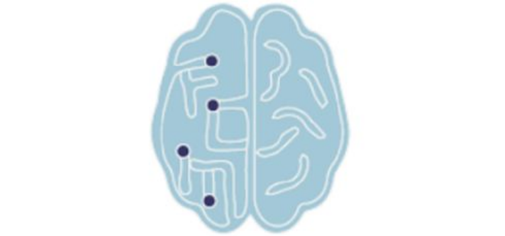
김주원, 박은영, 장원준,허연후

2023-11-22

FOM NLP MiniProject



FOM

Focus On Data Mining

INDEX

01 NLP pipeline

NLP

1. 텍스트 전처리 (Text Preporcessing)
   a. 토큰화 (영어), 형태소 분석 (한국어)
      i. 형태소 분석도 토큰화의 한 종류 (한국어의 특징으로 인해 한글은 형태소 분석으로 토큰화)
   b. 불용어 처리
      i. and, the, i 등을 제거, 은(는), 이(가) 등을 제거
   c. 정제
      i. 영어의 경우 대소문자 통일, 동음이의어 통일화 등

2. 피처 엔지니어링 (Feature Engineering)
   a. 단어 벡터화
      i. Word2Vec, GloVe, FastText (KoWord2Vec,KoBERT, KoFastText)
         단어 간의 의미적, 문법적 관계를 포착하는 데 사용 -> 단어 수준의 의미를 포착
   b. 단어 문서화
      i. TF-IDF
         문서 내 단어들의 특성을 종합하여 전체 문서를 벡터화 -> 문서 전체의 의미와 특성 보존

3. 모델링 (Modeling)
   a. 딥러닝 및 머신러닝을 활용
      i. Keras, Pytorch, LogisticRegression,...

02 nsmc preprocessing

NLP

- 널 데이터 및 중복 데이터 제거
  - df.dropna()
  - df.drop_diplaicates()
    - dropna, drop_duplocates 실행 시 특정 column을 기준으로 실행 시 subset = [‘column_name’]으로 지정 후 실행

- 널 데이터 (결측치 제거)

```
# 결측치 제거

naver_train = naver_train.dropna()
naver_test = naver_test.dropna()
print('학습 데이터셋 수 : {}'.format(naver_train.shape[0]))
print('테스트 데이터셋 수 : {}'.format(naver_test.shape[0]))
```

```
학습 데이터셋 수 : 149995
테스트 데이터셋 수 : 49997
```

-> 기존의 150000개의 train은 5개의 결측치 drop
기존의 50000개의 test는 3개의 결측치 drop

- 중복 데이터 제거

```
# 중복된 데이터 제거

naver_train=naver_train.drop_duplicates(subset=['document'])
naver_test=naver_test.drop_duplicates(subset=['document'])
print('학습 데이터셋 수 : {}'.format(naver_train.shape[0]))
print('테스트 데이터셋 수 : {}'.format(naver_test.shape[0]))
```

```
학습 데이터셋 수 : 146182
테스트 데이터셋 수 : 49157
```

-> 결측치를 제거한 149995개의 train은 3813개의 중복 데이터 drop
결측치를 제거한 49997개의 test는 840개의 중복 데이터 drop

02 | nsmc preprocessing

NLP

- 한글과 숫자, 공백, 일부 특수문자를 제외하고 다른 값들은 모두 띄어쓰기로 대체
  - 공백 또는 특수문자로만 이루어진 데이터 drop
    -> 앞의 과정에서 띄어쓰기로 대체하였으니 공백 데이터도 drop
  - 잘 drop되었는지 확인

- 띄어쓰기로 대체

```
# 한글과 숫자, 공백, 그리고 일부 특수문자를 제외한 모든 특수문자를 띄어쓰기로 변환
pattern = "[^0-9ㄱ-ㅎㅏ-ㅣ가-힣.,?! ]"
naver_train['document'] = naver_train['document'].replace(pattern, " ", regex=True)
naver_test['document'] = naver_test['document'].replace(pattern, " ", regex=True)
```

- 공백, 특수문자로만 이루어진 데이터 제거

```
# 공백 또는 특수문자로만 구성된 데이터 제거
pattern = '^[.,?! ]+$'
naver_train = naver_train.loc[naver_train['document'].str.match(pattern) == False]
naver_test = naver_test.loc[naver_test['document'].str.match(pattern) == False]
```

- train, test에 drop이 잘 됐는지 확인

```
Empty DataFrame
Columns: [document, label]
Index: []
Empty DataFrame
Columns: [document, label]
Index: []
```

02 nsmc preprocessing

Word Tokenizing

- OKT
  - Open Korean Text
  - 한글의 형태소를 기반으로 토큰화
- Kkma
  - Korean KoNLP Module for Python
  - 한글의 형태소를 기반으로 토큰화 (Okt와 같이 형태소로 토큰화하지만 문법 규칙이 달라 토큰화 방식이 다름)
  - EX) “ 자연어 처리는 매우 재미있지만 너무 어렵다. ”
    Okt 토큰화 => 자연어 처리 는 매우 재미있지만 너무 어렵다 .
    Kkma 토큰화 => 자연어 처리 는 매우 재미있 지만 너무 어렵 다 .

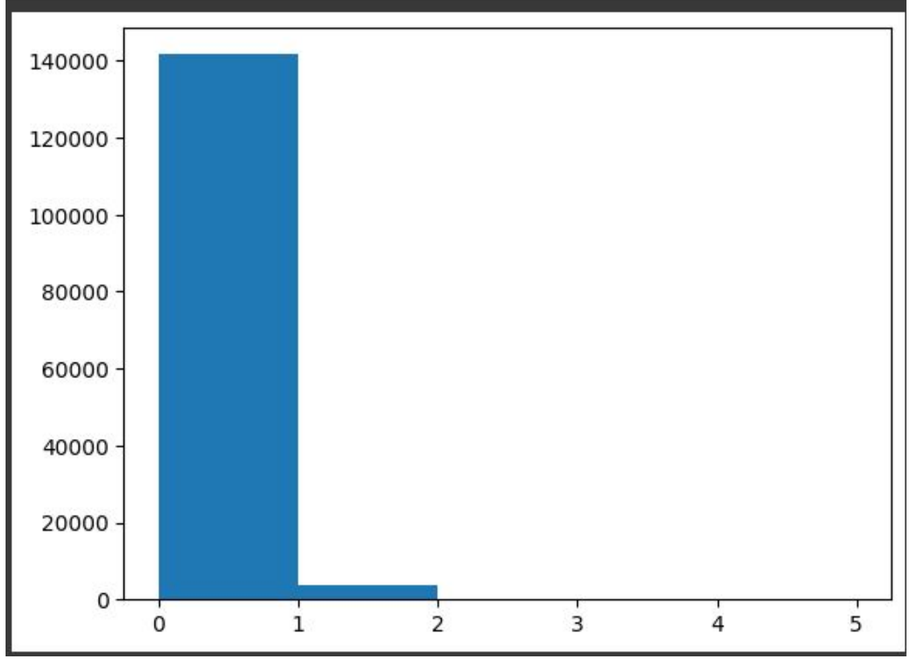
	id	document_token	label
0	9976970	아 더빙 진짜 짜증나네요 목소리	0
1	3819312	흠 포스터 보고 초딩 영화 줄 오버 연기 조차 가볍지 않구나	1
2	10265843	너 무재 밓었 다그 래서 보는것을 추천 한 다	0
3	9045019	교도소 이야기 구먼 솔직히 재미 는 없다 평점 조정	0
4	6483659	사이 몬페 그 의 익살스런 연기 가 돋보였던 영화 스파이더맨 에서 늙어 보이기만...	1

02 nsmc preprocessing

Plot

- 리뷰에 '감독' 이라는 단어가 들어간 문장의 수

```
data = np.asarray(bag[:, count.vocabulary_['감독']].todense()).reshape(-1)
plt.hist(data, bins=range(max(data)+2))
plt.show()
```



- x축 : 등장한 횟수
- y축 : i 번 등장한 리뷰의 수
  - (0,140000) : 리뷰에 '감독'이라는 단어가 0번 등장한 리뷰의 수가 140000개

02 nsmc preprocessing

EDA

- 가장 길고 가장 짧은 리뷰 파악

```
df_filtered['word_count'] = df_filtered['document_token'].apply(lambda x: len(x.split()))
max_word_row = df_filtered.loc[df_filtered['word_count'].idxmax()]
min_word_row = df_filtered.loc[df_filtered['word_count'].idxmin()]

print("길이가 가장 긴 문장:", max_word_row['document_token'])
print("가장 긴 문장의 길이:", max_word_row['word_count'])

print("길이가 가장 짧은 문장:", min_word_row['document_token'])
print("가장 짧은 문장의 길이:", min_word_row['word_count'])
```

- **길이가 가장 긴 문장:** 내 평점 은 쓰레기 임 진심 평점 줬 더 만일분 만에 삭제 댓 음 티비 서 내 돈 주 고만 원 내 고 제 돈 주고 내 평썻더 만 삭제 답 없다 같이 본 친구 폰 으로 다시 적음 내 가 영화 를 사랑 하 지만 영화 보면 서너 무 화 가나 서 처음 써 봄 정말 핫 바리 영화 다 알바 들 수고 한다 실시 간평 지운다고 피해자 일인 돈 아깝다 평점 싫으면 환불 해 라
- **가장 긴 문장의 길이:** 74
- **길이가 가장 짧은 문장**: 백봉기 언제 나오나요
- **가장 짧은 문장의 길이:** 3

02 nsmc preprocessing

stopwords

- 불용어 제거 후 X_train에 저장

```
stopwords = set(['의','가','이','은','들','는','좀','잘','걍','과','도','를','으로','자','에','와','한','하다'])
X_train = []

for sentence in data_train['document_token']:
    words = sentence.split()  # 문장을 공백 기준으로 나눔
    filtered_words = [word for word in words if word not in stopwords]
    X_train.append(filtered_words)

print(X_train[:3])
```

```
[['아', '더빙', '진짜', '짜증나네요', '목소리'], ['흠', '포스터', '보고', '초딩', '영화', '줄', '오버', '연기', '조차', '가볍지', '않구나'],
```

- 토큰화 된 텍스트 데이터중 stopword가 있는 word는 빼고 X_train에 append
  - stopwords = 조사, 짧은 부사 등 의미상으로 중요하지 않은 단어 (한글 기준)

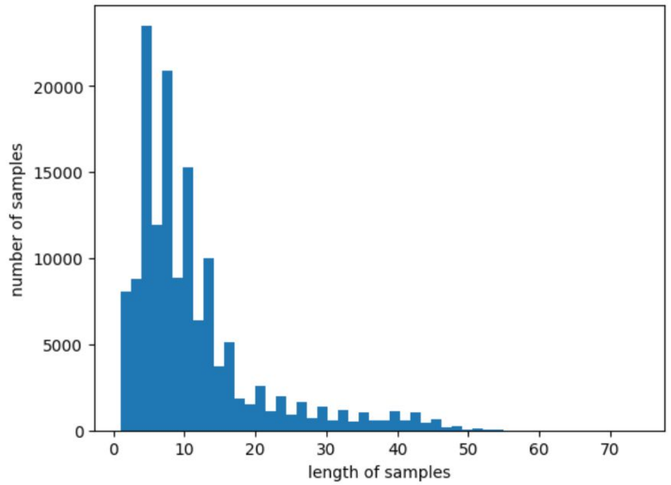
02 nsmc preprocessing

padding

- 패딩
  : 서로 다른 길이의 데이터들을 동일한 길이로 맞춰준다.

```
print('리뷰의 최대 길이 : ',max(len(l) for l in X_train))
print('리뷰의 평균 길이 : ',sum(map(len, X_train))/len(X_train))
plt.hist([len(s) for s in X_train], bins=50)
plt.xlabel('length of samples')
plt.ylabel('number of samples')
plt.show()
```

```
리뷰의 최대 길이 :  74
리뷰의 평균 길이 :  11.241673659737794
```



- 전체 훈련데이터 중 약 96%가 35이하의 길이를 가짐
- 문장 길이가 35이상인 데이터 수가 너무 적어, max_len를 35로 설정하여 데이터셋이 sparse해지는 것을 방지

```
전체 샘플 중 길이가 35 이하인 샘플의 비율: 95.7078592948233
```

```
X_train = pad_sequences(X_train, maxlen = max_len)
X_test = pad_sequences(X_test, maxlen = max_len)
```

03 nsmc Feature Engineering

정수 인코딩

정수 인코딩

- 기계가 텍스트를 숫자로 처리할 수 있도록 정수 인코딩 수행
- tensorflow의 Tokenizer(oov_token = 'OOV')를 수행하여 정수로 인코딩
  - ✔ 이때 OOV는 Out-Of-Vocabulary 라는 뜻으로 단어 집합에 없는 단어는 'OOV'로 토큰화 한다는 뜻
  - ✔ Tokenizer 자체가 정수로 인코딩 하는 인코더

```
print(X_train[:3])
print(X_test[:3])
```

```
[[38, 417, 9, 6826, 624], [880, 418, 33, 574, 2, 185, 1520, 13, 938, 6017, 25658], [348, 2807, 1, 6395, 8238, 11952, 189, 5]]
[[767, 71], [51, 143, 15, 4656, 1409, 11, 973, 838, 37, 754, 3631], [14555, 2430, 76, 309, 94, 90, 361, 133, 239]]
```

- 아 -> 38, 더빙 -> 417, 진짜 -> 9…… 으로 정수 인코딩 됨

```
print(X_train[:3])
print(X_test[:3])
```

```
[['아', '더빙', '진짜', '짜증나네요', '목소리'], ['흠', '포스터', '보고', '초딩', '영화', '줄', '오버', '연기', '조차', '가볍지', '않구나'], ['너', '무재', '밓었', '다그', '래서', '보는것을', '추천', '다']]
[['굳', 'ㅋ'], ['뭐', '야', '평점', '나쁘진', '않지만', '점', '짜', '리', '더', '더욱', '아니잖아'], ['지루하지는', '않은데', '완전', '막장', '임', '돈', '주고', '보기', '에는']]
```

- X_train[:3], X_train[3:]들이 모두 정수 인코딩 되어서 나오는 것을 볼 수 있음

04 nsmc Modeling

LogisticRegression

- TF-IDF를 사용해 단어 문서화
  - model = LogisticRegression(random_state = 0)
  - model.fit(X_train,y_train)

  - y_pred = model.predict(X_test)
    - 위의 코드를 통해 로지스틱 회귀 진행

- 성능 평가 :
  - Accuracy
    - 0.838
    - Recall : 0.83 , 0.85
    - Precision : 0.85, 0.83

```
Accuracy: 0.838

[[20686  3677]
 [ 4230 20310]]

              precision    recall  f1-score   support

           0       0.83      0.85      0.84     24363
           1       0.85      0.83      0.84     24540

    accuracy                           0.84     48903
   macro avg       0.84      0.84      0.84     48903
weighted avg       0.84      0.84      0.84     48903
```

04 nsmc Modeling

Naive Bayes

- 로지스틱 회귀와 같은 조건으로 나이브 베이즈 분류기 실행
- 성능 평가 :
  - Accuracy
    - 0.841
    - Recall : 0.85 , 0.84
    - Precision : 0.83, 0.85

Accuracy: 0.8409709015806801

	precision	recall	f1-score	support
0	0.85	0.83	0.84	24363
1	0.84	0.85	0.84	24540
accuracy			0.84	48903
macro avg	0.84	0.84	0.84	48903
weighted avg	0.84	0.84	0.84	48903

04 nsmc Modeling

Keras

```
model = Sequential()

model.add(Embedding(vocab_size, 4, input_length=max_len))  // input_dim, output_dim, input_length
model.add(GRU(8))

model.add(Dense(1, activation='sigmoid'))

model.save('my_model_3.h5')
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Embedding 레이어
- vocab_size: 전체 단어 개수
- output_dim: 각 단어를 임베딩 하는 데 사용하는 차원의 수, 즉 120차원의 벡터로 표현
- input_length: 입력 시퀀스의 길이

GRU layer
- GRU(32) // 32개의 유닛 사용

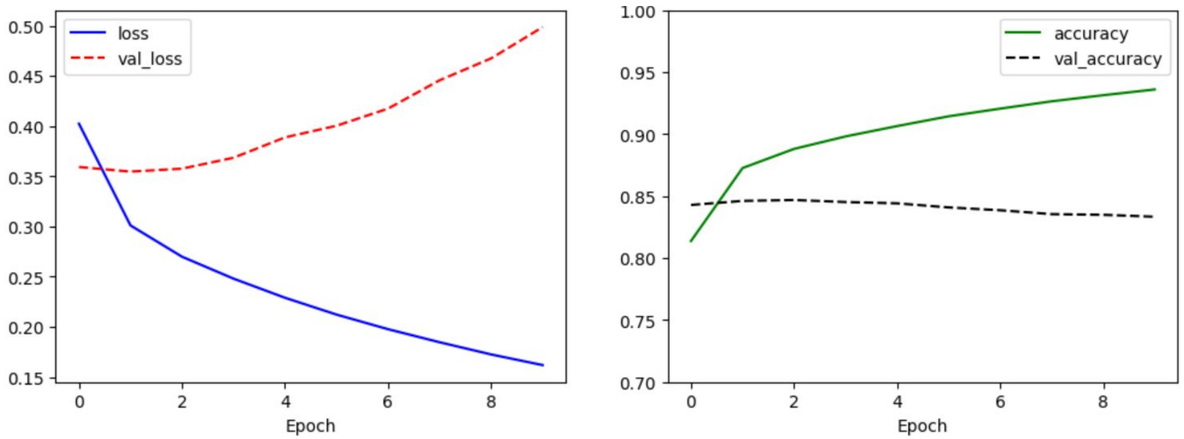
Dense 레이어
- Dense(1, activation='sigmoid') // 하나의 노드로 출력, 이진분류를 위해 sigmoid활성화 함수 사용

04 nsmc Modeling

Keras

```
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))
```

```
Epoch 10/10
2272/2272 [==============================] - 31s 14ms/step - loss: 0.1618 - accuracy: 0.9360 - val_loss: 0.4986 - val_accuracy: 0.8334
```



training data의 loss는 감소하지만, validation data의 loss는 증가한다. => 과적합 발생

- output_dim와 input_length, batch_size를 조정하였지만 loss와 정확도에 큰 변화가 없었다.
- train data의 레이블이 부정확한 것으로 추정

04 nsmc Modeling

Pytorch

- 돌려도 RAM에서 세션 다운 납니다.....나도 맥북 줘.....나도 GPU 줘.....

Q & A

감사합니다