기말프로젝트

Word Clustering with G&I Corpus

김혜지(1771018) | 인공지능 | 7 월 2 일

Hyeji Kim: https://orcid.org/0000-0001-8711-178X

https://github.com/Maeg5854/Ewha-AI/Word Clustering with GNI Corpus.ipynb

Task: GNI 코퍼스 Word Clustering

Word2Vec 모델을 활용하여 G&I 코퍼스의 단어들을 클러스터링 한다. 또한 클러스터 결과를 시각화 하고 특정 단어가 포함된 클러스터를 출력하여 참조할 수 있게 기능을 구현한다. 코퍼스는 sentence tokenized 폴더 내의 G&I 코퍼스를 사용하였다. Gensim패키지의 Word2Vec모델과 Skit-learn 패키지의 K-Mean을 활용하였다. 전체 코드의 흐름은 다음과 같다.

1. 라이브러리 import
2. G&I corpus reader
3. Preprocessing
4. Word2Vec Modeling and Training
5. Word Clustering
6. Multidimensional Scaling (MDS)
7. Visualization of Word Vector
8. Searching the cluster with a word

코드설명

1. 라이브러리 import

```
# import nltk
import nltk
from nltk.corpus import *
from nltk import *

# libraries for word clustering
from gensim.models import Word2Vec
from sklearn.cluster import KMeans
import joblib

# import libraries for preprocessing
import re

# libraries for visualization
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.manifold import MDS
from sklearn.metrics.pairwise import cosine_similarity

import time
```

Word clustering과 Visualization에 필요한 라이브러리를 import한다. G&I 코퍼스의 Word Embedding을 위해서 Word2Vec를, Word clustering을 위해서 KMeans를 추가한다. 전처리에 사용되는 모듈은 re와 nltk.corpus 내의 stopwords와 nltk.WordNetLemmatizer이며, 클러스터링 한 결과를 시각화 하기 위해서 matplotlib.pyplot, sklearn 패키지의 MDS, cosine_similarity를 추가한다.

2. G&I corpus reader

```
# using sentence tokenized gni corpus.
corpus_root = 'C:/Users/Maeg/nltk_data/Genomics-Informatics-Corpus-master/Genomics-Ir
GNIcorpus = PlaintextCorpusReader(corpus_root,'.*₩.txt',encoding='utf-8')
```

Nltk.corpus.PlaintextCorpusReader 를 이용하여 sentence_tokenized 폴더 내의 모든 G&I 코퍼스들을 읽는다.

3. Preprocessing

```
# preprocessing each sentence
def gni_preprocessed_sents(gni_sents):

    # load stopwords and initialize preprocessed sentences list
    stops = set(stopwords.words("english"))
    wnl = nltk.WordNetLemmatizer()
    gni_pre_sents = []

    # remove non-letter words and stop words in each sentence
    for s in gni_sents :
        sent = []
        for w in s :
            if (re.match(r'[a-zA-Z0-9]+',w) != None) & (not w in stops) :
                sent.append(wnl.lemmatize(w.lower()))
        gni_pre_sents.append(sent)

    # return preprocessed sentences list
    return gni_pre_sents
```

```
# preprocessing GNI corpus
print("Start preprocessing... : G&I sentences")
gni_sents = gni_preprocessed_sents(GNIcorpus.sents())
print("End preprocessing: G&I sentences")
```

G&I 코퍼스를 전처리한다. Sentence_tokenized 폴더 내의 G&I 코퍼스는 문장 분할이 된 상태이다. 따라서 GNIcorpus.sents()를 이용하여 문장과 각 단어를 분할한다. Word Clustering 은 G&I 논문의 키워드 중심으로 클러스터링이 되므로, 코퍼스 문장 내에서 stop word, non-letter 들을 제거한다. 또한 복수표현, 시제표현을 WordNetLemmatizer()로 원형으로 전환해 입력한다.

전처리한 문장 리스트는 Word2Vec 의 input 의 shape 와 통일 시켜, 단어로 토큰화 된 문장의 리스트로 리턴한다.

```
Start preprocessing... : G&I sentences
End preprocessing: G&I sentences
```

그림 1 전처리 output

4. Word2Vec Modeling and Training

```
n_features = 300        # the number of Word2Vector Dimension
min_wordcount = 40      # Words which counts at least 20 times are available
n_workers = 4           # the number of parallel work
size_boundary = 10       # contextual window size
downsampling = 1e-3     # down sampling with frequent words

# Initialize and train Word2Vec model
print("Start Training ...")
model = Word2Vec(gni_sents,
                 workers = n_workers,
                 size = n_features,
                 min_count = min_wordcount,
                 window = size_boundary,
                 sample = downsampling)
print("End Training ")

# manage memory
model.init_sims(replace=True)

# save trained model
model.save("GNI_"+str(n_features)+"features_"+str(min_wordcount)+"minwords_"+str(size
```

Word2Vec 모델을 구축하고 학습시킨다. 모델의 파라미터는 다음과 같다. (workers) 모델 학습 시 병렬 학습을 위한 모듈 수는 4로 하였다. (size) 모델의 특징 개수은 300으로 설정하였다. (min_count) 최소한 40번 이상의 빈도수를 가진 단어들만 모델에 입력된다. (window) 문장 내에서 단어의 전후 10단어의 단어를 고려하여 문맥을 파악한다. (sample) 너무 자주 등장하는 단어에 의해서 편향된 결과가 나타나는 것을 방지하기 위해 다운 샘플링 값을 0.001로 설정한다.

학습 후에는 효율적인 메모리 관리를 위해 model.init_sims()로 불필요한 메모리를 정리한다.

학습된 모델을 특징개수, 최소 빈도수, window개수 정보를 이름에 포함하여 저장한다.

```
Start Training ...
End Training
```

그림 2 Word2Vec Modeling and Training output

5. Word Clustering

```
# clustering start time
start = time.time()
print("Start: Clustering")

# word vector of word2Vec model
gniwv = model.wv

# set the number of cluster
num_clusters = (int)(len(gniwv.vocab) / 5)

# Initialize a k-means object and use it to extract centroids(cluster)
kmeans_clustering = KMeans( n_clusters = num_clusters)
index = kmeans_clustering.fit_predict(gniwv.vectors)

end = time.time()
clustering_time = end - start
print("End  : Clustering")

# print Clustering information
print("₩n============== Clustering Information ==============")
print("Clustering time:", clustering_time)
print("# of Word vectors:", len(gniwv.vocab))
print("# of Clusters:", num_clusters)
```

Word2Vec 모델에서 wv(Word vector) attribute를 참조할 수 있다. wv는 해당 모델에 입력된 단어(vocab)와 해당 단어들의 벡터값(vectors)를 포함하여 클러스터링에 필요한 벡터값을 얻을 수 있다. 클러스터의 개수는 모델의 단어 개수의 5분의 1로 설정하였다.

KMeans 객체에 클러스터의 개수(n_clusters)를 설정하고, fit_predict() 메소드에 인자를 단어의 벡터값 gniwv.vectors)을 입력하여 클러스터링을 진행한다. 클러스터링 후 클러스터링에 소요된 시간, 벡터의 개수, 클러스터의 개수와 같은 클러스터링 결과를 요약하여 출력한다.

```
Start: Clustering
End  : Clustering

============== Clustering Information ==============
Clustering time: 38.91790795326233
# of Word vectors: 2191
# of Clusters: 438
```

그림 3 클러스터 개요

```
# save the cluster models
joblib.dump(kmeans_clustering, 'GNI_words_cluster.pkl')
```

클러스터링한 결과를 .pkl 파일로 저장한다.

```
['GNI_words_cluster.pkl']
```

그림 4 클러스터링 결과 저장

```
# word - cluster number matching
word_centroid_map = dict(zip(gniwv.index2word, index))

# Print whole clutering results
for cluster in range(num_clusters) :
    print('₩nCluster',cluster)
    words = []
    for k, c in word_centroid_map.items():
        if c == cluster :
            words.append(k)
    print(words)
```

Word_centroid_map에 모델의 각 단어와 해당 단어가 포함된 클러스터 넘버를 딕셔너리 형태로 저장한다. 클러스터링 결과를 클러스터 별로 출력한다.

```
['approach', 'developed', 'knowledge', 'make', 'technique', 'computational', 'learni
g', 'efficient']

Cluster 10
['increase', 'negative', 'reduced', 'decrease', 'specificity']

Cluster 11
['search', 'query', 'blast']

Cluster 12
['cell', 'mouse', 'liver', 'brain', 'carcinoma', 'death', 'heart', 'prostate', 'colo
n', 'lymphoma', 'renal', 'adenocarcinoma', 'pancreatic']

Cluster 13
['proportion', 'substitution', 'explained', 'assumed']

Cluster 14
['most', 'exist', 'newly', 'indeed', 'substantial', 'extent', 'accumulation']

Cluster 15
['needed', 'developing', 'appropriate', 'trial', 'personalized', 'benefit', 'core'
```

그림 5 클러스터 결과 출력

6. Multidimensional Scaling(MDS)

```
# calculate distribution
dist = 1-cosine_similarity(gniwv.vectors)

# initialize MDS
MDS()

# scaling multidimensions
mds = MDS(n_components=2, dissimilarity = "precomputed", random_state=1)
pos = mds.fit_transform(dist)

xs, ys = pos[:,0], pos[:,1]
```

결과를 시각화 하기 위해서 단어 벡터들을 2차원의 형태로 변환시킨다. 우선 각 단어 벡터들 간의 코사인 유사도(cosine_similarity())를 구한 후, 1-코사인유사도로 분포 값을 정의한다. Dist는 각 벡터간의 거리를 의미하며 (단어의 개수)*(단어의 개수) 크기의 배열이다.

MDS을 활용하여 차원을 압축한다. (n_components) 결과로 나오는 차원의 개수는 2차원이므로 2로 설정하였다. (dissimilarity) 차원 압축시에는 거리 매트릭스(distance matrix)가 필요한데, dist를 통해 단어간 거리값을 입력해 주므로 dissimilarity 옵션을 precomputed로 설정한다.

Pos에 2차원으로 변환된 각 단어들의 위치가 저장된다. 각 단어의 X좌표를 xs에, Y좌표를 ys에 저장한다.

7. Visualization of Word Clustering

```
# set colors and names of clusters
cluster_colors = {35: '#1b9e77', 36: '#d95f02', 37: '#7570b3', 38: '#e7298a', 39: '#6
cluster_names = {35: 'Cluster35',
                 36: 'Cluster36',
                 37: 'Cluster37',
                 38: 'Cluster38',
                 39: 'Cluster39'
                }
```

```
%matplotlib inline

# Create (x-position(2D), y-position(2D), word, cluster number) table
df_all = pd.DataFrame(dict(x=xs, y=ys,word=gniwv.index2word, label=index))

# Show only five Clusters, 35, 36, 37, 38, 39
condition = ( df_all['label'] >= 35 ) & ( df_all['label'] < 40 )
df = df_all[condition]

# grouping with cluster number
groups = df.groupby('label')
```

모든 클러스터의 개수는 465개로 한 번에 표시하기에 너무 큰 개수였다. 따라서 대표로 35부터 39까지 5개의 클러스터만 골라 출력하기로 한다. 시각화될 5개의 클러스터들의 이름과 표시 색을 지정한다.

2차원으로 변환된 x좌표(x=xs)와 y좌표(y=ys), 단어(word=gniwv.index2word), 클러스터 번호(label=index) 4가지 열로 구성된 데이터 프레임 df_all을 구축한다. Label 열의 값을 35이상 40미만으로 필터링하여 cluster35~39 데이터만 df에 저장한다. 클러스터(label)별로 시각화해야 하기위해 df.groupby('label')로 그룹화한다.

	x	y	word	label
57	0.512553	0.532738	genomic	35
126	-0.502866	0.519568	http	39
374	-0.425101	0.501599	www	39
380	-0.461749	0.379313	org	39
621	0.206540	0.371589	amount	35
681	0.026347	-0.568387	sva	36
905	-0.034764	0.482661	simple	38
906	-0.047593	-0.145413	contain	36

그림 6 df의 일부

```
fig, ax = plt.subplots(figsize=(17, 9)) # set size

ax.margins(0.05)

for cluseter_n, group in groups:
    ax.plot(group.x, group.y, marker='o', linestyle='', ms=12,
                label=cluster_names[cluseter_n],
            color=cluster_colors[cluseter_n],
            mec='none')
    ax.set_aspect('auto')
    ax.tick_params(₩
        axis= 'x',          # changes apply to the x-axis
        which='both',      # both major and minor ticks are affected
        bottom='off',      # ticks along the bottom edge are off
        top='off',         # ticks along the top edge are off
        labelbottom='off')
    ax.tick_params(₩
        axis= 'y',         # changes apply to the y-axis
        which='both',      # both major and minor ticks are affected
        left='off',      # ticks along the bottom edge are off
        top='off',         # ticks along the top edge are off
        labelleft='off')

ax.legend(numpoints=1)

for i in range(len(df)):
    ax.text(df.iloc[i]['x'],
            df.iloc[i]['y'],
            df.iloc[i]['word'], size=12)
plt.show()
```

df를 시각화 한다. ax는 시각화 될 내용을 정한다. 여백은 0.05로 설정하였다. 클러스터별(name)로 x좌표(group.x), y좌표(group.y)를 원모양('o')으로 사전의 설정한 색과 이름을 부여한다. Df를 다시 참조하여 각 좌표에 단어(df.iloc[i]['word'])를 크기 12로 표시한다.

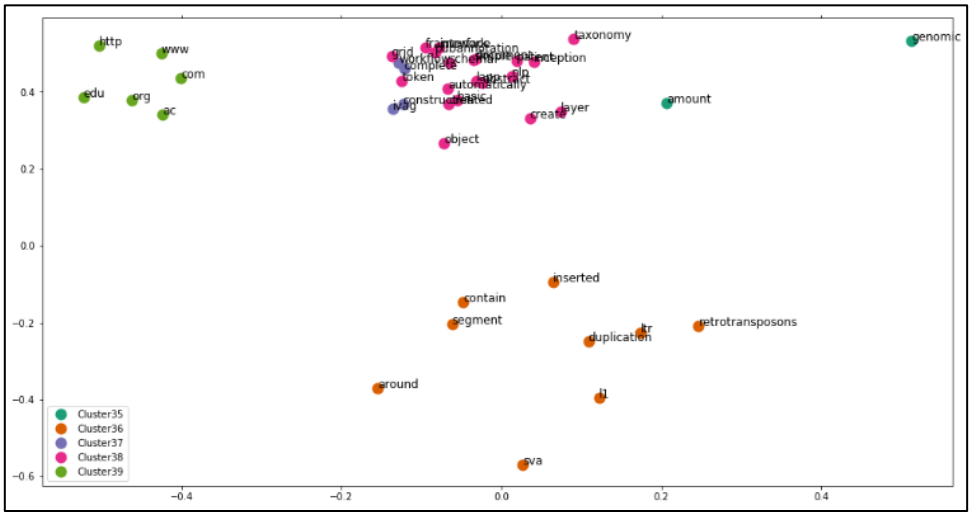

그림 7 Word 클러스터링 시각화 결과

8. Searching the cluster with a word

```
# find cluster result with specipic keyword
def find_cluster_with_key(key):
    # if there is not key in vocab, do nothing
    if not key in word_centroid_map.keys() :
        print("That word is not in keys")
    # if there is key in vocab, print cluster result
    else :
        cluster = word_centroid_map[key]
        print('₩nCluster',cluster)
        words = []
        for k, c in word_centroid_map.items():
            if c == cluster :
                words.append(k)
        print(words)
```

```
key = input("============= Find Clustering ==============₩nEnter a keyword: ")
find_cluster_with_key(key)
```

주어진 키워드가 포함된 클러스터를 출력하는 함수 find_cluster_with_key()를 정의하였다. 해당 함수는 키워드(key)를 인자로 받는다. 주어진 키워드가 어떤 클러스터에도 존재하지 않을 경우, 오류 메세지를 출력한다. 반대의 경우, 해당 키워드가 포함된 클러스터의 번호와 같은 클러스터에 포함된 키워드들을 모두 출력한다.

```
============= Find Clustering ==============
Enter a keyword: liver

Cluster 12
['cell', 'mouse', 'liver', 'brain', 'carcinoma', 'death', 'heart', 'prostate', 'colo
n', 'lymphoma', 'renal', 'adenocarcinoma', 'pancreatic']
```

그림 8 클러스터 키워드 탐색 결과

결과 분석

클러스터링은 Unsupervised Learning에 해당하는 내용이므로 정확도 분석을 위한 Y데이터를 구축할 수 없어 Accuracy를 계산할 수는 없다. 비록 정확성은 떨어지지만 각 클러스터 내의 단어 간의 연관도를 분석하여 클러스터 결과를 유의미하다고 생각되는 것(1)과 절반 가량 유의미한 것(0.5), 그렇지 않은 것(0)으로 직접 분류하여 정확도를 분석하고자 한다. 이 분석에 사용된 클러스터링 결과는 'https://guthub.com/Maeg5854/Ewha-AI/GNI_Word_Clustering.txt'에서 확인할 수 있다.

클러스터 내에서 70-80%가량 연관이 있는 단어들이 존재한다면 유의미한 클러스터로, 40-50%가량 존재한다면 절반 가량 유의미한 클러스터로, 전혀 규칙성을 찾을 수 없을 경우에는 그렇지 않은 클러스터로 분류하였다. 각 카테고리로 분류한 클러스터의 개수와 해당되는 클러스터 번호는 다음과 같다.

표 1 Word Clustering Hand-Evaluation

	유의미한 클러스터	절반 가량 유의미한 클러스터	그렇지 않은 클러스터
개수	56	22	360
비율	12.8%	5.00%	82.2%
클러스터 번호	2 8 10 12 13 23 29 31 34 39 46 54 57 58 63 70 74 80 97 98 99 103 107 117 126 128 135 143 148 165 171 177 181 191 201 225 231 234 235 248 249 268 302 306 310 319 330 336 347 350 357 400 403 417 432 434	3 5 11 22 25 26 48 64 106 113 118 147 162 190 233 290 329 359 371 385 393 398	0 1 4 6 7 9 14 15 16 17 18 19 20 21 24 27 28 30 32 33 35 36 37 38 40 41 42 43 44 45 47 49 50 51 52 53 55 56 59 60 61 62 65 66 67 68 69 71 72 73 75 76 77 78 79 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 100 101 102 104 105 108 109 110 111 112 114 115 116 119 120 121 122 123 124 125 127 129 130 131 132 133 134 136 137 138 139 140 141 142 144 145 146 149 150 151 152 153 154 155 156 157 158 159 160 161 163 164 166 167 168 169 170 172 173 174 175 176 178 179 180 182 183 184 185 186 187 188 189 192 193 194 195 196 197 198 199 200 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 226 227 228 229 230 232 236 237 238 239 240 241 242 243 244 245 246 247 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 291 292 293 294 295 296 297 298 299 300 301 303 304 305 307 308 309 311 312 313 314 315 316 317 318 320 321 322 323 324 325 326 327 328 331 332 333 334 335 337 338 339 340 341 342 343 344 345 346 348 349 351 352 353 354 355 356 358 360 361 362 363 364 365 366 367 368 369 370 372 373 374 375 376 377 378 379 380 381 382 383 384 386 387 388 389 390 391 392 394 395 396 397 399 401 402 404 405 406 407 408 409 410 411 412 413 414 415 416 418 419 420 421 422 423 424 425 426 427 428 429 430 431 433 435 436 437

클러스터링 평가 결과 전체 438개의 클러스터 중 82.2%에 해당하는 360개의 클러스터가 단어 간의 연관성이 떨어지는 것으로 나타났다. 또한 12.8%에 해당하는 56개의 클러스터가 유의미하였으며, 5.00%에 해당하는 22개의 클러스터가 절반정도의 유효성을 보여주는 클러스터였다. 각 클러스터링의 예시는 다음과 같다.

표 2 카테고리별 클러스터 예시 (window size = 10)

유의미한 클러스터	절반 가량 유의미한 클러스터	그렇지 않은 클러스터
Cluster 46 ['statistical', 'performance', 'accuracy', 'procedure', 'mdr', 'predictive']	Cluster 5 ['population', 'asian', 'breed', 'cattle', 'hanwoo', 'holstein', 'african', 'pig', 'ancestry', 'east']	Cluster 0 ['still', 'microbial', 'limited', 'advance', 'epigenome', 'effort', 'metagenomic', 'technical', 'enabled']

이와 같은 결과가 나타나는 이유는 문맥 window의 크기가 큼과 데이터 전처리의 부족으로 추정된다. 현재 클러스터링에 사용된 문맥 window의 크기는 10으로 특정 단어 앞뒤로 각각 10개 즉 20개의 단어와 연관성을 Word2Vec에 반영한다. 이 크기가 문맥을 파악하기에 너무 큰 수치여서 연관성이 크지 않은 단어들도 함께 클러스터링 된 것으로 추정되어 window의 크기를 6으로 설정하여 다시 클러스터링 하였다. 그리고 표2의 결과와 비교하기 위해 유의미한 클러스터의 'statistical'과 절반 가량 유의미한 클러스터의 'population'이 포함된 클러스터를 각각 출력하였다. 그 결과 같음과 같이 클러스터링 결과가 나아짐을 확인할 수 있었다.

표 3 카테고리별 클러스터 예시 (window size = 6)

유의미한 클러스터	절반 가량 유의미한 클러스터
Cluster 197 ['test', 'variable', 'statistical', 'regression', 'linear', 'logistic']	Cluster 384 ['population', 'breed', 'cattle', 'hanwoo', 'ne', 'holstein', 'african', 'pig', 'ancestry']

또한 데이터 전처리의 부족이 있었다. 각 단어를 원형의 형태로 일반화해 단어간 연관성을 높이려 WordNetLemmatizer를 사용했다. 그러나 클러스터 결과를 보면 'dataset'과 'datasets'가 서로 다른 단어로 인식되어 실제 전처리가 제대로 적용되지 않은 것을 알 수 있었다.

따라서 문맥 window의 크기의 적정값을 찾아 적용하고, 전처리 단계에서 단어의 원형으로 온전히 전환시킨 후 클러스터링을 진행하면 더 높은 유효성을 보여줄 것으로 기대된다.