

텍스트 분석 - 2

▶ 5조 박근주 배경은 신주찬 이주희

CONTENTS

Text Analysis



텍스트 분류 실습 - 20 뉴스그룹 분류

하양이 p. 487~496



감성분석

하양이 p. 497~512



토픽 모델링 (Topic Modeling) - 20 뉴스그룹

하양이 p. 512~516



문서 군집화 소개와 실습 - Opinion Review

하양이 P.516~528

Step1. 텍스트 분류 실습



텍스트 분류 실습

- 20 뉴스그룹 분류

하양이 p.487~496



감성 분석

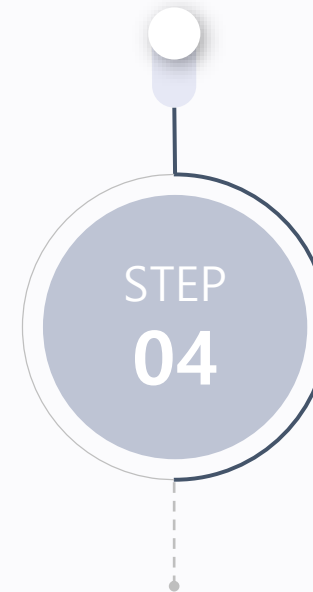
하양이 p.497~512



토픽 모델링

- 20 뉴스그룹

하양이 p. 512~516

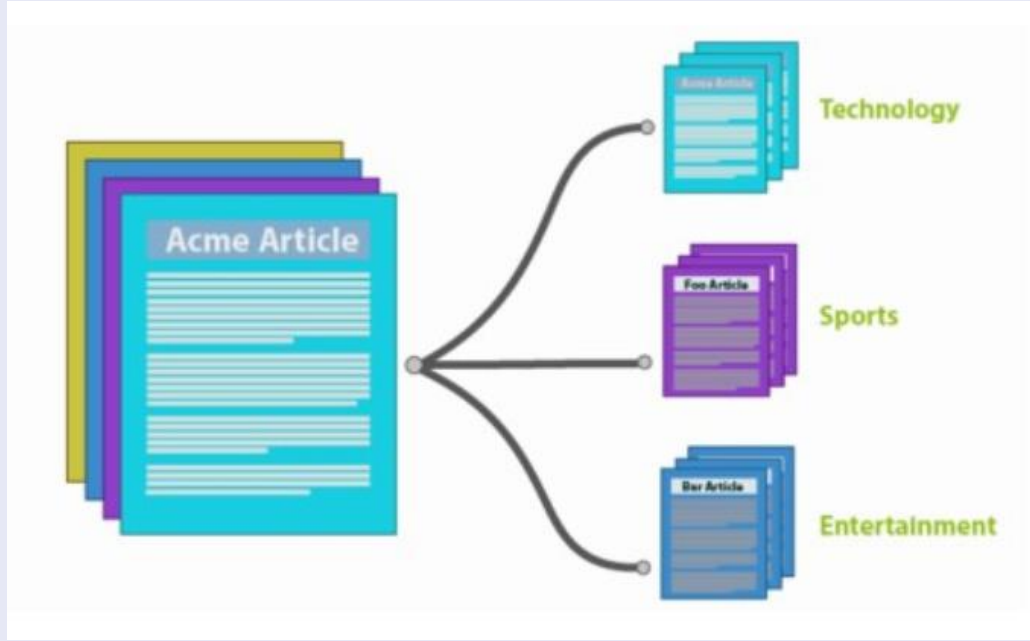


문서 군집화 소개와 실습

Opinion Review 데이터 세트

하양이 p.516~528

텍스트 분류



Step1) 텍스트 정규화

Step2) 정규화된 텍스트의 피처 벡터화

Step2) 학습 모델을 이용하여 다른 문서의 분류를 예측

사이킷 런

- `fetch_20newsgroups()`를 통해 예제 데이터 제공

텍스트의 피처 벡터화

- 희소행렬 형태로 변환

분류 처리

- 로지스틱 회귀
- 선형 서포트 벡터 머신
- 나이브 베이즈

1) 텍스트 정규화

1-1

```
from sklearn.datasets import fetch_20newsgroups

news_data = fetch_20newsgroups(subset='all', random_state=156)

print(news_data.keys())

dict_keys(['data', 'filenames', 'target_names', 'target', 'DESCR'])

import pandas as pd

print('target 클래스의 값과 분포도 \n', pd.Series(news_data.target).value_counts().sort_index())
print('target 클래스의 이름들 \n', news_data.target_names)
```

```
target 클래스의 값과 분포도
0    799
1    973
2    985
3    982
4    963
5    988
6    975
7    990
8    996
9    994
10   999
11   991
12   984
13   990
14   987
15   997
16   910
17   940
18   775
19   628
dtype: int64
```

```
target 클래스의 이름들
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']
```

1. fetch_20newsgroups() 다운로드

2. Key 값 확인하기

3. Target 클래스 구성 확인하기

A

Fetch_20newsgroups()

- 20ro의 뉴스그룹(일종의 주제별 커뮤니티 게시판)에서 모은 게시물로 이루어져 있음
- Scikit-learn에서 제공

B

Target 클래스 구성 확인하기

- Filenames: 인터넷에서 내려 받아 로컬 컴퓨터에 저장하는 디렉터리와 파일명을 지칭

1) 텍스트 정규화

```
print(news_data.data[0])
```

```
From: egreen@east.sun.com (Ed Green - Pixel Cruncher)
Subject: Re: Observation re: helmets
Organization: Sun Microsystems, RTP, NC
Lines: 21
Distribution: world
Reply-To: egreen@east.sun.com
NNTP-Posting-Host: laser.east.sun.com
```

```
In article 211353@mavenry.altcit.eskimo.com, maven@mavenry.altcit.eskimo.com (Norma
n Hamer) writes:
```

```
>
```

```
> The question for the day is re: passenger helmets, if you don't know for
> certain who's gonna ride with you (like say you meet them at a ... church
> meeting, yeah, that's the ticket)... What are some guidelines? Should I just
> pick up another shoe in my size to have a backup helmet (XL), or should I
> maybe get an inexpensive one of a smaller size to accomodate my likely
> passenger?
```

```
If your primary concern is protecting the passenger in the event of a
crash, have him or her fitted for a helmet that is their size. If your
primary concern is complying with stupid helmet laws, carry a real big
spare (you can put a big or small head in a big helmet, but not in a
small one).
```

```
---
```

```
Ed Green, former Ninjaite || I was drinking last night with a biker,
Ed.Green@East.Sun.COM || and I showed him a picture of you. I said,
DoD #0111 (919)460-8302 || "Go on, get to know her, you'll like her!"
(The Grateful Dead) --> || It seemed like the least I could do...
```

4. 개별 데이터의 텍스트 구성 확인하기

: 한 개만 추출하여 확인

텍스트 데이터 확인 결과

기사 내용, 뉴스그룹 제목, 작성자, 소속, 이메일 등의 다양한 정보 있음

→ 내용 제외하고 제목 등의 다른 정보는 제거하기

→ Why? 제목, 소속, 이메일 주소 등의 헤더와 푸터 정보들은 뉴스그룹 분류의 Target 클래스 값과 유사성이 높음

→ 목적: 순수한 텍스트만으로 구성된 기사내용으로 어떤 뉴스그룹에 속하는지 분류하기 위함

1) 텍스트 정규화

5. 뉴스그룹 기사의 헤더, 푸터 등 제거하기

→ remove() 사용

```
from sklearn.datasets import fetch_20newsgroups
```

```
# subset='train'으로 학습용(Train) 데이터만 추출, remove=('headers', 'footers', 'quotes')로 내용만 추출
train_news = fetch_20newsgroups(subset='train', remove=('headers', 'footers', 'quotes'), random_state=156)
X_train = train_news.data
y_train = train_news.target
print(type(X_train))
```

```
# subset='test'으로 테스트(Test) 데이터만 추출, remove=('headers', 'footers', 'quotes')로 내용만 추출
test_news = fetch_20newsgroups(subset='test', remove=('headers', 'footers', 'quotes'), random_state=156)
X_test = test_news.data
y_test = test_news.target
print('학습 데이터 크기 {0} , 테스트 데이터 크기 {1}'.format(len(train_news.data) , len(test_news.data)))
```

```
<class 'list'>
```

```
학습 데이터 크기 11314 , 테스트 데이터 크기 7532
```

A subset 파라미터

학습 데이터 세트와 테스트
데이터 세트 분리하기

B remove()

뉴스그룹 기사의 헤더(header),
푸터/footer) 등 제거

분석 결과

- 학습데이터: 리스트 형태의 11314개의 뉴스그룹 문서
- 테스트데이터: 리스트 형태의 7532개 문서

2) 피쳐 벡터화 변환과 머신러닝 모델 학습/예측/평가

1-2

1. 학습데이터의 텍스트를 피쳐 벡터화 하기

→ CountVectorizer 사용

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
# Count Vectorization으로 feature extraction 변환 수행.
```

```
cnt_vect = CountVectorizer()
```

```
# 개정판 소스 코드 변경(2019. 12. 24)
```

```
cnt_vect.fit(X_train)
```

```
X_train_cnt_vect = cnt_vect.transform(X_train)
```

```
# 학습 데이터로 fit( )된 CountVectorizer를 이용하여 테스트 데이터를 feature extraction 변환 수행.
```

```
X_test_cnt_vect = cnt_vect.transform(X_test)
```

```
print('학습 데이터 Text의 CountVectorizer Shape:', X_train_cnt_vect.shape)
```

```
학습 데이터 Text의 CountVectorizer Shape: (11314, 101631)
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score
```

```
# LogisticRegression을 이용하여 학습/예측/평가 수행.
```

```
lr_clf = LogisticRegression()
```

```
lr_clf.fit(X_train_cnt_vect, y_train)
```

```
pred = lr_clf.predict(X_test_cnt_vect)
```

```
print('CountVectorized Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test, pred)))
```

```
C:\Users\KwonChulmin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

```
FutureWarning)
```

```
C:\Users\KwonChulmin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify the multi_class option to silence this warning.
```

```
"this warning.", FutureWarning)
```

```
CountVectorized Logistic Regression 의 예측 정확도는 0.617
```

유의 사항

테스트 데이터 피쳐 벡터화 수행 시

- 학습데이터를 이용하여 fit()이 수행된 CountVectorizer 객체를 이용하여 테스트 데이터를 변환해야 함

→ 학습 시 설정된 CountVectorizer 피쳐 개수
(=)

테스트 데이터를 CountVectorizer로 변환할
피쳐 개수

- 학습데이터에 사용된 CountVectorizer 객체 변수인 cnt_vect.transform()를 이용해 변환

* fit_transform() 사용 X : 학습시 사용된 피쳐
개수와 예측사용할 피쳐 개수가 달라짐

2) 피처 벡터화 변환과 머신러닝 모델 학습/예측/평가

2. 뉴스그룹에 대한 분류 예측

A Count기반 로지스틱 회귀 분석

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# LogisticRegression을 이용하여 학습/예측/평가 수행.
lr_clf = LogisticRegression()
lr_clf.fit(X_train_cnt_vect, y_train)
pred = lr_clf.predict(X_test_cnt_vect)
print('CountVectorized Logistic Regression 의 예측 정확도는
      {0: .3f}'.format(accuracy_score(y_test, pred)))
```

CountVectorized Logistic Regression 의 예측 정확도는 0.617

B TF-IDF 기반으로 벡터 변경 후

```
from sklearn.feature_extraction.text import TfidfVectorizer

# TF-IDF Vectorization 적용하여 학습 데이터셋과 테스트 데이터 셋 변환.
tfidf_vect = TfidfVectorizer()
tfidf_vect.fit(X_train)
X_train_tfidf_vect = tfidf_vect.transform(X_train)
X_test_tfidf_vect = tfidf_vect.transform(X_test)

# LogisticRegression을 이용하여 학습/예측/평가 수행.
lr_clf = LogisticRegression()
lr_clf.fit(X_train_tfidf_vect, y_train)
pred = lr_clf.predict(X_test_tfidf_vect)
print('TF-IDF Logistic Regression 의 예측 정확도는
      {0: .3f}'.format(accuracy_score(y_test, pred)))
```

TF-IDF Logistic Regression 의 예측 정확도는 0.678

예측 모델 수행 결과

$A < B$

일반적으로 텍스트 분석에서
카운트 벡터화보다 TF-IDF 벡터화가 좋은 결과 도출

2) 피쳐 벡터화 변환과 머신러닝 모델 학습/예측/평가

3. 모델 성능 향상 시키기

방법1. 최적의 ML 알고리즘 선택하기

방법2. 최상의 피쳐 전처리 수행하기

A

- TfidfVectorizer 클래스 스톱 원드 변경

: None → English

- ngram_range

: (1,1) → (1,2)

- max_df=300

B

i. 가장 좋은 예측 성능

: C=10 일때 GridSearchCV의 교차 검증 테스트 세트

ii. 성능 수치 향상

A 다양한 파라미터 적용하여 성능 예측하기

```
# stop words 필터링을 추가하고 ngram을 기본(1,1)에서 (1,2)로 변경하여 Feature Vectorization 적용.
tfidf_vect = TfidfVectorizer(stop_words='english', ngram_range=(1,2), max_df=300)
tfidf_vect.fit(X_train)
X_train_tfidf_vect = tfidf_vect.transform(X_train)
X_test_tfidf_vect = tfidf_vect.transform(X_test)

lr_clf = LogisticRegression()
lr_clf.fit(X_train_tfidf_vect, y_train)
pred = lr_clf.predict(X_test_tfidf_vect)
print('TF-IDF Vectorized Logistic Regression의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test, pred)))
```

TF-IDF Vectorized Logistic Regression의 예측 정확도는 0.690

B GridSearchCV 이용한 로지스틱 회귀의 하이퍼 파라미터 최적화 수행

```
from sklearn.model_selection import GridSearchCV

# 최적 C 값 도출 튜닝 수행. CV는 3 Fold셋으로 설정.
params = {'C': [0.01, 0.1, 1, 5, 10]}
grid_cv_lr = GridSearchCV(lr_clf, param_grid=params, cv=3, scoring='accuracy', verbose=1)
grid_cv_lr.fit(X_train_tfidf_vect, y_train)
print('Logistic Regression best C parameter:', grid_cv_lr.best_params_)

# 최적 C 값으로 학습된 grid_ov로 예측 수행하고 정확도 평가.
pred = grid_cv_lr.predict(X_test_tfidf_vect)
print('TF-IDF Vectorized Logistic Regression의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test, pred)))
```

Fitting 3 folds for each of 5 candidates, totalling 15 fits

Logistic Regression best C parameter: {'C': 10}

TF-IDF Vectorized Logistic Regression의 예측 정확도는 0.704

3) 사이킷런 파이프라인 사용 및 GridSearchCV와의 결합

1-3

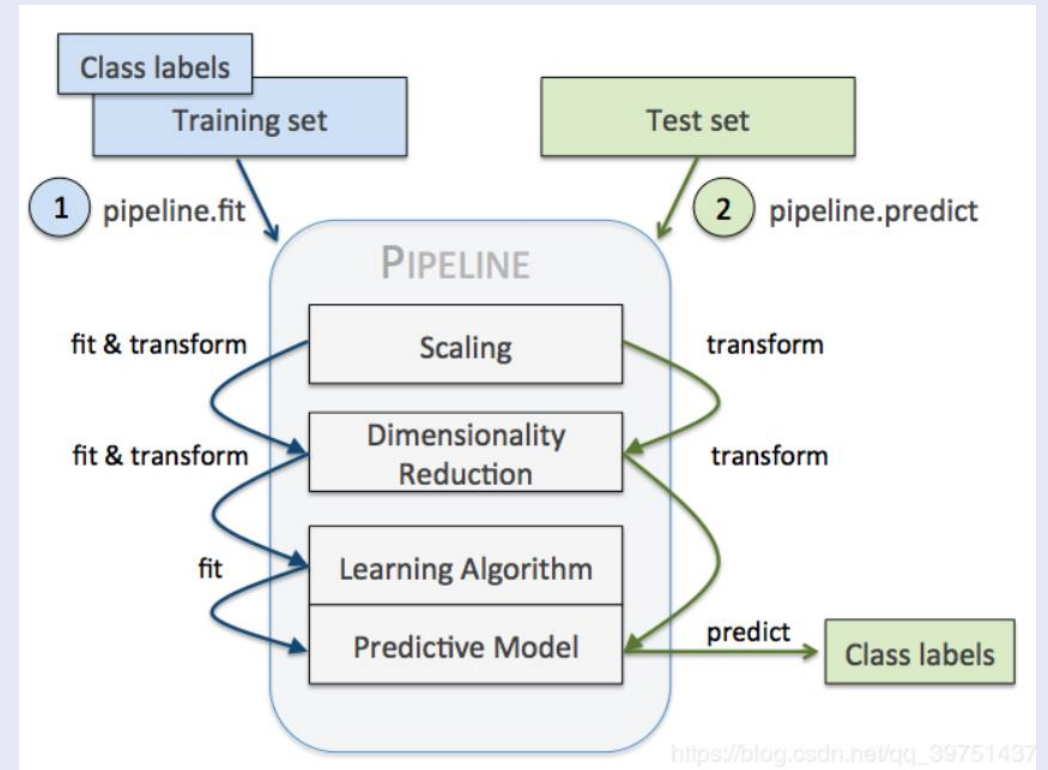
사이킷런 파이프라인 사용

- 연속된 변환을 순차적으로 처리하는 기능 제공하는 Wrapper 도구
- 피처 벡터화와 ML 알고리즘 학습/예측 코드 작성 동시 진행 가능
- 데이터의 전처리 & 머신러닝 학습과정
→ 통일된 API 기반에서 처리 가능 : 직관적인 ML 모델 코드 생성 가능
- 수행 시간 절약
- 텍스트 기반의 피처 벡터화 뿐만 아니라 모든 데이터 전처리 작업과 Estimator 결합 가능

Pipeline 객체 선언

```
pipeline = Pipeline([('tfidf_vect', TfidfVectorizer(stop_words='english')),  
                     ('lr_clf', LogisticRegression(random_state=156))])
```

- * TfidfVectorizer 객체 → tfidf_vect라는 객체 변수명으로,
LogisticRegression 객체 → lr_clf라는 객체 변수명으로 생성
- * 두개의 객체를 파이프라인으로 연결하는 Pipeline 객체 생성



3) 사이킷런 파이프라인 사용 및 GridSearchCV와의 결합

pipeline방식 적용

```
from sklearn.pipeline import Pipeline

# TfidfVectorizer 객체를 tfidf_vect 객체명으로, LogisticRegression 객체를 lr_clf 객체명으로 생성하는 Pipeline 생성
pipeline = Pipeline([
    ('tfidf_vect', TfidfVectorizer(stop_words='english', ngram_range=(1,2), max_df=300)),
    ('lr_clf', LogisticRegression(C=10))
])

# 별도의 TfidfVectorizer 객체의 fit_transform( )과 LogisticRegression의 fit(), predict( )가 필요 없음.
# pipeline의 fit( ) 과 predict( ) 만으로 한꺼번에 Feature Vectorization과 ML 학습/예측이 가능.
pipeline.fit(X_train, y_train)
pred = pipeline.predict(X_test)
print('Pipeline을 통한 Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test ,pred)))
```

Pipeline을 통한 Logistic Regression 의 예측 정확도는 0.704

- 머신러닝 코드를 직관적이고 쉽게 작성 가능
- GridSearchCV를 이용해 “피처 벡터화를 위한 파라미터”와 “ML 알고리즘의 하이퍼 파라미터”를 한번에 최적화 할 수 있음

3) 사이킷런 파이프라인 사용 및 GridSearchCV와의 결합

예제. Pipeline + GridSearchCV

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('tfidf_vect', TfidfVectorizer(stop_words='english')),
    ('lr_clf', LogisticRegression())
])

# Pipeline에 기술된 각각의 객체 변수에 언더바(_)2개를 연달아 붙여 GridSearchCV에 사용될
# 파라미터/하이퍼 파라미터 이름과 값을 설정.
params = { 'tfidf_vect__ngram_range': [(1,1), (1,2), (1,3)],
           'tfidf_vect__max_df': [100, 300, 700],
           'lr_clf__C': [1,5,10]
}

# GridSearchCV의 생성자에 Estimator가 아닌 Pipeline 객체 입력
grid_cv_pipe = GridSearchCV(pipeline, param_grid=params, cv=3, scoring='accuracy', verbose=1)
grid_cv_pipe.fit(X_train, y_train)
print(grid_cv_pipe.best_params_, grid_cv_pipe.best_score_)

pred = grid_cv_pipe.predict(X_test)
print('Pipeline을 통한 Logistic Regression 의 예측 정확도는 {0:.3f}'.format(accuracy_score(y_test, pred)))
```

Fitting 3 folds for each of 27 candidates, totalling 81 fits

```
{'lr_clf__C': 10, 'tfidf_vect__max_df': 700, 'tfidf_vect__ngram_range': (1, 2)} 0.755524129397207
```

Pipeline을 통한 Logistic Regression 의 예측 정확도는 0.702

플러스 알파

- 희소 행렬 기반의 텍스트 분류에
자주 사용되는 머신러닝 알고리즘
- 서포트 벡터머신
 - 나이브 베이즈 알고리즘

주의할 점

최적화하기 위해 많은 튜닝시간이 소모됨

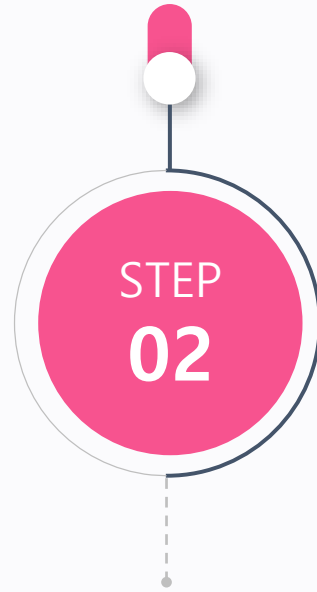
Step2. 감성분석



텍스트 분류 실습

- 20 뉴스그룹 분류

하양이 p.487~496



감성 분석

하양이 p.497~512



토픽 모델링

- 20 뉴스그룹

하양이 p. 512~516



문서 군집화 소개와 실습

Opinion Review 데이터 세트

하양이 p.516~528

감성 분석

감성 분석(Sentiment Analysis)

- 문서의 주관적인 감성/의견/감정/기분 등을 파악하기 위한 방법
- 여러 SNS, 여론조사, 온라인 리뷰, 피드백 등 다양한 분야에서 활용
- 감성 수치 계산: 긍정 감성 지수/부정 감성 지수 를 합산하여 결정

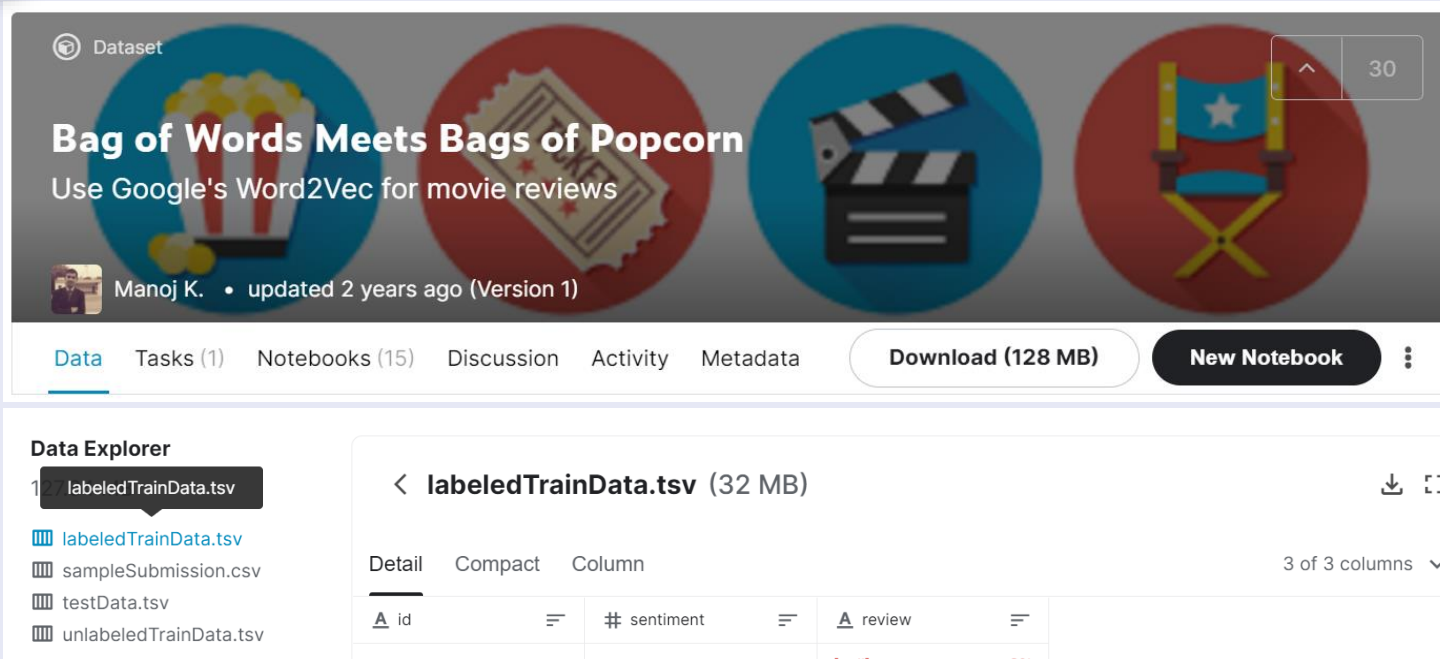
머신러닝 관점

지도학습	비지도학습
<ul style="list-style-type: none">- 학습 데이터와 타깃 레이블 값을 기반으로 감성 분석 학습 수행- 이를 기반으로 다른 데이터의 감성 분석 예측- 일반적인 텍스트 기반의 분류와 거의 동일	<ul style="list-style-type: none">- 'Lexicon'이라는 감성 어휘 사전 이용- Lexicon: 감성 분석을 위한 용어와 문맥에 대한 정보를 갖고 있음

1) 지도학습 기반 감성 분석 실습 – IMDB 영화평

2-1

1 Kaggle페이지>'Bag of Words Meets Bags of Popcorn' >'labeledTrainData.tsv' 다운(파일 첨부)



Dataset

Bag of Words Meets Bags of Popcorn

Use Google's Word2Vec for movie reviews

Manoj K. • updated 2 years ago (Version 1)

Data Tasks (1) Notebooks (15) Discussion Activity Metadata

Download (128 MB) New Notebook

Data Explorer

1 labeledTrainData.tsv

labeledTrainData.tsv

sampleSubmission.csv

testData.tsv

unlabeledTrainData.tsv

< labeledTrainData.tsv (32 MB)

Detail Compact Column

3 of 3 columns

id	sentiment	review
0	1	"With all this stuff going down at the moment ...
1	1	"\"The Classic War of the Worlds\" by Timothy ...
2	0	"The film starts with a manager (Nicholas Bell...

2 DataFrame으로 로딩

```
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...

데이터 피쳐 분석

- id: 각 데이터의 id
- Sentiment
: 영화평의 Sentiment 결과값(Target Label)
*1: 긍정적 평가 / 0: 부정적 평가
- Review
: 영화평 텍스트

1) 지도학습 기반 감성 분석 실습 – IMDB 영화평

3 텍스트 구성 요소 살펴보기: review칼럼의 한 개의 텍스트 값 확인

```
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 i watched The Wiz and watched Moonwalker again. Maybe i just want to get a certain insight into this guy who i thought was real the eighties just to maybe make up my mind whether he is guilty or innocent. Moonwalker is part biography, part feature film member going to see at the cinema when it was originally released. Some of it has subtle messages about MJ's feeling towards and also the obvious message of drugs are bad m'kay.
Visually impressive but of course this is all about Michael J. unless you remotely like MJ in anyway then you are going to hate this and find it boring. Some may call MJ an egotist for con: 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 h
The actual feature film bit when it finally starts is only on for 20 minutes or so excluding the Smooth Criminal sequence Pesci is convincing as a psychopathic all powerful drug lord. Why he wants MJ dead so bad is beyond me. Because MJ overheard I Nah, Joe Pesci's character ranted that he wanted people to know it is he who is supplying drugs etc so i dunno, maybe he just J's music.
Lots of cool things in this like MJ turning into a car and a robot and the whole Speed Demon sequence. directors must have had the patience of a saint when it came to filming the kiddy Bad sequence as usually directors hate workin

```
import re
```

```
# <br> html 태그는 replace 함수로 공백으로 변환
```

```
review_df['review'] = review_df['review'].str.replace('<br />', ' ')
```

```
# 파이썬의 정규 표현식 모듈인 re를 이용하여 영어 문자열이 아닌 문자는 모두 공백으로 변환
```

```
review_df['review'] = review_df['review'].apply( lambda x : re.sub("[^a-zA-Z]", " ", x) )
```

```
from sklearn.model_selection import train_test_split
```

```
class_df = review_df['sentiment']
```

```
feature_df = review_df.drop(['id', 'sentiment'], axis=1, inplace=False)
```

```
X_train, X_test, y_train, y_test= train_test_split(feature_df, class_df, test_size=0.3, random_state=156)
```

```
X_train.shape, X_test.shape
```

```
((17500, 1), (7500, 1))
```

1.
태그 여전히 존재 → 삭제하기
2. replace() :
태그 → 공백으로 변경
3. 숫자/특수문자 → 공란으로 변경: 정규표현식 이용
*정규 표현식 [^a-zA-Z]
: 영어 대/소문자 제외 모든 문자 찾기
*re.sub()사용
4. 결정값 sentiment칼럼 별도 추출
→ 결정값 데이터 세트 생성
→ 원본 데이터 세트에서 id와 sentiment칼럼 삭제
→ 피쳐 데이터 세트 생성
5. 학습용, 테스트용 데이터 세트 분리

1) 지도학습 기반 감성 분석 실습 – IMDB 영화평

4 ML 분류 알고리즘 적용하여 예측 성능 측정

```
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 CountVectorization수행.
# LogisticRegression의 C는 10으로 설정.
pipeline = Pipeline([
    ('cnt_vect', CountVectorizer(stop_words='english', ngram_range=(1,2) )),
    ('lr_clf', LogisticRegression(C=10))])

# Pipeline 객체를 이용하여 fit(), predict()로 학습/예측 수행. predict_proba()는 roc_auc때문에 수행.
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 {0:.4f}, ROC-AUC는 {1:.4f}'.format(accuracy_score(y_test, pred),
                                                         roc_auc_score(y_test, pred_probs)))
```

예측 정확도는 0.8860, ROC-AUC는 0.9503

1) 지도학습 기반 감성 분석 실습 – IMDB 영화평

5 TF-IDF 벡터화 적용하여 예측 성능 측정

```
# 스톱 워드는 english, filtering, ngram은 (1,2)로 설정해 TF-IDF 벡터화 수행.  
# LogisticRegression의 C는 10으로 설정.  
pipeline = Pipeline([  
    ('tfidf_vect', TfidfVectorizer(stop_words='english', ngram_range=(1,2) )),  
    ('lr_clf', LogisticRegression(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 {0:.4f}, ROC-AUC는 {1:.4f}'.format(accuracy_score(y_test, pred),  
                                                                roc_auc_score(y_test, pred_probs)))
```

예측 정확도는 0.8936, ROC-AUC는 0.9598

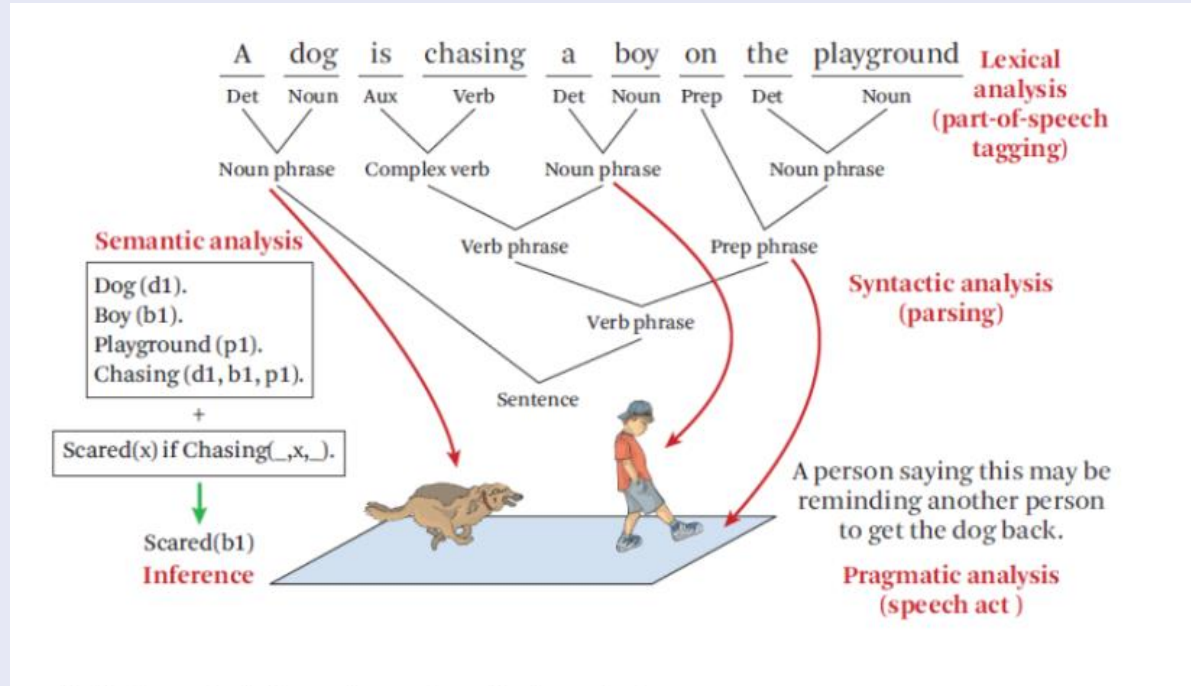
4 예측 정확도는 0.8860, ROC-AUC는 0.9503

- count 벡터화를 적용하여 예측 성능 측정
- TF-IDF 벡터화 적용
- Classifier: 회귀 분석 이용
- 이진 분류 → 테스트 데이터 세트의 정확도와 ROC-AUC 모두 측정

예측 성능 개선됨

2) 비지도학습 기반 감성 분석 소개

2-2



- Lexicon 기반(감성 분석용 데이터가 결정된 레이블 값을 가지고 있지 않은 경우 사용)
- Lexicon: 감성 어휘 사전
 - *감성 어휘 사전
 - 감성지수: 긍정, 부정 감성의 정도를 의미하는 수치
 - 감성지수 결정요인: 단어위치/주변단어/문맥/POS(Part of Speech)
- NLTK패키지: 감성 사전 구현 가능한 패키지(Lexicon모듈 포함 많은 서브 모듈 갖고 있음)

2) 비지도학습 기반 감성 분석 소개

A 시맨틱(semantic)

“Present”

- 1) 선물
- 2) 현재



상황/문맥/화자의 몸짓/어조 등에 따라서 다르게 표현되거나 이해될 수 있음

B WordNet

- 다양한 상황에서 같은 어휘라도 다르게 사용되는 어휘의 시맨틱 정보 제공

C Synset

- Sets of cognitive synonyms
- 각 품사로 구성된 개별 단어 표현
- 단순한 하나의 단어가 아니라
그 단어가 가지는 문맥, 시맨틱 정보를 제공하는 WordNet의 핵심 개념

A NLTK 감성사전

- (+) 감성에 대한 사전 역할 제공
- (-) 예측 성능이 낮음
 - NLTK 포함한 감성 사전
 - * 주로 실무에서는 NLTK 패키지가 아닌 다른 감성 사전 적용
 - SentiWordNet/VADER/ Pattern



SentiWordNet을 이용한 감성 분석

- I. WordNet Synset과 SentiWordNet SentiSynset 클래스의 이해
- II. SentiWordNet을 이용한 영화 감상평 감성분석

3) SentiWordNet을 이용한 감성 분석

2-3

I WordNet Synset과 SentiWordNet SentiSynset 클래스 이해

Step1. NLTK 데이터 세트와 패키지 다운로드

```
import nltk
nltk.download('all')
```

```
[nltk_data] Downloading collection 'all'
[nltk_data] |
[nltk_data] | Downloading package abc to
[nltk_data] | C:\Users\chkwon\AppData\Roaming\nltk_data...
[nltk_data] | Package abc is already up-to-date!
[nltk_data] | Downloading package alpino to
[nltk_data] | C:\Users\chkwon\AppData\Roaming\nltk_data...
[nltk_data] | Package alpino is already up-to-date!
[nltk_data] | Downloading package biocreative_ppi to
[nltk_data] | C:\Users\chkwon\AppData\Roaming\nltk_data...
[nltk_data] | Package biocreative_ppi is already up-to-date!
[nltk_data] | Downloading package brown to
[nltk_data] | C:\Users\chkwon\AppData\Roaming\nltk_data...
[nltk_data] | Package brown is already up-to-date!
[nltk_data] | Downloading package brown_tgz to
[nltk_data] | C:\Users\chkwon\AppData\Roaming\nltk_data...
[nltk_data] | Package brown_tgz is already up-to-date!
[nltk_data] | Downloading package conll to
[nltk_data] | C:\Users\chkwon\AppData\Roaming\nltk_data...
[nltk_data] | Package conll is already up-to-date!
```

```
from nltk.corpus import wordnet as wn
term = 'present'

# 'present'라는 단어로 wordnet의 synsets 생성.
synsets = wn.synsets(term)
print('synsets() 반환 type:', type(synsets))
print('synsets() 반환 값 갯수:', len(synsets))
print('synsets() 반환 값:', synsets)
```

```
synsets() 반환 type: <class 'list'>
synsets() 반환 값 갯수: 18
synsets() 반환 값: [Synset('present.n.01'), Synset('present.n.02'), Synset('present.n.03'), Synset('show.v.01'), Synset('present.v.02'), Synset('stage.v.01'), Synset('present.v.04'), Synset('present.v.05'), Synset('award.v.01'), Synset('give.v.08'), Synset('deliver.v.01'), Synset('introduce.v.01'), Synset('portray.v.04'), Synset('confront.v.03'), Synset('present.v.12'), Synset('salute.v.06'), Synset('present.a.01'), Synset('present.a.02')]
```

A

POS 태그-('present.n.02')

- present: 의미
- n: 품사_명사
- 02: 다른 의미 구분하는 인덱스

Step2. WordNet 모듈 импорт

- 'present'단어의 Synset 추출

A

3) SentiWordNet을 이용한 감성 분석

I WordNet Synset과 SentiWordNet SentiSynset 클래스 이해

Step3. synset 객체 속성 살펴보기

```
for synset in synsets :  
    print('##### Synset name : ', synset.name(), '#####')  
    print('POS : ', synset.lexname())  
    print('Definition: ', synset.definition())  
    print('Lemmas: ', synset.lemma_names())
```

Synset name : present.n.01

POS : noun.time

Definition: the period of time that is happening now; any continuous stretch of time including the moment of speech

Lemmas: ['present', 'nowadays']

Synset name : present.n.02

POS : noun.possession

Definition: something presented as a gift

Lemmas: ['present']

Synset name : present.n.03

POS : noun.communication

Definition: a verb tense that expresses actions or states at the time of speaking

Lemmas: ['present', 'present_tense']

Synset name : show.v.01

POS : verb.perception

Definition: give an exhibition of to an interested audience

Lemmas: ['show', 'demo', 'exhibit', 'present', 'demonstrate']

Synset name : present.v.02

POS : verb.communication

품사 동일/의미 다름

3) SentiWordNet을 이용한 감성 분석

I WordNet Synset과 SentiWordNet SentiSynset 클래스 이해

Step4. WordNet-단어의 상호 유사도 살펴보기

- path_similarity() 이용

```
# synset 객체를 단어별로 생성합니다.
tree = wn.synset('tree.n.01')
lion = wn.synset('lion.n.01')
tiger = wn.synset('tiger.n.02')
cat = wn.synset('cat.n.01')
dog = wn.synset('dog.n.01')

entities = [tree, lion, tiger, cat, dog]
similarities = []
entity_names = [entity.name().split('.')[0] for entity in entities]

# 단어별 synset 들을 iteration 하면서 다른 단어들의 synset과 유사도를 측정합니다.
for entity in entities:
    similarity = [round(entity.path_similarity(compared_entity), 2) for compared_entity in entities]
    similarities.append(similarity)

# 개별 단어별 synset과 다른 단어의 synset과의 유사도를 DataFrame형태로 저장합니다.
similarity_df = pd.DataFrame(similarities, columns=entity_names, index=entity_names)
```

	tree	lion	tiger	cat	dog
tree	1.00	0.07	0.07	0.08	0.12
lion	0.07	1.00	0.33	0.25	0.17
tiger	0.07	0.33	1.00	0.25	0.17
cat	0.08	0.25	0.25	1.00	0.20
dog	0.12	0.17	0.17	0.20	1.00

Lion의 유사도

- 가장 유사도 낮음: Tree
- 가장 유사도 높음: tiger

3) SentiWordNet을 이용한 감성 분석

I WordNet Synset과 SentiWordNet SentiSynset 클래스 이해

Step5. SentiWordNet - senti_synsets()

```
import nltk
from nltk.corpus import sentiwordnet as swn

senti_synsets = list(swn.senti_synsets('slow'))
print('senti_synsets() 반환 type:', type(senti_synsets))
print('senti_synsets() 반환 값 갯수:', len(senti_synsets))
print('senti_synsets() 반환 값:', senti_synsets)
```

```
senti_synsets() 반환 type: <class 'list'>
senti_synsets() 반환 값 갯수: 11
senti_synsets() 반환 값: [SentiSynset('decelerate.v.01'), SentiSynset('slow.v.02'), SentiSynset('slow.v.03'), SentiSynset('slow.a.01'), SentiSynset('slow.a.02'), SentiSynset('dense.s.04'), SentiSynset('slow.a.04'), SentiSynset('boring.s.01'), SentiSynset('dull.s.08'), SentiSynset('slowly.r.01'), SentiSynset('behind.r.03')]
```

Step5-1. 감성지수와 객관성 지수

```
import nltk
from nltk.corpus import sentiwordnet as swn

father = swn.senti_synset('father.n.01')
print('father 긍정감성 지수:', father.pos_score())
print('father 부정감성 지수:', father.neg_score())
print('father 객관성 지수:', father.obj_score())
print('\n')
fabulous = swn.senti_synset('fabulous.a.01')
print('fabulous 긍정감성 지수:', fabulous.pos_score())
print('fabulous 부정감성 지수:', fabulous.neg_score())
```

```
father 긍정감성 지수: 0.0
father 부정감성 지수: 0.0
father 객관성 지수: 1.0
```

```
fabulous 긍정감성 지수: 0.875
fabulous 부정감성 지수: 0.125
```

객관성 지수

0

1

객관적 ↑

3) SentiWordNet을 이용한 감성 분석

II SentiWordNet을 이용한 영화 감상평 감성 분석

순서

1. 문서를 문장 단위로 분해
2. 다시 문장을 단어단위로 토큰화하고 품사 태깅
3. 품사 태깅된 단어 기반으로 synset객체와 senti_synset 객체 생성
4. Senti_synset에서 긍정/부정 감성 지수 구하고 합산하여 특정 임계치 값 이상일 때 긍정감성으로 그렇지 않을 때는 부정감성으로 결정

Step1. 품사 태깅 내부 함수 생성

```
from nltk.corpus import wordnet as wn

# 간단한 NLTK PennTreebank Tag를 기반으로 WordNet기반의 품사 Tag로 변환
def penn_to_wn(tag):
    if tag.startswith('J'):
        return wn.ADJ
    elif tag.startswith('N'):
        return wn.NOUN
    elif tag.startswith('R'):
        return wn.ADV
    elif tag.startswith('V'):
        return wn.VERB
    return
```

3) SentiWordNet을 이용한 감성 분석

II SentiWordNet을 이용한 영화 감상평 감성 분석

Step2. swn_polarity(text) 함수 생성

A. 문서 → 문장 → 단어 토큰 → 품사 태깅

B. SentiSynset 클래스 생성

C. Polarity Score 합산하는 함수 생성

* (긍정지수)+(부정지수) > 0 → 긍정 감성

(긍정지수)+(부정지수) < 0 → 부정 감성

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import sentiwordnet as swn
from nltk import sent_tokenize, word_tokenize, pos_tag
```

```
def swn_polarity(text):
    # 감성 지수 초기화
    sentiment = 0.0
    tokens_count = 0

    A lemmatizer = WordNetLemmatizer()
    raw_sentences = sent_tokenize(text)
    # 분해된 문장별로 단어 토큰 → 품사 태깅 후에 SentiSynset 생성 → 감성 지수 합산
    for raw_sentence in raw_sentences:
        # NLTK 기반의 품사 태깅 문장 추출
        tagged_sentence = pos_tag(word_tokenize(raw_sentence))
        for word, tag in tagged_sentence:
```

```
            # WordNet 기반 품사 태깅과 어근 추출
            wn_tag = penn_to_wn(tag)
            if wn_tag not in (wn.NOUN, wn.ADJ, wn.ADV):
                continue
            lemma = lemmatizer.lemmatize(word, pos=wn_tag)
            if not lemma:
                continue
```

```
            B 어근을 추출한 단어와 WordNet 기반 품사 태깅을 입력해 Synset 객체를 생성.
            synsets = wn.synsets(lemma, pos=wn_tag)
            if not synsets:
                continue
            # sentiwordnet의 감성 단어 분석으로 감성 synset 추출
            # 모든 단어에 대해 긍정 감성 지수는 +로 부정 감성 지수는 -로 합산해 감성 지수 계산.
            synset = synsets[0]
            swn_synset = swn.senti_synset(synset.name())
```

```
            C sentiment += (swn_synset.pos_score() - swn_synset.neg_score())
            tokens_count += 1

    if not tokens_count:
        return 0

    # 총 score가 0 이상일 경우 긍정(Positive) 1, 그렇지 않을 경우 부정(Negative) 0 반환
    if sentiment >= 0:
        return 1

    return 0
```

3) SentiWordNet을 이용한 감성 분석

II SentiWordNet을 이용한 영화 감상평 감성 분석

Step3. IMDB 감성평의 개별 문서에 대해 긍정 및 부정 감성 예측하기

```
review_df['preds'] = review_df['review'].apply(lambda x : swn_polarity(x) )  
y_target = review_df['sentiment'].values  
preds = review_df['preds'].values
```

개별 감상평 텍스트에
swn_polarity(text) 적용

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score  
from sklearn.metrics import recall_score, f1_score, roc_auc_score  
import numpy as np
```

새로운 칼럼 'preds' 추가
= swn_polarity(text)로
반환된 감성평가

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

```
[[7668 4832]  
 [3636 8864]]  
정확도: 0.6613  
정밀도: 0.6472  
재현율: 0.7091
```

4) VADER를 이용한 감성 분석

2-4

VADER를 NLTK 서브 모듈로 설치(3. SentiWordNet이용한 감성 분석 참고)

```
import nltk  
nltk.download('all')
```

VADER 사용법

- 소셜 미디어의 감성 분석 용도로 만들어진 룰 기반의 Lexicon
- SentimentIntensityAnalyzer 임포트
- IMDB 감상평 한개만 감성 분석 수행한 결과

```
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}
```

4) VADER를 이용한 감성 분석

vader_polarity() 함수 생성

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

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

- 입력 파라미터: 긍정/부정 결정하는 임계값
 - Polarity_scores()메서드 호출
- 감성 결과 반환

VADER의 예측 성능 측정

```
# apply lambda 식을 이용하여 레코드별로 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))
```

[[6736 5764]
 [1867 10633]]
정확도: 0.6948
정밀도: 0.6485
재현율: 0.8506

VS

정확도: 0.6613
정밀도: 0.6472
재현율: 0.7091

SentiWordNet 결과

- 각 문서 별 감성결과
- : vader_preds라는
review_df의 새로운 칼럼으로 저장

Step3. 토픽 모델링



텍스트 분류 실습

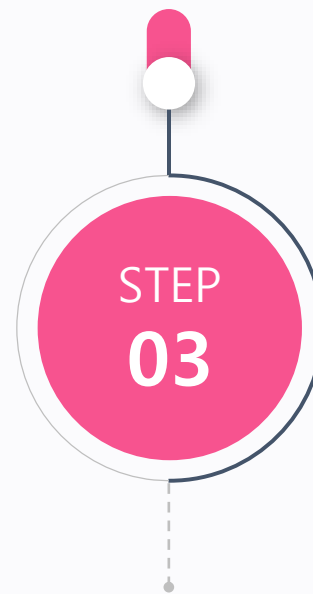
- 20 뉴스그룹 분류

하양이 p.487~496



감성 분석

하양이 p.497~512



토픽 모델링

- 20 뉴스그룹

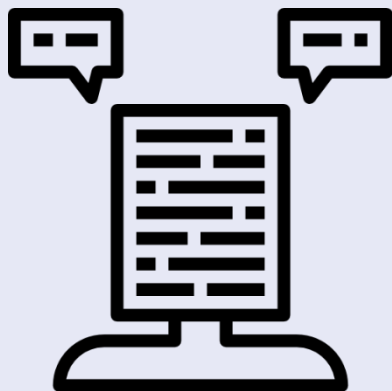
하양이 p. 512~516



문서 군집화 소개와 실습

Opinion Review 데이터 세트

하양이 p.516~528



- 토픽 모델링이란 문서 집합에 숨어있는 주제를 찾아내는 것이다.

많은 양의 문서가 있을 때 사람이 문서를 다 읽고 핵심 주제를 찾는 것은 매우 많은 시간이 소모된다. 이 경우에 토픽모델링을 적용해서 숨어있는 중요 주제를 효과적으로 찾을 수 있다.

- 머신러닝 기반의 토픽 모델은 숨겨진 주제를 효과적으로 표현할 수 있는 중심 단어를 함축적으로 추출한다.

- 머신러닝 기반의 토픽 모델링에 자주 사용되는 기법:

LSA(Latent Semantic Analysis)와 LDA(Latent Dirichlet Allocation)

LSA (Latent Semantic Analysis)

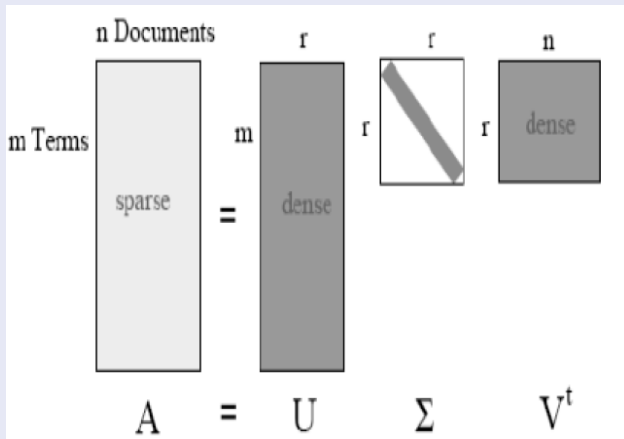
: LSA는 절단된 SVD(truncated SVD)를 통해 차원축소를 하여 잠재적의미를 이끌어낸다.

SVD: 임의의 $m \times n$ 차원의 행렬에 대하여 3개의 행렬의 곱으로 분해하는 방법. (2-3주차 차원축소)

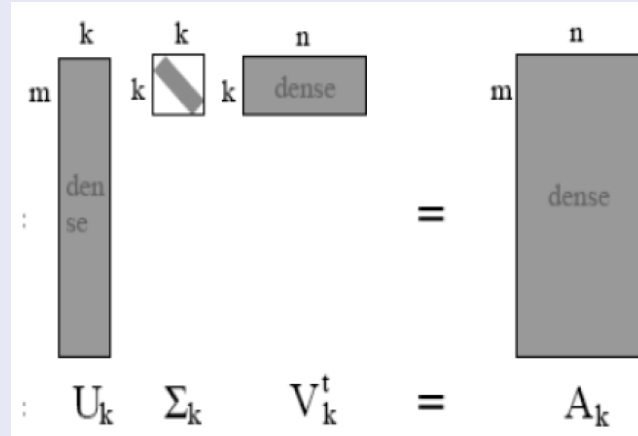
Truncated SVD: Σ 행렬의 대각원소(특이값) 가운데 상위 k 개만 골라낸 형태.

행렬 A 를 복원할 수 없지만, 데이터 정보를 상당히 압축했음에도 행렬 A 를 근사할 수 있다.

Full SVD



Truncated SVD



- 행렬 A 는 n 개의 문서를 m 개의 단어로 표현한 입력데이터
- k 만큼의 특이값만 남기고 $k \times k$ 크기를 갖는 Σ_k 를 만든다.
- U 와 V^t 행렬에 대해서도 k 개만 남기고 $m \times k$ 크기를 갖는 U_k 와 $k \times n$ 크기를 갖는 V_k^t 를 만들어준다.

→ **A_k** 행렬에는 불필요한 토픽은 제거되고, 문서의 잠재 의미(주요 주제, 중요 단어)만 남게 된다.

U_k : n 개의 문서는 원래 단어 수 m 보다 훨씬 작은 k 개 변수로 표현된 결과

V_k : m 개의 단어가 원래 문서 수 n 보다 작은 k 개 변수로 표현한 결과.

LDA (Latent Dirichlet Allocation)

POINT

- LSA의 단점을 개선하여 탄생한 알고리즘으로 토픽 모델링에 보다 적합한 알고리즘
- 단어가 특정 토픽에 존재할 확률과 문서에 특정 토픽이 존재할 확률을 결합확률로 추정하여 토픽을 추출

- 문서1 : 저는 사과랑 바나나를 먹어요
- 문서2 : 우리는 귀여운 강아지가 좋아요
- 문서3 : 저의 깜찍하고 귀여운 강아지가 바나나를 먹어요

LDA는 각 문서의 토픽 분포와 각 토픽 내의 단어 분포를 추정.

<각 문서의 토픽 분포>

문서1 : 토픽 A 100%

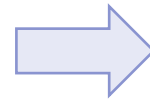
문서2 : 토픽 B 100%

문서3 : 토픽 B 60%, 토픽 A 40%

<각 토픽의 단어 분포>

토픽A : 사과 20%, 바나나 40%, 먹어요 40%, 귀여운 0%, 강아지 0%, 깜찍하고 0%, 좋아요 0%

토픽B : 사과 0%, 바나나 0%, 먹어요 0%, 귀여운 33%, 강아지 33%, 깜찍하고 16%, 좋아요 16%



LDA가 3개의 문서로부터 2개의 토픽을 찾은 결과

토픽 A: “과일”
토픽 B: “강아지”

LDA는 실제로 토픽의 제목을 정해주지 않지만, 위 결과로부터 과일과 강아지에 대한 토픽이라고 판단할 수 있다.

LDA (Latent Dirichlet Allocation)

LDA 수행과정

1) 사용자는 토픽의 개수 k 를 지정.

문서 집합에서 토픽이 몇 개가 존재할지는 사용자가 가정해야 한다. K 의 값을 잘못 선택하면 원치 않는 이상한 결과가 나올 수 있다.

2) 모든 단어를 k 개 중 하나의 토픽에 할당.

LDA는 모든 문서의 모든 단어에 대해서 k 개 중 하나의 토픽을 랜덤으로 할당한다. 이 작업이 끝나면 각 문서는 토픽을 가지며, 토픽은 단어 분포를 가진다.

3) 단어 w 는 아래의 두 가지 기준에 따라서 토픽이 재할당. 모든 문서의 모든 단어에 대해서 반복.(iterative)

(어떤 문서의 각 단어 w 는 잘못된 토픽에 할당되어져 있지만, 다른 단어들은 전부 올바른 토픽에 할당되어져 있는 상태라고 가정.)

- $p(\text{topic } t \mid \text{document } d)$: 문서 d 의 단어들 중 토픽 t 에 해당하는 단어들의 비율

- $p(\text{word } w \mid \text{topic } t)$: 단어 w 를 갖고 있는 모든 문서들 중 토픽 t 가 할당된 비율

이를 반복하면, 모든 할당이 완료된 수렴 상태가 된다.

LDA (Latent Dirichlet Allocation)

3) 단어 w 는 아래의 두 가지 기준에 따라서 토픽이 재할당.

두 가지 기준을 참고하여 LDA는 doc1의 apple을 어떤 토픽에 할당할지 결정한다.

doc1					
word	apple	banana	apple	dog	dog
topic	B	B	???	A	A

doc2					
word	cute	book	king	apple	apple
topic	B	B	B	B	B

- $p(\text{topic } t \mid \text{document } d)$: 문서 d 의 단어들 중 토픽 t 에 해당하는 단어들의 비율

doc1의 모든 단어들은 토픽 A와 토픽 B에 50 대 50의 비율로 할당되어져 있으므로, 단어 apple은 토픽 A 또는 토픽 B 둘 중 어디에도 속할 가능성이 있다.

doc1					
word	apple	banana	apple	dog	dog
topic	B	B	???	A	A

doc2					
word	cute	book	king	apple	apple
topic	B	B	B	B	B

- $p(\text{word } w \mid \text{topic } t)$: 단어 w 를 갖고 있는 모든 문서들 중 토픽 t 가 할당된 비율

단어 apple이 전체 문서에서 어떤 토픽에 할당되어져 있는지를 확인한다. apple은 토픽 B에 할당될 가능성이 높다.

토픽 모델링 – 20 뉴스그룹

실습

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

LDA 기반의 토픽모델링을 사이킷런에서
LatentDirichletAllocation 클래스로 제공한다

오토바이, 야구, 그래픽스, 윈도우즈, 중동, 기독교, 의학, 우주 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 category만 추출. fetch_20newsgroups()의 categories에 cats 입력

```
news_df = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'),
                             categories=cats, random_state=0)
```

LDA 는 Count기반의 Vectorizer만 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)
```

fetch_20newsgroups() API를 통해 카테고리 파라미터를
통해 필요한 주제만 필터링해 추출하고,
추출된 텍스트를 count 기반으로 벡터화 변환한다.

토픽 모델링 – 20 뉴스그룹

#피처 벡터화된 데이터셋을 기반으로 LDA 토픽 모델링을 수행

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

```
LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
                           evaluate_every=-1, learning_decay=0.7,
                           learning_method='batch', learning_offset=10.0,
                           max_doc_update_iter=100, max_iter=10, mean_change_tol=0.001,
                           n_components=8, n_jobs=None, n_topics=None, perp_tol=0.1,
                           random_state=0, topic_word_prior=None,
                           total_samples=1000000.0, verbose=0)
```

LDA의 components_ 속성값

```
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],
       ...,
       [3.60204965e+01, 2.08640688e+01, 4.29606813e+00, ...,
        1.45056650e+01, 8.33854413e+00, 1.55690009e+01],
       [1.25128711e-01, 1.25247756e-01, 1.25005143e-01, ...,
        9.17278769e+01, 1.25177668e-01, 3.74575887e+01],
       [5.49258690e+01, 4.47009532e+00, 9.88524814e+00, ...,
        4.87048440e+01, 1.25034678e-01, 1.25074632e-01]])
```

LDA의 components_ 속성값

- components_ 는 개별 토픽별로 각 word 피처가 얼마나 많이 그 토픽에 할당됐는지에 대한 수치를 가지고 있다.
- 높은 값일수록 해당 word피처는 그 토픽의 중심 word가 된다.

→ 8개의 토픽별로 1000개의 word피처가 해당 토픽별로 연관도를 가지고 있다고 볼 수 있다.

토픽 모델링 – 20 뉴스그룹

- 각 토픽별로 연관도 높은 순으로 word 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 index를 반환.
        topic_word_indexes = topic.argsort()[::-1]
        top_indexes = topic_word_indexes[:no_top_words]

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

# CountVectorizer 객체내의 전체 word들의 명칭을 get_feature_names()를 통해 추출
feature_names = count_vect.get_feature_names()

# Topic별 가장 연관도가 높은 word를 15개만 추출
display_topics(lda, feature_names, 15)

Topic # 0
year 10 game medical health team 12 20 disease cancer 1993 games years patients good
Topic # 1
don just like know people said think time ve didn right going say ll way
Topic # 2
image file jpeg program gif images output format files color entry 00 use bit 03
Topic # 3
like know don think use does just good time book read information people used post
Topic # 4
armenian israel armenians jews turkish people israeli jewish government war dos dos turkey arab armenia 000
Topic # 5
edu com available graphics ftp data pub motif mail widget software mit information version sun
Topic # 6
god people jesus church believe christ does christian say think christians bible faith sin life
Topic # 7
use dos thanks windows using window does display help like problem server need know run
```

topic 0 : 의학 관련 주제어
topic 1 : 명확하지 않고 일반적인 단어가 주를 이룸
topic 2 : 그래픽스 관련 주제어
topic 3 : 일반적인 단어로 주제어가 추출됨
topic 4 : 중동 관련 주제어
topic 5 : 윈도우 관련 주제어
topic 6 : 기독교 관련 주제어
topic 7 : 윈도우 운영체제 관련 주제어

- topic 1과 3은 애매한 주제어가 추출됨
- 모토사이클, 야구의 경우 명확한 주제어가 추출되지 않음

Step4. 문서 군집화 소개와 실습



텍스트 분류 실습

- 20 뉴스그룹 분류

하양이 p.487~496



감성 분석

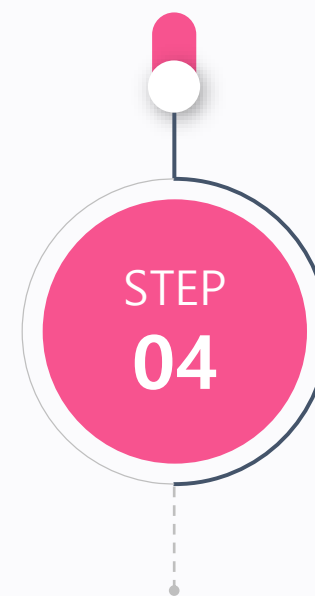
하양이 p.497~512



토픽 모델링

- 20 뉴스그룹

하양이 p. 512~516



문서 군집화 소개와 실습

Opinion Review 데이터 세트

하양이 p.516~528

텍스트 분류

Step1) 특정 문서의 분류를 학습데이터를 통해 학습해 모델 생성한 뒤

Step2) 학습 모델을 이용하여 다른 문서의 분류를 예측

텍스트 분류기반의 문서분류는 사전에 결정 카테고리 값을 가진 학습 데이터 세트가 필요하다.

문서 군집화

: 비슷한 텍스트 구성의 문서를 군집화하는 것

동일한 군집에 속하는 문서를 같은 카테고리 소속으로 분류할 수 있으므로 앞에서 소개한 텍스트 분류기반의 문서 분류와 유사하다.

하지만 문서 군집화는 학습 데이터 세트가 필요 없는 비지도 학습 기반으로 동작한다.

Opinion Review 데이터셋

- 데이터셋 설명
 - 문서 군집화를 수행할 데이터 세트: Opinion Review 데이터셋
 - 해당 데이터셋은 51개의 텍스트 파일로 구성되어 있고, 각 파일은 호텔, 자동차, 전자제품 사이트에서 가져온 리뷰 문서이다.
- 데이터셋 다운 경로 <https://archive.ics.uci.edu/ml/datasets/Opinosis+Opinion+%26frac%3B+Review>
Data Folder → OpinosisDataset1.0.zip



Opinosis Opinion / Review Data Set

Download [Data Folder](#), [Data Set Description](#)

Index of /ml/machine-learning-databases/opinion

- [Parent Directory](#)
- [OpinosisDataset1.0.zip](#)
- [opinion.names](#)













Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips SVN/1.7.14 Phusion_Passenger/4.0.53 mod_perl/2.0.11 Perl/

Opinion Review 데이터셋 로드

```
import pandas as pd
import glob, os

path=r'C:\Users\각자의 디렉토리 설정\OpinionDataset1.0\topics'

# path로 지정한 디렉터리 밑에 있는 모든 .data 파일의 파일명을 리스트로 취합
all_files=glob.glob(os.path.join(path, "*.data"))
filename_list=[]
opinion_text=[]
```

이름	수정한 날짜	유형
 accuracy_garmin_nuvi_255W_gps.txt.data	2010-03-06 오전 2:57	DATA 파일
 bathroom_bestwestern_hotel_sfo.txt.data	2010-03-06 오전 2:57	DATA 파일
 battery-life_amazon_kindle.txt.data	2010-03-06 오전 2:57	DATA 파일
 battery-life_ipod_nano_8gb.txt.data	2010-03-06 오전 2:57	DATA 파일
 battery-life_netbook_1005ha.txt.data	2010-03-06 오전 2:57	DATA 파일
 buttons_amazon_kindle.txt.data	2010-03-06 오전 2:57	DATA 파일
 comfort_honda_accord_2008.txt.data	2010-03-06 오전 2:57	DATA 파일
 comfort_toyota_camry_2007.txt.data	2010-03-06 오전 2:57	DATA 파일
 directions_garmin_nuvi_255W_gps.txt.data	2010-03-06 오전 2:57	DATA 파일
 display_garmin_nuvi_255W_gps.txt.data	2010-03-06 오전 2:57	DATA 파일

Opinion Review 데이터셋 로드

개별 파일을 하나씩 읽어서 파일명과 파일리뷰를 하나의 DataFrame으로 로드

```
for file_ in all_files:
    # 개별 파일을 dataframe으로 생성
    df=pd.read_table(file_,index_col=None, header=0, encoding='latin1')

    #절대 경로로 주어진 파일명에서 파일명.txt만 추출
    filename_=file_.split('\\\\')[1]
    filename=filename_.split('.')[0]

    filename_list.append(filename)
    opinion_text.append(df.to_string())

document_df=pd.DataFrame({'filename':filename_list, 'opinion_text':opinion_text})
document_df.head()
```

	filename	opinion_text
0	accuracy_garmin_nuvi_255W_gps	, and is very, very acc...
1	bathroom_bestwestern_hotel_sfo	The room was not overly big, but clean and...
2	battery-life_amazon_kindle	After I plugged it in to my USB hub on my ...
3	battery-life_ipod_nano_8gb	short battery life I moved up from a...
4	battery-life_netbook_1005ha	6GHz 533FSB cpu, glossy display, 3, Cell 2...

```
from sklearn.feature_extraction.text import TfidfVectorizer

# tokenizer는 Lemmatization을 구현한 LemNormalize() 함수를 이용
tfidf_vect=TfidfVectorizer(tokenizer=LemNormalize, stop_words='english',
                           ngram_range=(1,2), min_df=0.05, max_df=0.85)

# opinion_text 칼럼 값으로 피쳐벡터화 수행
feature_vect=tfidf_vect.fit_transform(document_df['opinion_text'])
```

LemTokens와 LemNormalize 함수

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

remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
lemmar = WordNetLemmatizer()

# 단어의 원형을 찾음
def LemTokens(tokens):
    return [lemmar.lemmatize(token) for token in tokens]

# 주어진 텍스트를 소문자로 만들고 remove_punct_dict에 대한 참조를 사용하여 텍스트 내 문장 부호를 제거
# 이후 단어토큰화를 수행하고 LemToken()을 수행
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

K-Means로 군집화

4-2

```
from sklearn.cluster import KMeans
```

3개의 집합으로 군집화

```
km_cluster = KMeans(n_clusters=3, max_iter=10000, random_state=0)
```

```
km_cluster.fit(feature_vect)
```

```
cluster_label = km_cluster.labels_
```

```
document_df['cluster_label'] = cluster_label
```

```
document_df.head()
```

	filename	opinion_text	cluster_label
0	accuracy_garmin_nuvi_255W_gps	, and is very, very acc...	1
1	bathroom_bestwestern_hotel_sfo	The room was not overly big, but clean and...	2
2	battery-life_amazon_kindle	After I plugged it in to my USB hub on my ...	1
3	battery-life_ipod_nano_8gb	short battery life I moved up from a...	1
4	battery-life_netbook_1005ha	6GHz 533FSB cpu, glossy display, 3, Cell 2...	1

자동차 리뷰

```
document_df[document_df['cluster_label']==0].sort_values(by='filename').head()
```

	filename	opinion_text	cluster_label
6	comfort_honda_accord_2008	Drivers seat not comfortable, the car its...	0
57	comfort_honda_accord_2008	Drivers seat not comfortable, the car its...	0
7	comfort_toyota_camry_2007	Ride seems comfortable and gas mileage fa...	0
58	comfort_toyota_camry_2007	Ride seems comfortable and gas mileage fa...	0
16	gas_mileage_toyota_camry_2007	Ride seems comfortable and gas mileage fa...	0

전자기기 리뷰

```
document_df[document_df['cluster_label']==1].sort_values(by='filename').head()
```

	filename	opinion_text	cluster_label
0	accuracy_garmin_nuvi_255W_gps	, and is very, very acc...	1
51	accuracy_garmin_nuvi_255W_gps	, and is very, very acc...	1
53	battery-life_amazon_kindle	After I plugged it in to my USB hub on my ...	1
2	battery-life_amazon_kindle	After I plugged it in to my USB hub on my ...	1
3	battery-life_ipod_nano_8gb	short battery life I moved up from a...	1

호텔 리뷰

```
document_df[document_df['cluster_label']==2].sort_values(by='filename').head()
```

	filename	opinion_text	cluster_label
1	bathroom_bestwestern_hotel_sfo	The room was not overly big, but clean and...	2
52	bathroom_bestwestern_hotel_sfo	The room was not overly big, but clean and...	2
13	food_holiday_inn_london	The room was packed to capacity with queu...	2
64	food_holiday_inn_london	The room was packed to capacity with queu...	2
14	food_swissotel_chicago	The food for our event was deli...	2

군집별 핵심단어 추출

4-3

각 군집에 속한 문서는 핵심 단어를 주축으로 군집화 되어 있다.
각 군집을 구성하는 핵심단어가 무엇이 있는지 알아보자.

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

```
cluster_centers shape: (3, 2409)  
[[0.          0.00155233 0.          ... 0.          0.          0.          ]  
 [0.0174125  0.          0.          ... 0.          0.          0.00462558]  
 [0.          0.00174184 0.0026023  ... 0.00331835 0.00357495 0.          ]]
```

- k-means 객체의 `cluster_centers_`는 각 군집을 구성하는 단어 피처가 군집의 중심(Centroid)을 기준으로 얼마나 가깝게 위치해 있는지를 보여준다.
- `cluster_centers_`는 배열 값으로 제공되며, 행은 개별 군집을, 열은 개별 피처를 의미한다.
- 예를 들어 `cluster_centers[0,1]`은 0번 군집에서 두번째 피처의 위치 값이다.
- 각 행의 배열 값은 각 군집 내의 2409개 피처가 각 군집의 중심과 얼마나 가까운지를 상대 값으로 나타낸 것이다. 0~1의 값을 가질 수 있으며, 1에 가까울수록 중심과 가까운 값을 의미한다.

군집별 핵심단어 추출

군집별 상위 10개의 핵심단어, 그 단어의 중심 위치 상대값, 대상 파일명들을 반환함.

```
def get_cluster_details(cluster_model, cluster_data, feature_names, clusters_num, top_n_features=10):
    cluster_details = {} # 핵심단어 이름과 상대위치 값을 추출해 저장하는 딕셔너리

    # cluster_centers array 의 값이 큰 순으로 정렬된 index 값을 반환
    # 군집 중심점(centroid)별 할당된 word 피쳐들의 거리값이 큰 순으로 값을 구하기 위함.
    centroid_feature_ordered_ind = cluster_model.cluster_centers_.argsort()[::-1]

    # 개별 군집별로 iteration하면서 핵심단어, 그 단어의 중심 위치 상대값, 대상 파일명 입력
    for cluster_num in range(clusters_num):
        # 개별 군집별 정보를 담은 데이터 초기화.
        cluster_details[cluster_num] = {}
        cluster_details[cluster_num]['cluster'] = cluster_num

        # cluster_centers_.argsort()[::-1] 로 구한 index 를 이용하여 top n 피쳐 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 피쳐 단어의 중심 위치 상대값 구함
        top_feature_values = cluster_model.cluster_centers_[cluster_num, top_feature_indexes].tolist()

        # cluster_details 딕셔너리 객체에 개별 군집별 핵심 단어와 중심위치 상대값, 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
```

```
feature_names = tfidf_vect.get_feature_names()

cluster_details = get_cluster_details(cluster_model=km_cluster, cluster_data=document_df,
                                     feature_names=feature_names, clusters_num=3, top_n_features=10 )
```

Get_cluster_details() 함수 생성

- cluster_centers_ 배열 내
가장 값이 큰 데이터의 위치 인덱스 추출
- 해당 인덱스를 이용해
핵심 단어 이름과 그때의 상대 위치 값 추출
- cluster_details라는 Dict 객체 변수에 기록 및 반환

인자

- Kmeans 군집화 객체
- document_df
- 피쳐명 리스트
- 전체 군집개수
- 핵심단어 추출 개수

군집별 핵심단어 추출

각 cluster의 상위 feature와 파일명 출력

```
for cluster_num, cluster_detail in cluster_details.items():
    print('##### Cluster {0}'.format(cluster_num))
    print('Top features:', cluster_detail['top_features'])
    print('Reviews 파일명 :', cluster_detail['filenames'][:7])
    print('=====')
```

```
##### Cluster 0
Top features: ['interior', 'seat', 'mileage', 'comfortable', 'car', 'gas', 'gas mileage', 'comfort', 'ride',
'performance']
Reviews 파일명 : ['comfort_honda_accord_2008', 'comfort_toyota_camry_2007', 'gas_mileage_toyota_camry_2007',
'interior_honda_accord_2008', 'interior_toyota_camry_2007', 'mileage_honda_accord_2008', 'performance_honda_ac
cord_2008']
=====
```

```
##### Cluster 1
Top features: ['screen', 'battery', 'life', 'battery life', 'keyboard', 'kindle', 'size', 'button', 'voice',
'easy']
Reviews 파일명 : ['accuracy_garmin_nuvi_255W_gps', 'battery-life_amazon_kindle', 'battery-life_ipod_nano_8gb',
'battery-life_netbook_1005ha', 'buttons_amazon_kindle', 'directions_garmin_nuvi_255W_gps', 'display_garmin_nuv
i_255W_gps']
=====
```

```
##### Cluster 2
Top features: ['room', 'hotel', 'service', 'location', 'staff', 'food', 'clean', 'bathroom', 'parking', 'room
wa']
Reviews 파일명 : ['bathroom_bestwestern_hotel_sfo', 'food_holiday_inn_london', 'food_swissotel_chicago', 'free
_bestwestern_hotel_sfo', 'location_bestwestern_hotel_sfo', 'location_holiday_inn_london', 'parking_bestwestern
_hotel_sfo']
=====
```

- cluster0은 **자동차** 리뷰 군집으로 실내 인테리어, 좌석, 연료 효율 등이 핵심단어로 군집화 되었다.
- Cluster1은 **전자제품** 리뷰 군집으로 화면과 배터리 수명 등이 핵심단어로 군집화 되었다.
- Cluster2는 **호텔** 리뷰 군집으로 룸과 서비스 등이 핵심 단어로 군집화 되었다.

Thank you for listening

Q & A