Project 3.

Document search engine & Classification and Clustering

TEAM 06 김세찬 오재원 이승아 표승희 홍지현

Part 1. Document search engine

1.Document Pre-processing & Indexing

(1) Stopwords

문서 분석) Document에서 자주 나오는 단어 중 100번 이상 검색된 단어(총 45개)

[‘wing’ , ‘hypersonic’ , ‘)’ , ‘were’ , ‘solution’ , ‘transition’ , ‘body’ , ‘(‘, ‘temperature’ , ‘surface’ , ‘from’ , ‘plate’ , ‘be’ , ‘buckling’ , ‘heat’ , ‘flutter’ , ‘wa’ 'layer', 'theory', 'method', 'number', 'mach', 'an', 'that', 'jet', 'shock', 'boundary', 'by', 'pressure', 'on', 'at', 'with', 'are', 'flow', 'for', 'is', 'to', 'in', 'a', 'and', ',', '.', 'of', 'the']

- 총 document 개수(1400개)를 고려하였을 때, 상당히 작은 개수
- 표시된 단어들의 경우, 품사가 명사인 단어들이 상당수 존재 : stopword로 사용하기 무리가 있음
- 실제 성능또한 nltk의 stopwords를 사용했을 때보다 떨어짐

∴ nltk에서 제공하는 stopwords를 사용하여 전처리

1.Document Pre-processing & Indexing

(2) Pos-tagging & lemmatize

- Query term 중 명사와 그 주위의 형용사(수식)에 대해 가중치 부여 방안 선택
- Document 또한 pos-tagging을 통한 품사 분석 & lemmatize 필요

∴ nltk에서 제공하는 pos-tag와 WordNetLemmatizer을 사용하여 전처리

Ex) Words : Pos-tag "NNS" > lemmatize > Word : Pos-tag "NN"

2. Query Pre-processing & Query Expansion

(1) Query Pre-Processing

- Whoosh 검색 엔진은 query와 document 내의 동일한 단어를 매칭함
- Query와 document는 동일한 방식으로 전처리되어야 함

∴ document와 마찬가지로 nltk에서 제공하는
pos-tag와 WordNetLemmatizer을 사용하여 전처리

2. Query Pre-processing & Query Expansion

(2) Term Weighting based on Linguistics

- 품사를 추출하고, 품사 간의 위치에 따라 가중치를 부여하여 성능을 높일 수 있음
- Query의 품사 및 단어 간 위치에 따라 다른 가중치를 부여

⇒ ‘명사’ 와 ‘명사 주위를 수식하는 형용사’ 에 가중치를 부여할 때 높은 성능을 보임

2. Query Pre-processing & Query Expansion

(3) Query Expansion_Keyword

- Query와 연관 있는 keyword를 query에 추가하는 방식
- whoosh key_terms를 사용하여 검색 수행 결과 상위 document에서 자주 나오는 단어를 keyword로 추출
- 본 프로젝트에서는 상위 10개의 문서에서 자주 나오는 단어 7개를 선정하여 query에 추가

2. Query Pre-processing & Query Expansion

(3) Query Expansion_n-grams

- Query의 연속된 term에 대한 가중치를 부여
- Unigram 과 bigram을 혼합하여 사용하는 경우 가장 높은 성능을 보임

type	score
unigram	0.3699976150960649
bigram	0.28644325928144615
trigram	0.16368008603698192
unigram + bigram	0.34733665544845316
unigram^1.5 + bigram	0.3563802002219599
unigram^3 + bigram	0.37181867273084646
unigram^3.5 + bigram	0.3727205642890792
unigram^4 + bigram	0.37185477314507837
unigram^5 + bigram	0.3700769526692274
unigram^7 + bigram	0.368968747121309

3. Scoring

- Tf, Idf 등의 정보를 바탕으로 scoring function을 정의
- BM25의 다양한 변형 꼴에 대해 검색 엔진 수행

BM25	$\left(\log\left(\frac{dc}{df+1}\right)+1\right)\cdot\frac{tf\cdot(K1+1)}{K_1\cdot\left(1-B+B\cdot\left(\frac{fl}{avgfl}\right)\right)+tf}$	0.391465760632322
Robertson et al.	$\log\left(\frac{dc-df+0.5}{df+0.5}\right)\cdot\frac{tf}{K_1\cdot\left(1-B+B\cdot\left(\frac{fl}{avgfl}\right)\right)+tf}$	0.390086739707641
Lucene (accurate)	$\log\left(1+\frac{dc-df+0.5}{df+0.5}\right)\cdot\frac{tf}{K_1\cdot\left(1-B+B\cdot\left(\frac{fl}{avgfl}\right)\right)+tf}$	0.390392830810121
ATIRE	$\log\left(\frac{dc}{df}\right)\cdot\frac{tf\cdot(K1+1)}{K_1\cdot\left(1-B+B\cdot\left(\frac{fl}{avgfl}\right)\right)+tf}$	0.390627671549542
$TF_{l\circ\delta\circ\rho}\times IDF$	$\log\left(\frac{dc+1}{df}\right)\cdot\left(1+\log\left(1+\log\left(\frac{tf}{\left(1-B+B\cdot\left(\frac{fl}{avgfl}\right)\right)}+\delta\right)\right)\right)$	0.395575565451715

⇒ 평균 BPREF가 가장 높은 $TF_{l\circ\delta\circ\rho}\times IDF$ 을 선택

4. Searching

- 총 두 번의 검색을 통해 검색 결과를 보여줌
- 1. 전처리 및 품사에 따른 weighting이 된 query로 첫 번째 결과를 도출
- 2. 추출한 keyword, bi-gram에 작은 가중치를 두어 query에 추가 expansion된 query로 두 번째 결과를 도출

	평균 BPREF	score가 0인 문서
첫 번째 검색만 사용	0.3681165190164179	15개
두 번째 검색까지 사용	0.3955755654517156	18개

Part 2. Classification and Clustering

R2-1. 영어 신문 기사 분류

(1) 데이터 전처리

① 특수 문자 제거(특수 문자는 중요한 정보가 없는 경우가 많음)
② 길이가 3 이하인 단어 제거(영어의 경우, 길이가 짧은 단어는 대부분 불용어)
③ 전체 단어 소문자 변환(document의 대부분 글자가 소문자)

```
import pandas as pd
import nltk

def preProcess(documents):
    news_df = pd.DataFrame({'document':documents})
    # 특수 문자 제거
    news_df['clean_doc'] = news_df['document'].str.replace("[^a-zA-Z]", " ")
    # 길이가 3이하인 단어는 제거 (길이가 짧은 단어 제거)
    news_df['clean_doc'] = news_df['clean_doc'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>3]))
    # 전체 단어에 대한 소문자 변환
    news_df['clean_doc'] = news_df['clean_doc'].apply(lambda x: x.lower())

    return news_df['clean_doc']
```

R2-1. 영어 신문 기사 분류

(2) Naive Bayes Classifier

① CountVectorizer로 Stopwords 제거 후, 벡터화
② TfidfTransformer로 tf-idf 가중치 부여
③ MultinomialNB로 classification 진행. Alpha값은 0.01 ~ 0.5의 값을 0.01 단위로 grid_search하여 가장 적합한 파라미터 적용

```
# TODO - 2-1-1. Build pipeline for Naive Bayes Classifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
clf_nb = Pipeline([('vect', CountVectorizer(stop_words = 'english')),
                   ('tfidf', TfidfTransformer(use_idf=True)),
                   ('clf', MultinomialNB(alpha=0.05))])
clf_nb.fit(preProcess(train_data.data), train_data.target)
docs_test = test_data.data

predicted = clf_nb.predict(docs_test)
print("NB accuracy : %d / %d" % (np.sum(predicted==test_data.target), len(test_data.target)))
print(metrics.classification_report(test_data.target, predicted, target_names=test_data.target_names))
print(metrics.confusion_matrix(test_data.target, predicted))
```

R2-1. 영어 신문 기사 분류

(2) Naive Bayes Classifier

① 44개의 test 데이터 중에서 32개를 정확하게 분류, accuracy = 73%
② Books와 Opinion 카테고리 분류에서 낮은 성능

```
NB accuracy : 32 / 44
              precision    recall  f1-score   support

        arts       1.00      0.80      0.89         5
       books       0.50      0.83      0.62         6
    business       0.71      1.00      0.83         5
      movies       0.75      0.50      0.60         6
     opinion       0.50      0.20      0.29         5
      sports       1.00      0.80      0.89         5
          us       0.83      0.83      0.83         6
       world       0.71      0.83      0.77         6

    accuracy                           0.73        44
   macro avg       0.75      0.73      0.72        44
weighted avg       0.75      0.73      0.71        44

[[4 1 0 0 0 0 0 0]
 [0 5 0 0 1 0 0 0]
 [0 0 5 0 0 0 0 0]
 [0 3 0 3 0 0 0 0]
 [0 0 2 0 1 0 1 1]
 [0 1 0 0 0 4 0 0]
 [0 0 0 0 0 0 5 1]
 [0 0 0 1 0 0 0 5]]
```

R2-1. 영어 신문 기사 분류

(3) SVM Classifier

① CountVectorizer로 Stopwords 제거 후, 벡터화
② TfidfTransformer로 tf-idf 가중치 부여
③ SVM로 classification 진행. C = 1, gamma = 1, kernel = 'linear'
C는 1 ~ 5의 값을 1 단위로, gamma도 1 ~ 5의 값을 1 단위로 , kernel은 linear, poly, rbf로 grid_search하여 적합한 파라미터 적용

```
# TODO - 2-1-2. Build pipeline for SVM Classifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
clf_svm = Pipeline([('vect', CountVectorizer(stop_words='english')),
                    ('tfidf', TfidfTransformer()),
                     ('clf', SVC(C = 1, gamma = 1, kernel = 'linear'))])
clf_svm.fit(preProcess(train_data.data), train_data.target)

predicted = clf_svm.predict(docs_test)
print("SVM accuracy : %d / %d" % (np.sum(predicted==test_data.target), len(test_data.target)))
print(metrics.classification_report(test_data.target, predicted, target_names=test_data.target_names))
print(metrics.confusion_matrix(test_data.target, predicted))
```

R2-1. 영어 신문 기사 분류

(3) SVM Classifier

① 44개의 test 데이터 중에서 35개를 정확하게 분류, accuracy = 80%
② Opinion 카테고리 분류에서 낮은 성능
③ MultinomialNB보다 높은 정확도

```
SVM accuracy : 35 / 44
              precision    recall  f1-score   support

        arts       1.00      0.80      0.89         5
       books       0.75      1.00      0.86         6
    business       0.71      1.00      0.83         5
      movies       1.00      0.83      0.91         6
     opinion       0.33      0.20      0.25         5
      sports       0.80      0.80      0.80         5
          us       0.83      0.83      0.83         6
       world       0.83      0.83      0.83         6

    accuracy                           0.80        44
   macro avg       0.78      0.79      0.78        44
weighted avg       0.79      0.80      0.78        44

[[4 0 0 0 1 0 0 0]
 [0 6 0 0 0 0 0 0]
 [0 0 5 0 0 0 0 0]
 [0 1 0 5 0 0 0 0]
 [0 0 2 0 1 0 1 1]
 [0 1 0 0 0 4 0 0]
 [0 0 0 0 1 0 5 0]
 [0 0 0 0 0 1 0 5]]
```

R2-2. 영어 신문 기사 군집화

(1) 데이터 전처리

순서	내용	세부내용
①	Dataframe	load한 파일을 dataframe으로 생성함 label과 data를 column으로 부름
②	Text Lower	data를 모두 소문자로 전환함
③	Title Weight	data 중 title에 가중치를 부여함
④	Token > P1,P2 > lemma	tokenize 후 P1이나 P2를 거쳐 lemmatize함 (P1의 성능이 더 우수함)
⑤	Stopwords	NLTK 패키지의 stopwords를 이용함

*P1 : Tokenize > Pos_tagging > Lemmatize
*P2 : Tokenize > Stemming > Lemmatize

R2-2. 영어 신문 기사 군집화

(1) 데이터 전처리

```
# TODO - Data preprocessing and clustering
# dataframe, lower
dataframe = pd.DataFrame(data.data, data.target)
dataframe.columns=['text']

dataframe['text'] = dataframe['text'].str.lower()

# title split & weight
dataframe['text'] = dataframe['text'].str.split('\n')
dataframe2 = dataframe['text'].apply(pd.Series)

dataframe2.columns=['title', 'body', 'NAN']
del dataframe2['NAN']

dataframe2['total'] = (dataframe2['title'] + ' ')*2 + ' ' + dataframe2['body']
```

```
# nomalization (tokenize, lemmatize)
w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
lemmatizer = nltk.stem.WordNetLemmatizer()

def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

def pos_tag(text):
    pos_tokens1 = [nltk.pos_tag(w_tokenizer.tokenize(text))]
    pos_tokens2 = [[(lemmatizer.lemmatize(word, get_wordnet_pos(pos_tag))) for (word, pos_tag) in pos] for pos in pos_tokens1]
    return pos_tokens2[0]

dataframe2['lemma'] = dataframe2.total.apply(pos_tag)

# stopwords remove
sw = stopwords.words('english')
dataframe2['le_st'] = dataframe2['lemma'].apply(lambda x: ' '.join([word for word in x if word not in sw]))
```

R2-2. 영어 신문 기사 군집화

(2) Parameter 변경

Index	parameter	비고
V3	Vectorizer종류, Stop_words, analyzer, min_df, max_df, ngram_range, max_feature	7개
V4	norm, use_idf, smooth_idf, sublinear	4개
V5	random_state, init, n_init, max_iter, algorithm, precompute_distance,	6개

R2-2. 영어 신문 기사 군집화

(2) Parameter 변경

```
70      # vectorize
71      for k in range(1, 12):
72          for m in list(np.arange(0.5, 1.0, 0.05)):
73              for n in range(0, 15):
74
75                  V_list = [(k,m,n)]
76
77                  for a, b, c in V_list:
78                      vectorizer = CountVectorizer(stop_words='english',  analyzer='word', min_df=a, max_df=b) ## V3
79                      data_trans1 = vectorizer.fit_transform((dataframe2['le_st']))
80                      data_trans2 = TfidfTransformer().fit_transform(data_trans1) ## V4
81
82                      # cluster
83                      clst = KMeans(n_clusters=8, random_state=c) ## V5
84                      clst.fit(data_trans2)
85
86                      print("end %d %f %d" % (a, b, c))
87                      print(metrics.v_measure_score(data.target, clst.labels_))
```

R2-2. 영어 신문 기사 군집화

(3) 성능 향상 과정

1. Base
2. Countvectorizer : stopwords
3. Countvectorizer : min_df, max_df
   Kmeans : random_state
4. Countvectorizer : max_features
   Kmeans : n_init, max_iter
5. Text-preprocessing (Title weight)
6. 기타

	추가 variables/parameter	성능
1	(Base)	0.20652
2	V3 : Stopwords (English)	0.39794
3,4	V3 : min_df (9) V5 : random_state (10)	0.40513
5	추가적인 Preprocessing V1 : title weight (2)	0.48766
6	V4 : smooth_idf (False)	0.48891

R2-2. 영어 신문 기사 군집화

(4) Clustering 결과

```
keywords:
 ['israel', 'israeli', 'gaza', 'palestinian', 'hamas', 'netanyahu', 'rocket', 'jerusalem', 'arab', 'minister']
```

```
keywords:
 ['trump', 'biden', 'president', 'mr', 'republican', 'senate', 'vote', 'election', 'party', 'capitol']
```

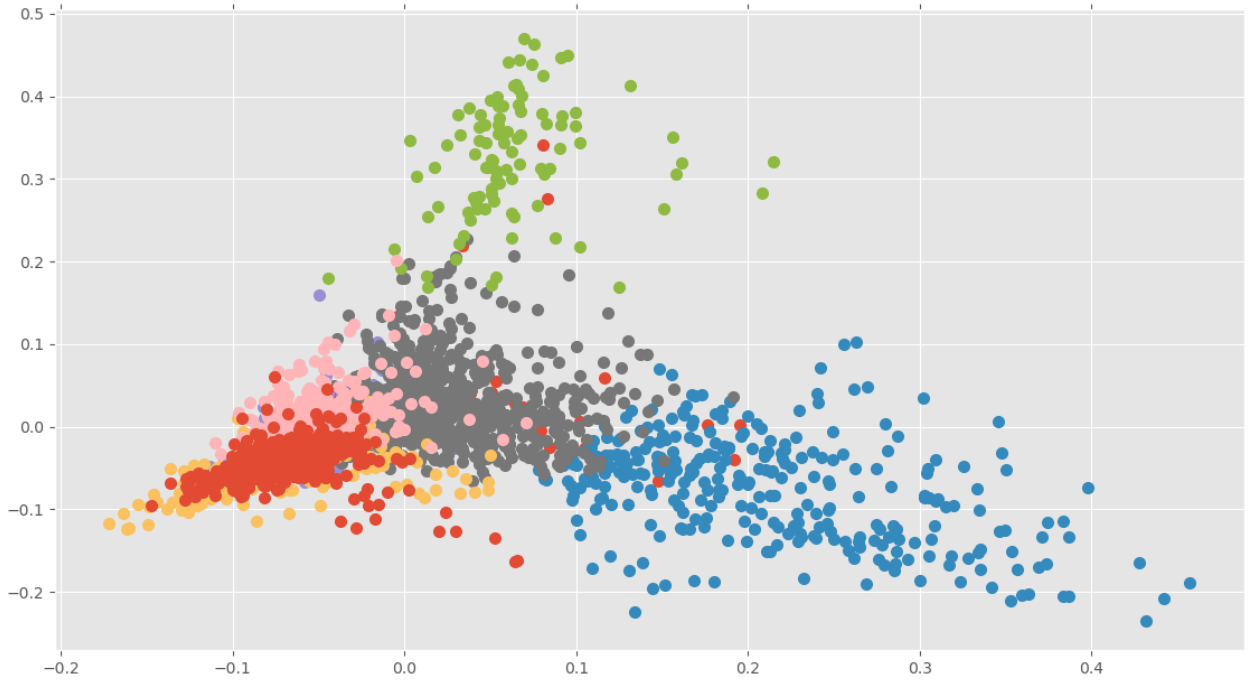
```
keywords:
 ['art', 'music', 'museum', 'artist', 'new', 'york', 'work', 'song', 'gallery', 'album']
```

대략적으로 비슷한 유형의 키워드끼리 묶임

But 여전히 'israeli' 나 'mr' 등 이상한 단어 존재
또한 'israel' 과 'trump' 는 "world" 라는 카테고리에 함께 있지 않음

R2-2. 영어 신문 기사 군집화

(4) Clustering 결과



Project 3.

Document search engine & Classification and Clustering

TEAM 06 김세찬 오재원 이승아 표승희 홍지현