PART 1. 문서 검색 엔진 성능 개선

1-1. CustomerScoring 함수 변경

```
return idf * ((tf * (K1 + 1)) / (tf + K1 * ((1 - B) + B * fl / avgfl)))
```

K1 = 0.13 / B = 0.4로 설정

```
Counter({'': 140037, 'n': 73527, 'the': 7229, 'of': 4955, 'us': 2543, '
circuit': 2348, 'frequ': 2323, 'field': 2317, 'and': 2224, 'electron':
2173, 'vary': 1764, 'magnet': 1758, 'in': 1618, 'is': 1566, 'method': 1
566, 'effect': 1523, 'ampl': 1519, 'giv': 1444, 'two': 1386, 'high': 13
35, 'are': 1307, 'wav': 1295, 'describ': 1294, 'on': 1272, 'result': 12
54, 'observ': 1229, 'radio': 1217, 'to': 1154, 'design': 1143, 'system'
: 1085, 'analys': 1084, 'meas': 1066, 'oscil': 1062, 'ionosph': 1052, '
for': 1039, 'low': 1038, 'ion': 1036, 'with': 991, 'apply': 983, 'show'
: 976, 'typ': 964, 'obtain': 895, 'equ': 878, 'network': 878, 'tim': 87
3, 'gen': 863, 'discuss': 860, 'cur': 857, 'puls': 822, 'lay': 809, 'ma
k': 798, 'sol': 796, 'rel': 795, 'reg': 785, 'bas': 780, 'volt': 775, '
op': 731, 'transist': 725, 'an': 717, 'nois': 708, 'der': 697, 'mod': 6
97, 'by': 696, 'stat': 690, 'energy': 685, 'calc': 680, 'charact': 679,
'expery': 674, 'output': 673, 'sign': 673, 'valu': 668, 'funct': 658, '
pow': 653, 'at': 650, 'rady': 649, 'filt': 648, 'height': 643, 'band':
642, 'distribut': 629, 'diff': 627, 'dens': 626, 'ear': 621, 'also': 61
6, 'elect': 613, 'discussed': 609, 'consid': 607, 'resist': 607, 'deter
min': 605, 'part': 599, 'frequency': 598, 'form': 590, 'described': 582
, 'stabl': 578, 'phas': 578, 'pres': 575, 'reson': 573, 'temp': 572, 'm
ay': 569, 'plasm': 556, 'control': 553, 'emit': 552, 'input': 545, 'atm
```

Term frequency 분석 결과의 일부분 예시

```
return ((idf**parameter) * ((tf * (K1 + 1)) / (tf + K1 * ((1 - B) + B * fl / avgfl))))
```

Idf의 비중을 높이고자 idf에 parameter를 제곱해주었고 parameter = 1 ~ 약 1.7까지 개선 경향 보임.

1-2. OrGroup 변수 변경

```
parser = QueryParser("contents", schema=ix.schema, group=OrGroup.factory(0.999))
```

OrGroup 변수 = 0.999로 설정

1-3. 문장 부호 제거

```
for qid, q in query_dict.items():
    table = str.maketrans('\n?.,!', '     ')
    q = q.translate(table)
    new_q = ''
```

Query 문장 부호 제거

```
table = str.maketrans('\n?.,!', '     ')
doc_text = doc_text.translate(table)

new_doc_text = ''
for word in doc_text.split(' '):
    if word.lower() not in stopWords:
        # word = n.lemmatize(word)
        # word_stem = stemmizer.stem(word.lower())
        # new_doc_text += word_stem + ' '
        new_doc_text += word + ' '
```

document 문장 부호 제거

1-4. Stemming 및 Lemmatizing 적용

```
n = WordNetLemmatizer()
st3 = LancasterStemmer()
```

```
new_doc_text = ''
for word in doc_text.split(' '):
    if word.lower() not in stopWords:
        word = n.lemmatize(word.lower())
        word_stem = st3.stem(word)
        new_doc_text += word_stem + ' '
```

Document에 Stemming 및 Lemmatizing 적용

(LancasterStemmer , WorldNetLemmatizer)

```
n = WordNetLemmatizer()
#s1 = PorterStemmer()
#s2 = SnowballStemmer('english')
s3 = LancasterStemmer()
```

```
new_q = ''
for word in q.split(' '):
#for word in q.split(' '):
    if word.lower() not in stopWords:

        word = n.lemmatize(word.lower())
        #new_q += word + ' '
        new_q += s3.stem(word) + ' '
```

Query에 Stemming 및 Lemmatizing 적용

(LancasterStemmer , WorldNetLemmatizer)

1-5. Query Expansion

```
syns = wordnet.synsets(s3.stem(word).lower())
if len(syns) > 4:
    # new_q += syns[].lemmas()[0].name() + ' '
    # new_q += syns[2].lemmas()[0].name() + ' '
    new_q += s3.stem(n.lemmatize(syns[1].lemmas()[0].name())) + ' '
    new_q += s3.stem(n.lemmatize(syns[2].lemmas()[0].name())) + ' '
    new_q += s3.stem(n.lemmatize(syns[3].lemmas()[0].name())) + ' '
```

성능 향상X → 적용X

1-6. 최종 결과

Query stopword 제거

↓

Query 문장 부호 제거

↓

Query에 Stemming 및 Lemmatizing 적용

(LancasterStemmer , WorldNetLemmatizer)

document stopword 제거

↓

document 문장 부호 제거

↓

Document에 Stemming 및 Lemmatizing 적용

(LancasterStemmer , WorldNetLemmatizer)

↓

OrGroup 변수 = 0.999로 설정

↓

```
return ((idf**parameter) * ((tf * (K1 + 1)) / (tf + K1 * ((1 - B) + B * fl / avgfl))))
```

Parameter = 1.5, k1 = 0.13, B = 0.5로 조정

1-7. 최종 개선된 성능

```
0.25908098603535284
```

PART 2. 문서 분류 및 군집화

2-1-1. naïve bayes

```
categories = ['Business', 'Entertainment', 'Living', 'Metro', 'Shopping', 'Sports', 'Tech']

train_data = load_files(container_path='text/train', categories=categories, shuffle=True,
                        encoding='utf-8', decode_error='replace')

# TODO - 2-1-1. Build pipeline for Naive Bayes Classifier
clf_nb = Pipeline([
                  ('vect', CountVectorizer(lowercase=True, stop_words='english', ngram_range=(1,1))),
                  ('tfidf', TfidfTransformer(use_idf=True, smooth_idf=True, sublinear_tf=False)),
                  ('clf', MultinomialNB(alpha=0.01))
                  ])
clf_nb.fit(train_data.data, train_data.target)
```

2-1-2. svd

```
# TODO - 2-1-2. Build pipeline for SVM Classifier
clf_svm = Pipeline([
                    ('vect', CountVectorizer(lowercase=True, stop_words='english', ngram_range=(1,1))),
                    ('tfidf', TfidfTransformer(use_idf=True, smooth_idf=True, sublinear_tf=False)),
                    ('clf', LinearSVC(C=1.0, random_state=0))
                    ])
clf_svm.fit(train_data.data, train_data.target)
```

2-1-3. 성능 평가

```
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))

predicted1 = clf_svm.predict(docs_test)
print("SVM accuracy: %d / %d" % (np.sum(predicted1 == test_data.target), len(test_data.target)))
# print(metrics.classification_report(test_data.target, predicted1, target_names=test_data.target_names))
# print(metrics.confusion_matrix(test_data.target, predicted1))
```

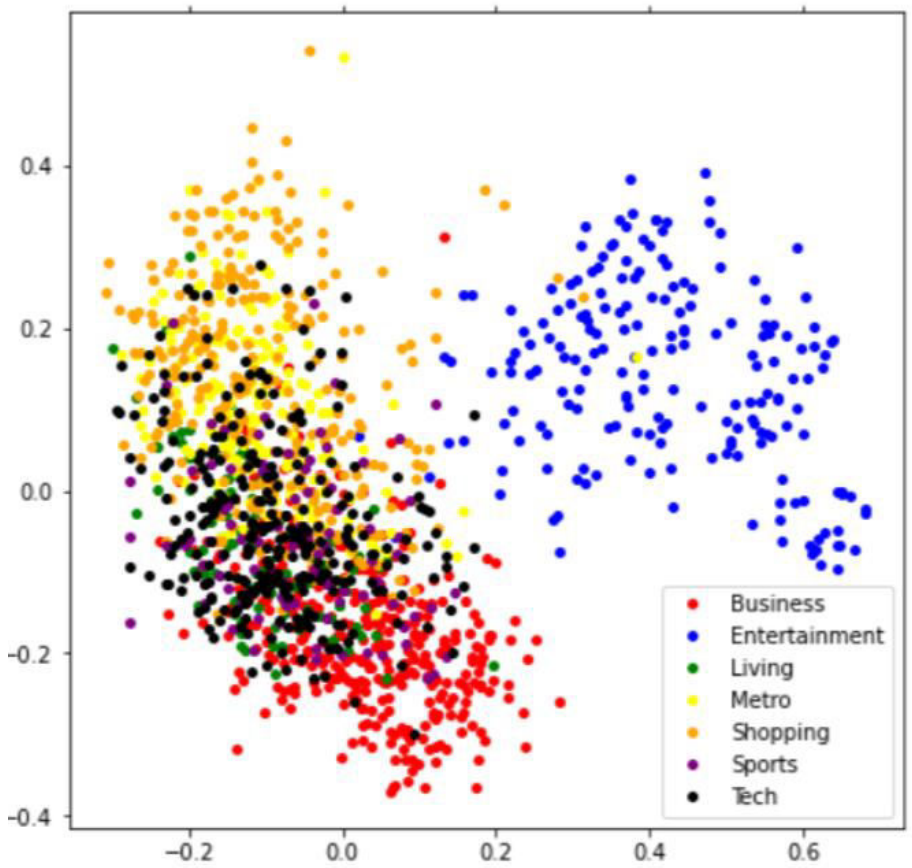
2-1-4. pkl 형태로 저장

```
TEAM = 11

with open('DMA_project3_team%02d_nb.pkl' % TEAM, 'wb') as f1:
    pickle.dump(clf_nb, f1)

with open('DMA_project3_team%02d_svm.pkl' % TEAM, 'wb') as f2:
    pickle.dump(clf_svm, f2)
```

2-2-1. 초기 모델 분석

```
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn import metrics
from sklearn.cluster import KMeans
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
from sklearn.pipeline import make_pipeline

categories = ['Business', 'Entertainment', 'Living', 'Metro', 'Shopping', 'Sports', 'Tech']

data1 = load_files(container_path='C:/Users/User/Downloads/DMA_project3/DMA_project3/CC/text_all', categories=categories, shuffle=True,
                   encoding='utf-8', decode_error='replace')

# TODO - Data preprocessing and clustering
data_trans=TfidfTransformer().fit_transform(CountVectorizer().fit_transform(data1.data))

clst1=KMeans(n_clusters=7,random_state=0)
clst1.fit(data_trans)
print(metrics.v_measure_score(data.target, clst1.labels_))
```

2-2-1. 초기 모델 분석


0.4
0.2
0.0
-0.2
-0.4
−0.2
0.0
0.2
0.4
0.6
Business
Entertainment
Living
Metro
Shopping
Sports
Tech

2-2-2. CountVectorier 수정

```
from nltk.corpus import stopwords

stop_words=set(stopwords.words('english'))
count_vec=CountVectorizer(stop_words=stop_words,min_df=10, max_df=600)
vec_data =count_vec.fit_transform(data.data)
data_tf  =TfidfTransformer().fit_transform(vec_data)
```

Stop_words: English
Min_df, max_df 설정
Tf, idf vectorize

2-2-3. svd normalizer 사용

```
<1501x32932 sparse matrix of type '<class 'numpy.float64'>'
        with 346257 stored elements in Compressed Sparse Row format>
```

```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
from sklearn.pipeline import make_pipeline

svd = TruncatedSVD(n_components=100, random_state=44)
normalizer = Normalizer(copy=False)
lsa = make_pipeline(svd,normalizer)
data_lsa=lsa.fit_transform(data_tf)
```

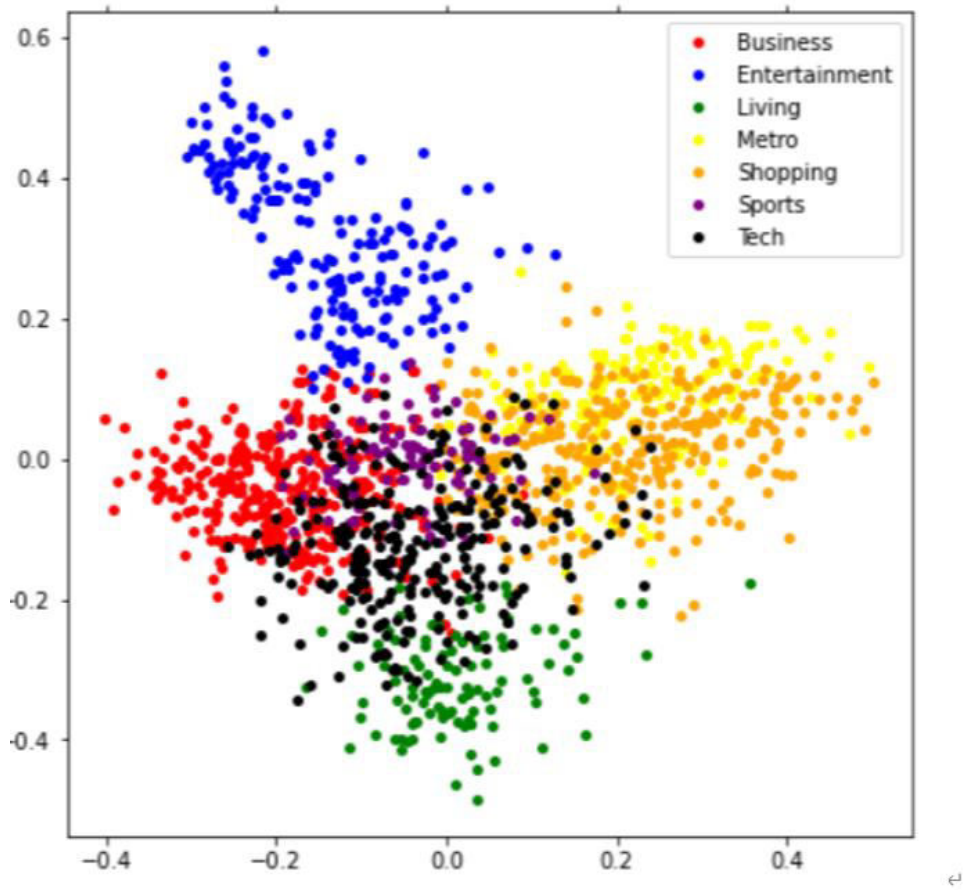
2-2-4. random state 값 조정

```
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn import metrics
from sklearn.cluster import KMeans
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
from sklearn.pipeline import make_pipeline

categories = ['Business', 'Entertainment', 'Living', 'Metro', 'Shopping', 'Sports', 'Tech']

data = load_files(container_path='C:/Users/User/Downloads/DMA_project3/DMA_project3/CC/text_all', categories=categories, shuffle=True,
                  encoding='utf-8', decode_error='replace')

# TODO - Data preprocessing and clustering
stop_words=set(stopwords.words('english'))
count_vec=CountVectorizer(stop_words=stop_words,min_df=10, max_df=600)
vec_data =count_vec.fit_transform(data.data)
data_tf  =TfidfTransformer().fit_transform(vec_data)

svd = TruncatedSVD(n_components=100, random_state=44)
normalizer = Normalizer(copy=False)
lsa = make_pipeline(svd,normalizer)
data_lsa=lsa.fit_transform(data_tf)
clst = KMeans(n_clusters=7, random_state=0)
clst.fit(data_lsa)
print(metrics.v_measure_score(data.target, clst.labels_))
```

2-2-5. 최종 모델 평가

```
from sklearn.decomposition import PCA
pca = PCA(2)
pca.fit(data_lsa1)
X_PCA = pca.transform(data_lsa1)
X_PCA.shape

x, y = X_PCA[:, 0], X_PCA[:, 1]

colors = {0: 'red', 1: 'blue',2: 'green', 3: 'yellow', 4: 'orange',  5:'purple',6:'black'}
names = {0: 'Business', 1: 'Entertainment', 2: 'Living', 3: 'Metro', 4: 'Shopping',5:'Sports', 6:'Tech'}
df = pd.DataFrame({'x': x, 'y':y, 'label':labels})
groups = df.groupby('label')

fig, ax = plt.subplots(figsize=(7, 7))

for name, group in groups:
    ax.plot(group.x, group.y, marker='o', linestyle='', ms=5,
            color=colors[name],label=names[name], mec='none')
    ax.set_aspect('auto')
    ax.tick_params(axis='x',which='both',bottom='off',top='off',labelbottom='off')
    ax.tick_params(axis= 'y',which='both',left='off',top='off',labelleft='off')

ax.legend()

plt.show()
```

2-2-5. 최종 모델 평가


Business
Entertainment
Living
Metro
Shopping
Sports
Tech
0.6
0.4
0.2
0.0
-0.2
-0.4
−0.4
−0.2
0.0
0.2
0.4