텍스트 마이닝과 데이터 마이닝

Part 04. 감정 분석

정 정 민

Chapter 10. 감정 분석 모델 알고리즘

1. 텍스트 데이터 전처리
2. 감정 분석 실습

텍스트 데이터 전처리

전처리 과정

- 텍스트 데이터를 이용하는 분석을 위해서는 분석에 사용할 수 있는 형태로 전처리 필요

- 일반적으로 아래의 과정을 진행
  - **Tokenize**
    - 원문 글을 분석에 사용할 기본 개념 단위로 분리하는 과정
    - 이 단위가 글을 분석하는 과정의 가장 작은 의미 단위가 됨
  - **Stop Words 제거**
    - 개념 단위로 나뉜 개체에서 의미가 없는 개체를 제거하는 과정
    - 데이터의 복잡도를 줄이고 분석 정확도를 올리는 역할을 함
  - **Stemming**
    - 나뉜 개체에서 접두사나 접미사를 제거해 기본 의미를 갖도록 함

Tokenize

- Token이란,
  - 분석의 기본 단위가 되는 개체를 의미함
  - 쉬운 예로, 띄어쓰기로 글을 나누면 단어가 Token이 됨
  - 문장 자체 혹은 문단 자체가 Token이 되기도 함
  - Token의 정의는 문제에 따라 사용자가 정의하기 나름!

- 원래 글을 Token으로 나누는 과정을 Tokenize라고 함

"문장을 토큰으로 나누는 과정을 토크나이즈라고 해요"

문장을,
토큰으로,
나누는,
과정을,
토크나이즈라고,
해요

Stop Words 제거

- 빈번히 많이 사용되지만 의미는 없어 분석에 도움이 되지 않는 Token을 제거하는 과정

- 가령, 단어 단위로 Token을 설정했을 때,
  - 한글에서는, '그리고', '아', '내가' 등등은 큰 의미가 없음
  - 영어의 경우, 'The', 'a', 'and' 등도 큰 의미가 없음
  - 하지만 이런 단어는 매우 매우 많이 쓰임

- 텍스트 마이닝 분석의 중요한 철학 중 하나는
  - "많이 나온 Token은 중요한 역할을 한다"
  - Stop words는 이 철학에 배반하는 Token

- 일반적으로 stop words는 미리 사전 정의하고
- 해당 단어가 나오면 제거하는 형태

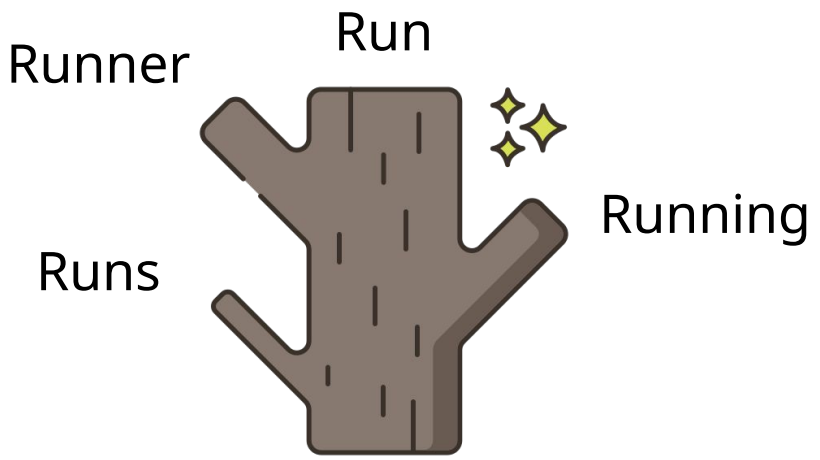
Stemming

- 단어 기반의 Token을 사용하는 경우
- 특정 단어를 그것의 기본 형태(어간)으로 축소하는 과정

- 접두사나 접미사를 제거하여 단어의 기본 줄기(stem)를 찾는 과정

- 예를 들어,
  - Running → Run (‘-ing’ 제거)
  - Runner → Run (‘-er’ 제거)

- 빠르고 간단하게 처리할 수 있지만,
- 문맥을 고려하지 않아 잘못된 결과를 반환할 수 있음
  - University와 Universe를 같은 줄기(stem)로 처리할 가능성이 있음



[참고] Lemmatization

- 단어를 그 의미론적 기본 형태 (Lemma)로 변환하는 과정

- 특정 단어의 품사와 문맥을 고려해
- 정확한 기본 형태를 찾아냄

- 예를 들어,
  - ‘are’ , ‘is’, ‘am’ 모두 ‘be’로 변경
  - ‘Running’, ‘Runner’ 모두 ‘Run’으로 변경하고 더불어
  - ‘Ran’도 ‘Run’으로 변경

- Stemming에 비해 복잡하고 시간이 오래 걸리지만 정확도가 좋음

딥러닝을 기반으로 하는 전처리

- 딥러닝을 활용한 언어 처리 방법은
- 전통적인 방식에서 중요하게 생각했던 처리 과정을 일부 변형하여 사용

- **Tokenize**
  - 전통적으로 단어 단위의 Token을 선택했지만
  - 단어를 더 쪼개는 Subword 방식의 Tokenize를 사용
  - 기반 모델이 선택한 Tokenize 방식을 사용

- **Stop Words**
  - 딥러닝 모델은 종종 문맥 속에서 단어의 중요성을 자동으로 파악
  - 명시적으로 Stop Words를 제거할 필요가 적음

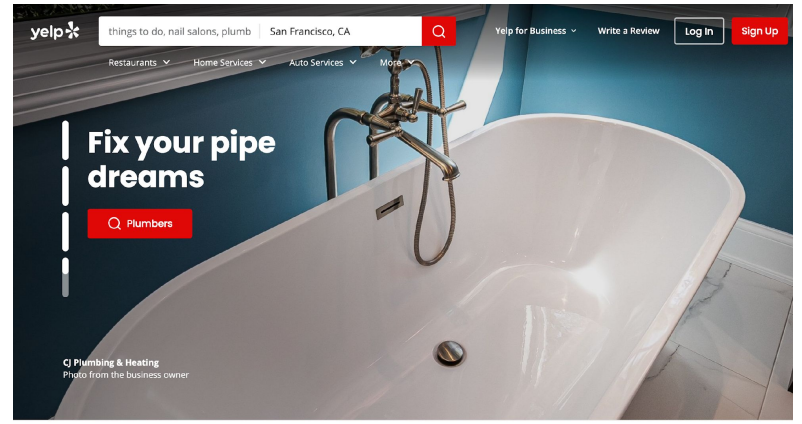
- **Stemming & Lemmatization**
  - 단어 형태의 표준화는 과정 자체를
  - 딥러닝 모델이 스스로 학습하도록 처리

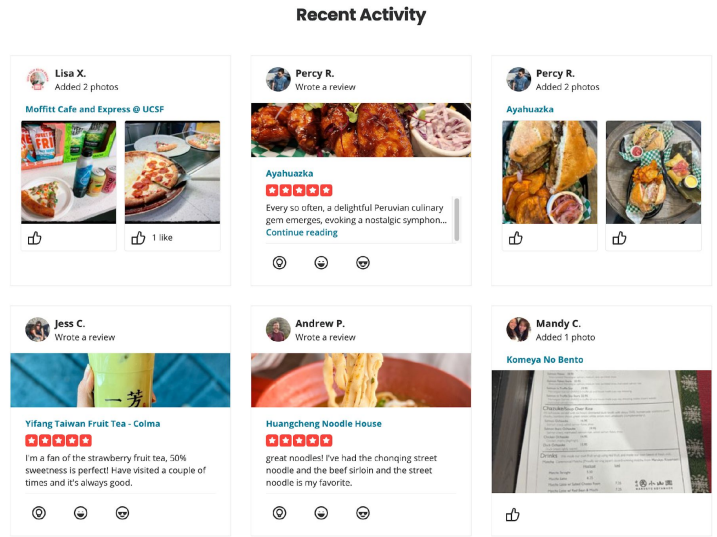
감정 분석 실습

강의 실습 코드 : 링크

Yelp 데이터셋

- Yelp란, 사용자들이 다양한 로컬 음식점에 대한 리뷰를 공유하는 플랫폼
- 1000개의 리뷰를 모아 학습용 데이터로 구축한 Kaggle 데이터 활용 (링크)
  - 다운로드 받아주세요!
  - Amazon_cells 데이터와 imdb 데이터는 사용하지 않음
- 음식점과 음식에 대한 텍스트 리뷰와 그 만족도 (1: 만족 / 0: 불만족)





yelp_labelled.txt — 텍스트 편집기(으)로 열기

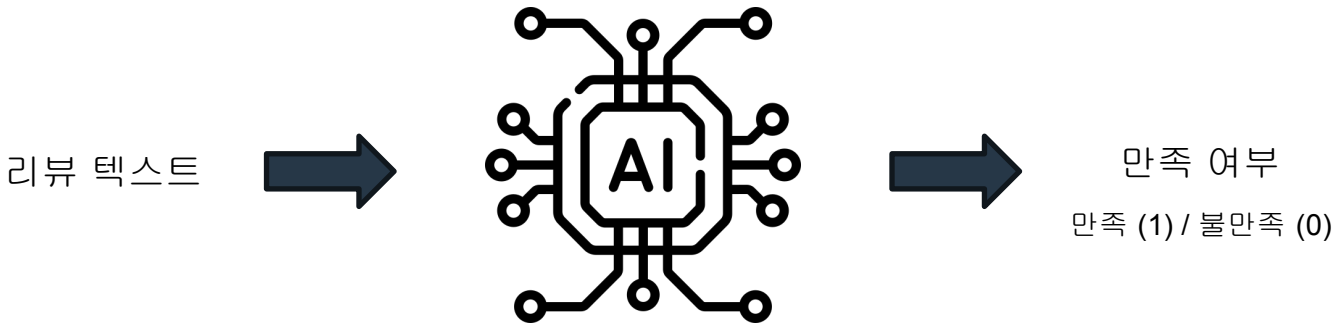
```
Wow... Loved this place.        1
Crust is not good.      0
Not tasty and the texture was just nasty.       0
Stopped by during the late May bank holiday off Rick Steve recommendation and loved it. 1
The selection on the menu was great and so were the prices.     1
Now I am getting angry and I want my damn pho.  0
Honeslty it didn't taste THAT fresh.)   0
The potatoes were like rubber and you could tell they had been made up ahead of time being kept under a warmer. 0
The fries were great too.       1
A great touch.  1
Service was very prompt.        1
Would not go back.      0
The cashier had no care what so ever on what I had to say it still ended up being wayyy overpriced.     0
I tried the Cape Cod ravoli, chicken,with cranberry...mmmm!     1
I was disgusted because I was pretty sure that was human hair.  0
I was shocked because no signs indicate cash only.      0
Highly recommended.     1
Waitress was a little slow in service.  0
This place is not worth your time, let alone Vegas.     0
did not like at all.    0
The Burrittos Blah!     0
The food, amazing.      1
Service is also cute.   1
I could care less... The interior is just beautiful.    1
So they performed.      1
That's right....the red velvet cake.....ohhh this stuff is so good.     1
- They never brought a salad we asked for.      0
This hole in the wall has great Mexican street tacos, and friendly staff.       1
Took an hour to get our food only 4 tables in restaurant my food was Luke warm, Our sever was running around like he
```

https://www.yelp.com/

문제 정의

- 풀어야 하는 문제
  - 사용자의 텍스트 리뷰를 바탕으로 그들의 감정 상태를 예측

- 입력과 출력
  - 입력 : 텍스트 리뷰 문장(들)
  - 출력 : 감정 상태
    - 1 : 만족
    - 0 : 불만족



진행 과정

- 이번 실습은 아래 과정을 포함

- **전처리**
  - 단어 기반 Tokenize
  - Stop Words 제거 + 기타 전처리
  - Stemming은 사용 x (아래 GloVe는 학습 과정에서 Stemming 한 데이터를 사용하지 x)

- **모델 적용 및 학습**
  - GloVe를 통해 단어 임베딩
  - 문장에 존재하는 임베딩 값들 기반으로 문장의 임베딩을 생성
    - TF-IDF 가중치를 추가한 단어들의 임베딩 값을 평균!
  - 이진 분류(긍정/부정) 문제로 Logistic Regression 적용

- **결과 확인**
  - 예측 결과를 프린트

전처리

- GloVe를 사용하기 위한 전처리 과정을 진행

- 단어를 기반으로 Tokenize를 진행하고
- Stop Words를 제거
  - Stop Words는 내장 패키지에서 다운로드 후 사용

- GloVe는 학습 당시 단어의 정확한 형태와 문맥을 기반으로 학습됨
- 따라서 Stemming 과정은 진행하지 않음
  - 이를 진행하면 원래 의미와 문맥 정보의 손실 가능성

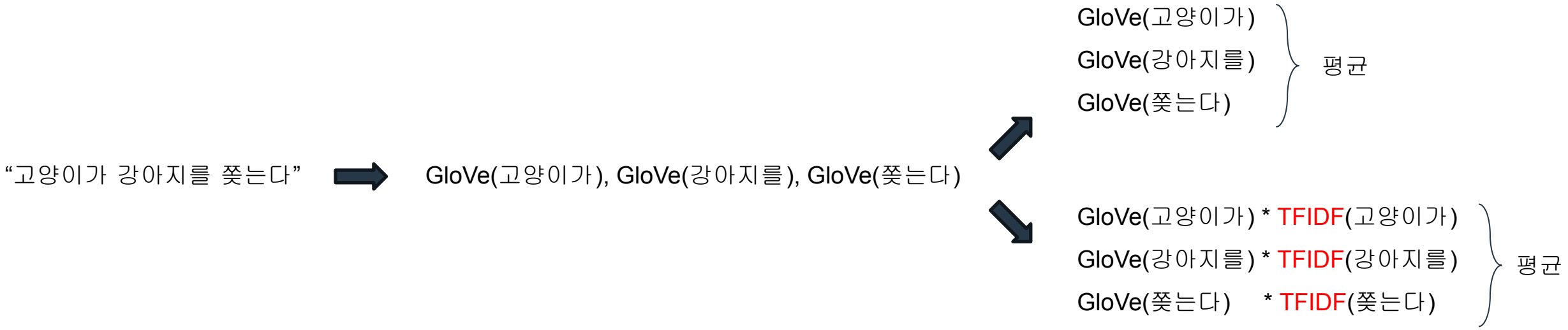
- TF-IDF 과정에서도 서로 어간 추출 없이 원본 단어의 중요도를 판단

```
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def preprocessing(text) :
    text = text.lower()
    text = re.sub(r'\W', ' ', text)
    text = text.split()
    text = [t for t in text if t not in stop_words]
    return text
```

문장 임베딩 생성

- GloVe로 단어들의 임베딩 값을 생성
- 이 값들을 단순히 평균을 내면 서로 다른 의미의 문장임에도 같은 결과를 내보낼 수 있음

- 따라서, 각 단어마다 중요도를 나타내는 수치값을 적용한 임베딩을 활용
  - 이 수치를 TF-IDF라고 함



TF-IDF (1/2)

- TF-IDF는
  - 문서 안에서 특정 단어의 중요도를 평가하는 통계적인 방법
  - TF 파트와 IDF 파트의 곱으로 계산

- **TF (단어의 빈도, Term Frequency)**
  - 특정 단어가 문서 내에 얼마나 자주 등장하는지
  - 문서 내 전체 단어 중 해당 단어가 얼마나 출현했는지 빈도

$$TF(t,d) = \frac{\text{문서 } d\text{안에서 단어 } t\text{의 출현 횟수}}{\text{문서 } d\text{의 전체 단어 수}}$$

TF-IDF (2/2)

- **IDF (역 문서 빈도, Inverse Document Frequency, IDF)**
  - 특정 단어가 얼마나 여러 문서에서 등장하는지
  - 모든 문서에 자주 등장한 단어 : 중요도가 낮음
  - 특정 문서에서만 자주 등장 : 중요도가 높음

$$IDF(t, D) = \log\left(\frac{\text{전체 문서 } D\text{의 개수}}{\text{단어 } t\text{를 포함한 문서의 수}}\right)$$

- D : 전체 문서 집합을 의미

- TF-IDF 계산은 TF와 IDF 값을 곱해서 산출
- 이 값은 문서 d 안에서 단어 t가 갖는 상대적 중요도
  - 높은 TF-IDF 값을 갖는 단어는 해당 문서에서 더 많은 정보를 제공
  - 전체 문서 집합 D에서 보다 의미 있는 특징을 갖고 있음

문장 임베딩 코드

- TF-IDF가 계산된 tfidf_matrix를 활용
  - 행 : 문서
  - 열 : 전체 단어를 포괄하는 vector에 해당 문서에 들어있는 단어들에 값이 존재

- GloVe와 TF-IDF 계산에서 사용될 수 있는 단어들을 선택해야 함

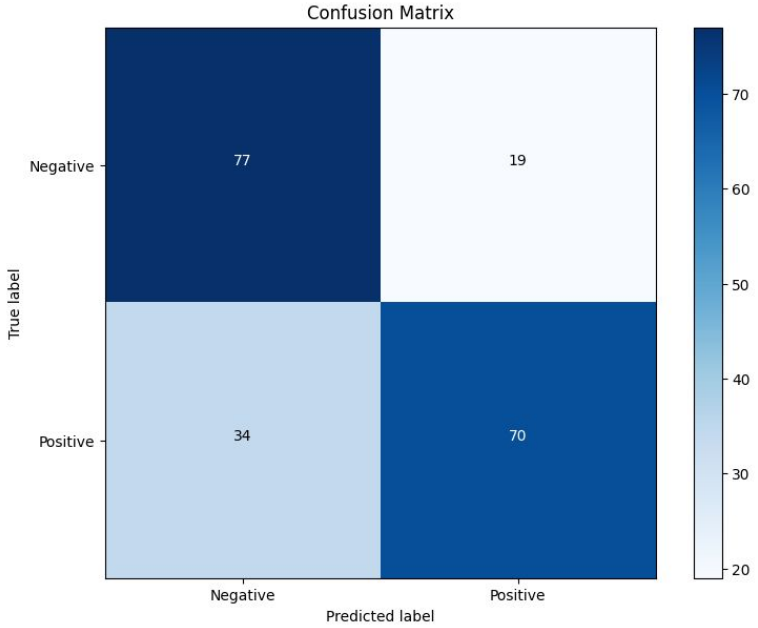
```
def sentence_embedding(doc, doc_idx):
    embeddings = []
    for word in doc
        if word in glove and word in tfidf_feature_names:
            word_idx = np.where(tfidf_feature_names==word)[0][0]
            tfidf_weight = tfidf_matrix[doc_idx, word_idx]

            embeddings.append(glove[word] * tfidf_weight)
    return np.mean(embeddings, axis=0) if embeddings else np.zeros(100)
```

모델 학습 및 평가

- Logistic Regression 모델을 돌려서 학습 및 평가를 진행

- 학습 과정에서 입력 데이터는 Sentence Embedding인 100 크기의 Vector
- 이를 학습에 사용할 수 있는 형태[1000(전체 데이터), 100(vector 크기)]로 변환

- 분류 문제로 평가 진행



Accuracy: 0.73

Precision: 0.79

Recall: 0.67

F1-Score: 0.73

실제 있을 법한 가상 데이터 생성 후 결과 도출

- 머신 러닝 모델을 생성하고 평가로 끝내는 것이 아니라,
- 실제 고객이 작성할 법한 예시 문장을 활용해 추론 진행

- 진행한 전처리를 동일하게 적용
  - 앞서 설명한 전처리
  - TF-IDF 결과로 가중된 GloVe 임베딩 벡터

- 이후, 모델 입력으로 출력 결과 표시

```
# 예제 문장 전처리
preprocessed_examples = [preprocessing(text) for text in examples]

# TF-IDF 임베딩과 GloVe 임베딩을 결합하여 문장 임베딩 생성
example_sentence_embs = []
for doc_idx, doc in enumerate(preprocessed_examples):
    example_sentence_embs.append(sentence_embedding(doc, doc_idx))

# 모델을 이용해 감정 분석 수행
example_sentence_embs = np.stack(example_sentence_embs)
predictions = model.predict(example_sentence_embs)

# 결과 출력
for idx, (text, pred) in enumerate(zip(examples, predictions)):
    origin_sent = '긍정적' if idx % 2 == 0 else '부정적'
    pred_sent = '긍정적' if pred == 1 else '부정적'

    print(f"문장: {text} \n원래 감정 : {origin_sent} / 예측 : {pred_sent}", end='\n\n')
```

E.O.D