# PROJECT #3

## : Document search engine & Classification and Clustering

Team 05 - 김민영, 김해찬, 오태양, 하창우

# PROJECT #3

PART I : Document Searching Engine

PART II : Document Classifier & Clustering

# PART I： Document Searching Engine

Requirement 1-1： make_index.py

Requirement 1-2： QueryResult.py

Requirement 1-3： CustomScoring.py

PART Ⅰ : Document Searching Engine

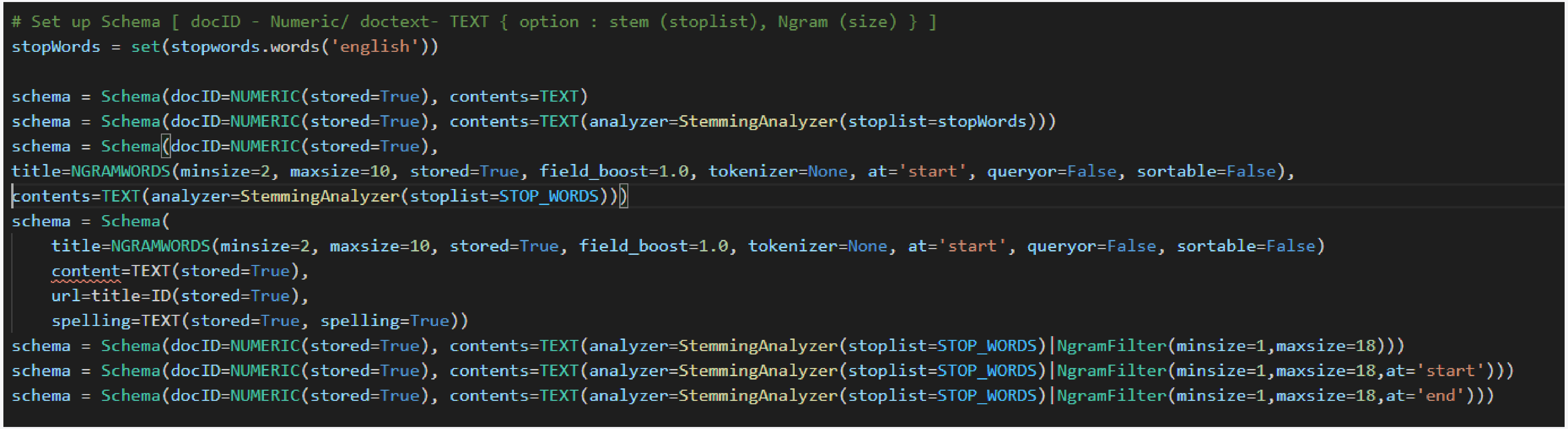
Requirement 1-1

Data Processing & Algorithm coding

- ✓ Stemming & Lemmatizing
- ✓ Removing Stopwords & Special Characters
- ✓ Spell check
- ✓ Ngram
- ✓ Giving Weightness

## PART I : Document Searching Engine

### Requirement 1-1 : make_index.py

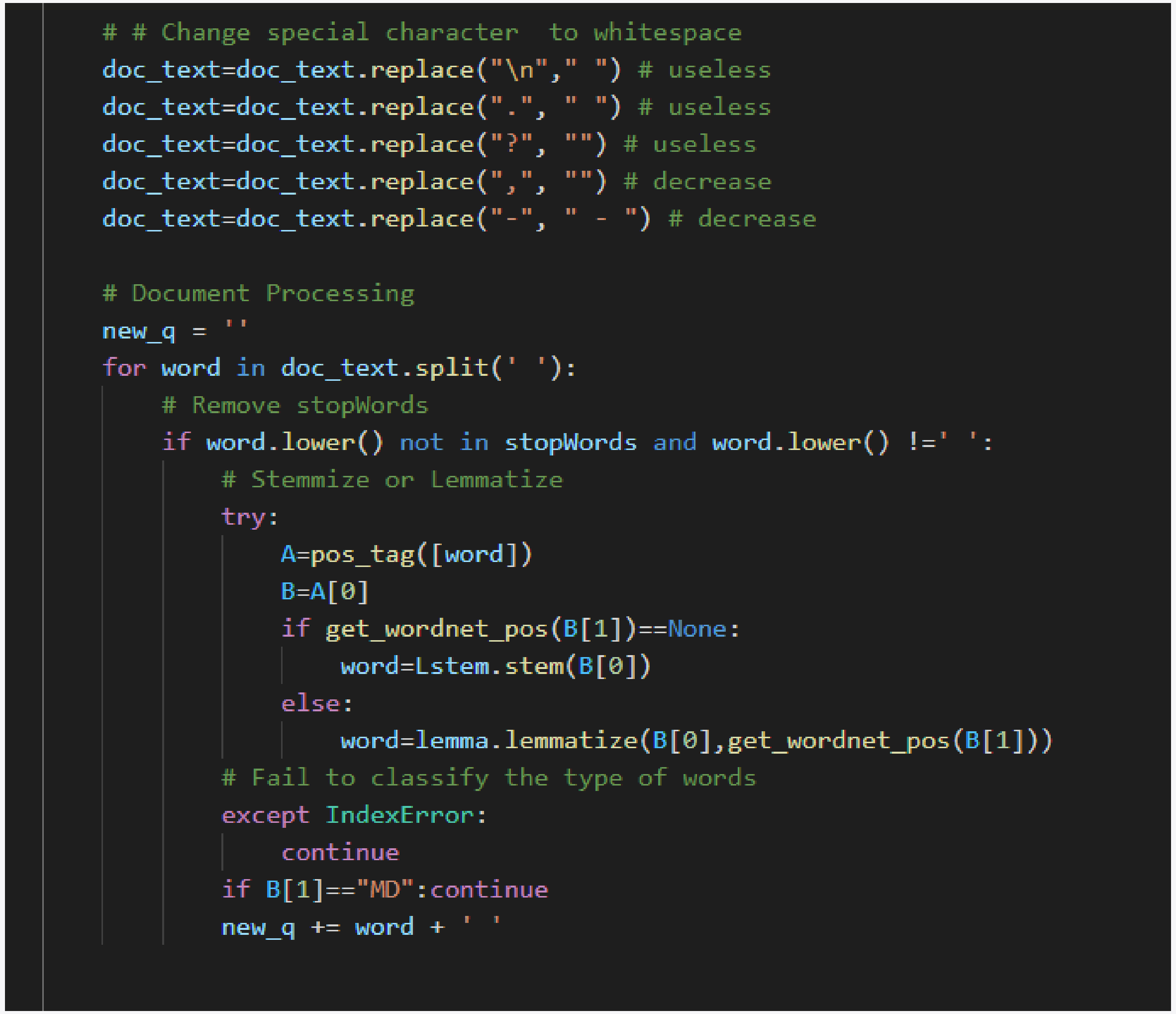


```
# Set up Schema [ docID - Numeric/ doctext- TEXT { option : stem (stoplist), Ngram (size) } ]
stopWords = set(stopwords.words('english'))

schema = Schema(docID=NUMERIC(stored=True), contents=TEXT)
schema = Schema(docID=NUMERIC(stored=True), contents=TEXT(analyzer=StemmingAnalyzer(stoplist=stopWords)))
schema = Schema(docID=NUMERIC(stored=True),
title=NGRAMWORDS(minsize=2, maxsize=10, stored=True, field_boost=1.0, tokenizer=None, at='start', queryor=False, sortable=False),
contents=TEXT(analyzer=StemmingAnalyzer(stoplist=STOP_WORDS)))
schema = Schema(
    title=NGRAMWORDS(minsize=2, maxsize=10, stored=True, field_boost=1.0, tokenizer=None, at='start', queryor=False, sortable=False)
    content=TEXT(stored=True),
    url=title=ID(stored=True),
    spelling=TEXT(stored=True, spelling=True))
schema = Schema(docID=NUMERIC(stored=True), contents=TEXT(analyzer=StemmingAnalyzer(stoplist=STOP_WORDS)|NgramFilter(minsize=1,maxsize=18)))
schema = Schema(docID=NUMERIC(stored=True), contents=TEXT(analyzer=StemmingAnalyzer(stoplist=STOP_WORDS)|NgramFilter(minsize=1,maxsize=18,at='start')))
schema = Schema(docID=NUMERIC(stored=True), contents=TEXT(analyzer=StemmingAnalyzer(stoplist=STOP_WORDS)|NgramFilter(minsize=1,maxsize=18,at='end')))
```

- ✓ Stemming
- ✓ Removing Stopwords – Which stopwords list is best?
- ✓ Ngram – Is it useful?

# PART I： Document Searching Engine

## Requirement 1-1 ： make_index.py

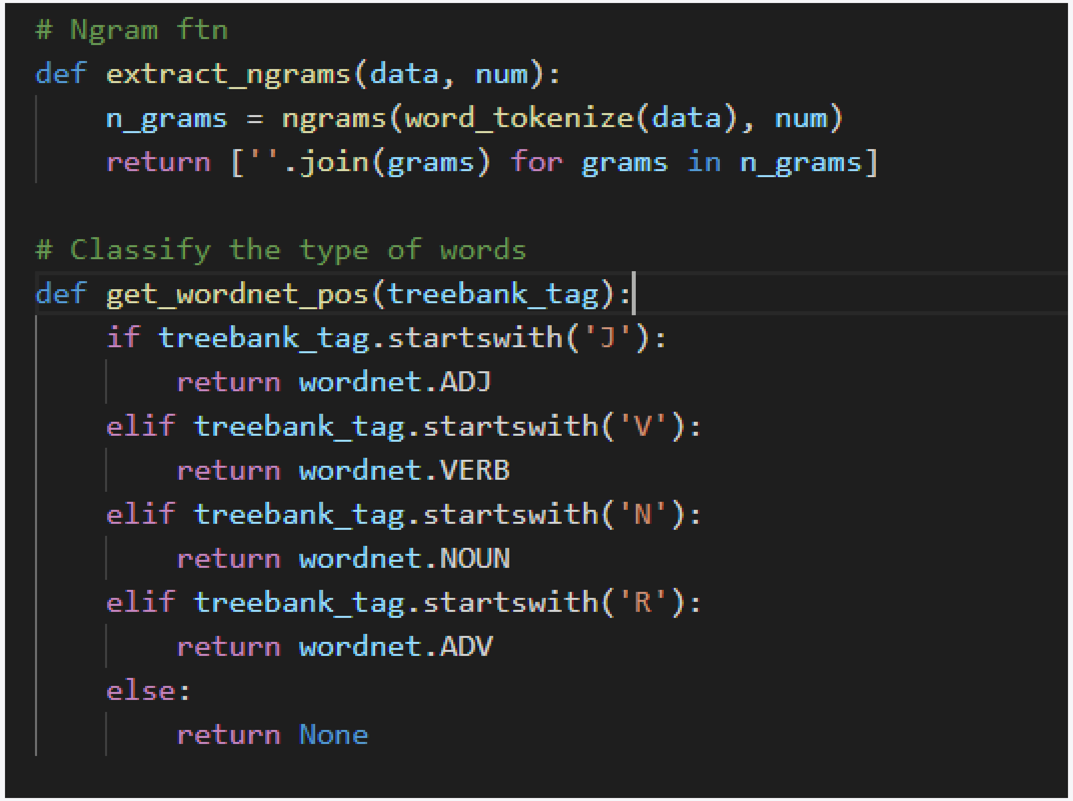


```
# # Change special character  to whitespace
doc_text=doc_text.replace("\n"," ") # useless
doc_text=doc_text.replace(".", " ") # useless
doc_text=doc_text.replace("?", "") # useless
doc_text=doc_text.replace(",", "") # decrease
doc_text=doc_text.replace("-", " - ") # decrease

# Document Processing
new_q = ''
for word in doc_text.split(' '):
    # Remove stopWords
    if word.lower() not in stopWords and word.lower() !=' ':
        # Stemmize or Lemmatize
        try:
            A=pos_tag([word])
            B=A[0]
            if get_wordnet_pos(B[1])==None:
                word=Lstem.stem(B[0])
            else:
                word=lemma.lemmatize(B[0],get_wordnet_pos(B[1]))
        # Fail to classify the type of words
        except IndexError:
            continue
        if B[1]=="MD":continue
        new_q += word + ' '
```

- ✓ Stemming & Lemmatizing
- ✓ Removing Stopwords & Special Characters

# PART I : Document Searching Engine

## Requirement 1-2 : QueryResult.py

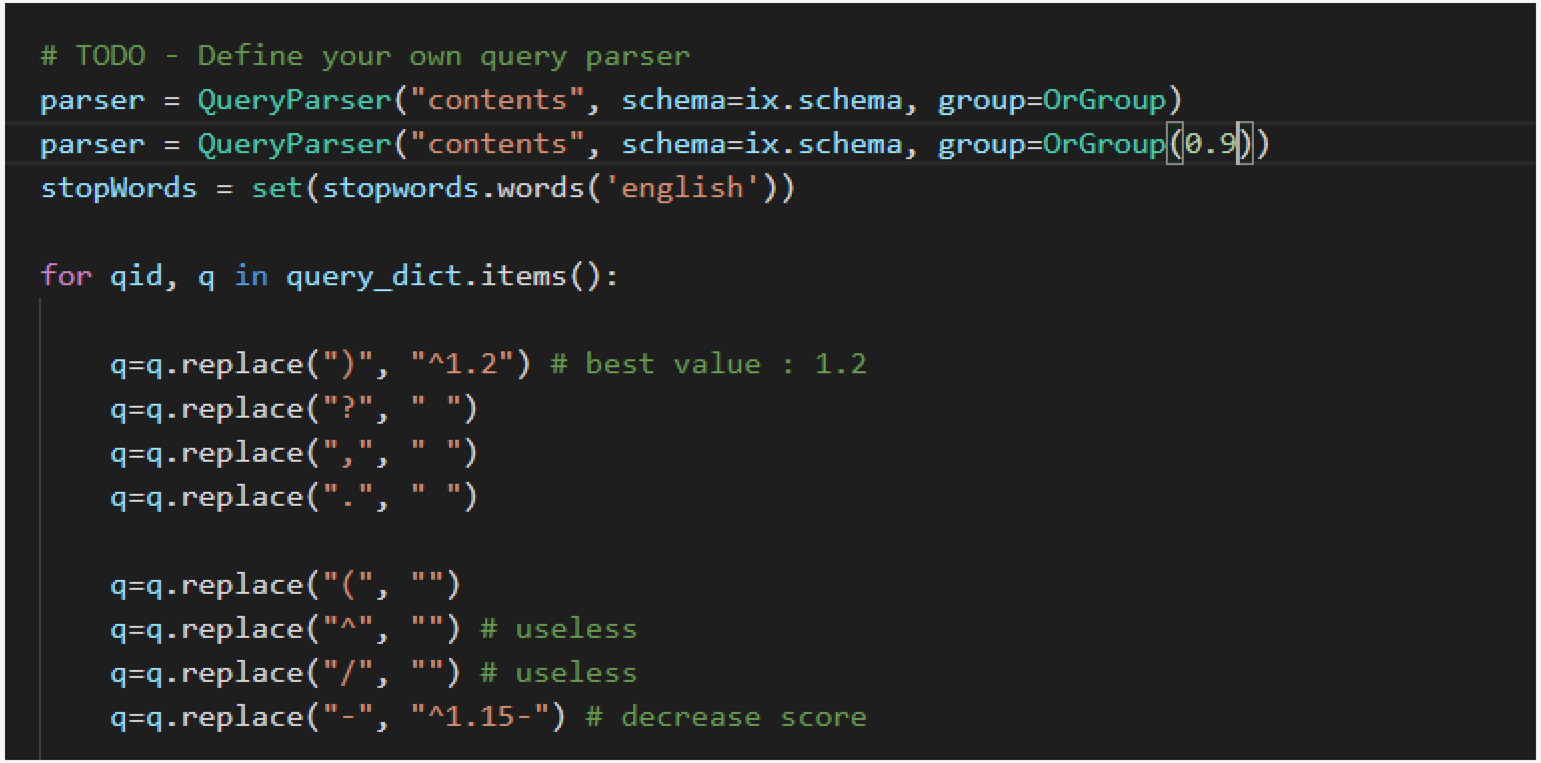


```python
# Ngram ftn
def extract_ngrams(data, num):
    n_grams = ngrams(word_tokenize(data), num)
    return [''.join(grams) for grams in n_grams]

# Classify the type of words
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 None
```

- ✓ Stemming & Lemmatizing
- ✓ Ngram
- ✓ Giving Weightness

# PART I : Document Searching Engine

## Requirement 1-2 : QueryResult.py

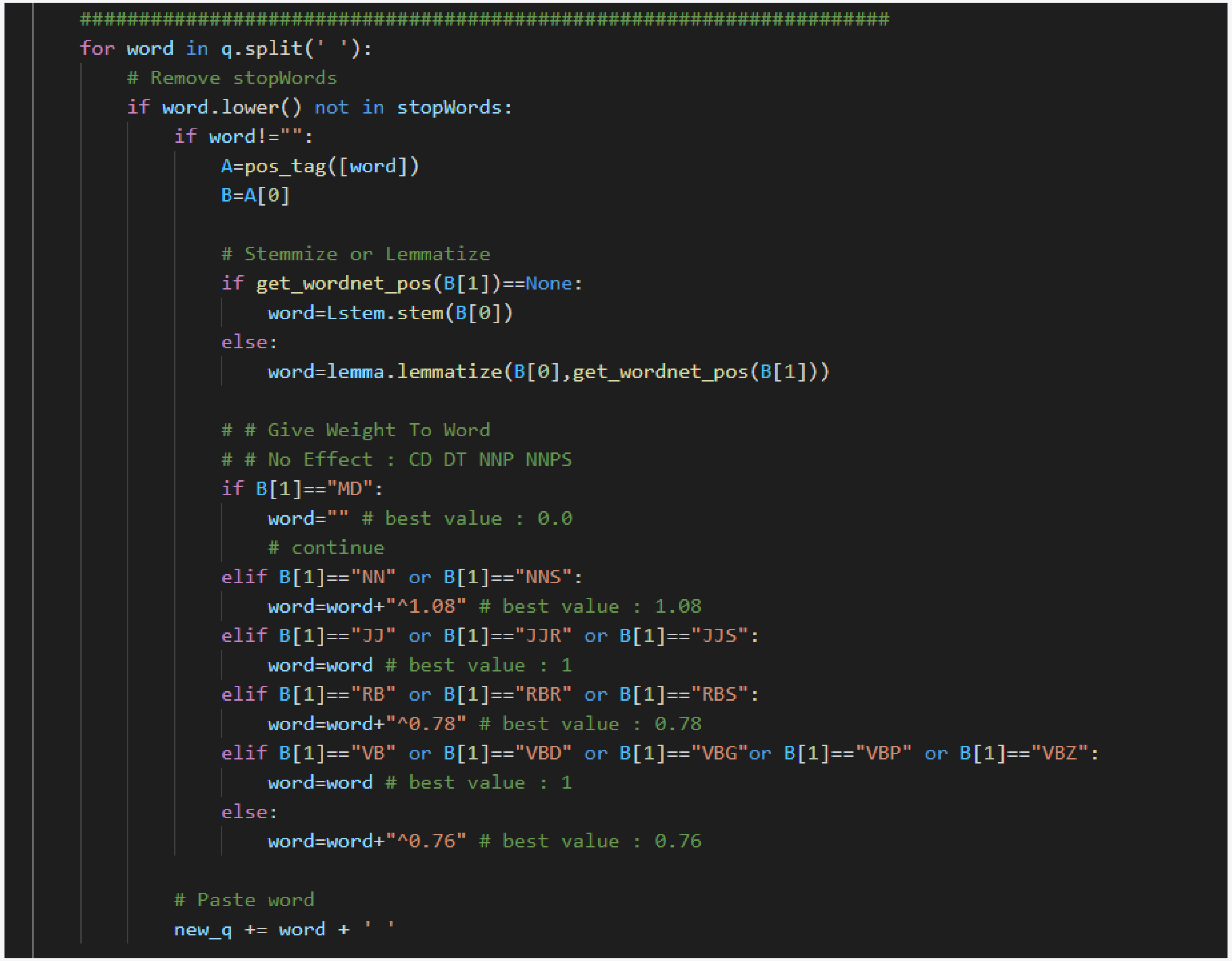


```
# TODO - Define your own query parser
parser = QueryParser("contents", schema=ix.schema, group=OrGroup)
parser = QueryParser("contents", schema=ix.schema, group=OrGroup(0.9))
stopWords = set(stopwords.words('english'))

for qid, q in query_dict.items():

    q=q.replace(")", "^1.2") # best value : 1.2
    q=q.replace("?", " ")
    q=q.replace(",", " ")
    q=q.replace(".", " ")

    q=q.replace("(", "")
    q=q.replace("^", "") # useless
    q=q.replace("/", "") # useless
    q=q.replace("-", "^1.15-") # decrease score
```

- ✓ Removing Special Characters
- ✓ Considering Redundant word
- ✓ Giving Weightness

# PART I： Document Searching Engine

## Requirement 1-2： QueryResult.py

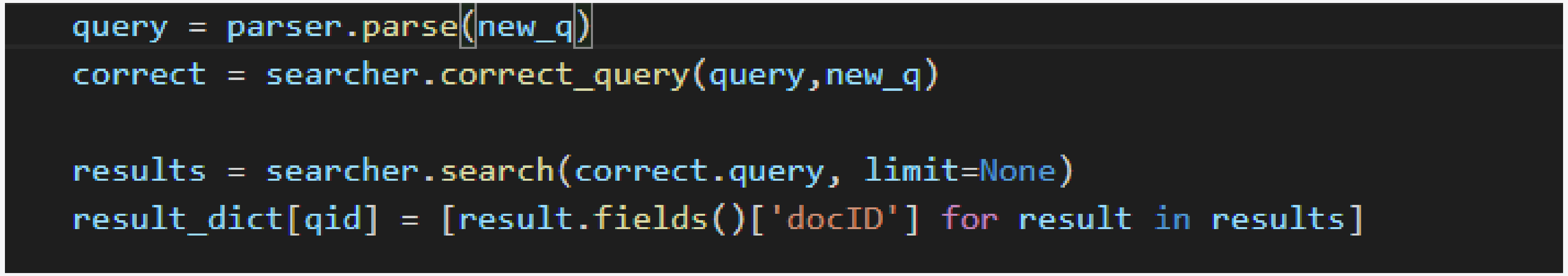


```
####################################################################
for word in q.split(' '):
    # Remove stopWords
    if word.lower() not in stopWords:
        if word!="":
            A=pos_tag([word])
            B=A[0]

            # Stemmize or Lemmatize
            if get_wordnet_pos(B[1])==None:
                word=Lstem.stem(B[0])
            else:
                word=lemma.lemmatize(B[0],get_wordnet_pos(B[1]))

            # # Give Weight To Word
            # # No Effect : CD DT NNP NNPS
            if B[1]=="MD":
                word="" # best value : 0.0
                # continue
            elif B[1]=="NN" or B[1]=="NNS":
                word=word+"^1.08" # best value : 1.08
            elif B[1]=="JJ" or B[1]=="JJR" or B[1]=="JJS":
                word=word # best value : 1
            elif B[1]=="RB" or B[1]=="RBR" or B[1]=="RBS":
                word=word+"^0.78" # best value : 0.78
            elif B[1]=="VB" or B[1]=="VBD" or B[1]=="VBG"or B[1]=="VBP" or B[1]=="VBZ":
                word=word # best value : 1
            else:
                word=word+"^0.76" # best value : 0.76

        # Paste word
        new_q += word + ' '
```

- ✓ Stemming & Lemmatizing
- ✓ Removing Stopwords
- ✓ Giving Weightness

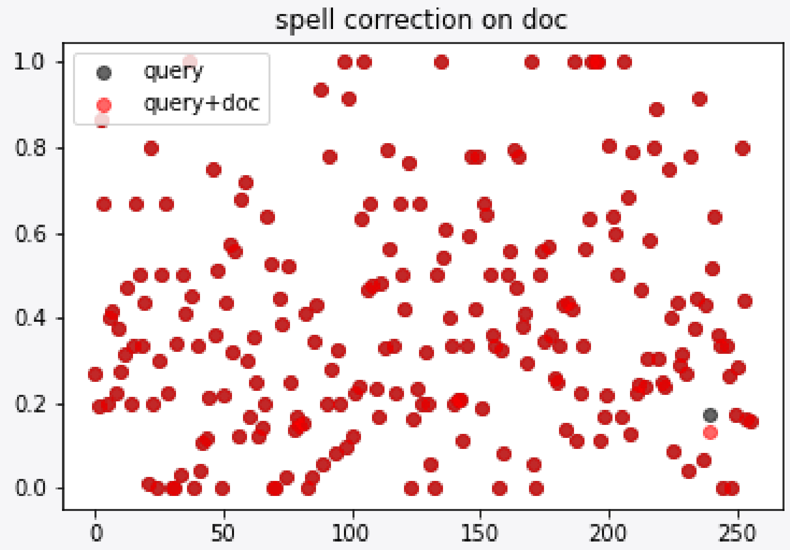
## PART I： Document Searching Engine

**Requirement 1-1 & 2：** make_index.py & QueryResult.py

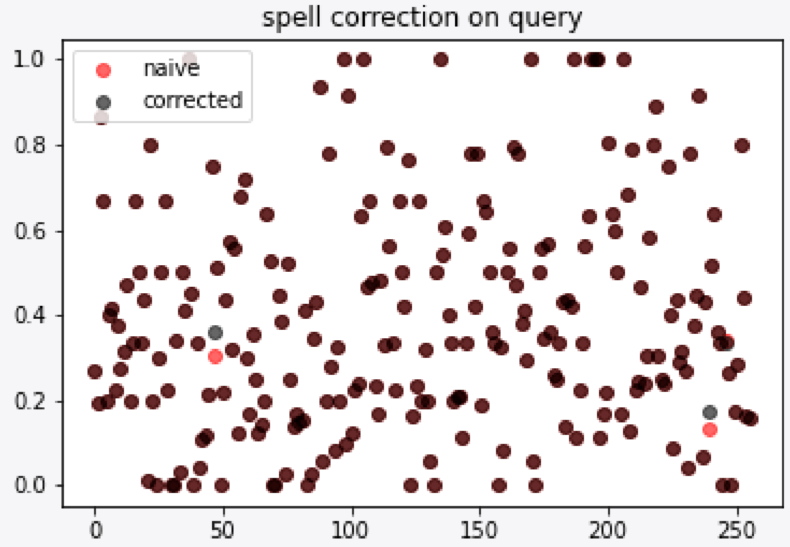


```
query = parser.parse(new_q)
correct = searcher.correct_query(query,new_q)

results = searcher.search(correct.query, limit=None)
result_dict[qid] = [result.fields()['docID'] for result in results]
```

✓ Spell check



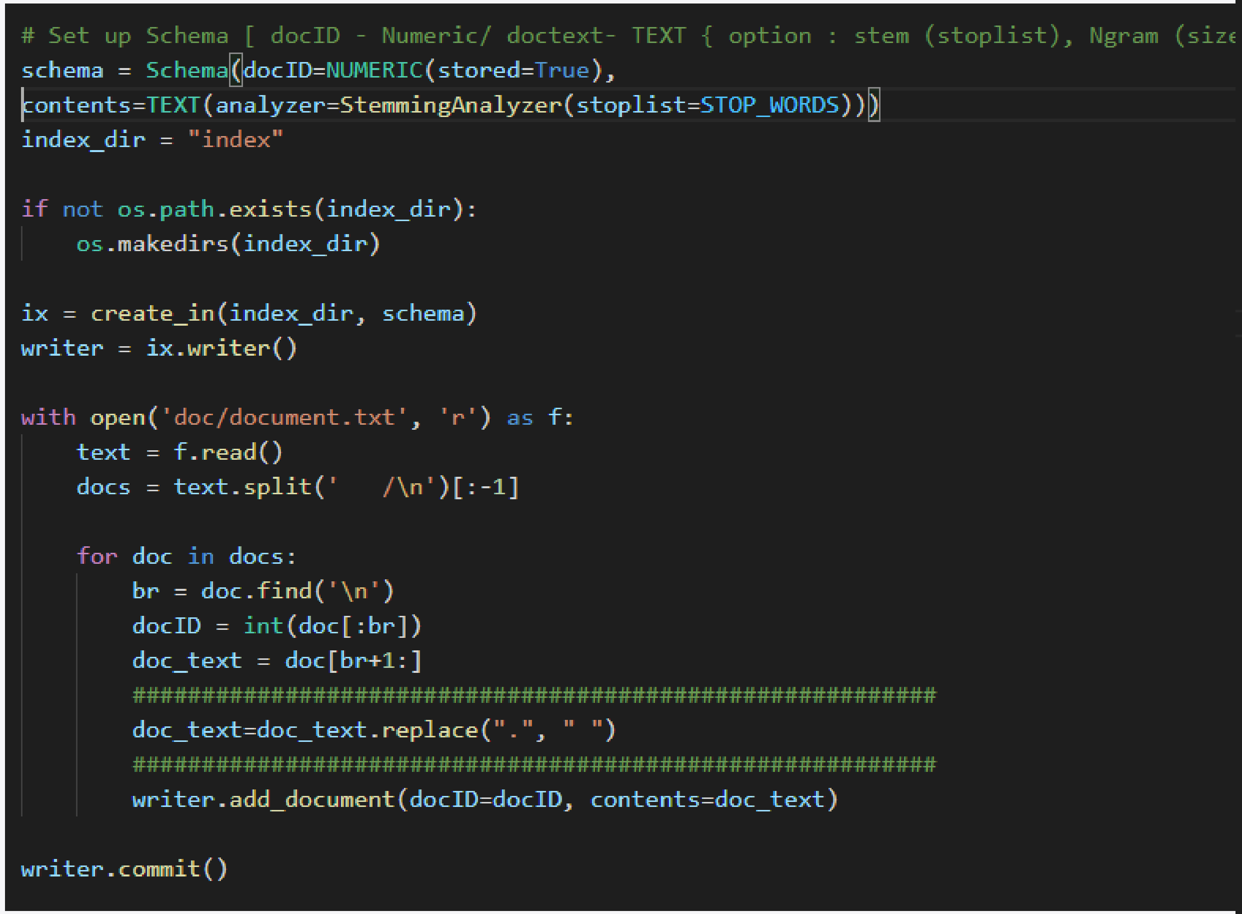
spell correction on doc



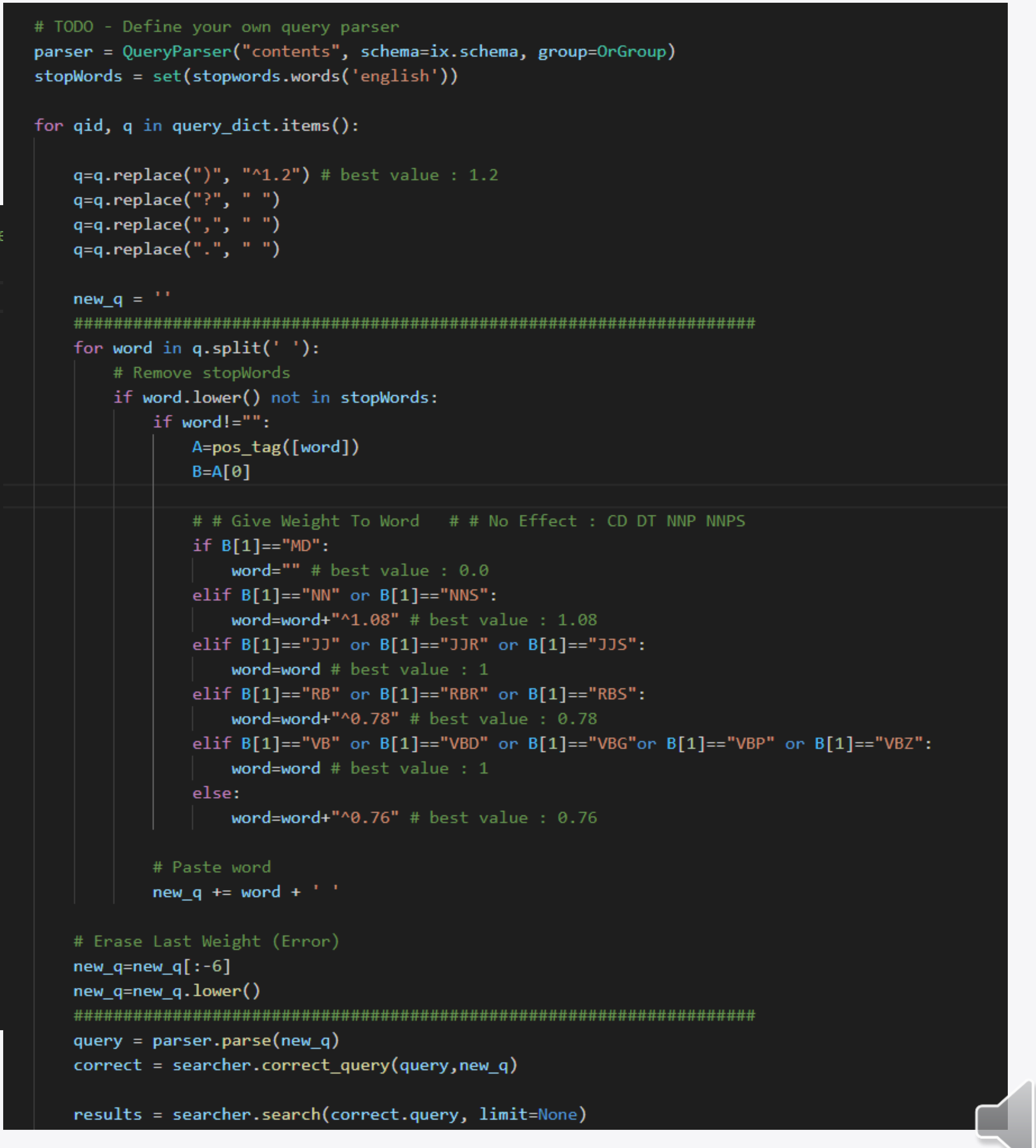
spell correction on query

# PART I : Document Searching Engine

## Requirement 1-1 & 2 : make_index.py & QueryResult.py



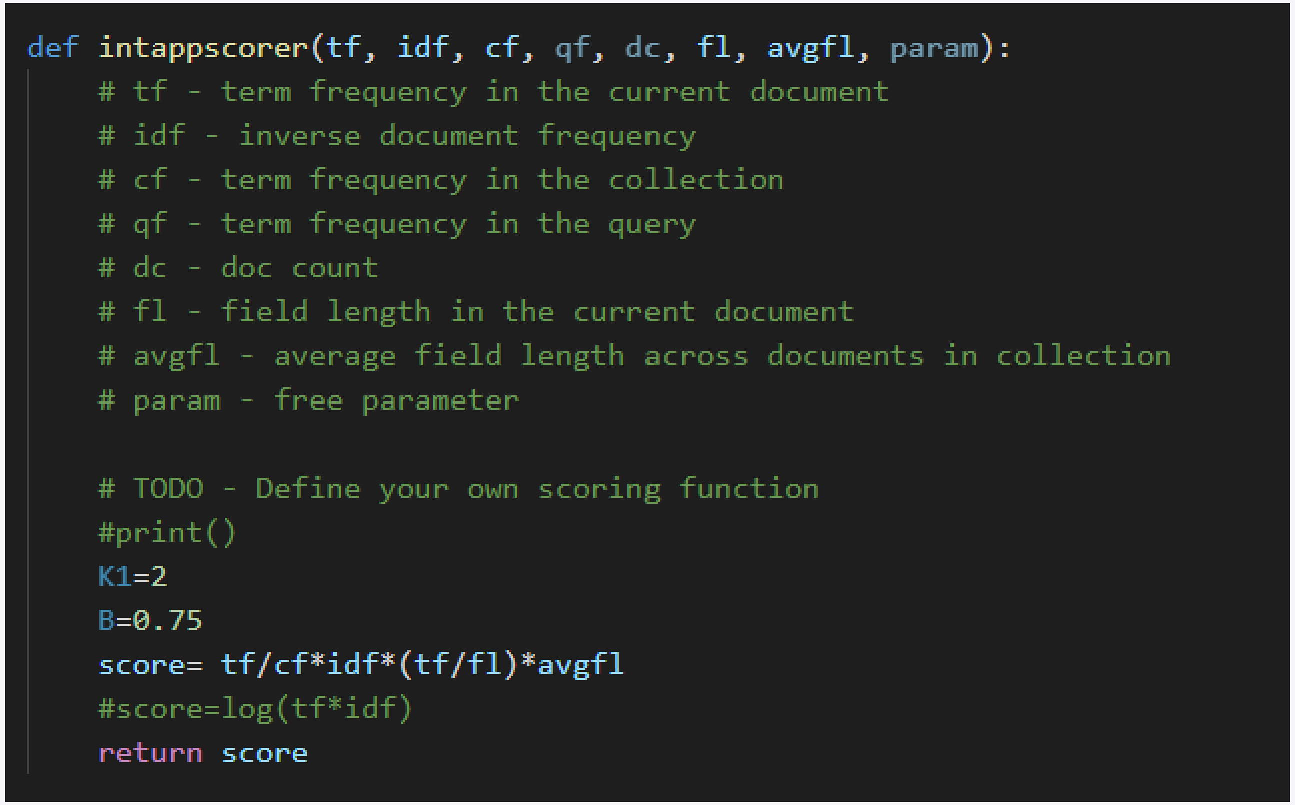
```
# Set up Schema [ docID - Numeric/ doctext- TEXT { option : stem (stoplist), Ngram (size
schema = Schema(docID=NUMERIC(stored=True),
contents=TEXT(analyzer=StemmingAnalyzer(stoplist=STOP_WORDS)))
index_dir = "index"

if not os.path.exists(index_dir):
    os.makedirs(index_dir)

ix = create_in(index_dir, schema)
writer = ix.writer()

with open('doc/document.txt', 'r') as f:
    text = f.read()
    docs = text.split('   /\n')[:-1]

    for doc in docs:
        br = doc.find('\n')
        docID = int(doc[:br])
        doc_text = doc[br+1:]
        ##########################################################
        doc_text=doc_text.replace(".", " ")
        ##########################################################
        writer.add_document(docID=docID, contents=doc_text)

writer.commit()
```



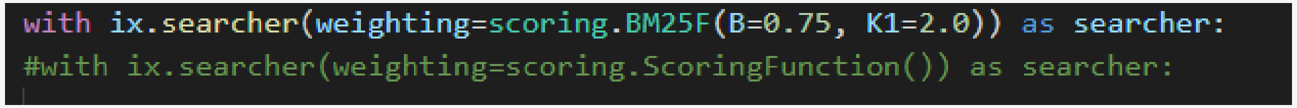
```
# TODO - Define your own query parser
parser = QueryParser("contents", schema=ix.schema, group=OrGroup)
stopWords = set(stopwords.words('english'))

for qid, q in query_dict.items():

    q=q.replace(")", "^1.2") # best value : 1.2
    q=q.replace("?", " ")
    q=q.replace(",", " ")
    q=q.replace(".", " ")

    new_q = ''
    #####################################################################
    for word in q.split(' '):
        # Remove stopWords
        if word.lower() not in stopWords:
            if word!="":
                A=pos_tag([word])
                B=A[0]

                # # Give Weight To Word   # # No Effect : CD DT NNP NNPS
                if B[1]=="MD":
                    word="" # best value : 0.0
                elif B[1]=="NN" or B[1]=="NNS":
                    word=word+"^1.08" # best value : 1.08
                elif B[1]=="JJ" or B[1]=="JJR" or B[1]=="JJS":
                    word=word # best value : 1
                elif B[1]=="RB" or B[1]=="RBR" or B[1]=="RBS":
                    word=word+"^0.78" # best value : 0.78
                elif B[1]=="VB" or B[1]=="VBD" or B[1]=="VBG"or B[1]=="VBP" or B[1]=="VBZ":
                    word=word # best value : 1
                else:
                    word=word+"^0.76" # best value : 0.76

            # Paste word
            new_q += word + ' '

    # Erase Last Weight (Error)
    new_q=new_q[:-6]
    new_q=new_q.lower()
    #####################################################################
    query = parser.parse(new_q)
    correct = searcher.correct_query(query,new_q)

    results = searcher.search(correct.query, limit=None)
```

# PART I： Document Searching Engine

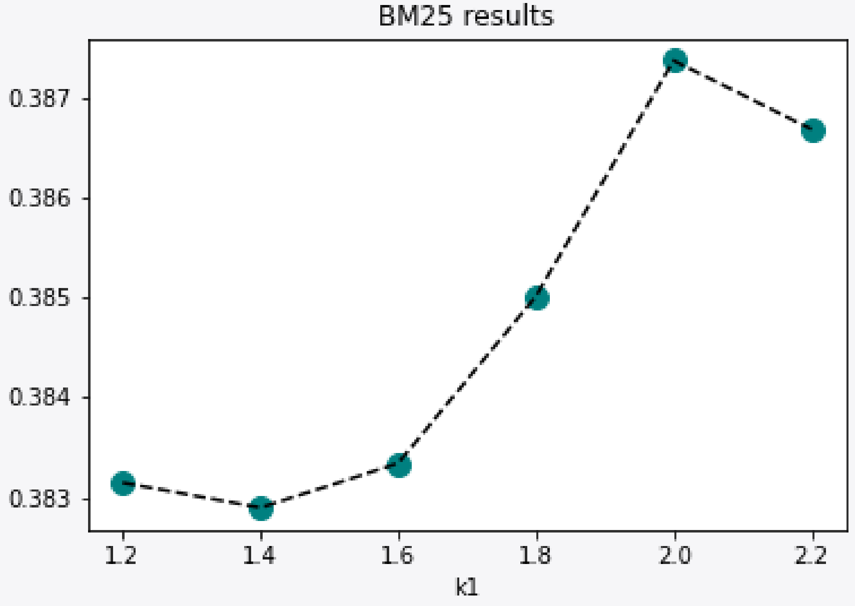
## Requirement 1-3： CustomScoring.py



```
def intappscorer(tf, idf, cf, qf, dc, fl, avgfl, param):
    # tf - term frequency in the current document
    # idf - inverse document frequency
    # cf - term frequency in the collection
    # qf - term frequency in the query
    # dc - doc count
    # fl - field length in the current document
    # avgfl - average field length across documents in collection
    # param - free parameter

    # TODO - Define your own scoring function
    #print()
    K1=2
    B=0.75
    score= tf/cf*idf*(tf/fl)*avgfl
    #score=log(tf*idf)
    return score
```



```
with ix.searcher(weighting=scoring.BM25F(B=0.75, K1=2.0)) as searcher:
#with ix.searcher(weighting=scoring.ScoringFunction()) as searcher:
```



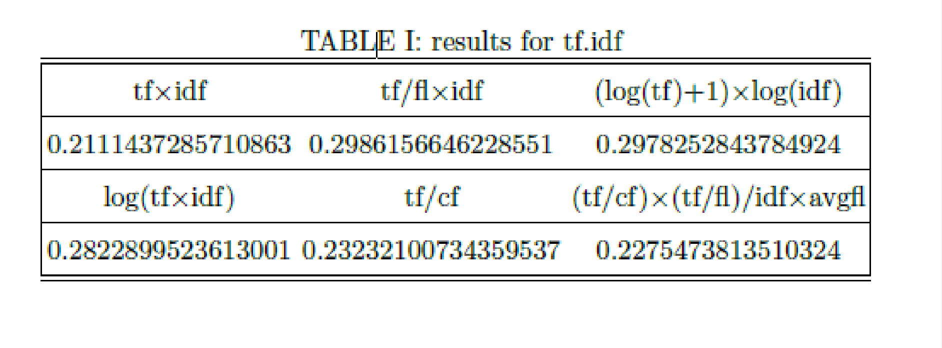


TABLE I: results for tf.idf

| | | |
|---|---|---|
| tf×idf | tf/fl×idf | (log(tf)+1)×log(idf) |
| 0.2111437285710863 | 0.2986156646228551 | 0.2978252843784924 |
| log(tf×idf) | tf/cf | (tf/cf)×(tf/fl)/idf×avgfl |
| 0.2822899523613001 | 0.23232100734359537 | 0.2275473813510324 |

# Part II : 문서 분류 및 군집화

## Requirement2-1 : Classification

- Naïve bayes Classifier
- Support Vector Machine

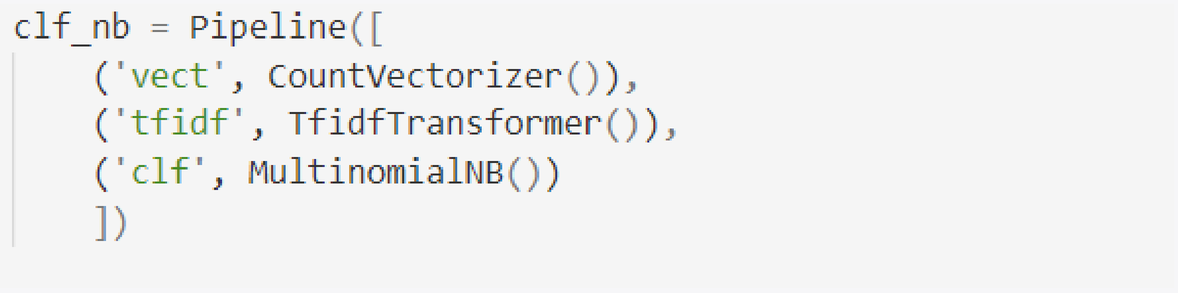
## Requirement2-2 : Clustering

# Part II : 문서 분류 및 군집화

## Requirement2-1 : Classification - NaÏve bayes Classifier

- Pipeline : CountVecorizer, TFidfTransformer, MultinomialNB

- CountVectorizer : stop_words, ngram_range, min_df, max_df
- MultinomialNB : alpha



```
clf_nb = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB())
    ])
```

# Part II : 문서 분류 및 군집화

## Requirement2-1 : Classification - Naïve Bayes Classifier

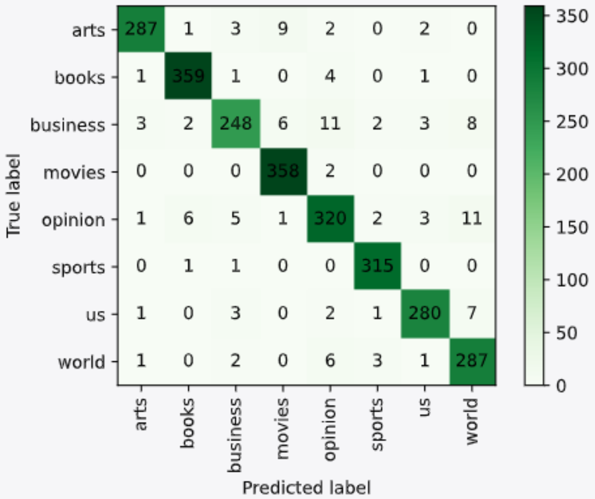
Ngram_range (min, max)

- 과적합을 방지하기 위해 (1, 1)과 (1, 2)를 사용
- (1, 1)에서 더 좋은 성능

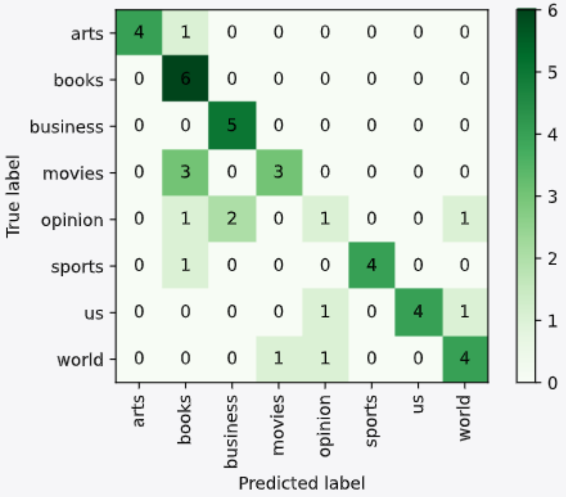
max_df, min_df

- ‘총 단어 수 32,431개(ngram_range = (1,2) 일 때는 286,455)
  -> max_df, min_df로 사용 단어 수를 줄일 필요가 있음.

| Parameter | 값 |
|---|---|
| Stop_words | ‘english’ |
| Ngram_range | (1, 1) |
| Max_df | 700 |
| Min_df | 2 |
| alpha | 0.4 |
| Test 정확도 | 31/44 |
| Unigram만을 사용했을 때의 모델 | |



| | arts | books | business | movies | opinion | sports | us | world |
|---|---|---|---|---|---|---|---|---|
| arts | 287 | 1 | 3 | 9 | 2 | 0 | 2 | 0 |
| books | 1 | 359 | 1 | 0 | 4 | 0 | 1 | 0 |
| business | 3 | 2 | 248 | 6 | 11 | 2 | 3 | 8 |
| movies | 0 | 0 | 0 | 358 | 2 | 0 | 0 | 0 |
| opinion | 1 | 6 | 5 | 1 | 320 | 2 | 3 | 11 |
| sports | 0 | 1 | 1 | 0 | 0 | 315 | 0 | 0 |
| us | 1 | 0 | 3 | 0 | 2 | 1 | 280 | 7 |
| world | 1 | 0 | 2 | 0 | 6 | 3 | 1 | 287 |



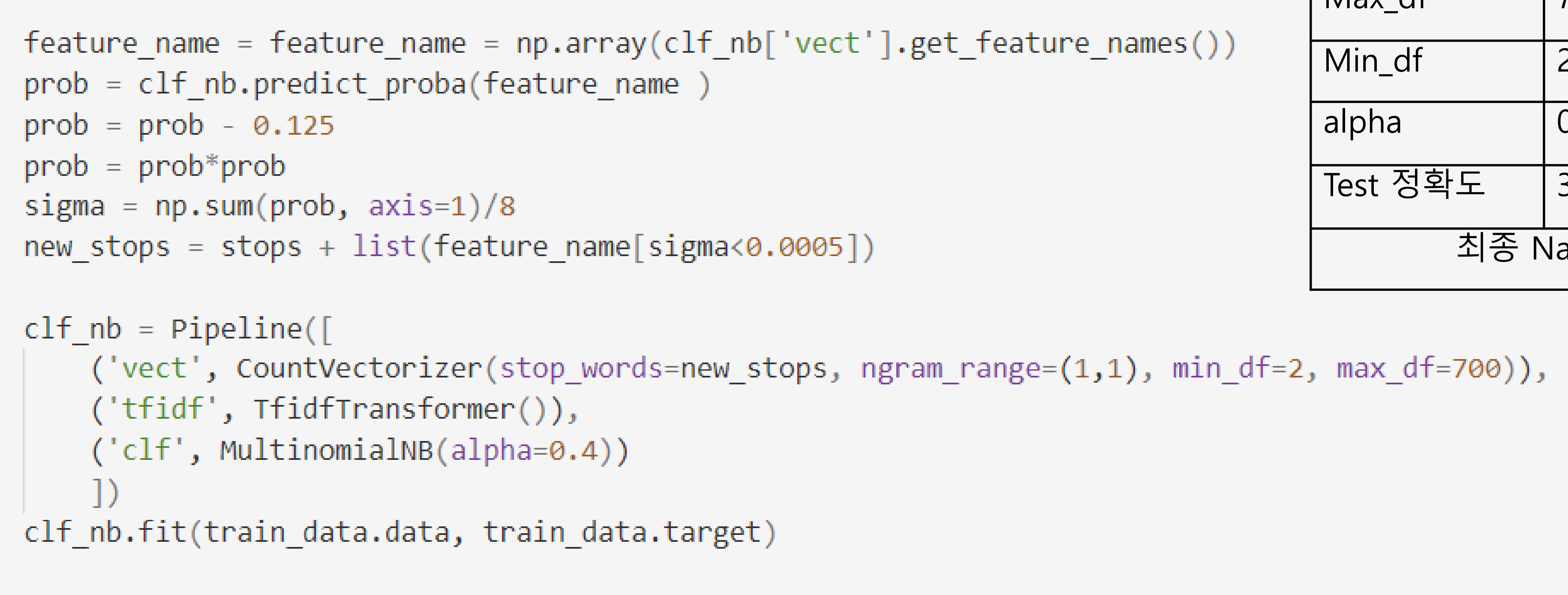
| | arts | books | business | movies | opinion | sports | us | world |
|---|---|---|---|---|---|---|---|---|
| arts | 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| books | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| business | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 |
| movies | 0 | 3 | 0 | 3 | 0 | 0 | 0 | 0 |
| opinion | 0 | 1 | 2 | 0 | 1 | 0 | 0 | 1 |
| sports | 0 | 1 | 0 | 0 | 0 | 4 | 0 | 0 |
| us | 0 | 0 | 0 | 0 | 1 | 0 | 4 | 1 |
| world | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 4 |

# Part II : 문서 분류 및 군집화

## Requirement2-1 : Classification - Naïve Bayes Classifier

Stop_word 재정의

- 각 단어들의 확률을 측정
- 8개 그룹의 확률의 분산이 0.0005미만인 것들을 stops에 추가



```
feature_name = feature_name = np.array(clf_nb['vect'].get_feature_names())
prob = clf_nb.predict_proba(feature_name )
prob = prob - 0.125
prob = prob*prob
sigma = np.sum(prob, axis=1)/8
new_stops = stops + list(feature_name[sigma<0.0005])

clf_nb = Pipeline([
    ('vect', CountVectorizer(stop_words=new_stops, ngram_range=(1,1), min_df=2, max_df=700)),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB(alpha=0.4))
    ])
clf_nb.fit(train_data.data, train_data.target)
```

| Parameter | 값 |
|---|---|
| Stop_words | 새로 설정한 stopword |
| Ngram_range | (1, 1) |
| Max_df | 700 |
| Min_df | 2 |
| alpha | 0.4 |
| Test 정확도 | 33/44 |
| 최종 Naïve bayes 모델 | |

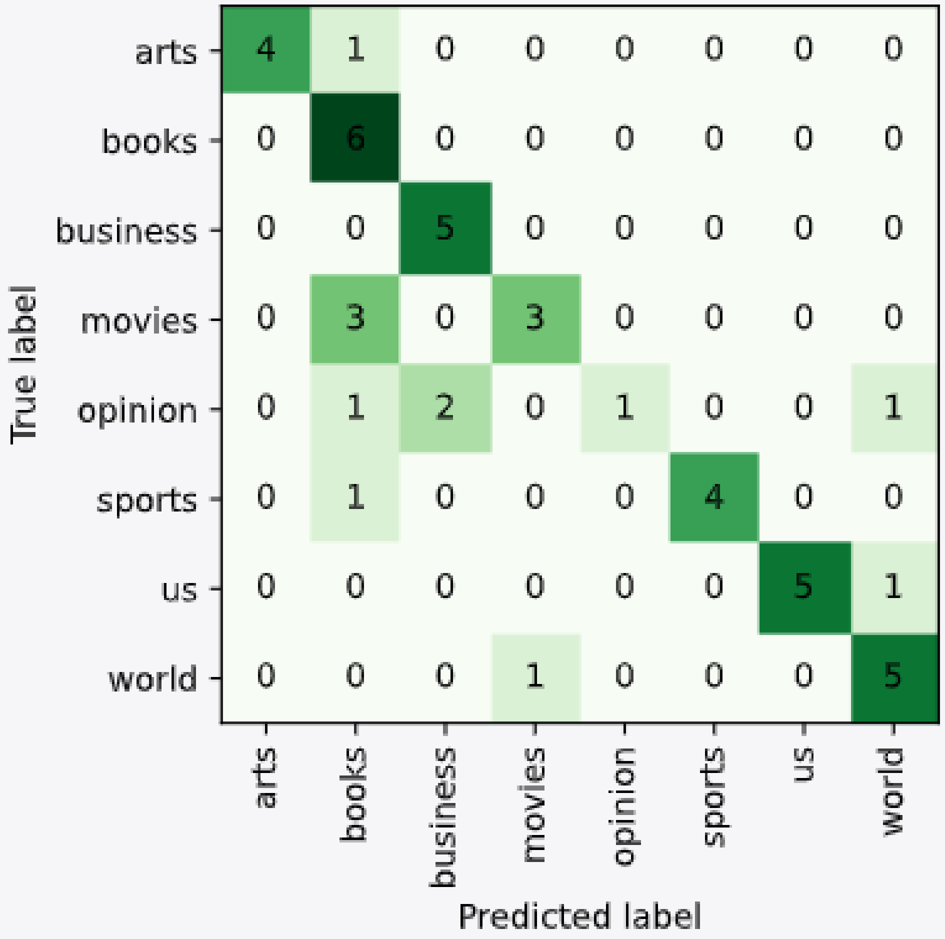
# Part II : 문서 분류 및 군집화

## Requirement2-1 : Classification – Naïve Bayes Classifier

Result

- 맞춘 Test case : 33개
- Precision에서 상향된 결과

| | precision | recall | f1-score |
|---|---|---|---|
| arts | 1 | 0.8 | 0.89 |
| books | 0.5 | 1 | 0.67 |
| business | 0.71 | 1 | 0.83 |
| movies | 0.75 | 0.5 | 0.6 |
| opinion | 1 | 0.2 | 0.33 |
| sports | 1 | 0.8 | 0.89 |
| us | 1 | 0.83 | 0.91 |
| world | 0.71 | 0.83 | 0.77 |
| accuracy | | | 0.75(33/44) |



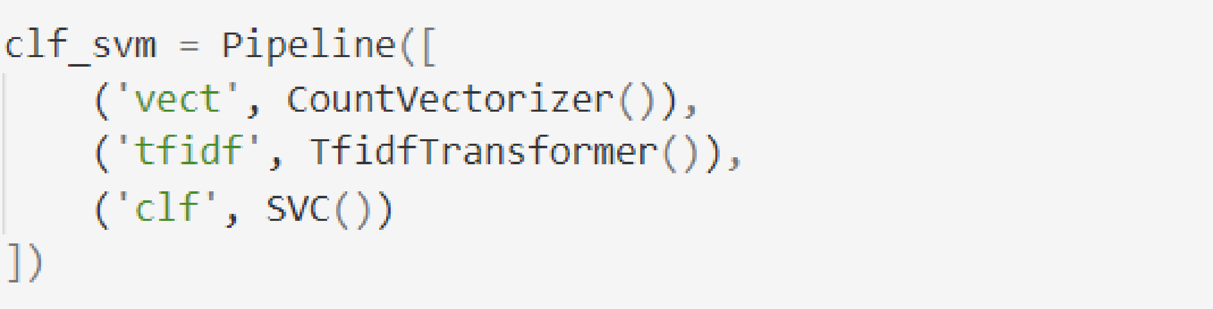
| | arts | books | business | movies | opinion | sports | us | world |
|---|---|---|---|---|---|---|---|---|
| arts | 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| books | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| business | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 |
| movies | 0 | 3 | 0 | 3 | 0 | 0 | 0 | 0 |
| opinion | 0 | 1 | 2 | 0 | 1 | 0 | 0 | 1 |
| sports | 0 | 1 | 0 | 0 | 0 | 4 | 0 | 0 |
| us | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 1 |
| world | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 5 |

# Part II : 문서 분류 및 군집화

## Requirement2-1 : Classification   Support Vector Machine

- Pipeline : CountVecorizer, TFidfTransformer, SVC

- CountVectorizer : stop_words, ngram_range, min_df, max_df
- SVC : kernel, C, gamma, coeff



```
clf_svm = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', SVC())
])
```

# Part II : 문서 분류 및 군집화

## Requirement2-1 : Classification     Support Vector Machine

Kernel
- rbf : 과적합
- Linear : Training의 정확도가 완전하지는 않지만, 그럼에도 test data에서 높은 정확도를 보인다.
- Poly : 과적합
- Sigmoid : training과 test 모두 높지 않음.

| kernel | Train accuracy | Test accuracy |
|---|---|---|
| rbf(default) | 2565 | 33 |
| linear | 2553 | 34 |
| Poly(degree = 2) | 2573 | 34 |
| sigmoid | 2211 | 29 |

# Part II : 문서 분류 및 군집화

## Requirement2-1 : Classification  Support Vector Machine

GridSearchCV

- CountVectorizer에서 최적의 parmeter을 찾아냄
- ngram_range =(1,1), max_df=950, min_df=4
- Linear과 sigmoid 중 linear에서 더 높은 성능

Linear

| Parameter | 값 |
|---|---|
| Stop_words | 'english' |
| Ngram_range | (1, 1) |
| Max_df | 950 |
| Min_df | 4 |
| kernel | Linear |
| C | 1 |
| Decision_function_shape | 'ovr' |
| Test 정확도 | 36/44 |

Sigmoid

| Parameter | 값 |
|---|---|
| Stop_words | 'english' |
| Ngram_range | (1, 1) |
| Max_df | 950 |
| Min_df | 4 |
| kernel | Sigmoid |
| C | 25 |
| coef0 | 2.5 |
| gamma | 2.5 |
| Decision_function_shape | 'ovr' |
| Test 정확도 | 35/44 |

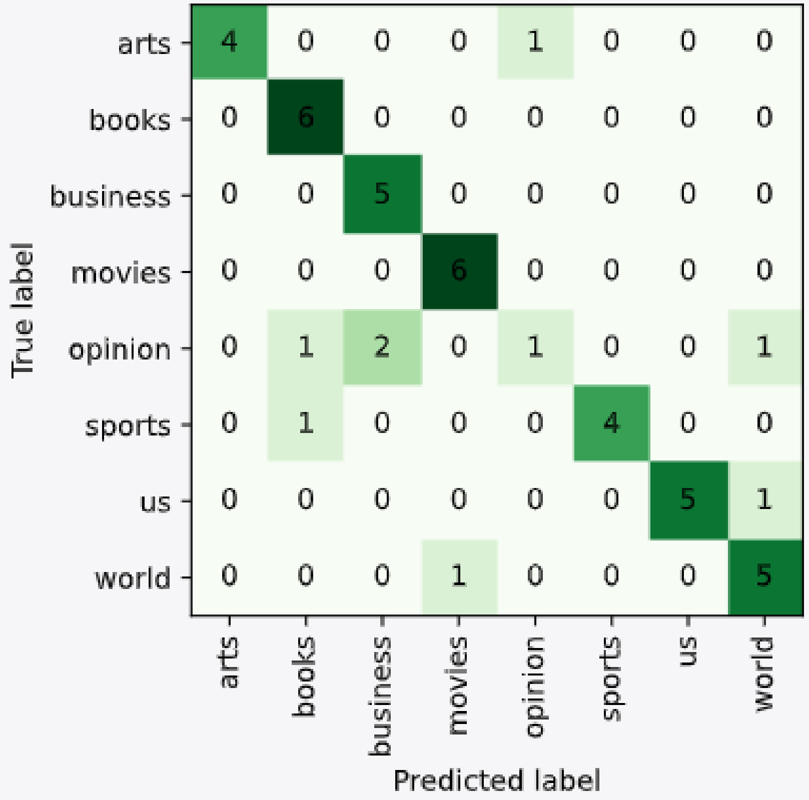
# Part II : 문서 분류 및 군집화

## Requirement2-1 : Classification Support Vector Machine

Result

- 맞춘 Test case : 36개
- Naïve Bayes에 비해 좋은 성능

| Parameter | 값 |
|---|---|
| Stop_words | 'english' |
| Ngram_range | (1, 2) |
| Max_df | 950 |
| Min_df | 4 |
| kernel | Linear |
| Decision_function_shape | 'ovr' |
| Test 정확도 | 36/44 |



| | arts | books | business | movies | opinion | sports | us | world |
|---|---|---|---|---|---|---|---|---|
| arts | 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| books | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| business | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 |
| movies | 0 | 0 | 0 | 6 | 0 | 0 | 0 | 0 |
| opinion | 0 | 1 | 2 | 0 | 1 | 0 | 0 | 1 |
| sports | 0 | 1 | 0 | 0 | 0 | 4 | 0 | 0 |
| us | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 1 |
| world | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 5 |

PART II : Clustering

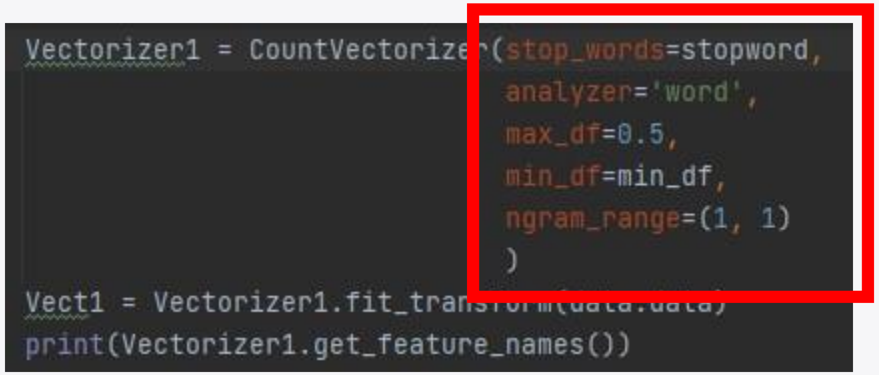
1st Grid searching

2nd Grid searching

결과 분석

# PART II : Clustering – 1st Grid Searching

## CountVectorizer



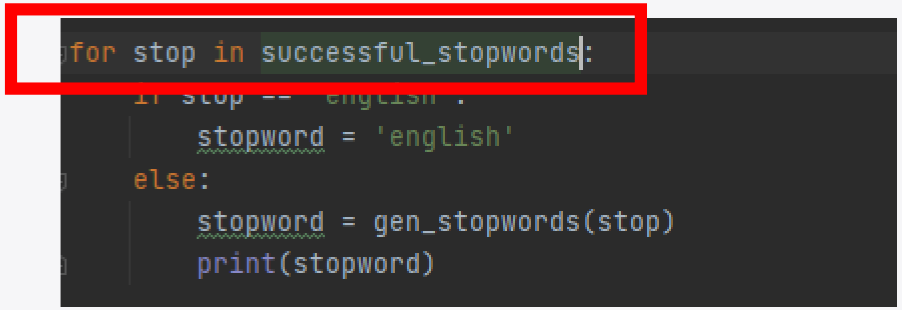
```
Vectorizer1 = CountVectorizer(stop_words=stopword,
                              analyzer='word',
                              max_df=0.5,
                              min_df=min_df,
                              ngram_range=(1, 1)
                              )
Vect1 = Vectorizer1.fit_transform(data.data)
print(Vectorizer1.get_feature_names())
```

1. min_df = 20~30
2. max_df = 0.5
3. ngram-range = (1,1)
4. Stopword = ‘English’

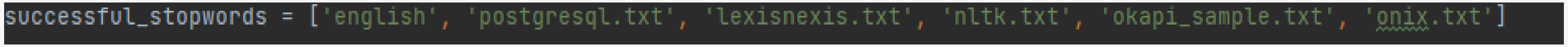
# PART II : Clustering    1st Grid Searching

## CountVectorizer

| stopword | avg v_measure | max v_measure |
|---|---|---|
| onix.txt | 0.421243788 | 0.459918709 |
| okapi_sample.txt | 0.402373513 | 0.409524552 |
| okapi_sample_expanded.txt | 0.399628088 | 0.405999241 |
| lingpipe.txt | 0.397162201 | 0.418990854 |
| weka.txt | 0.396200509 | 0.426215476 |
| vw_lda.txt | 0.39416418 | 0.426223624 |
| taporware.txt | 0.393116287 | 0.403632107 |
| zettair.txt | 0.391398998 | 0.422517918 |
| nltk.txt | 0.390841977 | 0.455236456 |
| gilner_morales.txt | 0.388789925 | 0.434379852 |
| lexisnexis.txt | 0.388384372 | 0.451394562 |
| voyant_taporware.txt | 0.387601008 | 0.436093459 |
| smart.txt | 0.386881889 | 0.424393739 |
| mysql_myisam.txt | 0.385825582 | 0.401896468 |
| gate_keyphrase.txt | 0.384068101 | 0.426676731 |
| postgresql.txt | 0.383622548 | 0.44789582 |
| … | … | … |



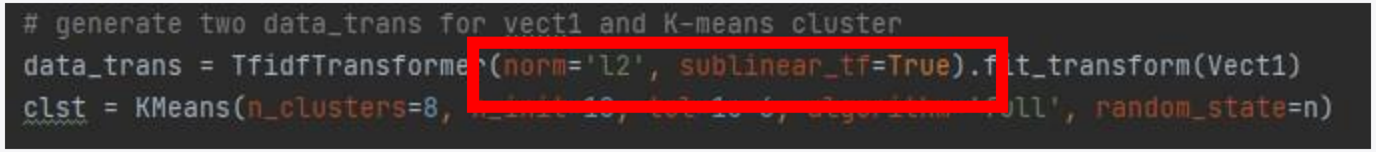
```
for stop in successful_stopwords:
    if stop == 'english':
        stopword = 'english'
    else:
        stopword = gen_stopwords(stop)
        print(stopword)
```



```
successful_stopwords = ['english', 'postgresql.txt', 'lexisnexis.txt', 'nltk.txt', 'okapi_sample.txt', 'onix.txt']
```

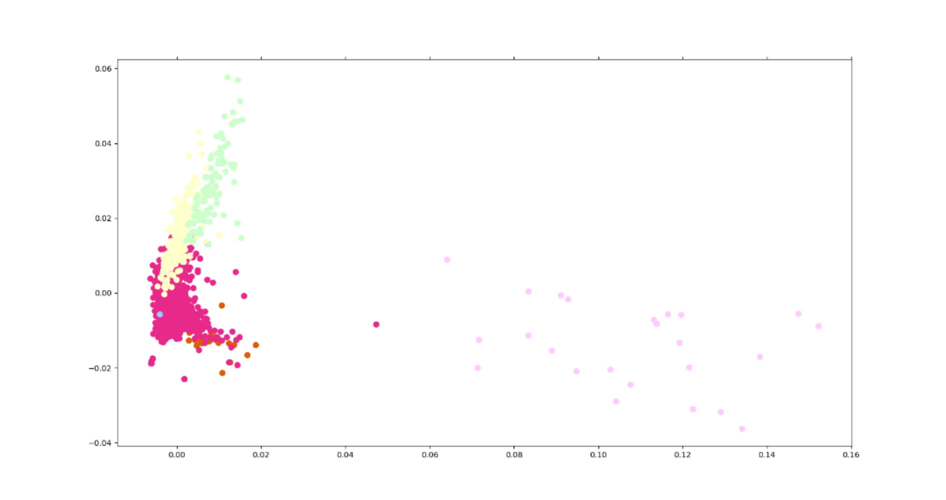
# PART II : Clustering    1st Grid Searching

## TfidfTransformer



```
# generate two data_trans for vect1 and K-means cluster
data_trans = TfidfTransformer(norm='l2', sublinear_tf=True).fit_transform(Vect1)
clst = KMeans(n_clusters=8, n_init=10, tol=1e-6, algorithm='full', random_state=n)
```

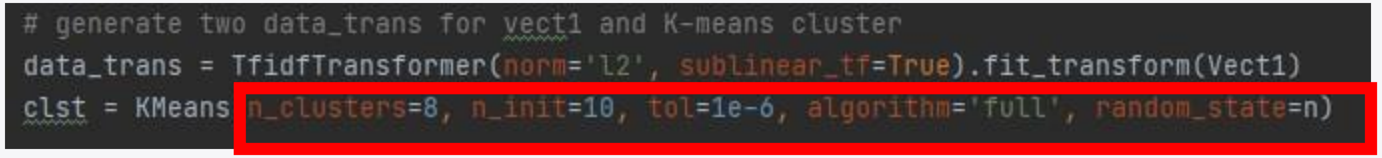
1. norm : ‘l2’
2. Sublinear_tf : True



norm=‘l1’

# PART II : Clustering 1st Grid Searching

## KMeans



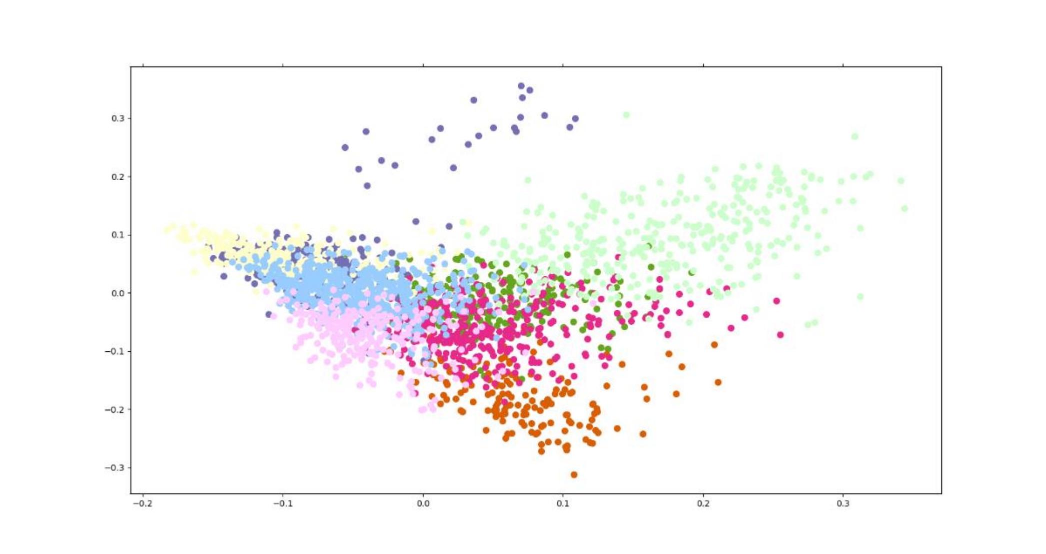
```
# generate two data_trans for vect1 and K-means cluster
data_trans = TfidfTransformer(norm='l2', sublinear_tf=True).fit_transform(Vect1)
clst = KMeans(n_clusters=8, n_init=10, tol=1e-6, algorithm='full', random_state=n)
```

1. n_clusters = 8
2. n_init = 10
3. tol = 1e-6
4. algorithm = ‘full’
5. random_state

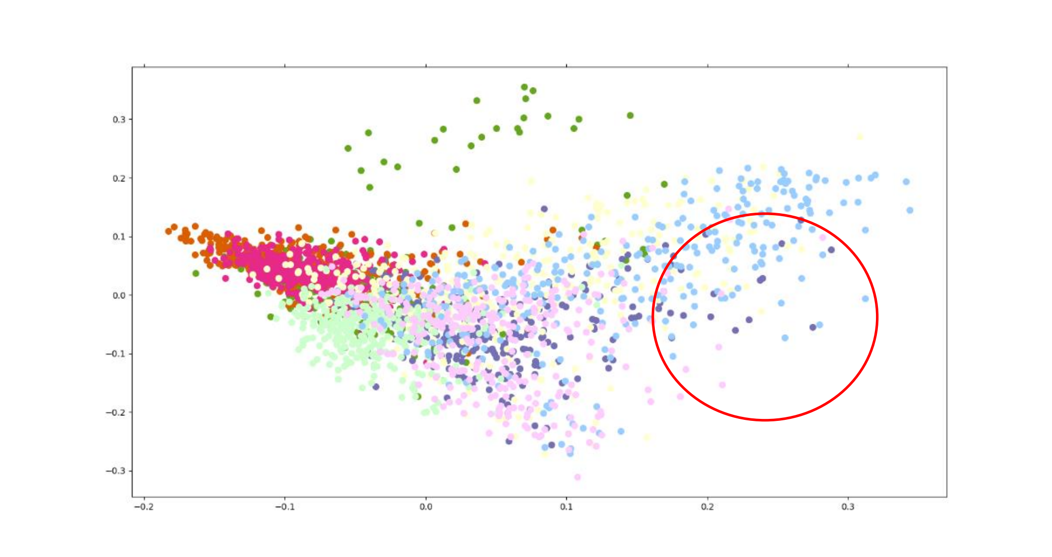
# PART II : Clustering 1st Grid Searching

## Semi Result Analysis

Clustering



Target Category

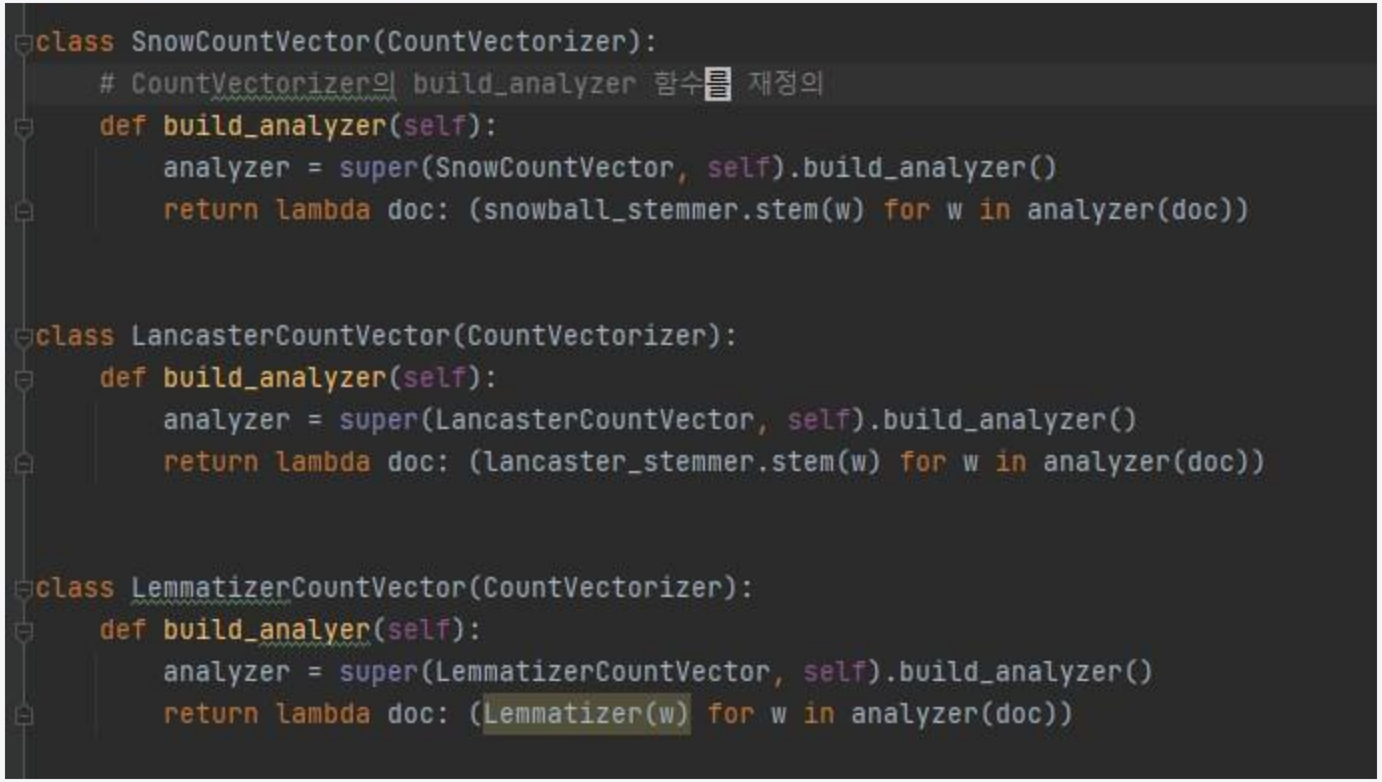


1. Target Category의 복잡한 분포 확인
2. 산개해 있는 경우 분류 어려움

→ 더 정교한 feature 선택 시도

## PART II : Clustering    2nd Grid Searching

### Tokenization, Lemmatizer



```
class SnowCountVector(CountVectorizer):
    # CountVectorizer의 build_analyzer 함수를 재정의
    def build_analyzer(self):
        analyzer = super(SnowCountVector, self).build_analyzer()
        return lambda doc: (snowball_stemmer.stem(w) for w in analyzer(doc))


class LancasterCountVector(CountVectorizer):
    def build_analyzer(self):
        analyzer = super(LancasterCountVector, self).build_analyzer()
        return lambda doc: (lancaster_stemmer.stem(w) for w in analyzer(doc))


class LemmatizerCountVector(CountVectorizer):
    def build_analyer(self):
        analyzer = super(LemmatizerCountVector, self).build_analyzer()
        return lambda doc: (Lemmatizer(w) for w in analyzer(doc))
```

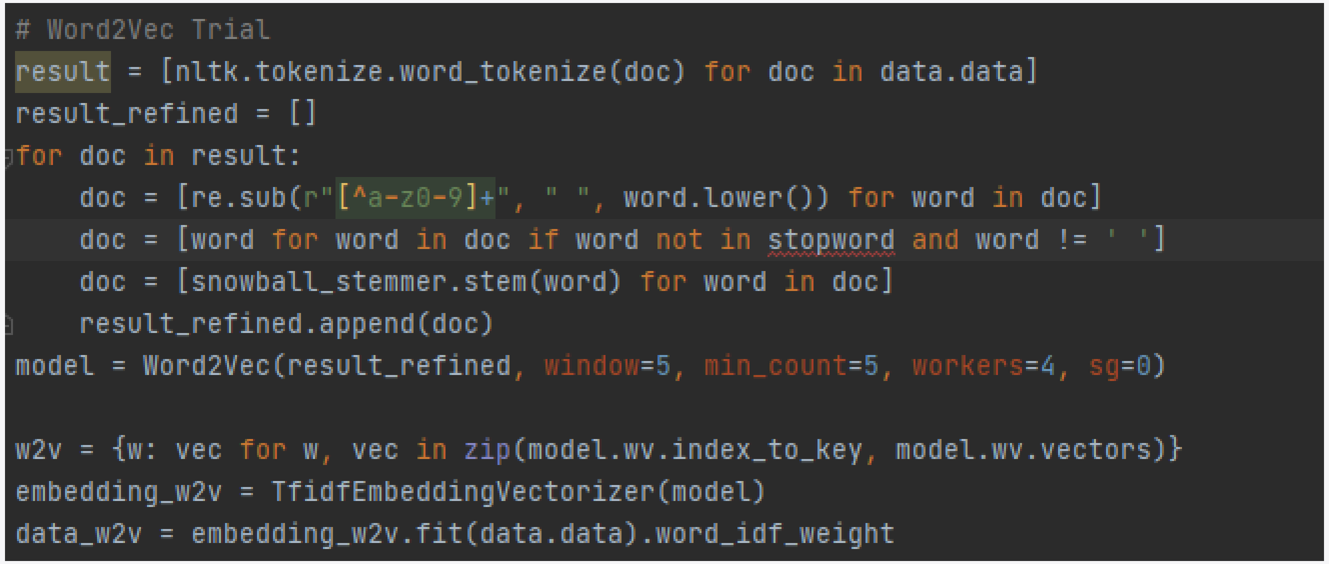
NLTK 패키지

1. Snowball Stemmer, etc
2. Lancaster Stemmer, etc
3. Lemmatizer

avg = 0.438076911 max=0.46716645
avg = 0.438052360 max=0.45759459
avg = 0.44875163, max= 0.48838172

# PART II : Clustering 2nd Grid Searching

## Vectorizer with Word2Vec



```
# Word2Vec Trial
result = [nltk.tokenize.word_tokenize(doc) for doc in data.data]
result_refined = []
for doc in result:
    doc = [re.sub(r"[^a-z0-9]+", " ", word.lower()) for word in doc]
    doc = [word for word in doc if word not in stopword and word != ' ']
    doc = [snowball_stemmer.stem(word) for word in doc]
    result_refined.append(doc)
model = Word2Vec(result_refined, window=5, min_count=5, workers=4, sg=0)

w2v = {w: vec for w, vec in zip(model.wv.index_to_key, model.wv.vectors)}
embedding_w2v = TfidfEmbeddingVectorizer(model)
data_w2v = embedding_w2v.fit(data.data).word_idf_weight
```

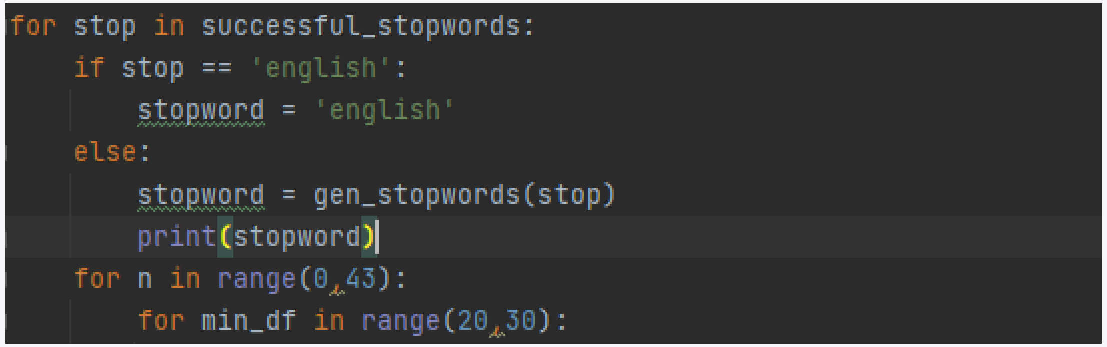
Word2Vec

1. Embedding Vector
2. 단어 간의 유사도 사용

유의미한 개선 X

# PART II : Clustering 2nd Grid Searching

## More Strict Grid Searching



```python
for stop in successful_stopwords:
    if stop == 'english':
        stopword = 'english'
    else:
        stopword = gen_stopwords(stop)
        print(stopword)
    for n in range(0,43):
        for min_df in range(20,30):
```

| | |
|---|---|
| CountVectorizer | : stopwords / min_df / max_df |
| TfidfTransformer | : Sublinear_tf |
| KMeans | : n_init, tol, algorithm, random_state |

# PART II : Clustering 2nd Grid Searching

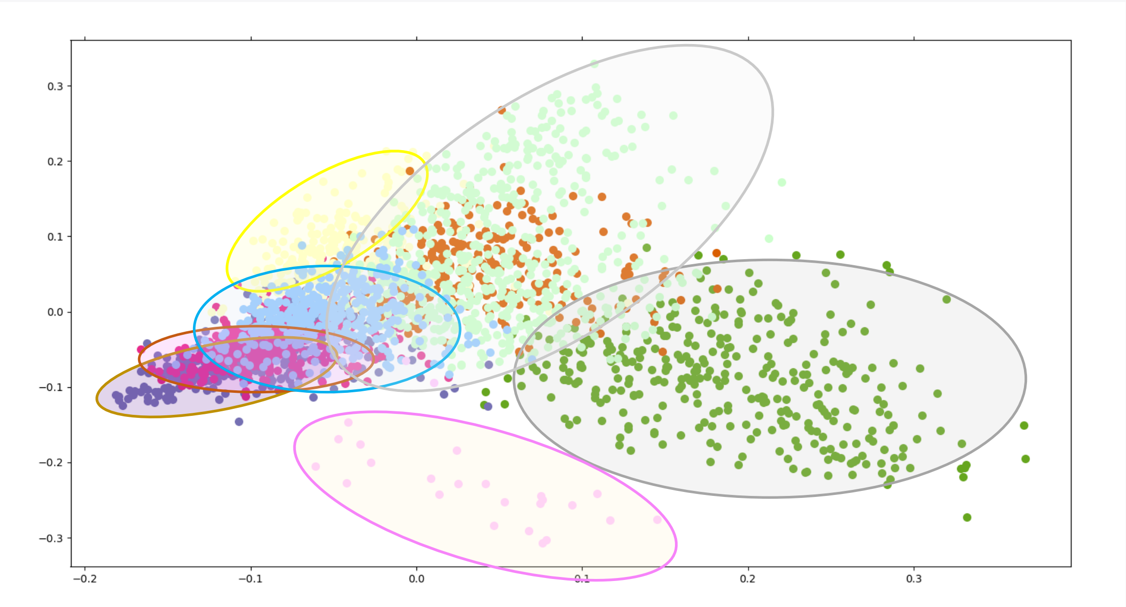
## Best Model

| Class | Parameter |
|---|---|
| CountVectorizer | stop_words=’okapi_samplel.txt’(adjusted), max_df=0.5, min_df=24 |
| TfidfTransformer | norm=’l2’, sublinear_tf=True |
| KMeans | n_cluster=8, n_init=10, tol=6e-1, algorithm=’full’, random_state=17 |
| v_measure | 0.5114715913 |

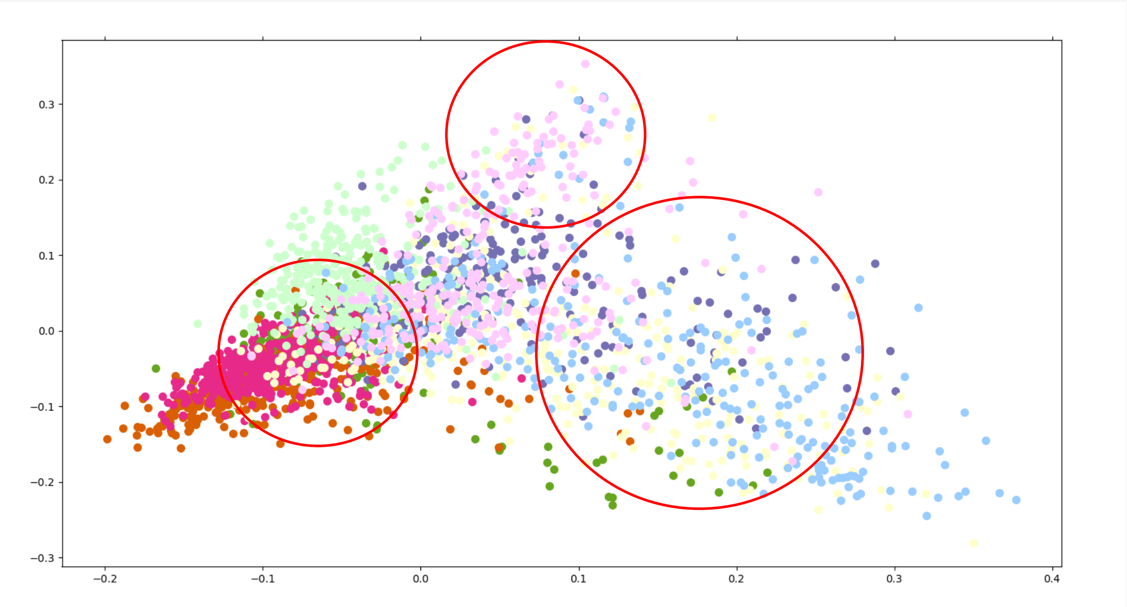
# PART II : Clustering 결과 분석(1)

## PCA 차원 축소 그래프

Clustering



Target Category



## PART II : Clustering 결과 분석(2)

### Confusion Matrix : Row-wise

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Total |
|---|---|---|---|---|---|---|---|---|---|
| Opinion | 8 | 18 | 16 | 22 | 0 | 5 | 215 | 25 | 309 |
| Business | 6 | 12 | 310 | 4 | 1 | 18 | 21 | 0 | 372 |
| World | 30 | 175 | 6 | 6 | 2 | 40 | 29 | 0 | 288 |
| US | 0 | 3 | 8 | 330 | 0 | 2 | 23 | 0 | 366 |
| Arts | 125 | 46 | 31 | 2 | 1 | 114 | 35 | 0 | 354 |
| Sports | 1 | 8 | 2 | 2 | 252 | 27 | 30 | 0 | 322 |
| Books | 158 | 11 | 2 | 0 | 1 | 118 | 10 | 0 | 300 |
| Movies | 9 | 9 | 1 | 5 | 2 | 254 | 26 | 0 | 306 |

Opinion, Business, World, US, Sports, Movies : 1 big cluster + others

# PART II : Clustering 결과 분석(3)

## Confusion Matrix : Row-wise

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Total |
|---|---|---|---|---|---|---|---|---|---|
| Opinion | 8 | 18 | 16 | 22 | 0 | 5 | 215 | 25 | 309 |
| Business | 6 | 12 | 310 | 4 | 1 | 18 | 21 | 0 | 372 |
| World | 30 | 175 | 6 | 6 | 2 | 40 | 29 | 0 | 288 |
| Us | 0 | 3 | 8 | 330 | 0 | 2 | 23 | 0 | 366 |
| Arts | 125 | 46 | 31 | 2 | 1 | 114 | 35 | 0 | 354 |
| Sports | 1 | 8 | 2 | 2 | 252 | 27 | 30 | 0 | 322 |
| Books | 158 | 11 | 2 | 0 | 1 | 118 | 10 | 0 | 300 |
| Movies | 9 | 9 | 1 | 5 | 2 | 254 | 26 | 0 | 306 |

Arts, Books : 2 big cluster + others

# PART II : Clustering 결과 분석(4)

## Confusion Matrix : Column-wise

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Total |
|---|---|---|---|---|---|---|---|---|---|
| Opinion | 8 | 18 | 16 | 22 | 0 | 5 | 215 | 25 | 309 |
| Business | 6 | 12 | 310 | 4 | 1 | 18 | 21 | 0 | 372 |
| World | 30 | 175 | 6 | 6 | 2 | 40 | 29 | 0 | 288 |
| Us | 0 | 3 | 8 | 330 | 0 | 2 | 23 | 0 | 366 |
| Arts | 125 | 46 | 31 | 2 | 1 | 114 | 35 | 0 | 354 |
| Sports | 1 | 8 | 2 | 2 | 252 | 27 | 30 | 0 | 322 |
| Books | 158 | 11 | 2 | 0 | 1 | 118 | 10 | 0 | 300 |
| Movies | 9 | 9 | 1 | 5 | 2 | 254 | 26 | 0 | 306 |

Arts. Books, Movies -> Confusing Cluster

## PART II : Clustering 결과 분석(5)

### Confusion Matrix : Column-wise

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Total |
|---|---|---|---|---|---|---|---|---|---|
| Opinion | 8 | 18 | 16 | 22 | 0 | 5 | 215 | 25 | 309 |
| Business | 6 | 12 | 310 | 4 | 1 | 18 | 21 | 0 | 372 |
| World | 30 | 175 | 6 | 6 | 2 | 40 | 29 | 0 | 288 |
| Us | 0 | 3 | 8 | 330 | 0 | 2 | 23 | 0 | 366 |
| Arts | 125 | 46 | 31 | 2 | 1 | 114 | 35 | 0 | 354 |
| Sports | 1 | 8 | 2 | 2 | 252 | 27 | 30 | 0 | 322 |
| Books | 158 | 11 | 2 | 0 | 1 | 118 | 10 | 0 | 300 |
| Movies | 9 | 9 | 1 | 5 | 2 | 254 | 26 | 0 | 306 |

Opinion – Cluster with several semi-clusters

# PROJECT #3

Team 05
감사합니다