX_SEQUENCE_LENGTH = 174
71
72  train_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
73  print('Shape of train data: ', train_inputs.shape)
74
75  train_labels = np.array(train_data['sentiment'])
76  print('Shape of label tensor:', train_labels.shape)
```

각 리뷰가 텍스트가 아닌 인덱스의 벡터로 구성될 것

- [라인59] 제일 첫 번째 리뷰를 출력해서 확인 함

Step 2: Preprocessing (Part 3)

```
56  tokenizer = Tokenizer()
57  tokenizer.fit_on_texts(clean_train_reviews)
58  text_sequences = tokenizer.texts_to_sequences(clean_train_reviews)
59  print(text_sequences[0])
60
61  word_vocab = tokenizer.word_index
62  print(word_vocab)
63
64  print("전체 단어 개수: ", len(word_vocab) + 1)
65
66  data_configs = {}
67  data_configs['vocab'] = word_vocab
68  data_configs['vocab_size'] = len(word_vocab)
69
70  MAX_SEQUENCE_LENGTH = 174
71
72  train_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
73  print('Shape of train data: ', train_inputs.shape)
74
75  train_labels = np.array(train_data['sentiment'])
76  print('Shape of label tensor:', train_labels.shape)
```

전체 데이터가 인덱스로 구성되었을 텐데 각 인덱스가 어떤 단어를 의미하는지를 파악함 (단어 사전 필요)

- [라인61] 단어 사전 → word_vocab

Step 2: Preprocessing (Part 3)

```
56  tokenizer = Tokenizer()
57  tokenizer.fit_on_texts(clean_train_reviews)
58  text_sequences = tokenizer.texts_to_sequences(clean_train_reviews)
59  print(text_sequences[0])
60
61  word_vocab = tokenizer.word_index
62  print(word_vocab)
63
64  print("전체 단어 개수: ", len(word_vocab) + 1)
65
66  data_configs = {}
67  data_configs['vocab'] = word_vocab
68  data_configs['vocab_size'] = len(word_vocab)
69
70  MAX_SEQUENCE_LENGTH = 174
71
72  train_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
73  print('Shape of train data: ', train_inputs.shape)
74
75  train_labels = np.array(train_data['sentiment'])
76  print('Shape of label tensor:', train_labels.shape)
```

word_vocab의 기본적인 값을 저장해 둠

Step 2: Preprocessing (Part 3)

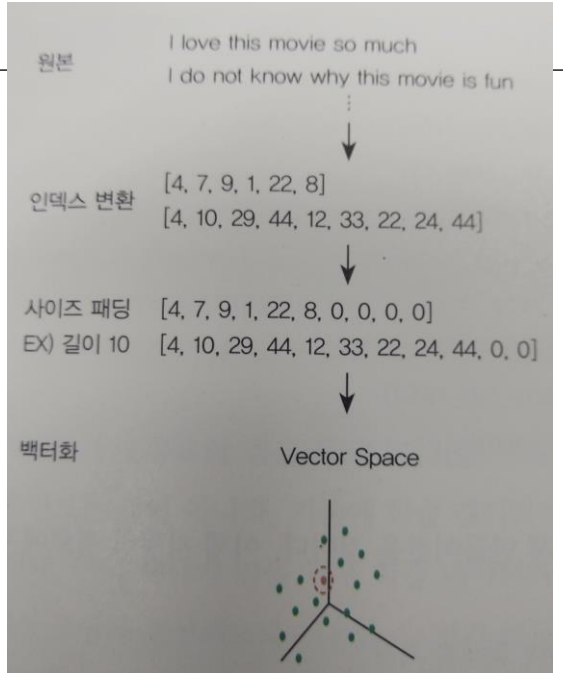
```
56  tokenizer = Tokenizer()
57  tokenizer.fit_on_texts(clean_train_reviews)
58  text_sequences = tokenizer.texts_to_sequences(clean_train_reviews)
59  print(text_sequences[0])
60
61  word_vocab = tokenizer.word_index
62  print(word_vocab)
63
64  print("전체 단어 개수: ", len(word_vocab) + 1)
65
66  data_configs = {}
67  data_configs['vocab'] = word_vocab
68  data_configs['vocab_size'] = len(word_vocab)
69
70  MAX_SEQUENCE_LENGTH = 174
71
72  train_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
73  print('Shape of train data: ', train_inputs.shape)
74
75  train_labels = np.array(train_data['sentiment'])
76  print('Shape of label tensor:', train_labels.shape)
```

- 현재 각 데이터는 길이가 서로 다른데 이 길이를 하나로 통일해야 이후 모델에 바로 적용할 수 있음
  따라서 특정 길이를 최대 길이로 정하고([라인70]에 의하여 174), 더 긴 데이터의 경우에는 뒷부분을 자르고 짧은 데이터의 경우에는 0값을 패딩하는 작업을 수행함
- [라인73] 25000개의 데이터의 길이가 고정되었으므로 share로 확인
  Shape of train data: (25000, 174)

Shape of label tensor: (25000,)

Step 2: Preprocessing (Part 3)

```
56  tokenizer = Tokenizer()
57  tokenizer.fit_on_texts(clean_train_reviews)
58  text_sequences = tokenizer.texts_to_sequences(clean_train_reviews)
59  print(text_sequences[0])
60
61  word_vocab = tokenizer.word_index
62  print(word_vocab)
63
64  print("전체 단어 개수: ", len(word_vocab) + 1)
65
66  data_configs = {}
67  data_configs['vocab'] = word_vocab
68  data_configs['vocab_size'] = len(word_vocab)
69
70  MAX_SEQUENCE_LENGTH = 174
71
72  train_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
73  print('Shape of train data: ', train_inputs.shape)
74
75  train_labels = np.array(train_data['sentiment'])
76  print('Shape of label tensor:', train_labels.shape)
```



Step 2: Preprocessing (Part 4)

```
78  TRAIN_INPUT_DATA = 'train_input.npy'
79  TRAIN_LABEL_DATA = 'train_label.npy'
80  TRAIN_CLEAN_DATA = 'train_clean.csv'
81  DATA_CONFIGS = 'data_configs.json'
82
83  import os
84  # 저장하는 디렉토리가 존재하지 않으면 생성
85  if not os.path.exists(DATA_IN_PATH):
86      os.makedirs(DATA_IN_PATH)
87
88  # 전처리 된 데이터를 넘파이 형태로 저장
89  np.save(open(DATA_IN_PATH + TRAIN_INPUT_DATA, 'wb'), train_inputs)
90  np.save(open(DATA_IN_PATH + TRAIN_LABEL_DATA, 'wb'), train_labels)
91
92  # 정제된 텍스트를 csv 형태로 저장
93  clean_train_df.to_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA, index = False)
94
95  # 데이터 사전을 json 형태로 저장
96  json.dump(data_configs, open(DATA_IN_PATH + DATA_CONFIGS, 'w'), ensure_ascii=False)
```

- 텍스트 데이터의 경우 CSV파일로 저장
- 벡터화한 데이터와 정답 라벨의 경우 넘파이 파일로 저장
- 마지막 데이터 정보의 경우 딕셔너리 형태이므로 JSON 파일로 저장

Preprocessing is done!

Step 2: Preprocessing (Part 5)

```
test_data = pd.read_csv(DATA_IN_PATH + "testData.tsv", header=0, delimiter="\t", quoting=3)

clean_test_reviews = []
for review in test_data['review']:
    clean_test_reviews.append(preprocessing(review, remove_stopwords = True))

clean_test_df = pd.DataFrame({'review': clean_test_reviews, 'id': test_data['id']})
test_id = np.array(test_data['id'])

text_sequences = tokenizer.texts_to_sequences(clean_test_reviews)
test_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')

TEST_INPUT_DATA = 'test_input.npy'
TEST_CLEAN_DATA = 'test_clean.csv'
TEST_ID_DATA = 'test_id.npy'

np.save(open(DATA_IN_PATH + TEST_INPUT_DATA, 'wb'), test_inputs)
np.save(open(DATA_IN_PATH + TEST_ID_DATA, 'wb'), test_id)
clean_test_df.to_csv(DATA_IN_PATH + TEST_CLEAN_DATA, index = False)
```

Step 2: Preprocessing (Part 5)

```
test_data = pd.read_csv(DATA_IN_PATH + "testData.tsv", header=0, delimiter="\t", quoting=3)

clean_test_reviews = []
for review in test_data['review']:
    clean_test_reviews.append(preprocessing(review, remove_stopwords = True))

clean_test_df = pd.DataFrame({'review': clean_test_reviews, 'id': test_data['id']})
test_id = np.array(test_data['id'])

text_sequences = tokenizer.texts_to_sequences(clean_test_reviews)
test_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')

TEST_INPUT_DATA = 'test_input.npy'
TEST_CLEAN_DATA = 'test_clean.csv'
TEST_ID_DATA = 'test_id.npy'

np.save(open(DATA_IN_PATH + TEST_INPUT_DATA, '
np.save(open(DATA_IN_PATH + TEST_ID_DATA, 'wb'
clean_test_df.to_csv(DATA_IN_PATH + TEST_CLEAN
```

- 평가 데이터에 대해서도 동일한 과정으로 전처리 진행
- 다른점: 평가 데이터의 경우 라벨이 없기 때문에 라벨은 따로 저장하지 않아도 되고 데이터 정보인 단어사전과 단어개수에 대한 정보도 학습데이터의 것을 사용하므로 저장하지 않아도 됨
- 추가로 평가 데이터에 대해 저장해야 하는 값은 각 리뷰 데이터에 대해 리뷰어에 대한 id값임
- 토크나이저를 통해 인덱스 벡터로 만들 때 토크나이징 객체로 새롭게 만드는 것이 아니라, 기존에 학습 데이터에 적용한 토크나이저 객체를 사용해야 함
  → (단어들의 인덱스를 맞추기 위하여)

3단계: 알고리즘 모델링

- 1단계: 데이터를 불러옴
- 2단계: 데이터 전처리 및 데이터 분석
- **3단계: 알고리즘 모델링**
  - **회귀모델: TF-IDF**

Step 3: TF-IDF Computation

```
import os
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

DATA_IN_PATH = './data_in/'
DATA_OUT_PATH = './data_out/'
TRAIN_CLEAN_DATA = 'train_clean.csv'
RANDOM_SEED = 42
TEST_SPLIT = 0.2

train_data = pd.read_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA)
reviews = list(train_data['review'])
sentiments = list(train_data['sentiment'])
vectorizer = TfidfVectorizer(min_df = 0.0, analyzer="char", sublinear_tf=True, ngram_range=(1,3), max_features=5000)
X = vectorizer.fit_transform(reviews)
y = np.array(sentiments)

X_train, X_eval, y_train, y_eval = train_test_split(X, y, test_size=TEST_SPLIT, random_state=RANDOM_SEED)

lgs = LogisticRegression(class_weight='balanced')
lgs.fit(X_train, y_train)
predicted = lgs.predict(X_eval)
print("Accuracy: %f" % lgs.score(X_eval, y_eval))
```

Step 3: TF-IDF Computation

```
1  import os
2  import pandas as pd
3  import numpy as np
4  from sklearn.feature_extraction.text import TfidfVectorizer
5  from sklearn.model_selection import train_test_split
6  from sklearn.linear_model import LogisticRegression
7
8  DATA_IN_PATH = './data_in/'
9  DATA_OUT_PATH = './data_out/'
10 TRAIN_CLEAN_DATA = 'train_clean.csv'
11 RANDOM_SEED = 42
12 TEST_SPLIT = 0.2
13
14 train_data = pd.read_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA)
15 reviews = list(train_data['review'])
16 sentiments = list(train_data['sentiment'])
17 vectorizer = TfidfVectorizer(min_df = 0.0, analyzer="char", sublinear_tf=True, ngram_range=(1,3), max_features=5000)
18 X = vectorizer.fit_transform(reviews)
19 y = np.array(sentiments)
20
21 X_train, X_eval, y_train, y_eval = train_test_split(X, y, test_size=TEST_SPLIT, random_state=RANDOM_SEED)
22
23 lgs = LogisticRegression(class_weight='balanced')
24 lgs.fit(X_train, y_train)
25 predicted = lgs.predict(X_eval)
26 print("Accuracy: %f" % lgs.score(X_eval, y_eval))
```

[라인17] TF-IDF값으로 벡터화
- min_df: 설정한 값보다 특정 토큰의 df값이 더 적게 나오면 제거
- analyzer: 분석하기 위한 기준 단위(word나 char)
- sublinear_tf: 문서의 단어 빈도 수에 대한 스무딩 여부를 설정
- ngram_range: 빈도의 기본 단위를 어느 범위의 n-gram으로 설정하는지 정의
- max_features: 각 벡터의 최대 길이

[라인18]
- fit_transform: 전체 문장에 대한 특징벡터 X를 생성

Step 3: TF-IDF Computation

```
1  import os
2  import pandas as pd
3  import numpy as np
4  from sklearn.feature_extraction.text import TfidfVectorizer
5  from sklearn.model_selection import train_test_split
6  from sklearn.linear_model import LogisticRegression
7
8  DATA_IN_PATH = './data_in/'
9  DATA_OUT_PATH = './data_out/'
10 TRAIN_CLEAN_DATA = 'train_clean.csv'
11 RANDOM_SEED = 42
12 TEST_SPLIT = 0.2
13
14 train_data = pd.read_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA)
15 reviews = list(train_data['review'])
16 sentiments = list(train_data['sentiment'])
17 vectorizer = TfidfVectorizer(min_df = 0.0, analyzer="char", sublinear_tf=True, ngram_range=(1,3), max_features=5000)
18 X = vectorizer.fit_transform(reviews)
19 y = np.array(sentiments)
20
21 X_train, X_eval, y_train, y_eval = train_test_split(X, y, test_size=TEST_SPLIT, random_state=RANDOM_SEED)
22
23 lgs = LogisticRegression(class_weight='balanced')
24 lgs.fit(X_train, y_train)
25 predicted = lgs.predict(X_eval)
26 print("Accuracy: %f" % lgs.score(X_eval, y_eval))
```

[라인21] 학습데이터 생성
- X_train → (20000, 5000)
- X_eval → (5000, 5000)
- y_train → (20000,)
- y_eval → (5000,)

Step 3: TF-IDF Computation

```
import os
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

DATA_IN_PATH = './data_in/'
DATA_OUT_PATH = './data_out/'
TRAIN_CLEAN_DATA = 'train_clean.csv'
RANDOM_SEED = 42
TEST_SPLIT = 0.2

train_data = pd.read_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA)
reviews = list(train_data['review'])
sentiments = list(train_data['sentiment'])
vectorizer = TfidfVectorizer(min_df = 0.0, analyzer="char", sublinear_tf=True, ngram_range=(1,3), max_features=5000)
X = vectorizer.fit_transform(reviews)
y = np.array(sentiments)

X_train, X_eval, y_train, y_eval = train_test_split(X, y,

lgs = LogisticRegression(class_weight='balanced')
lgs.fit(X_train, y_train)
predicted = lgs.predict(X_eval)
print("Accuracy: %f" % lgs.score(X_eval, y_eval))
```

[라인23] LogisticRegression 모델
- balanced: 각 라벨에 대해 균형있게 학습할 수 있게 한 것

정확도: 0.859600