

정보검색과데이터마이닝 기말 과제

소프트웨어학부 20191650 이한정

1. Toknizer

1.1. 음절 Tokenizer

1.1.1. Bigram Tokenizer

```
def vectorize_document(doc_arr):  
    vectorizer = TfidfVectorizer(min_df=1)  
    X = vectorizer.fit_transform(doc_arr)  
    X = X.todok()  
  
    return X
```

min_df는 TfidfVectorizer()의 인자값 중 하나로 DF(document frequency)의 최소 빈도값을 설정해주는 파라미터이다. 이 값을 키우면 불용어 제거와 비슷한 효과를 나타낼 수 있다. 이를 여러 값으로 변경해가며 min_df 값이 커질수록 분류 성능이 좋아지는지를 알아보았다.

1.1.2. Trigram Tokenizer

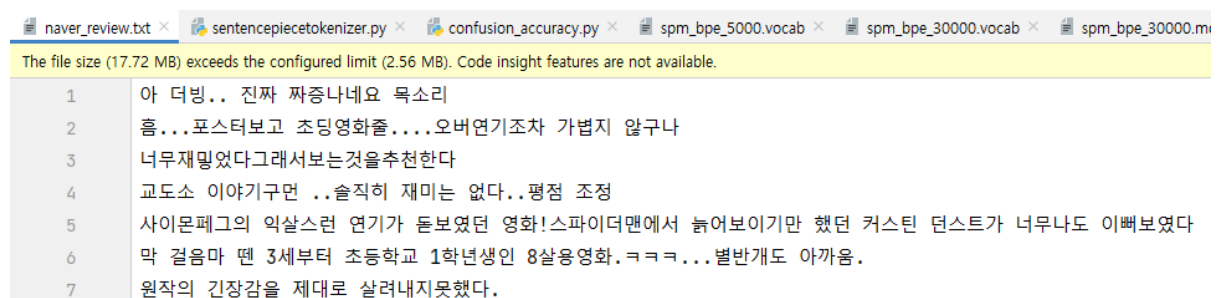
위의 Bigram Tokenizer와 동일하게 진행하였다.

1.2. SentencePiece Tokenizer

1.2.1 vocab_size = 5000(default)

```
spm.SentencePieceTrainer.Train(input='naver_review.txt', model_prefix='spm_bpe_5000', model_type='bpe')
```

vocab_size는 SentencePieceTrainer.Train()의 인자값 중 하나로 말 그대로 vocab size 즉, 사전 크기를 나타낸다. Default 값은 5000이고, 나는 이 값이 커질수록 성능이 좋아질 것이라는 생각에 아래의 vocab_size = 30000으로도 실험해보았다. 익히 알고있는 Etri kobert는 이 vocab_size가 32000개, Skt kobert는 8000개라고 한다.



```
1 아 더빙.. 진짜 짜증나네요 목소리  
2 흠...포스터보고 초딩영화줄....오버연기조차 가법지 않구나  
3 너무재밌었다그래서보는것을추천한다  
4 교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정  
5 사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 던스트가 너무나도 이뻐보였다  
6 막 걸음마 댔 3세부터 초등학교 1학년생인 8살용영화.ㅋㅋㅋ...별반개도 아까움.  
7 원작의 긴장감을 제대로 살려내지못했다.
```

입력으로는 다음과 같이 ratings_train.txt와 ratings_test.txt의 document 값만을 뽑아 저장한 naver_review.txt(20만개의 문서로 이루어짐) 파일을 주었다.

출력으로는 spm_bpe_5000.model과 spm_bpe_5000.vocab 파일이 생성된다.

```
f_all = open("naver_review.txt", 'r', encoding="utf-8")
f_train_sp = open("train_spm_5000.txt", 'w', encoding="utf-8")
f_test_sp = open("test_spm_5000.txt", 'w', encoding="utf-8")

all = f_all.readlines()
f_all.close()

sp = spm.SentencePieceProcessor()
vocab_file = "spm_bpe_5000.model"
sp.load(vocab_file)

tokenized_list = []

for line in all[:150000]:
    tokens_list = sp.encode_as_pieces(line)
    tokenized_list.append(tokens_list)
    for t in tokens_list:
        tokenized_list.append(t)
        f_train_sp.write(t + ' ')
    f_train_sp.write('\n')

f_train_sp.close()

for line in all[150000:]:
    tokens_list = sp.encode_as_pieces(line)
    tokenized_list.append(tokens_list)
    for t in tokens_list:
        tokenized_list.append(t)
        f_test_sp.write(t + ' ')
    f_test_sp.write('\n')

f_test_sp.close()
```

이를 이용하여 sentencepiece tokenizer로 토큰화한 document 파일을 만들었다. 이는 앞서 합친 train과 test를 나누어서 저장해주었다.

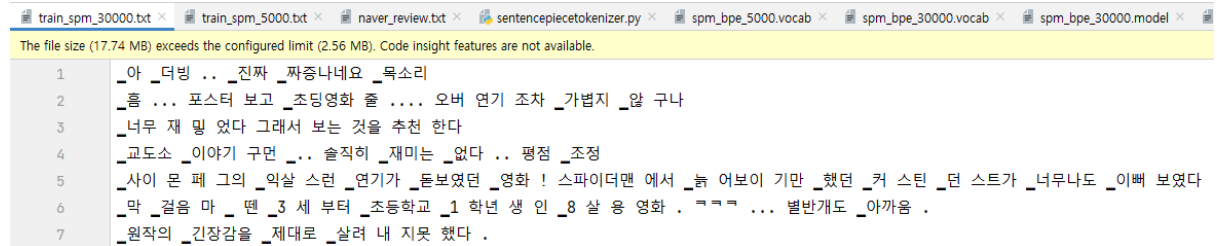
The screenshot shows a code editor with multiple tabs open. The active tab is 'train_spm_5000.txt'. Below the tabs, a message states: 'The file size (18.2 MB) exceeds the configured limit (2.56 MB). Code insight features are not available.' The main content area displays a table with 7 rows of tokenized Korean text. The text is as follows:

1	_아 _더빙 ... _진짜 _짜증나 네요 _목소리
2	_홀 ... _포스터 보고 _초딩 영화 줄 오 버 연기 조차 _가볍 지 _않 구나
3	_너무 재 밈 었다 그래서 보는 것을 추천 한다
4	_교 도 소 _이야기 구 먼 ... 솔직히 _재미는 _없다 .. 평점 _조 정
5	_사이 론 페 그 의 _익 살 스런 _연기가 _돋보 였던 _영화 ! 스 파이 더맨 에서 _늬 어 보이 기만 _했던 _커 스 틴 _던 스트 가 _너무나도 _이뻐 보 였다
6	_막 _걸 음 마 _텐 _3 세 부터 _초등학교 _1 학년 생 인 _8 살 용 영화 . ㅋㅋㅋ ... 별 반 개도 _아까움 .
7	_원작의 _긴장감 을 _제대로 _살려 내 지못 했다 .

결과는 위와 같다.

1.2.2 vocab_size = 30000

```
spm.SentencePieceTrainer.Train(input='naver_review.txt', model_prefix='spm_bpe_30000', model_type='bpe', vocab_size=30000)
```



```
1  _아 _더빙 .. _진짜 _짜증나네요 _목소리
2  _홀 ... 포스터 보고 _초딩영화 줄 .... 오버 연기 조차 _가법지 _않 구나
3  _너무 재 밉 었다 그래서 보는 것을 추천 한다
4  _교도소 _이야기 구면 ... 솔직히 _재미는 _없다 .. 평점 _조정
5  _사이 몬 페 그의 _악살 스런 _연기가 _돋보였던 _영화 ! 스파이더맨 에서 _늑 어보이 기만 _했던 _커 스티 _던 스트가 _너무나도 _이뻐 보였다
6  _막 _걸음 마 _텐 _3 세 부터 _초등학교 _1 학년 생 인 _8 살 용 영화 . ㅋㅋㅋ ... 별반개도 _아까움 .
7  _원작의 _긴장감을 _제대로 _살려 내 지못 했다 .
```

위의 vocab_size = 5000일 때와 모두 동일하게 진행하였다. 토큰화 된 결과를 보면 vocab_size = 30000인 경우가 더욱 단어 형성이 잘 되어있는 것을 볼 수 있다. (ex. 스 파이 더맨 <-> 스파이더맨)

2. Word Embedding

2.1. TF-IDF

```
##### vectorize #####
X = vectorize_document(bi_arr) # 문서번호, 단어번호

X = sorted(X.items(), key=lambda item:item[0])
#####

def vectorize_document(doc_arr):
    vectorizer = TfidfVectorizer(min_df=5)
    X = vectorizer.fit_transform(doc_arr)
    X = X.todok()

    return X
```

Sklearn의 내장함수인 TfidfVectorizer를 이용하여 토큰화된 결과를 입력으로 주고, 각 토큰의 TF-IDF 값을 계산하여 벡터화하였다.

```
f_doc_vec_train = open("document_vector_train_bi_mindf_5.dat", 'w', encoding="utf-8")
f_doc_vec_test = open("document_vector_test_bi_mindf_5.dat", 'w', encoding="utf-8")

cnt = 0
l = label[cnt]
f_doc_vec_train.write(str(l) + " ")

for doc in X:
    doc_id = doc[0][0]
    feature = doc[0][1]
    tfidf = doc[1]

    if (cnt < 150000):
        if (cnt == doc_id):
            f_doc_vec_train.write(str(feature) + ":" + str(tfidf) + " ") # 파일로 저장하기
        else:
            f_doc_vec_train.write("\n")
            cnt += 1
            if (cnt < 150000):
                l = label[cnt]
                f_doc_vec_train.write(str(l) + " ")
            if (cnt == 150000):
                l = label[cnt]
                f_doc_vec_test.write(str(l) + " ")
    else:
        if (cnt == doc_id):
            f_doc_vec_test.write(str(feature) + ":" + str(tfidf) + " ")
        else:
            f_doc_vec_test.write("\n")
            cnt += 1
            if (cnt < 200000):
                l = label[cnt]
                f_doc_vec_test.write(str(l) + " ")
```

위에서의 vectorize 결과를 가지고 SVM 파일 형식으로 만들어 저장해주었다.

2.2. Word2Vec

```
from sentencepiece.tokenizer import tokenized_list

result = tokenized_list

model = Word2Vec(sentences = result, vector_size = 100, window = 5, min_count = 1, workers = 4, sg = 0)

model_vocab = list(model.wv.index_to_key)
model_vocab_vec = [model.wv[v] for v in model_vocab]

sum_vocab_vec = []
for vec in model_vocab_vec:
    sum_vocab_vec.append(sum(vec))
```

Gensim의 Word2Vec 함수를 이용하여 임베딩해주었다. 입력으로는 sentencepiece.tokenizer 모듈에서 구해놓은 tokenized_list를 넣어주었다. Word2Vec도 중심 단어를 예측하기 위해 앞, 뒤로 몇 개의 단어를 볼지 결정하는 window의 값을 변경해가며 실행시켜보고 싶었지만, 시간이 부족하여 해보지 못하였다.

```
f_svm_test = open('svm_word2vec_spm_bpe_test_30000.dat', 'w', encoding="utf-8")
f_svm_train = open('svm_word2vec_spm_bpe_train_30000.dat', 'w', encoding="utf-8")

cnt = 0
for tokens in tokenized_list:
    if (cnt < 150000):
        l = label[cnt]
        f_svm_train.write(str(l) + " ")
    else:
        l = label[cnt]
        f_svm_test.write(str(l) + " ")

    dict = {}

    for token in tokens:
        idx = model_vocab.index(token)
        vec = sum_vocab_vec[idx]
        dict[idx] = vec

    sdct = sorted(dict.items()) # 왜냐하면 svm은 feature number가 오름차순이어야 함

    if (cnt < 150000):
        for key, val in sdct:
            f_svm_train.write(str(key + 1) + ":" + str(val) + " ") # key + 1 왜냐하면 svm은 feature number가 1 이상이어야 함
        f_svm_train.write("\n")
    else:
        for key, val in sdct:
            f_svm_test.write(str(key + 1) + ":" + str(val) + " ") # key + 1 왜냐하면 svm은 feature number가 1 이상이어야 함
        f_svm_test.write("\n")

    if (cnt % 1000 == 0):
        print(cnt)

    cnt += 1
```

위에서 구한 결과를 가지고 SVM 파일을 만들기 위하여 다음과 같은 코드를 작성하였다.

```
svm_word2vec_spm_bpe_train_30000.dat  svm_word2vec_spm_bpe_train_5000.dat  ratings_train.txt  ratings_test.txt  sentencepiecetokenizer.py  test_spm_30000.txt  train_spm_30000.txt  word2vec.py
The file is too large: 46.17 MB. Read-only mode.
-1 2:0.538402397884056 10:5.905627366155386 43:-7.519375376403332 986:-5.269392233341932 1653:-3.127878364175558 16826:-0.10725762916263193
1 3:-0.46918961146932095 17:-2.441836515441537 159:-19.4806663240380608 381:-5.360798786045052 421:-2.499343978241086 431:5.359352445229888 753:-4.498373749200255 1396:8.738978313282132 1433:4.963594960514456 6194:-2
.640690531115979 16879:3.196202892344445 16890:0.867772826294201 23097:0.5632544783002231
-1 7:-19.188266371493228 258:6.1456306939944625 503:8.646861001849174 551:0.013056600466370583 633:-4.379905128851533 1476:3.338869107887149 2431:2.1540655176884495 3144:1.0469899913296103 28832:-0.09682353015523404
-1 2:0.538402397884056 72:14.497599998489022 149:-5.888381662033498 193:12.423954413738102 792:-1.9874929941724986 904:8.132144411094487 3918:0.12570073063216114 7912:1.0459898639237508 9775:-0.10029697185382247 16302:0
.22485975810448173
1 4:15.954690819839016 9:13.765084664337337 52:-17.31037171476055 278:3.5839632083661854 1861:-10.63298703264445 1565:4.952266129432246 1866:-2.707570134778507 2264:-3.6868100026622415 2483:5.6064491285942495 2900:-3
.6473431270569563 3780:-3.9835353845264763 3819:-1.66121687553823 6086:-3.1470431110356003 6463:-0.01857373584061861 8534:0.6185237616300583 9594:-0.77575623207641 9774:0.8539359144342598 12986:1.78776582248247 21222:0
.22772768698632717 22121:-1.2152323468471877 23233:0.9034709383558948 24399:-0.5280407496611588 28923:-0.13032437744550407
-1 1:-0.4736430589109659 3:-0.46910961146932095 21:-1.8742062542587519 40:19.940276478417218 47:-15.755123026669025 75:6.219955525128171 89:-7.298122765496373 170:-3.708147222176194 250:0.6724596507847309 277:-2
.285232282736305 295:-10.30309536963701 326:-7.106805352494121 399:-5.428537989035249 461:-5.903858057750556 520:-12.801222019828856 847:-2.0758568793535233 1133:12.36076932400465 4943:4.103154834359884 5168:0
.8852215730585158 21228:0.2876181975007057 24411:1.6374244675971568 29706:0.300710068389898
-1 1:-0.4736430589109659 238:-0.47669152496382594 251:-1.4901646031066775 346:3.0737027041614056 3028:5.549850742332637 3669:0.529473926348146 5141:-3.6902978969737887 5928:3.0468937407713383

svm_word2vec_spm_bpe_train_5000.dat  svm_word2vec_spm_bpe_train_30000.dat  train_spm_30000.txt  train_spm_5000.txt  naver_review.txt  sentencepiecetokenizer.py  word2vec.py  svm_word2vec_spm_bpe_test_5000.dat  test_spm_5000.txt  test_spm_30000.txt
The file is too large: 54.91 MB. Read-only mode.
-1 2:-0.12400035886093974 26:-1.3016420528353572 57:-4.333948163315654 65:-14.731083145132288 1037:1.1327967583201826 2159:-3.1121436797002424 2720:4.770755448844284
1 3:-2.799595112912357 18:22.43519312515895 29:15.6335343811661 36:-1.8913750941865146 159:-8.862129061402847 168:1.7031022645533085 279:0.16872743144631386 431:3.095640581101179 454:4.693011028226465 500:3.955377226229757
510:-7.770253978902474 1656:-1.1971594695933163 1773:2.148756462149322 1901:7.5177427250089192 5362:1.078152148402296 5513:-8.404641381772235
-1 17:-8.572091794116803 275:-9.180562736466527 299:9.64097146311562 376:-14.151253159201580 557:-0.14872141182422638 1482:9.377659486606717 1970:-0.8780136608984321 4232:4.119420997914858 7933:-0.37154019171430264
-1 2:-0.12400035886093974 9:11.06421688709408 97:3.8816629201177782 118:-19.270560268312493 137:-11.802393739577383 193:-12.350913591682911 238:-5.73033464976866 302:1.299769977107644 307:0.5505021153949201 996:-5
.391258297022432 1181:-1.7131679872982204 1532:0.9183124025277793 1598:-11.653326015919447 5014:0.01893866325318813
1 4:4.608083601109635 8:8.034801422152668 14:-1.295908099040389 21:6.3155738124623895 34:-2.994377814233303 40:1.6581805683672420 62:-0.6231034067459404 94:3.695809952914715 190:-1.4053995572030544 388:-11.173043239861727
574:2.397251613321714 637:-1.0584374973550439 671:-12.302645278163254 1107:12.42080730667457 1492:-2.813524422208204 1758:-9.95441519562155 1926:-3.023587624076754 2062:4.053926145890728 2384:0.03756292560137808 2413:-3
.3841267006460911 2486:-10.423137257166341 2510:1.2353209351422265 2583:-7.69612179370597 2630:-2.0381574584171176 2783:-3.2503143437206745 3246:4.087321507744491 3948:-4.35600236186292 6545:2.4420124534517527 6595:-4
.241059581981972 6680:-4.793223076558206 6815:-1.8657844718565885
-1 1:-3.4172608596272767 3:-2.799595112912357 7:-17.769415549002588 29:15.6335343811661 39:-2.837627226486802 52:-6.5271695628762245 96:-4.995941434521228 141:-18.849851527251303 153:-12.898094927892089 196:-2
.5011461151298136 236:-3.870032020293176 263:17.48320273961072 310:-5.350650118663907 388:-11.173043239861727 470:-11.388733784648148 545:-1.416075203800574 577:2.413005432114005 672:-12.858919731806964 772:4
.533677666448057 1141:-1.8159572836011648 1858:-8.78879213752225 2628:-6.2159461081027985 3540:-10.35851847846061 6330:-2.0317314630374312 8168:0.38676764140836895
-1 1:-3.4172608596272767 10:32.200304763391614 202:8.300669117830694 248:-4.58209521509707 631:6.503886893391609 916:-5.30455582216382 3416:11.78187772430107 4121:-0.5205185990780592 4744:1.9000785453245044
```

위는 각각 vocab_size=30000, 5000일 때의 SVM 파일이다. 전자의 경우가 파일의 용량이 46.17MB, 후자의 경우가 54.91MB로 전자가 피쳐의 개수가 더 적다고 유추할 수 있다.(-> 원래 단어로 더 잘 묶음)

3. Classifier

3.1. SVM

```
document_vector_test_bi_mindf_1.dat
document_vector_test_bi_mindf_2.dat
document_vector_test_bi_mindf_3.dat
document_vector_test_bi_mindf_5.dat
document_vector_test_tri_mindf_1.dat
document_vector_test_tri_mindf_2.dat
document_vector_test_tri_mindf_3.dat
document_vector_train_bi_mindf_1.dat
document_vector_train_bi_mindf_2.dat
document_vector_train_bi_mindf_3.dat
document_vector_train_bi_mindf_5.dat
document_vector_train_tri_mindf_1.dat
document_vector_train_tri_mindf_2.dat
document_vector_train_tri_mindf_3.dat
svm_word2vec_bi_test_5.dat
svm_word2vec_bi_train_5.dat
svm_word2vec_spm_bpe_test_5000.dat
svm_word2vec_spm_bpe_test_30000.dat
svm_word2vec_spm_bpe_train_5000.dat
svm_word2vec_spm_bpe_train_30000.dat
svm_word2vec_test.dat
svm_word2vec_test_sorted.dat
svm_word2vec_train.dat
svm_word2vec_train_sorted.dat
svm_word2vec_tri_test.dat
svm_word2vec_tri_train.dat
```

위와 같이 여러 종류의 SVM 형식의 파일을 만들어 교수님께서 제공해주신 svm_learn.exe, svm_classify.exe를 이용하여 성능 측정을 해보았다.

3.2. KNN

각 test dataset의 document들을 train dataset의 모든 document들과 cosine similarity를 구하여 가장 유사도가 높은 다섯개의 train document를 구했다. 이 다섯개의 train document들의 라벨값(1/-1) 개수를 세어 많이 나온 라벨값과 test document의 라벨값을 비교하여 일치하는지, 불일치하는지를 판단해서 성능 평가를 하였다.

```
if __name__ == "__main__":
    arg = int(sys.argv[1]) # 이 번호까지의 영화평을 분석할거다.

    if len(sys.argv) != 2:
        print("Wrong Argument")
        sys.exit()

    start = time.time()
    dict_list, real_label_list = get_dict_list(arg, "document_vector_test_tri_mindf_3.dat")
    all_cos_sim_list = get_cos_list(dict_list, start) # 모든 cosine 유사도 구함
    pred_label_list = get_pos_neg(all_cos_sim_list) # 다섯개의 영화평 중 많이 나온 라벨이 들어있는 리스트를 반환한다.
    cm = confusion_matrix(real_label_list, pred_label_list)
    print(cm)

    print("긍정 영화평 중 일치 정확도:", cm[0][0]/sum(cm[0]))
    print("부정 영화평 중 일치 정확도:", cm[1][1]/sum(cm[1]))
```

최종 성능 평가는 (긍정 영화평 중 일치 정확도 + 부정 영화평 중 일치 정확도)/2 를 하여 구하였다.

4. 결과 정리

SVM

Tokenizer	Word Embedding	Classify Accuracy
Bigram + min_df=1	TfidfVectorizer	85.58%
Bigram + min_df=3	TfidfVectorizer	85.55%
Trigram + min_df=1	TfidfVectorizer	86.04%
Trigram + min_df=3	TfidfVectorizer	85.92%
Bigram + min_df=1	Word2Vec	82.81%
Trigram + min_df=1	Word2Vec	82.23%
SentencePiece + vocab_size=5000	Word2Vec	81.04%
SentencePiece + vocab_size=30000	Word2Vec	80.13%

KNN

Tokenizer	Word Embedding	Classify Accuracy
Bigram + min_df=1	TfidfVectorizer	81%
Bigram + min_df=3	TfidfVectorizer	81%
Trigram + min_df=1	TfidfVectorizer	81.95%
Trigram + min_df=3	TfidfVectorizer	81.95%
Bigram + min_df=1	Word2Vec	62.15%
Trigram + min_df=1	Word2Vec	60.9%
SentencePiece + vocab_size=5000	Word2Vec	72.05%
SentencePiece + vocab_size=30000	Word2Vec	63.05%

=> 분류 성능 측정 결과를 보았을 때, SVM 분류기가 KNN 분류기보다 성능이 좋고, Word Embedding의 경우 TF-IDF가 Word2Vec보다 우수함을 알 수 있다. min_df의 경우 높을수록 성능이 좋을 것이라 생각한 것과 다르게 min_df=1일 경우가 min_df=3일 경우보다 미세하게 성능이 좋았다. 다음 번에는 조금 더 차이를 뒤서 min_df=5, 10 등에 대하여 실험해보고싶다. SentencePieceTokenizer의 vocab_size의 경우에도 30000일 경우가 5000일 경우보다 성능이 우수할 것이라고 예상했지만 반대의 결과가 나왔다. 무조건적으로 vocab_size가 클수록 성능이 좋다가보다, 적절한 값이 있을 것이라는 생각이 들었다. 분류 성능면에서는 Trigram이 Bigram보다 미세하게 앞서지만, 직접 유사한 영화평을 출력해봤을 때엔 Bigram이 더 유사한 영화평을 잘 찾는다는 느낌이 들었다.