텍스트마이닝 과제2

20190784 식품영양학과 정한주

1. Sklearn의 20 뉴스그룹 문서를 활용하여 아래 물음에 답하시오. 각 문항에 대해 구현 코드 및 결과를 첨부하여 워드 파일로 제출하시오 (15).

① 'alt.atheism', 'sci.electronics'를 이용하여 별도의 전처리 없이 학습 데이터와 시험 데이터를 나눈 후, 각각에 대한 TF-IDF 행렬을 추출하시오 (3).

```
from sklearn.datasets import fetch_20newsgroups
# 20개의 토픽 중 alt.atheism, sci.electronics를 이용
categories = ['alt.atheism', 'sci.space']
# 학습 데이터셋
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)
# 검증 데이터셋
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)
X_train = newsgroups_train.data   #학습 데이터셋 문서
y_train = newsgroups_train.target #학습 데이터셋 라벨
X_test = newsgroups_test.data     #검증 데이터셋 문서
y_test = newsgroups_test.target   #검증 데이터셋 라벨
# TF-IDF 행렬
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
cachedStopWords = stopwords.words("english")
from nltk.tokenize import RegexpTokenizer
from nltk.stem.porter import PorterStemmer
RegTok = RegexpTokenizer("[₩w']{3,}") # 정규포현식으로 토크나이저를 정의
english_stops = set(stopwords.words('english')) #영어 불용어를 가져옴
def tokenizer(text):
    tokens = RegTok.tokenize(text.lower())
    # stopwords 제외
    words = [word for word in tokens if (word not in english_stops) and len(word) > 2]
```

```
    # porterstemmer 적용
    features = (list(map(lambda token: PorterStemmer().stem(token),words)))
    return features
tfidf = TfidfVectorizer(tokenizer=tokenizer)
X_train_tfidf = tfidf.fit_transform(X_train) # train set을 변환
X_test_tfidf = tfidf.transform(X_test) # test set을 변환
print(X_train_tfidf)
print(X_test_tfidf)
```

```
  (0, 12803)    0.05614745539668118
  (0, 15330)    0.062251412713291376
  (0, 8202)     0.08141029772502276
  (0, 9934)     0.04118619719939266
  (0, 8543)     0.074325210864740 86
  (0, 10439)    0.05625554187525934
  (0, 15246)    0.06283975064951941
  (0, 4673)     0.06554663281299161
  (0, 2845)     0.06098607375588154
  (0, 14278)    0.10366675188394683
  (0, 12424)    0.10478001905050269
  (0, 13794)    0.05519939104737248
  (0, 4756)     0.07567146978902138
  (0, 5866)     0.15422389398571743
  (0, 6886)     0.07931454859945888
  (0, 13642)    0.09143152296581589
  (0, 4794)     0.06947158739071178
  (0, 11475)    0.08833840471272068
  (0, 5363)     0.08254400204153153
  (0, 6921)     0.12224577545385738
  (0, 15280)    0.07512235919726824
  (0, 14291)    0.08733379440457896
  (0, 3094)     0.07233905280629284
  (0, 10991)    0.0599231060344091 1
  (0, 3291)     0.06455012737678571
  :     :
  (1072, 16112) 0.16974520247162575
  (1072, 12192) 0.10522785703718507
  (1072, 9968)  0.1930790567102756
  (1072, 12814) 0.06982831956221622
  (1072, 8690)  0.07876879381133498
  (1072, 15636) 0.06649436357191521
  (1072, 13325) 0.0560848444761479
  (1072, 13921) 0.21293510910833355
  (1072, 6812)  0.06613560443314076
  (1072, 10209) 0.09211332190334816
  (1072, 3289)  0.05814287841138872
  (1072, 6415)  0.059900293363187244
  (1072, 2897)  0.08773055775149467
  (1072, 10019) 0.05554810115775162
  (1072, 9590)  0.07004244451765697
  (1072, 15243) 0.071376531404198 54
  (1072, 15246) 0.06240735715542442
  (1072, 2845)  0.12113287042202682
  (1072, 9951)  0.04093559013660819
  (1072, 16252) 0.0733025418788615
  (1072, 15397) 0.08673286005769103
  (1072, 11204) 0.08197602573528572
  (1072, 11636) 0.04280729870053732
  (1072, 14766) 0.020448746397715913
  (1072, 3989)  0.36778283419966273
  (0, 16806)    0.25911003222417206
  (0, 16786)    0.03859349966499689
  (0, 16451)    0.08144840395494059
  (0, 16275)    0.24082794218398862
  (0, 15141)    0.13903442813402067
  (0, 15136)    0.2421477459767975
  (0, 15046)    0.1445185671153352
  (0, 14766)    0.028704468269173024
  (0, 14597)    0.1150720041114234
  (0, 13755)    0.1238765333398468
  (0, 13625)    0.23235231649775592
  (0, 13592)    0.13903442813402067
  (0, 13552)    0.07680857490820321
  (0, 13373)    0.10117745173814906
  (0, 13208)    0.12781481857882557
  (0, 12795)    0.1286717320122964
  (0, 11636)    0.03004489183199601
  (0, 10244)    0.14031039234224402
  (0, 10024)    0.07738556857219832
  (0, 9951)     0.028731207412484496
  (0, 8726)     0.09772247624024669
  (0, 8583)     0.25234823340488105
  (0, 8414)     0.10084560447837182
  (0, 8198)     0.2890371342306704
  (0, 8197)     0.1445185671153352
  :     :
  (712, 7602)   0.13107102461531
  (712, 7282)   0.06627976111661242
  (712, 6969)   0.09918253476084922
  (712, 6965)   0.07079697584905106
  (712, 6943)   0.09802766328299987
  (712, 6940)   0.08172761298231368
  (712, 6476)   0.07376691968969436
  (712, 6415)   0.05580162852014029
  (712, 6258)   0.06389546015860831
  (712, 5771)   0.05450363431591839
  (712, 5707)   0.10466274742376906
  (712, 4801)   0.09693882149091843
  (712, 4632)   0.0724789098659963
  (712, 4629)   0.188004642273886
  (712, 4561)   0.11238667335661968
  (712, 4344)   0.11493038566556409
  (712, 4342)   0.21627178034770922
  (712, 4291)   0.2975476042825477
  (712, 4285)   0.09918253476084922
  (712, 4155)   0.10632027458885365
  (712, 4108)   0.188004642273886
  (712, 3707)   0.0597195279376527
  (712, 3611)   0.12134002876962507
  (712, 2851)   0.07601075696229735
  (712, 702)    0.13879495054816063
```

② PCA를 이용하여 1000개 차원으로 축소하고 정보 손실을 정량화하시오 (3).
– Random state = 42 로 설정하시오.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=1000, random_state=42)

X_train_pca1 = pca.fit_transform(X_train_tfidf.toarray())

X_test_pca1 = pca.transform(X_test_tfidf.toarray())

print('Original tfidf matrix shape:', X_train_tfidf.shape)

print('PCA Converted matrix shape:', X_train_pca1.shape)

print('Sum of explained variance ratio: {:.3f}'.format(pca.explained_variance_ratio_.sum()))
```

```
Original tfidf matrix shape: (1073, 16962)
PCA Converted matrix shape: (1073, 1000)
Sum of explained variance ratio: 0.995
```

16962개의 단어를 1000개의 대표되는 새로운 단어조합으로 바꾸어 차원 16962개를 1000개로 줄였다. 실행 결과, 전체 데이터 분산의 99.5% 설명력을 가져 정보의 손실이 거의 없었다는 사실을 알 수 있었다. 즉 16962개를 다 쓰지 않고 1000개만 써도 성능에 문제가 없다.

③ 원본 시험 데이터와 변환된 시험 데이터에 대한 분류기를 각각 학습하고, 분류 점수를 도출하시오 (3).
- Sklearn에서 제공하는 RidgeClassifier를 이용하시오.
- Random state = 42 로 설정하고, 점수는 score 함수를 이용하시오.

```
from sklearn.linear_model import RidgeClassifier

from sklearn.metrics import accuracy_score

rc = RidgeClassifier()

rc.fit(X_train_tfidf, y_train) # 원본 시험 데이터에 대한 분류기 학습과 점수 도출

print('#Train set score: {:.3f}'.format(rc.score(X_train_tfidf, y_train)))

print('#Test set score: {:.3f}'.format(rc.score(X_test_tfidf, y_test)))

rc.fit(X_train_pca1, y_train) # 변환된 시험 데이터에 대한 분류기 학습과 점수 도출

print('#Train set score: {:.3f}'.format(rc.score(X_train_pca1, y_train)))

print('#Test set score: {:.3f}'.format(rc.score(X_test_pca, y_test)))
```

```
#Train set score: 1.000
#Test set score: 0.968
#Train set score: 1.000
#Test set score: 0.968
```

④ PCA를 이용하여 100개 차원과 10개 차원으로 축소한 데이터를 구축하고 각각에 대한 정보 손실 정도를 정량화하시오 (3).

- Random state = 42 로 설정하시오.

```
# 100개: pca = PCA(n_components=100, random_state=42)

X_train_pca2 = pca.fit_transform(X_train_tfidf.toarray())

X_test_pca = pca.transform(X_test_tfidf.toarray())

print('Original tfidf matrix shape:', X_train_tfidf.shape)

print('PCA Converted matrix shape:', X_train_pca2.shape)

print('Sum of explained variance ratio: {:.3f}'.format(pca.explained_variance_ratio_.sum()))

# 10개: pca = PCA(n_components=10, random_state=42)

X_train_pca3 = pca.fit_transform(X_train_tfidf.toarray())

X_test_pca = pca.transform(X_test_tfidf.toarray())

print('Original tfidf matrix shape:', X_train_tfidf.shape)

print('PCA Converted matrix shape:', X_train_pca3.shape)

print('Sum of explained variance ratio: {:.3f}'.format(pca.explained_variance_ratio_.sum()))
```

```
Original tfidf matrix shape: (1073, 16962)  Original tfidf matrix shape: (1073, 16962)
PCA Converted matrix shape: (1073, 100)     PCA Converted matrix shape: (1073, 10)
Sum of explained variance ratio: 0.381      Sum of explained variance ratio: 0.091
```

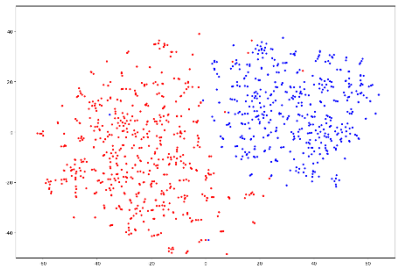
100개로 차원을 축소하였을 때는 전체 데이터 분산의 38.1%를 설명하였고, 10개로 차원을 축소하였을 때는 전체 데이터 분산의 9.1%밖에 설명하지 못했다.

⑤ 1000개, 100개, 10개로 축소된 각각의 데이터를 TSNE를 이용하여 2차원으로 시각화하시오(3).

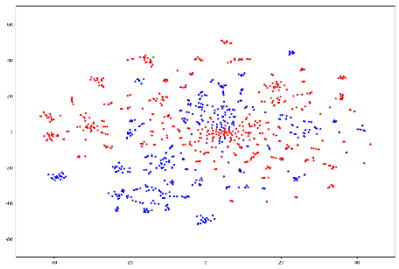
- Random state = 42 로 설정하시오.

```
import matplotlib.pyplot as plt

import matplotlib as mpl

mpl.rcParams['axes.unicode_minus'] = False # 그래프에서 마이너스 폰트 깨지는 문제에 대한 대처

from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=42, init='random')

tsne_tfidf = tsne.fit_transform(X_train_pca1) <- 여기서 X_train_pca만 변경(1: 1000개, 2: 100개, 3: 10개)

print('TSNE dimension:', tsne_tfidf.shape)

def tsne_graph(tsne_2, label, lim=None):

    colors = {0:'blue', 1:'red', 2:'green', 3:'purple'}

    x = tsne_2[:,0] #압축된 첫 차원을 x축으로 이용
```

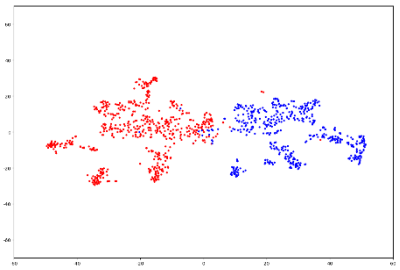
```
    y = tsne_2[:,1] #압축된 둘째 차원은 y축으로 이용
    plt.figure(figsize=(15,10))
    if lim == None:
        lim = [min(x), max(x), min(y), max(y)]
    plt.xlim(lim[0], lim[1])
    plt.ylim(lim[2], lim[3])
    #for i in range(500):
    for i in range(len(x)):
        #각 값에 대해 y값 즉 label에 따라 색을 바꿔가며 text로 그래프에 출력
        if (lim[0] < x[i] < lim[1]) and (lim[2] < y[i] < lim[3]):
            plt.text(x[i], y[i], label[i], color = colors[label[i]])
    plt.show()
tsne_graph(tsne_tfidf, y_train, (-70, 70, -50, 50))
```



1000차원



100차원



10차원

2. 첨부된 '한국어리뷰_최종.csv'문서를 활용하여 아래 물음에 답하시오. 각 문항에 대해 구현 코드 및 결과를 첨부하여 워드 파일로 제출하시오 (35).

① 데이터 전처리를 하기 전 타겟 데이터를 추출하라.(7)

```
import pandas as pd

df = pd.read_csv('한국어리뷰_최종.csv', encoding='CP949')

# 0784부터 1000개의 리뷰 데이터 추출

start_index = 784

num_reviews = 1000

target_data = df.iloc[start_index:start_index + num_reviews]
```

② 수업시간에 배운 다양한 종류의 데이터 전처리 기법을 활용하여 상품평에 대한 전처리를 수행하라 (7).

```
from konlpy.tag import Okt

import re

def tokenize(text): # 토큰화

    t = Okt()

    tokens = t.morphs(text)

    return tokens

def remove_stopwords(tokens): # 불용어 처리

    my_stopwords = ['니다', '요', '습니다', '있습니다', '입니다', '고', '이', '가', '은' ,'는']

    tokens = [token for token in tokens if token not in my_stopwords]

    return tokens

def preprocess(text):

    text = re.sub('[^가-힣ㄱ-ㅎㅏ-ㅣa-zA-Z]', ' ', text) # 특수 문자와 숫자를 공백으로 치환

    text = text.lower() # 소문자로 통일

    return text

def text_preprocessing(text): # 전체 전처리 함수

    text = preprocess(text)

    tokens = tokenize(text)

    tokens = remove_stopwords(tokens)

    return ' '.join(tokens)

target_data['Reviews'] = target_data['Reviews'].apply(text_preprocessing)
```

③ 전처리된 상품평을 기준으로 카운트벡터와 TF-IDF벡터를 생성하라 (7).
- 단어 feature는 본인 학번 기준 홀수는 40개 짝수는 50개로 지정

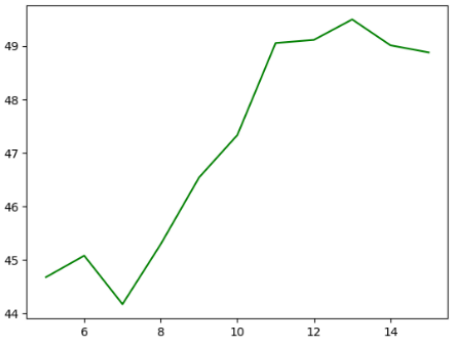
```
texts = target_data['Reviews'] # 전처리 끝낸 텍스트 리스트를 저장

# 카운트벡터 생성

from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features=50)

reviews_cv = cv.fit_transform(texts)

# TF-IDF벡터 생성

from sklearn.feature_extraction.text import TfidfVectorizer

tfv = TfidfVectorizer(max_features=50)

reviews_tfv = tfv.fit_transform(texts)
```

④ LDA를 활용하여 토픽 모델링을 수행하라 (gensim과 pyLDAvis를 활용할 것) (7).
- 토픽수별로 혼란도와 응집도를 계산하여 가장 좋은 토픽 수를 가진 모델을 도출
- 각 토픽 별 최대 단어 수는 10개로 지정

```
# LDA 토픽 모델링 실행

from sklearn.decomposition import LatentDirichletAllocation

import numpy as np

np.set_printoptions(precision=3)

lda = LatentDirichletAllocation(n_components = 10, # 추출할 topic의 수

                                max_iter=5,

                                topic_word_prior=0.1, doc_topic_prior=1.0, # topic = beta에 대한 파라미터 doc = alpha에 대한 파라미터

                                learning_method='online',

                                n_jobs= -1, # 사용 processor 수 (전부)

                                random_state=0)

reviews_topics = lda.fit_transform(reviews_cv)

print('#Shape of review_topics:', reviews_topics.shape)

print('#Sample of review_topics:', reviews_topics[0])

gross_topic_weights = np.mean(reviews_topics, axis=0)

print('#Sum of topic weights of documents:', gross_topic_weights)

print('#shape of topic word distribution:', lda.components_.shape)
```

```
import matplotlib.pyplot as plt

%matplotlib inline

# 혼란도를 계산하여 시각화 최적의 토픽 수 선택하기

def show_perplexity(cv, start=10, end=30, max_iter=15, topic_word_prior= 0.1,

                    doc_topic_prior=1.0):

    iter_num = []

    per_value = []

    for i in range(start, end + 1):

        lda = LatentDirichletAllocation(n_components = i, max_iter=max_iter,

                                        topic_word_prior= topic_word_prior,

                                        doc_topic_prior=doc_topic_prior,

                                        learning_method='batch', n_jobs= -1,

                                        random_state=7)

        lda.fit(cv)

        iter_num.append(i)

        pv = lda.perplexity(cv)

        per_value.append(pv)

        print(f'n_components: {i}, perplexity: {pv:0.3f}')

    plt.plot(iter_num, per_value, 'g-')

    plt.show()

    return start + per_value.index(min(per_value))

print("n_components with minimum perplexity:",

      show_perplexity(reviews_cv, start=5, end=15))
```

```
n_components: 5, perplexity: 44.677
n_components: 6, perplexity: 45.080
n_components: 7, perplexity: 44.173
n_components: 8, perplexity: 45.295
n_components: 9, perplexity: 46.541
n_components: 10, perplexity: 47.333
n_components: 11, perplexity: 49.053
n_components: 12, perplexity: 49.112
n_components: 13, perplexity: 49.494
n_components: 14, perplexity: 49.010
n_components: 15, perplexity: 48.877
```



혼란도가 가장 낮을 때: 토픽 수가 7일 때

```
from gensim.corpora.dictionary import Dictionary

# 전처리한 texts는 문자열이므로 doc2bow에 적용할 수 있도록 유니코드 토큰 배열로 변환

tokenized_texts = [text.split() for text in texts]

# 토큰화 결과로부터 dictionary 생성

dictionary = Dictionary(tokenized_texts)

print('#Number of initial unique words in documents:', len(dictionary))

# 문서 빈도수가 너무 적거나 높은 단어를 필터링하고 특성을 단어의 빈도 순으로 선택

dictionary.filter_extremes(keep_n=2000, no_below=5, no_above=0.5)

print('#Number of unique words after removing rare and common words:', len(dictionary))

# 카운트 벡터로 변환

corpus = [dictionary.doc2bow(tokens) for tokens in tokenized_texts]

print('#Number of unique tokens: %d' % len(dictionary))

print('#Number of documents: %d' % len(corpus))
```

```
#Number of initial unique words in documents: 3957
#Number of unique words after removing rare and common words: 586
#Number of unique tokens: 586
#Number of documents: 1000
```

```
from gensim.models import LdaModel

num_topics = 10

passes = 5

%time model = LdaModel(corpus=corpus, id2word=dictionary,₩

                       passes=passes, num_topics=num_topics, ₩

                       random_state=7)
```

```
CPU times: total: 5.12 s
Wall time: 9.22 s
```

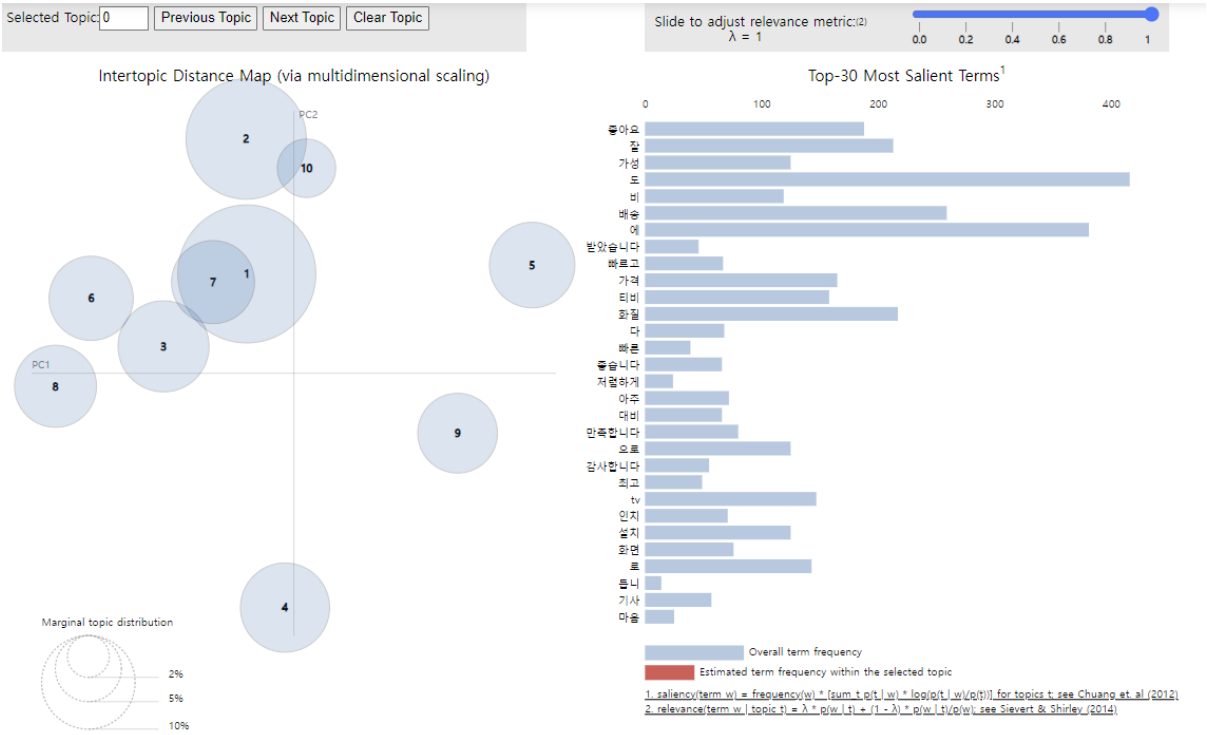
```
model.print_topics(num_words=10)
```

```
[(0,
  '0.072*"에" + 0.034*"도" + 0.029*"다" + 0.024*"으로" + 0.019*"로" + 0.017*"가격" + 0.015*"화질" + 0.014*"마음" + 0.014*"티비" + 0.013*"릅니"'),
 (1,
  '0.062*"가성" + 0.059*"비" + 0.032*"잘" + 0.020*"최고" + 0.018*"좋은" + 0.017*"로" + 0.017*"티비" + 0.015*"제품" + 0.015*"에" + 0.015*"tv"'),
 (2,
  '0.034*"도" + 0.028*"에" + 0.025*"tv" + 0.017*"배송" + 0.016*"화질" + 0.016*"가격" + 0.016*"했는데" + 0.015*"로" + 0.013*"제품" + 0.013*"너무"'),
 (3,
  '0.090*"도" + 0.060*"배송" + 0.041*"빠르고" + 0.025*"에" + 0.024*"화질" + 0.023*"설치" + 0.023*"제품" + 0.016*"기사" + 0.015*"님" + 0.011*"좋고"'),
 (4,
  '0.060*"도" + 0.029*"화질" + 0.021*"설치" + 0.020*"화면" + 0.019*"에" + 0.019*"티비" + 0.019*"배송" + 0.018*"으로" + 0.014*"사용" + 0.013*"가격"'),
 (5,
  '0.123*"좋아요" + 0.033*"아주" + 0.033*"도" + 0.031*"화질" + 0.017*"에" + 0.014*"티비" + 0.014*"너무" + 0.013*"좋네요" + 0.013*"생각" + 0.012*"제품"'),
 (6,
  '0.063*"배송" + 0.060*"에" + 0.049*"가격" + 0.030*"대비" + 0.026*"가성" + 0.025*"빠른" + 0.022*"감사합니다" + 0.020*"제품" + 0.019*"비" + 0.018*"도"'),
 (7,
  '0.034*"티비" + 0.032*"화질" + 0.029*"좋습니다" + 0.016*"도" + 0.015*"인치" + 0.015*"ㅅ" + 0.014*"ㄱ" + 0.013*"가성" + 0.012*"그냥" + 0.012*"으로"'),
 (8,
  '0.039*"에" + 0.022*"사용" + 0.022*"를" + 0.021*"을" + 0.020*"티비" + 0.019*"구매" + 0.018*"도" + 0.017*"제품" + 0.016*"tv" + 0.014*"의"'),
 (9,
  '0.104*"잘" + 0.042*"받았습니다" + 0.031*"도" + 0.026*"저렴하게" + 0.024*"만족합니다" + 0.019*"화질" + 0.017*"으로" + 0.017*"tv" + 0.016*"로" + 0.014*"만족"')]
```

```
print("#topic distribution of the first document: ", model.get_document_topics(corpus)[0])
```

```
#topic distribution of the first document:  [(0, 0.012502679), (1, 0.8874813), (2, 0.012501591), (3, 0.012501391), (4, 0.012501343), (5, 0.01250402), (6, 0.012501221), (7, 0.012500917), (8, 0.012501777), (9, 0.012503809)]
```

```
import pyLDAvis

import pyLDAvis.gensim_models as gensimvis

pyLDAvis.enable_notebook()

# feed the LDA model into the pyLDAvis instance

lda_viz = gensimvis.prepare(model, corpus, dictionary)

lda_viz
```



#혼란도와 토픽 응집도를 이용한 최적값 선택

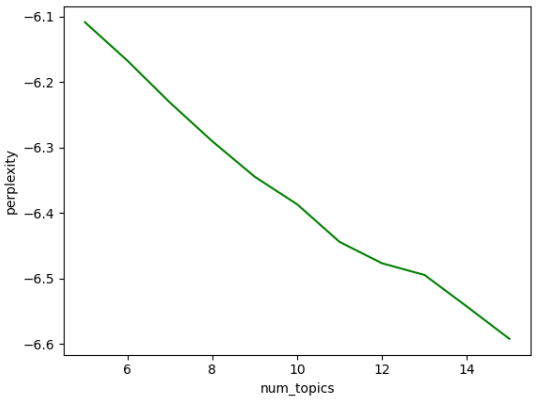
```
from gensim.models import CoherenceModel

cm = CoherenceModel(model=model, corpus=corpus, coherence='u_mass')

coherence = cm.get_coherence()

print(coherence)
```

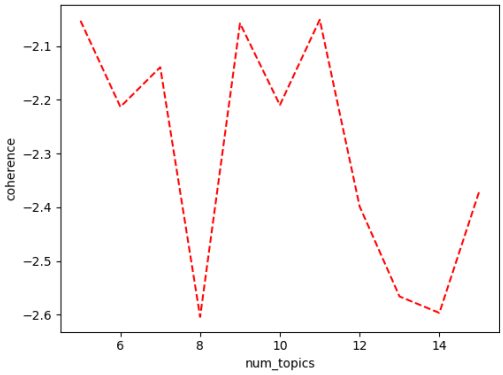
```
-3.00232934585742
```

```
def show_coherence(corpus, dictionary, start=5, end=15):

    iter_num = []

    per_value = []

    coh_value = []

    for i in range(start, end + 1):

        model = LdaModel(corpus=corpus, id2word=dictionary,

                chunksize=1000, num_topics=i,
```

```
                    random_state=7)
        iter_num.append(i)
        pv = model.log_perplexity(corpus)
        per_value.append(pv)
        cm = CoherenceModel(model=model, corpus=corpus,
                            coherence='u_mass')
        cv = cm.get_coherence()
        coh_value.append(cv)
        print(f'num_topics: {i}, perplexity: {pv:0.3f}, coherence: {cv:0.3f}')
    plt.plot(iter_num, per_value, 'g-')
    plt.xlabel("num_topics")
    plt.ylabel("perplexity")
    plt.show()
    plt.plot(iter_num, coh_value, 'r--')
    plt.xlabel("num_topics")
    plt.ylabel("coherence")
    plt.show()
show_coherence(corpus, dictionary, start=5, end=15)
```

```
num_topics: 5, perplexity: -6.109, coherence: -2.053
num_topics: 6, perplexity: -6.167, coherence: -2.213
num_topics: 7, perplexity: -6.231, coherence: -2.139
num_topics: 8, perplexity: -6.291, coherence: -2.604
num_topics: 9, perplexity: -6.345, coherence: -2.057
num_topics: 10, perplexity: -6.387, coherence: -2.210
num_topics: 11, perplexity: -6.444, coherence: -2.050
num_topics: 12, perplexity: -6.477, coherence: -2.398
num_topics: 13, perplexity: -6.495, coherence: -2.566
num_topics: 14, perplexity: -6.543, coherence: -2.597
num_topics: 15, perplexity: -6.592, coherence: -2.370
```





num_topics: 11, perplexity: -6.444, coherence: -2.050 인 모델

⑤ 전처리된 상품평과 원본 데이터에 있는 Score를 토대로 감성 분석을 수행하라 (7).

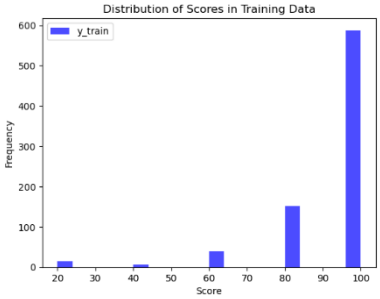
- 분류/회귀 모델을 사용하여 리뷰를 긍정/부정으로 분류/회귀
- 데이터를 훈련/테스트로 분리하여 모델을 훈련하고 평가
- 평가 지표는 정확도 및 F1점수를 사용

```
from sklearn.model_selection import train_test_split

# 전처리된 상품평 데이터

X = target_data['Reviews']

# 점수 데이터(Score)

y = target_data['Score']

# train-test 데이터 분리

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Train set size: ", len(X_train))

print("Test set size: ", len(X_test))

# TF-IDF 벡터화

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report

tfidf_vectorizer = TfidfVectorizer(max_features=50)

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

X_test_tfidf = tfidf_vectorizer.transform(X_test)

# 로지스틱 회귀 모델 학습: 긍정/부정

lm = LogisticRegression()

lm.fit(X_train_tfidf, y_train)

print('# Regression Train set R^2 score: {:.3f}'.format(lm.score(X_train_tfidf, y_train)))

print('# Regression Test set R^2 score: {:.3f}'.format(lm.score(X_test_tfidf, y_test)))
```

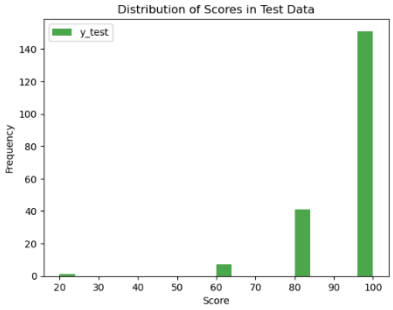
```
# Regression Train set R^2 score: 0.735
# Regression Test set R^2 score: 0.755
```

Score 몇 점부터 긍정으로 판단할 것인지 임계값을 정하기 위해 y의 분포를 먼저 확인하였다.

```
import matplotlib.pyplot as plt

# 학습 데이터의 y값 분포 확인

plt.hist(y_train, bins=20, color='blue', alpha=0.7, label='y_train')

plt.xlabel('Score')

plt.ylabel('Frequency')

plt.title('Distribution of Scores in Training Data')

plt.legend()

plt.show()
```




```
# 테스트 데이터의 y값 분포 확인

plt.hist(y_test, bins=20, color='green', alpha=0.7, label='y_test')

plt.xlabel('Score')

plt.ylabel('Frequency')

plt.title('Distribution of Scores in Test Data')

plt.legend()

plt.show()
```



60점까지를 부정, 그 이상을 긍정으로 분류하기로 했다. 즉 임계값을 60점으로 설정하였다.

```
y_train_senti = (y_train > 60)

y_test_senti = (y_test > 60)

y_train_predict = (lm.predict(X_train_tfidf) > 60)

y_test_predict = (lm.predict(X_test_tfidf) > 60)


from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score

print("# Accuracy for train set: {:.3f}".format(accuracy_score(y_train_senti, y_train_predict)))

print("# Precision for train set: {:.3f}".format(precision_score(y_train_senti, y_train_predict)))

print("# Recall for train set: {:.3f}".format(recall_score(y_train_senti, y_train_predict)))

print("# F1 for train set: {:.3f}".format(f1_score(y_train_senti, y_train_predict)))


print("# Accuracy for test set: {:.3f}".format(accuracy_score(y_test_senti, y_test_predict)))

print("# Precision for test set: {:.3f}".format(precision_score(y_test_senti, y_test_predict)))

print("# Recall for test set: {:.3f}".format(recall_score(y_test_senti, y_test_predict)))

print("# F1 for test set: {:.3f}".format(f1_score(y_test_senti, y_test_predict)))
```

```
# Accuracy for train set: 0.925
# Precision for train set: 0.925
# Recall for train set: 1.000
# F1 for train set: 0.961
# Accuracy for test set: 0.960
# Precision for test set: 0.960
# Recall for test set: 1.000
# F1 for test set: 0.980
```