



Alchemist Session

CHAP 08 텍스트 분석

CONTENTS

목차

1. 텍스트 분석 이해
2. 텍스트 전처리 - 텍스트 정규화
3. Bag Of Words - BOW
4. 텍스트 분류 실습 - 20 뉴스그룹 분류
5. 감성 분석
6. 토픽모델링 - 20 뉴스그룹
7. 문서 군집화 소개와 실습 (Opinion Review)
8. 문서 유사도
9. 한글 텍스트 처리 - 네이버 영화 평점 감성 분석

05. 감성 분석

감성 분석

- 주관적인 감성/의견/감정/기분 등을 파악하기 위한 방법
- 텍스트가 나타내는 주관적인 단어와 문맥을 기반으로 감성 수치를 계산
- 감정 지수 - 긍정감정지수, 부정감정지수

지도 학습	학습 데이터와 타깃 레이블 값을 기반으로 감성분석학습 수행
비지도 학습	‘Lexicon’이라는 일종의 감성 어휘 사전 이용. Lexicon을 이용해 긍정적, 부정적 감성 여부 판단

감성 분석

```
import pandas as pd  
  
review_df = pd.read_csv('./labeledTrainData.tsv', header=0, sep="\t", quoting=3)  
review_df.head(3)
```

	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... ...

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

"With all this stuff going down at the moment with MJ i've started listening to his music, watching the odd documentary here and there, watched The Wiz and watched Moonwalker again. Maybe i just want to get a certain insight into this guy who i thought was really cool in the eighties just to maybe make up my mind whether he is guilty or innocent. Moonwalker is part biography, part feature film which i remember going to see at the cinema when it was originally released. Some of it has subtle messages about MJ's feeling towards the press and also the obvious message of drugs are bad m'kay.

Visually impressive but of course this is all about Michael Jackson so unless you're remotely like MJ in anyway then you are going to hate this and find it boring. Some may call MJ an egotist for consenting to the making of this movie BUT MJ and most of his fans would say that he made it for the fans which if true is really nice of him.

The actual feature film bit when it finally starts is only on for 20 minutes or so

- Id : 각 데이터의 id
- sentiment : 영화평의 sentiment 결과값. (1=긍정적 평가. 0=부정적 평가) ✓ target !!
- review : 영화평 텍스트

감성 분석

- 텍스트 전처리

- (1)
태그 삭제

- str 속성 이용

- (2) 숫자/특수문자 제거

- 정규 표현식 이용

```
import re

# <br> html 태그는 replace 함수로 공백으로 변환
review_df['review'] = review_df['review'].str.replace('<br />', '')

# 파이썬의 정규 표현식 모듈인 re를 이용해 영어 문자열이 아닌 문자는 모두 공백으로 변환
review_df['review'] = review_df['review'].apply(lambda x : [x.sub("[^a-zA-Z]", " ", x))
```

감성 분석

- 피처 벡터화와 ML 분류 알고리즘

두 가지 과정 Pipeline 이용해 한번에 처리

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score

# 스톱 워드는 english, filtering, ngram은 (1,2)로 설정해 TF-IDF 벡터화 수행.      TF-IDF 결과
# LogisticRegression의 C는 10으로 설정.
pipeline = Pipeline([
    ('cnt_vect', TfidfVectorizer(stop_words='english', ngram_range=(1,2))),
    ('lr_clf', LogisticRegression(solver='liblinear', C=10))])

pipeline.fit(X_train['review'], y_train)
pred = pipeline.predict(X_test['review'])
pred_probs = pipeline.predict_proba(X_test['review'])[:,1]

print('예측 정확도는 {:.4f}, ROC-AUC는 {:.4f}'.format(accuracy_score(y_test, pred),
                                                       roc_auc_score(y_test, pred_probs)))
```

예측 정확도는 0.8939, ROC-AUC는 0.9598

감성 분석

- **비지도학습** 기반 감성 분석

Lexicon이라는 **감성사전**을 이용해 각 단어의 감정 지수 표시

*감정지수 : 긍정/감정 정도 의미하는 수치, 단어의 위치나 주변 단어/문맥/POS 등으로 결정

- NLTK(Natural Language Toolkit) 패키지의 WordNet은 영어 단어들의 의미 사전이며, 단어들 사이의 의미적, 어휘적 관계를 제공. 이는 사람의 언어 이해에 가까운 방식으로 단어와 문장을 이해하는 데 도움을 줌.
- Synset은 WordNet에서 동의어 그룹을 의미. 각 Synset은 고유한 의미를 가진 단어의 집합으로, 서로 대체 가능한 단어들을 모아놓은 것. 이런 Synset들을 이용하면 단어나 문장의 의미를 더 정확하게 파악하고 분석할 수 있음.
예를 들어) 'ship'이라는 단어에 대한 Synset은 'boat', 'cargo ship' 등을 포함.

감성 분석

- 비지도학습 기반 감성 분석

대표적인 감성 사전

SentiWordNet	감성 단어 전용의 WordNet 구현. WordNet의 Synset별로 3가지 감성 점수를 할당 (긍정/부정/객관적 감정 지수)
VADER	주로 소셜미디어 텍스트에 대한 감성 분석을 제공하기 위한 패키지.
Pattern	예측 성능 측면에서 가장 주목받는 패키지 (파이썬 2.X 버전에서만 동작)

감성 분석

- VADER를 이용한 감성 분석

VADER 감성 사전을 이용해 감성분석(비지도)을 수행한 뒤 예측 성능을 지도학습 기반의 분류와 비교

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer  
  
senti_analyzer = SentimentIntensityAnalyzer()  
senti_scores = senti_analyzer.polarity_scores(review_df['review'][0])  
print(senti_scores)  
  
{'neg': 0.13, 'neu': 0.743, 'pos': 0.127, 'compound': -0.7943}
```

VADER 사용법

Polarity_scores() - 감정 점수를 구한 뒤, 해당 감성 점수가 특정 임계값 이상이면 긍정, 또는 부정으로 판단

감성 분석

- VADER를 이용한 감성 분석

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

예러 발생시 3줄 코드 추가

```
def vader_polarity(review, threshold=0.1):
    analyzer = SentimentIntensityAnalyzer()
    scores = analyzer.polarity_scores(review)
```

compound 값에 기반해 threshold 입력값보다 크면 1, 그렇지 않으면 0을 반환
agg_score = scores['compound']
final_sentiment = 1 if agg_score >= threshold else 0
return final_sentiment

감성결과 반환

```
#apply lambda 식을 이용해 레코드 별로 vader_polarity()를 수행하고 결과를 'vader_preds'에 저장
review_df['vader_preds'] = review_df['review'].apply(lambda x : vader_polarity(x, 0.1))
y_target = review_df['sentiment'].values
vader_preds = review_df['vader_preds'].values
```

```
print(confusion_matrix(y_target, vader_preds))
print("정확도:", np.round(accuracy_score(y_target, vader_preds), 4))
print("정밀도:", np.round(precision_score(y_target, vader_preds), 4))
print("재현율:", np.round(recall_score(y_target, vader_preds), 4))
```

Vader_polarity()

- 입력 파라미터 : 영화 감상평 텍스트와 긍정/부정을 결정하는 임곗값
- Polarity_scores()메서드를 호출해 감성 결과를 반환

감성 분석

- VADER를 이용한 감성 분석

SentiWordNet과 VADER 예측 성능 비교

평가 지표	정확도	정밀도	재현율
SentiWordNet	0.6613	0.6472	0.7091
VADER	0.6956	0.6491	0.8514

지도학습 분류 기반의 예측 성능에 비해 비지도 학습은 낮은 수준

06. 토픽 모델링

20 뉴스그룹 분류

토픽모델링

- **토픽모델링**

문서 집합에 숨어있는 주제를 찾아내는 것. 머신러닝 기반 토픽 모델은 숨겨진 주제를 효과적으로 표현할 수 있는 중심 단어를 합축적으로 추출

토픽모델링 기법

- **LSA (Latent Semantic Analysis)**

문서와 단어의 잠재적인 의미를 발견하려는 방법입니다. 이는 특이값 분해(Singular Value Decomposition, SVD)라는 수학적 기법을 이용하여 문서-단어 행렬을 분해하고, 이렇게 분해된 행렬을 다시 조합하여 문서의 잠재적인 주제를 찾아냅니다

- **LDA (Latent Dirichlet Allocation)**

각 문서에서 토픽의 분포와 각 토픽 내에서 단어의 분포를 추정하는 확률적 생성 모델입니다. LDA는 각 문서가 여러 토픽들의 혼합으로 구성되어 있으며, 각 토픽은 확률 분포에 따라 단어를 생성한다는 가정하에 작동합니다. 이 방법을 통해 각 문서의 토픽 분포를 알아낼 수 있습니다.

토픽모델링

- LDA 적용 토픽 모델링 예제

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# 모터사이클, 야구, 그래픽스, 윈도우즈, 종동, 기독교, 전자공학, 의학 8개 주제를 추출
cats = ['rec.motorcycles', 'rec.sport.baseball', 'comp.graphics', 'comp.windows.x',
        'talk.politics.mideast', 'soc.religion.christian', 'sci.electronics', 'sci.med']

# 위에서 cats 변수로 기재된 카테고리만 추출. featch_20newsgroups( )의 categories에 cats 입력
news_df = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'),
                             categories=cats, random_state=0)

#LDA는 Count 기반의 벡터화만 적용합니다.
count_vect = CountVectorizer(max_df=0.95, max_features=1000, min_df=2, stop_words='english',
                            ngram_range=(1,2))
feat_vect = count_vect.fit_transform(news_df.data)
print('CountVectorizer Shape:', feat_vect.shape)
```

CountVectorizer Shape: (7862, 1000)

Categories 파라미터를 통해 필요한 주제만 필터링해 추출하고 추출된 텍스트를 Count 기반으로 벡터화 변환

토픽모델링

- LDA 적용 토픽 모델링 예제

토픽 8개로 설정한 후 LDA 토픽 모델링 수행

```
lda = LatentDirichletAllocation(n_components=8, random_state=0)
lda.fit(feat_vect)
```

LatentDirichletAllocation(n_components=8, random_state=0)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Components_는 개별 토픽별로 각 word 피처가 얼마나 많이 그 토픽에 할당됐는지에 대한 수치 가짐.

```
print(lda.components_.shape)
lda.components_
(8, 1000)

array([[3.60992018e+01, 1.35626798e+02, 2.15751867e+01, ...,
       3.02911688e+01, 8.66830093e+01, 6.79285199e+01],
       [1.25199920e-01, 1.44401815e+01, 1.25045596e-01, ...,
       1.81506995e+02, 1.25097844e-01, 9.39593286e+01],
       [3.34762663e+02, 1.25176265e-01, 1.46743299e+02, ...,
       1.25105772e-01, 3.63689741e+01, 1.25025218e-01],
       ...,
```

0번째 row, 10번째 col에 있는 값은 Topic #00에 대한 피처 벡터화 행렬에서 10번째 칼럼에 해당하는 피처(단어) 연관 수치값

토픽모델링

- LDA 적용 토픽 모델링 예제

Display_topics() : 각 토픽별로 연관도가 높은 순으로 Word를 나열

```
def display_topics(model, feature_names, no_top_words):
    for topic_index, topic in enumerate(model.components_):
        print('Topic #', topic_index)

        # components_array에서 가장 값이 큰 순으로 정렬했을 때, 그 값의 array 인덱스를 반환
        topic_word_indexes = topic.argsort()[:-1]
        top_indexes = topic_word_indexes[:no_top_words]

        # top_indexes대상인 인덱스별로 feature_names에 해당하는 word feature 추출 후 join으로 concat
        feature_concat = ' '.join([feature_names[i] for i in top_indexes])
        print(feature_concat)

#CountVectorizer 객체 내 전체 word 명칭을 get_features_names()를 통해 추출
feature_names = count_vect.get_feature_names_out() 로 수정하기 ✕
#토픽별 가장 연관도가 높은 word를 15개만 추출
display_topics(lda, feature_names, 15)
```

07. 문서 군집화 소개와 실습

Opinion Review 데이터 세트

문서 군집화 소개와 실습

- 문서 군집화

비슷한 텍스트 구성의 문서를 군집화. 토픽모델링과 유사하지만 문서 군집화는 학습 데이터 세트가 필요 없는 비지도 학습 기반으로 작동.

1) Data Import

파일명과 파일 내용을 칼럼으로 가지는 DataFrame을 생성

```
import pandas as pd
import glob, os
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_colwidth', 700)

path = '/Users/lenovo/Downloads/OpinosisDataset1.0_2/topics'
#path로 지정한 디렉토리 밑에 있는 .data 파일들의 파일명을 리스트로 취합
all_files = glob.glob(os.path.join(path, '*.data'))
filename_list = []
opinion_text = []

# 개별 파일들의 파일명은 filename_list 리스트로 취합,
# 개별 파일들의 파일 내용은 DataFrame 로딩후 다시 string으로 변환하여 opinion_text 리스트로 취합
for file_ in all_files:
    #개별 파일을 읽어서 DataFrame으로 생성
    df = pd.read_table(file_, index_col = None, header=0, encoding='latin1')
    # 절대 경로로 주어진 파일명을 가공.
    # 맨 마지막 .data 확장자도 제거
    filename_ = file_.split('\\')[-1]
    filename = filename_.split('.')[0]

    # 파일명 리스트와 파일 내용 리스트에 파일명과 파일 내용을 추가.
    filename_list.append(filename)
    opinion_text.append(df.to_string())

# 파일명 리스트와 파일 내용 리스트를 DataFrame으로 생성
document_df = pd.DataFrame({'filename':filename_list, 'opinion_text': opinion_text})
document_df.head()
```

문서 군집화 소개와 실습

- 문서 군집화

2) TF-IDF 시행

피처 벡터화 시행. TfidfVectorizer에

Lemmatization 기능 추가를 위해 tokenizer에

커스텀 어근변환 함수 LemNormalize() 함수 생성

```
from nltk.stem import WordNetLemmatizer
import nltk
import string

# 단어 원형 추출 함수
lemmar = WordNetLemmatizer()
def LemTokens(tokens):
    return [lemmar.lemmatize(token) for token in tokens]

# 특수 문자 사전 생성: {33: None ...}
# ord(): 아스키 코드 생성
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)

# 특수 문자 제거 및 단어 원형 추출
def LemNormalize(text):
    # 텍스트 소문자 변경 후 특수 문자 제거
    text_new = text.lower().translate(remove_punct_dict)

    # 단어 토큰화
    word_tokens = nltk.word_tokenize(text_new)

    # 단어 원형 추출
    return LemTokens(word_tokens)

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english',
                             ngram_range=(1,2), min_df = 0.05, max_df= 0.85)

#opinion_text 칼럼값으로 feature vectorization 수행
feature_vect = tfidf_vect.fit_transform(document_df['opinion_text'])
```

문서 군집화 소개와 실습

3) k-means clustering

문서의 유형은 전자제품, 자동차, 호텔로 돼 있음. 5개 중심 기반으로 어떻게 군집화 됐는지 확인

```
from sklearn.cluster import KMeans  
  
# 5개 집합으로 군집화 수행. 예제를 위해 동일한 클러스터링 결과 도출용 random_state=0  
km_cluster = KMeans(n_clusters=5, max_iter=10000, random_state=0)  
km_cluster.fit(feature_vect)  
cluster_label = km_cluster.labels_  
cluster_centers = km_cluster.cluster_centers_
```

문서 군집화 소개와 실습

- 문서 군집화

3) k-means clustering

5개 군집화 결과

Cluster #0	Cluster #1	Cluster #2	Cluster #3	Cluster #4
호텔에 대한 리뷰	포터풀 전자기기 및 주요 구성요소	주로 차량용 내비 게이션으로 군집이 구성	대부분 호텔에 대한 리뷰	자동차에 대한 리뷰

filename	opinion_text	cluster_label
1 /Users/lenovo/Downloads/OpinosisDataset1	Ride seems comfortable and gas mileage fairly good averaging 26 city and 30 open road .\n0 ...	0
18 /Users/lenovo,		
22 /Users/lenovo,	The voice prompts and maps are wonderful especially when driving after dark .\n0 I also thought the the voice	1
23 /Users/lenovo,	We arrived at 23,30 hours and they could not recommend a restaurant so we decided to go to Tesco, with very	
6 /Users/l	limited choices but when you are hingry you do not careNext day they rang the bell at 8,00 hours to clean the	
29 /Users/lenovo,	filename	
8 /Users/l	2 /Users/lenovo/Downloads/OpinosisDataset1	opinion_text 2 cluster_label
30 /Users	0 /Users/lenovo/Downloads/OpinosisDataset1	
7 /Users/leno	short battery life I moved up from an 8gb .\n0 I love this ipod except for the battery life .\n1 ...	3
10 /Users/l	3 /Users/lenovo/Downloads/OpinosisDataset1	
31 /Users	Great location for tube and we crammed in a fair amount of sightseeing in a short time .\n0 All in all, a normal	4
9 /Users/leno	chain hotel on a nice lo...	
46 /Users	4 /Users/lenovo/Downloads/OpinosisDataset1	Staff are friendl...
11 /Users/leno	13 /Users/lenovo/Downloads/OpinosisDataset1	
	Mediocre room and service for a very extravagant price .\n0 ...	4
	Both of us having worked in tourism for over 14 years were very disappointed at the level of service provided by	
16 /Users/lenovo/Downloads/OpinosisDataset1	this gentleman .\n0 The service was good, very friendly staff and we loved the free wine reception each night	
	.\n1 ...	4

문서 군집화 소개와 실습

- 문서 군집화

3) k-means clustering

3개 군집화 결과

Cluster #0	Cluster #1	Cluster #2
포터블 전자기기 리뷰	자동차 리뷰	호텔 리뷰

	filename	opinion_text	cluster_label
50	/Users/lenovo/Downloads/OpinosisDataset1	Parking was expensive but I think this is common for San Fran .\n0 there is a fee for parking but well worth it seeing no where to park if you do have a car .\n1 ...	0
27	/Users/ 48 /Users/lenovo/Downloads/OpinosisDataset1	3 quot widescreen display was a bonus .\n0 This made for smoother graphics on the 255w of the vehicle moving along displayed roads, where the 750's display was more of a jerky movement .\n1 ...	1
28	/Users/ 38 /Users/lenovo/Downloads/OpinosisDataset1	In fact, the entire navigation structure has been completely revised , I'm still getting used to it but it's a huge step forward .\n0 ...	1
36	/Users/lenov 18 /Users/lenovo/Downloads/OpinosisDataset1	Drivers seat not comfortable, the car itself compared to other models of similar class .\n0 ...	2
30	/Users/ 42 /Users/lenovo/Downloads/OpinosisDataset1	I previously owned a Toyota 4Runner which had incredible build quality and reliability .\n0 I bought the Camry because of Toyota reliability and qua...	2
0	/Users/lenov 43 /Users/lenovo/Downloads/OpinosisDataset1	Ride seems comfortable and gas mileage fairly good averaging 26 city and 30 open road .\n0 Seats are fine, in fact of all the smaller sedans this is the most comfortable I found for the price as I am 6', 2 and 250#. \n1 Great gas mileage and comfortable on long trips ...	2
10	/Users/lenov 29 /Users/lenovo/Downloads/OpinosisDataset1	Front seats are very uncomfortable .\n0 No memory seats, no trip computer, can only display outside temp with trip odometer .\n1 ...	2

문서 군집화 소개와 실습

- 문서 군집화

4) 군집별 핵심 단어 추출하기

각 군집 문서는 핵심 단어를 주축으로 군집화. Kmeans 객체에는 단어 피처가 군집의 중심을 기준을 얼마나 깊게 위치해 있는지 **cluster_centers_ 속성** 제공

```
cluster_centers = km_cluster.cluster_centers_
print('cluster_centers shape: ', cluster_centers.shape)
print(cluster_centers)
```

```
cluster_centers shape: (3, 4611)
[[0.          0.00099499 0.00174637 ... 0.          0.00183397 0.00144581]
 [0.01005322 0.          0.          ... 0.00706287 0.          0.          ]
 [0.          0.00092551 0.          ... 0.          0.          0.        ]]
```

*1에 가까울 수록 중심과 가까운 값 의미

ex) cluster[0,1] = 0번 군집에서 두번째 피처의 위치 값

문서 군집화 소개와 실습

- 문서 군집화

4) 군집별 핵심 단어 추출하기

```
# 군집별 top n 핵심 단어, 그 단어의 중심 위치 상댓값, 대상 파일명을 반환함.
def get_cluster_details(cluster_model, cluster_data, feature_names, clusters_num,
                       top_n_features=10):
    cluster_details = {}

    # cluster_centers array의 값이 큰 순으로 정렬된 인덱스 값을 반환
    # 군집 중심점(centroid)별 할당된 word 피쳐들의 거리값이 큰 순으로 값을 구하기 위한.
    centroid_feature_ordered_ind = cluster_model.cluster_centers_.argsort()[:, ::-1]

    #개별 군집별로 반복하면서 핵심 단어, 그 단어의 중심 위치 상댓값, 대상 파일명 입력
    for cluster_num in range(clusters_num):
        # 개별 군집별 정보를 담을 데이터 초기화.
        cluster_details[cluster_num] = {}
        cluster_details[cluster_num]['cluster'] = cluster_num

        # cluster argsort로 구한 인덱스를 이용해 top n 피처 단어를 구함
        top_feature_indexes = centroid_feature_ordered_ind[cluster_num, :top_n_features]
        top_features = [feature_names[ind] for ind in top_feature_indexes]

        #top_feature_indexes를 이용해 해당 피처 단어의 중심 위치 상댓값 구함.
        top_feature_values = cluster_model.cluster_centers_[cluster_num,
                                                          top_feature_indexes].tolist()

        #cluster_details 딕셔너리 객체에 개별 군집별 핵심단어와 중심위치 상댓값, 해당 파일명 입력
        cluster_details[cluster_num]['top_features'] = top_features
        cluster_details[cluster_num]['top_features_value']= top_feature_values
        filenames = cluster_data[cluster_data['cluster_label']==cluster_num]['filename']
        filenames = filenames.values.tolist()

        cluster_details[cluster_num]['filenames'] = filenames

    return cluster_details
```

get_cluster_details()

Cluster_centers_ 배열 내에서 가장 값이 큰 데이터의 위치
인덱스를 추출한 뒤, 해당 인덱스를 이용해 핵심 단어 이름
과 그때의 상대 위치 값을 추출

문서 군집화 소개와 실습

- #### • 문서 군집화

4) 군집별 핵심 단어 추출하기

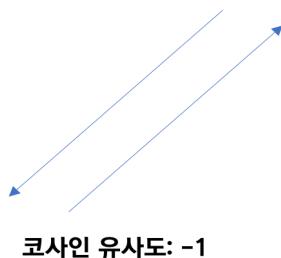
Cluster #0	Cluster #1	Cluster #2
‘screen’, ‘battery’, ‘battery life’	‘interior’, ‘seat’, ‘mileage’, ‘comfortable’	‘room’ ‘hotel’, ‘service’, ‘staff’

08. 문서 유사도

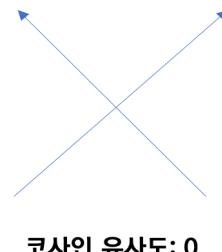
문서 유사도

- 코사인 유사도

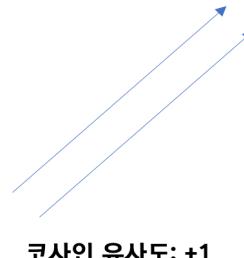
코사인 유사도는 벡터와 벡터 간의 유사도를 비교할 때 벡터의 상호 방향성이 얼마나 유사한지에 기반



반대관계인 벡터들



관련성이 없는 벡터들



유사 벡터들

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

문서 유사도

- 코사인 유사도

코사인 유사도 함수 `cos_similarity()` 함수 작성

```
import numpy as np

def cos_similarity(v1, v2):
    dot_product = np.dot(v1, v2)
    l2_norm = (np.sqrt(sum(np.square(v1)))* np.sqrt(sum(np.square(v2))))
    similarity = dot_product / l2_norm

    return similarity
```

1번째 문장과 2번째 문장의 코사인 유사도 측정

```
#TfidfVectorizer로 transform()한 결과는 회소 행렬이므로 밀집 행렬로 변환.
feature_vect_dense = feature_vect_simple.todense()

#첫번째 문장과 두번째 문장의 피처 벡터 추출
vect1= np.array(feature_vect_dense[0]).reshape(-1, )
vect2= np.array(feature_vect_dense[1]).reshape(-1, )

#첫번째 문장과 두번째 문장의 피처 벡터로 두 개 문장의 코사인 유사도 추출
similarity_simple = cos_similarity(vect1, vect2)
print('문장 1, 문장 2 Cosine 유사도: {:.3f}'.format(similarity_simple))
```

문장 1, 문장 2 Cosine 유사도: 0.402

문서 유사도

- 코사인 유사도

사이킷런 제공 API 사용

sklearn.metrics.pairwise.cosine_similarity API

```
from sklearn.metrics.pairwise import cosine_similarity
similarity_simple_pair = cosine_similarity(feature_vect_simple[0], feature_vect_simple)
print(similarity_simple_pair)
```

[[1. 0.40207758 0.40425045]]

비교되는 문서의 피처 행렬
비교기준이 되는 피처 행렬

문서 유사도

- Opinion Review 데이터 세트를 이용한 문서 유사도 측정

호텔 군집화 데이터 추출

추출된 데이터에 해당되는
TfidfVectorizer 데이터 추출

```
from sklearn.metrics.pairwise import cosine_similarity

# cluster_label=2인 데이터는 호텔로 군집화된 데이터임. DataFrame에서 해당 인덱스를 추출
hotel_indexes = document_df[document_df['cluster_label']==2].index
print('호텔로 클러스터링 된 문서들의 DataFrame Index:', hotel_indexes)

# 호텔로 군집화된 데이터 중 첫번째 문서를 추출해 파일명 표시
comparison_docname = document_df.iloc[hotel_indexes[0]]['filename']
print('##### 비교 기준 문서명 ', comparison_docname, ' 와 타 문서 유사도 #####')

'''document_df에서 추출한 Index 객체를 feature_vect로 입력해 호텔 군집화된 feature_vect 추출
이를 이용해 호텔로 군집화된 문서 중 첫 번째 문서와 다른 문서 간의 코사인 유사도 측정'''
similarity_pair = cosine_similarity(feature_vect[hotel_indexes[0]], feature_vect[hotel_indexes]) 문서간의 유사도 출력
print(similarity_pair)
```

```
호텔로 클러스터링 된 문서들의 DataFrame Index: Index([1, 18, 22, 23, 29, 35, 42, 43, 45, 47], dtype='int64')
##### 비교 기준 문서명 /Users/lenovo/Downloads/OpinosisDataset1 와 타 문서 유사도 #####
[[1.          0.15655631  0.0879083   0.08217817  0.06276647  0.96608144
  0.14398794  0.27273923  0.05452321  0.20206332]]
```

문서 유사도

- Opinion Review 데이터 세트를 이용한 문서 유사도 측정

문서 간에 유사도가 높은 순으로 정렬 및 시각화

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

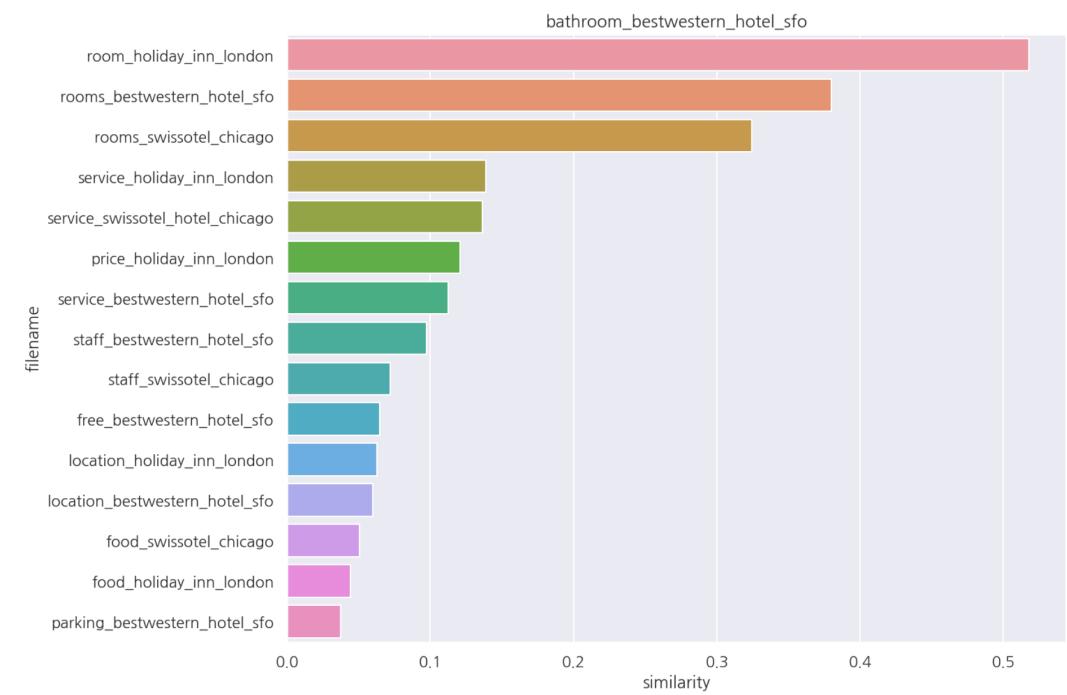
#첫번째 문서와 타 문서간 유사도가 큰 순으로 정렬한 인덱스 추출하되 자기 자신은 제외
sorted_index = similarity_pair.argsort()[:, ::-1]
sorted_index = sorted_index[:, 1:]

#유사도가 큰 순으로 유사도 값을 재정렬하되 자기자신은 제외
hotel_sorted_indexes = hotel_indexes[sorted_index.reshape(-1)]

#유사도가 큰 순으로 유사도 값을 재정렬하되 자기 자신은 제외
hotel_1_sim_value = np.sort(similarity_pair.reshape(-1))[:, :-1]
hotel_1_sim_value = hotel_1_sim_value[1:]

#유사도가 큰 순으로 정렬된 인덱스와 유사도 값을 이용해 파일명과 유사도 값을 막대 그래프로 시각화
hotel_1_sim_df = pd.DataFrame()
hotel_1_sim_df['filename'] = document_df.iloc[hotel_sorted_indexes]['filename']
hotel_1_sim_df['similarity'] = hotel_1_sim_value
print('가장 유사도가 큰 파일명 및 유사도:\n', hotel_1_sim_df.iloc[0:, :])

sns.barplot(x='similarity', y='filename', data=hotel_1_sim_df)
plt.title(comparison_docname)
```



09. 한글 텍스트 처리

네이버 영화 평점 감성 분석

한글 텍스트 처리

- **한글 NLP 처리의 어려움**

한글 언어 처리가 영어 등의 라틴어 처리보다 어려운 이유

① 띄어쓰기

한글은 띄어쓰기를 잘못하면 의미가 왜곡 되어 전달될 수 있음

(예) 아버지가 방에 들어가신다, 아버지가 가방에 들어가신다

이에 반해 영어는 띄어쓰기를 잘못하면 잘못된 혹은 없는 단어로 인식되는 것이 대부분임

(예) My father enters a room, My fatherenters a room

② 다양한 조사

주어나 목적어를 위해 추가되는 조사는 경우의 수가 많기 때문에 어근 추출(Stemming/ Lemmatization) 등의 전처리로 제거하기까 까다로움

(예) '너희 집은 어디 있니?'에서 "집은"이 "은"이 조사인지 명사인지 구분하기가 어렵다

한글 텍스트 처리

- KoNLPy

*형태소 : 단어로서 의미를 가지는 최소 단위

*형태소 분석 말뭉치를 형태소 어근 단위로 쪼개고 각 형태소에 품사 태깅(POS tagging)을 부착하는 작업

파이썬의 대표적인 한글 형태소 패키지

꼬꼬마(Kkma), 한나눔(hannanum), Komoran, 은전한닢 프로젝트(Mecab), Twitter 5개의 형태소 분석

#KoNLP 설치 참고 링크

- <https://velog.io/@chldnjstjr/KoNLP-%ED%8C%A8%ED%82%A4%EC%B9%98-%EC%98%A4%EB%A5%98-%EC%88%98%EB%8F%99%EC%84%A4%EC%B9%98>
- <https://mrchypark.github.io/post/KoNLP-%EC%84%A4%EC%B9%98-%EB%B0%A9%EB%B2%95/>
- (mac) <https://jmock.tistory.com/15>

한글 텍스트 처리

(1) 데이터 로딩

master 1 Branch 1 Tags

Go to file Code

e9t Fix spacing typo in partition.py cc0670e · 8 years ago 9 Commits

code Fix spacing typo in partition.py 8 years ago

raw Add raw data 9 years ago

README.md Upload README 9 years ago

ratings.txt Initial commit 9 years ago

ratings_test.txt Modify headers 9 years ago

ratings_train.txt Modify headers 9 years ago

synopses.json Add synopses data 9 years ago

README

Naver sentiment movie corpus v1.0

This is a movie review dataset in the Korean language. Reviews were scraped from [Naver Movies](#).

The dataset construction is based on the method noted in [Large movie review dataset](#) from Maas et al., 2011.

```
import pandas as pd

train_df = pd.read_csv('ratings_train.txt', sep='\t', encoding='UTF-8')
train_df.head(3)
```

	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밌었다그래서보는것을추천한다	0

```
train_df['label'].value_counts()
```

```
label
0    75173
1    74827
Name: count, dtype: int64
```

1이 긍정 0이 부정 감성

한글 텍스트 처리

(2) 데이터 전처리

```
import re

train_df = train_df.fillna(' ')
#정규 표현식을 이용해 숫자를 공백으로 변경(정규 표현식으로 \d는 숫자를 의미함.)
train_df['document'] = train_df['document'].apply(lambda x : re.sub(r"\d+", " ", x))

#테스트 데이터 세트를 로딩하고 동일하게 Null 및 숫자를 공백으로 변환
test_df = pd.read_csv('ratings_test.txt', sep='\t', encoding='UTF-8')
test_df = test_df.fillna(' ')
test_df['document'] = test_df['document'].apply(lambda x : re.sub(r"\d+", " ", x))
```

Null을 공백으로 변환

정규표현식 이용해 숫자를 공백으로 변환

```
# id 칼럼 삭제
train_df.drop('id', axis=1, inplace = True)
test_df.drop('id', axis=1, inplace = True)
```

한글 텍스트 처리

(2) 데이터 전처리

- TF-IDF 벡터화

한글 형태소 분석을 통해 형태소 단어로 토큰화

```
from konlpy.tag import Okt  
  
okt = Okt()  
def tw_tokenizer(text):  
    # 입력 인자로 들어온 텍스트를 형태소 단어로 토큰화해 리스트 형태로 반환  
    tokens_ko = okt.morphs(text)  
    return tokens_ko
```

Twitter 클래스 -> Okt 클래스!!

형태소 엔진으로 토큰화 수행

```
from sklearn.feature_extraction.text import TfidfVectorizer  
from sklearn.linear_model import LogisticRegression  
from sklearn.model_selection import GridSearchCV  
  
# Twitter 객체의 morphs() 객체를 이용한 tokenizer를 사용. ngram_range는 (1, 2)  
tfidf_vect = TfidfVectorizer(tokenizer=tw_tokenizer, ngram_range=(1,2), min_df=3, max_df=0.9)  
tfidf_vect.fit(train_df['document'])  
tfidf_matrix_train = tfidf_vect.transform(train_df['document'])
```

TF-IDF 시행

한글 텍스트 처리

(3) 모델 훈련

로지스틱 회귀를 이용해 분류 기반의 감성 분석 수행

```
#로지스틱 회귀를 이용해 감성 분석 분류 수행
lg_clf = LogisticRegression(random_state=0, solver='liblinear')

#파라미터 C 최적화를 위해 GridSearchCV를 이용
params = {'C': [1,3.5,4.5,5.5,10]}
grid_cv = GridSearchCV(lg_clf, param_grid=params, cv=3, scoring='accuracy', verbose=1)
grid_cv.fit(tfidf_matrix_train, train_df['label'])
print(grid_cv.best_params_, round(grid_cv.best_score_,4))
#etta photoshop
```

GridSearchCV를 이용해 로지스틱 회귀 하이퍼파라미터
C 최적화 수행

테스트 세트를 이용해 최종 감성 분석 예측 수행

```
from sklearn.metrics import accuracy_score

#학습데이터를 적용한 TfidfVectorizer를 이용해 테스트 데이터를 TF-IDF 값으로 피처 변환함.
tfidf_matrix_test = tfidf_vect.transform(test_df['document'])

#classifier는 GridSearchCV에서 최적 파라미터로 학습된 classifier를 그대로 이용
best_estimator = grid_cv.best_estimator_
preds = best_estimator.predict(tfidf_matrix_test)

print('Logistic Regression 정확도:', accuracy_score(test_df['label'], preds))
```

*테스트 예측할 때는 학습할 때 적용한 TfidfVectorizer 그대로 사용

수고하셨습니다