



감성분석_IMDB 영화평



kaggle_Bag of Words Meets Bags of Popcorn 활용



활용 데이터

Data List

Aa 이름	데이터 유형	파일	링크
labeled_train_data	tsv	labeledTrainData.tsv	https://drive.google.com/file/d/1I5siLPcZjICz2Dy_9Ztusp=share_link
unlabeled_train_data	tsv	unlabeledTrainData.tsv	https://drive.google.com/file/d/1femJ9utjWEGZOITVusp=share_link
test_data	tsv	testData.tsv	https://drive.google.com/file/d/1wFvx-L5mhKEsCJ9DRtVJebCgRDx_3hA8/view?usp=share_link
sample_submission	csv	sampleSubmission.csv	https://drive.google.com/file/d/1V6B3zYk_ZgjNg5s3cusp=share_link



문제정의

IMDB영화 사이트의 영화평을 이용해서 감성분석을 수행했습니다.

지도학습 기반 감성 분석이므로 텍스트 기반의 이진 분류와 유사한 방식을 사용했고 감성 분석 결과가 긍정 혹은 부정인지 예측하는 모델을 제작했습니다.



목차

0. Import

- Import Library
- Load Data

1. Data Preprocessing

2. Feature Vectorization

- CounterVectorizer

- TfidfVectorizer
- BOWVectorizer

3. Modeling

4. Feature Importance

분석

1. Data PreProcessing

```
print(review_df['review'][0])
```

```
"With all this stuff going down at the moment with MJ i've started listening to his music, watching th
```

'review'칼럼의 각 행에는 문장이 들어있습니다.

```
#target 칼럼 추출
class_df = review_df['sentiment']

#review 칼럼 추출
feature_df = review_df.drop(columns=['id', 'sentiment'], axis=1, inplace=False)

#review 칼럼 train, test분할(train_test_split(X,y))
X_train, X_test, y_train, y_test = train_test_split(feature_df, class_df, test_size = 0.3, random_state=42)
X_train.shape, X_test.shape
```

```
X_train
```

	review
3243	"As horror fans we all know that blind rentals...
3188	"Cut to the chase, this is one of the five wor...
3063	"W"Sky CaptainW" may be considered an homage t...
469	"The lovely Danish actress Sonja Richter steal...
2689	"OK - as far as the 2 versions of this movie. ...
...	...
2103	"This 1947 film stars and was directed and wri...
3842	"After having watched W"Guinea PigW", two ques...
1892	"This was the funniest piece of film/tape I ha...
3082	"I was going to give it an 8, but since you pe...
1442	"The film is poorly casted, except for some fa...
3500 rows x 1 columns	

review칼럼만 가진 X_train, X_test 데이터를 제작합니다.

2. Feature Vectorization

CountVectorizer

- 단어 피처에 값을 부여할 때 각 문서에서 해당 단어가 나타나는 횟수, 즉 Count를 부여하는 경우

```
#stop words = English, filtering, ngram = (1,2)로 설정 후 피처벡터화
cnt_vect = CountVectorizer(stop_words='english', ngram_range = (1,2))
cnt_vect.fit(X_train['review'])

#X_train데이터 피처 벡처화 변환 수행
X_train_cnt_vect = cnt_vect.transform(X_train['review'])
print('학습데이터 크기 : ',X_train_cnt_vect.shape)
#X_test데이터 피처 벡처화 변환 수행
X_test_cnt_vect = cnt_vect.transform(X_test['review'])
print('테스트데이터 크기 : ',X_test_cnt_vect.shape)
```

```
학습데이터 크기 : (3500, 342198)
테스트데이터 크기 : (1500, 342198)
```

TF-IDF Vectorizer

- 개별 문서에서 자주 나타나는 단어에 높은 가중치를 주되, 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 페널티

$$TFIDF_i = TF_i * \log \frac{N}{DF_i}$$

TF_i = 개별 문서에서의 단어 i 빈도
 DF_i = 단어 i 를 가지고 있는 문서 개수
 N = 전체 문서 개수

```
tfidf_vect = TfidfVectorizer(stop_words='english', ngram_range = (1,2),max_df=300)
tfidf_vect.fit(X_train['review'])

#X_train데이터 피쳐 백처화 변환 수행
X_train_tfidf_vect = tfidf_vect.transform(X_train['review'])
print('학습데이터 크기 : ',X_train_tfidf_vect.shape)
#X_test데이터 피쳐 백처화 변환 수행
X_test_tfidf_vect = tfidf_vect.transform(X_test['review'])
print('테스트데이터 크기 : ',X_test_tfidf_vect.shape)
```

```
학습데이터 크기 : (3500, 342091)
테스트데이터 크기 : (1500, 342091)
```

BOWVectorizer

1. 단어 토큰화를 진행합니다.

```
#단어 토큰화
word_map = {}
for i in tqdm(review_df['review'].index):
    text = review_df.loc[i,'review']
    words = word_tokenize(review_df.loc[i,'review'])
    word_map[i] = words

#words_map을 list형태로 변형
word_list = list(word_map.values())
word_list
```

2. 모든 단어를 중복 제거 후 칼럼 형태로 나열합니다. 각 단어에 고유의 인덱스를 부여합니다.

```
word_unique_list = []
for value in tqdm(word_list):
    word_unique_list.extend(value)
    word_unique_list = list(set(word_unique_list))

print(word_unique_list)
```

```
100%|██████████| 5000/5000 [00:22<00:00, 225.49it/s]
['Visions', 'Jerusalem', 'suplexing', 'Flu', 'ruminating', 'naif', 'thin.\\', 'self-loathing']
```

3. 개별 문장에서 해당 단어가 나타나는 횟수를 각 단어에 기재합니다.

```
word_df = pd.DataFrame(columns = word_unique_list)
word_df
```

```
Visions Jerusalem suplexing Flu ruminating naif thin. self-loathing Smart 'blind ... handle prem
0 rows x 57483 columns
```

```
datas = []
for i in tqdm(review_df['review'].index):
    word_series = pd.Series(collections.Counter(word_list[i])) #값별 개수를 도출
    data = word_series.to_dict()
    datas.append(data)
```

```
word_list_df = pd.DataFrame(datas, index = review_df.index)
word_list_df
```

```
With all this stuff going down at the moment ... alterego- Eamonn screwballs un
0 1 1.0 4.0 11.0 1.0 3.0 1.0 2.0 17.0 1.0 ... NaN NaN NaN
1 1 NaN NaN NaN NaN NaN NaN NaN 9.0 NaN ... NaN NaN NaN
2 1 NaN NaN NaN NaN NaN NaN NaN 17.0 NaN ... NaN NaN NaN
3 1 NaN NaN 3.0 NaN NaN NaN NaN 25.0 NaN ... NaN NaN NaN
4 1 NaN 2.0 NaN NaN NaN NaN 3.0 14.0 NaN ... NaN NaN NaN
...
4995 1 NaN 1.0 2.0 1.0 NaN NaN NaN 8.0 NaN ... NaN NaN NaN
4996 1 NaN 1.0 4.0 NaN NaN NaN NaN 3.0 NaN ... NaN NaN NaN
4997 1 NaN NaN 2.0 NaN NaN NaN NaN 3.0 NaN ... NaN NaN NaN
4998 1 NaN NaN NaN NaN NaN NaN NaN 2.0 NaN ... 1.0 1.0 1.0
4999 1 NaN NaN NaN NaN NaN 1.0 NaN 6.0 NaN ... NaN NaN NaN
5000 rows x 57483 columns
```

4. Sentiment칼럼을 붙입니다.

```
word_list_df['sentiment'] = review_df['sentiment']
word_list_df['sentiment']
```

```
0      1
1      1
2      0
3      0
4      1
..
4995    0
```

```

4996    1
4997    0
4998    1
4999    1
Name: sentiment, Length: 5000, dtype: int64

```

```

#NaN값을 0으로 채워줍니다.
word_list_df = word_list_df.fillna(0)
word_list_df

```

the moment ... alterego- Eamonn screwballs# unique.It rapidly.The mentors fantastic.It nowadays.I										
1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...
1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	...	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	...	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0

3. Modeling

```

#CounterVectorized 감성 예측
lr_clf = LogisticRegression(solver='liblinear',C=10)
lr_clf.fit(X_train_cnt_vect, y_train)
pred = lr_clf.predict(X_test_cnt_vect)
print('CounterVectorized LR 정확도 : ', accuracy_score(y_test,pred))

```

```
CounterVectorized LR 정확도 : 0.844
```

```

#TfidfVectorizer 감성 예측
lr_clf = LogisticRegression(solver='liblinear',C=10)
lr_clf.fit(X_train_tfidf_vect, y_train)
pred = lr_clf.predict(X_test_tfidf_vect)
print('TfidfVectorizer LR 정확도 : ', accuracy_score(y_test,pred))

```

```
TfidfVectorizer LR 정확도 : 0.8586666666666667
```

```
#BOWVectorizer 감성 예측
lr_clf = LogisticRegression(solver='liblinear',C=10)
lr_clf.fit(X_train, y_train)
pred = lr_clf.predict(X_test)
print('BOWVectorizer RF 정확도 : ', accuracy_score(y_test,pred))
```

```
BOWVectorizer RF 정확도 : 0.8273333333333334
```

4. Feature Importance

```
#BOW기반 피쳐 중요성
rf = RandomForestClassifier(n_estimators=100, random_state=0)
rf.fit(X_train, y_train)
imp = rf.feature_importances_
imp_df = pd.DataFrame(imp, index=X.columns, columns=["imp"])
imp_df = imp_df.sort_values("imp", ascending=False)
imp_df.head(10)
```

	imp
bad	0.008337
worst	0.007127
great	0.005132
waste	0.004547
awful	0.004507
and	0.004262
?	0.004231
n't	0.003585
nothing	0.003305
of	0.003001

결론

- 최종 백터화 및 성능

최종선택 피처벡터화 : **TfidfVectorizer**

최종 성능 : accuracy : **0.8586666666666667**

- 코드

https://drive.google.com/file/d/10B3MmJTupuVjSheWU0osg34T2aaA_HIT/view?usp=sharing

대소문자 처리

특수문자 처리

너무 적은거 제외