인공지능 기말 프로젝트

1685128 이유진
1829008 김민영

목차

0. 프로젝트 개요

1) 사용한 데이터

: PUBMED에서 bioinformatics 관련 저널 12개의 abstract (2015년)

2) 프로젝트 목적

: bioinformatics 관련 저널 12개를 다각도로 분석, 분류 모델 구축

3) 분석 방향

- 국가별, 학과별 word cloud 및 단어 분포 파악
- journal 내 paper 별로 유사도 분석, keyword extraction, clustering을 통해서 각 paper 별 특징 분석
- journal 별 (journal 내의 paper 구분 X) 유사도 분석, keyword extraction, 클러스터링 등을 통해 각 journal 별 특징 분석
- journal을 label로 하여 지도 학습을 수행하는 LSTM 분류 모델 학습

0. 프로젝트 개요 (ctd.)

4) 분석 기법

- Document Representation: TF-IDF, Doc2Vec
- Clustering: K-means (Non-hierarchical), Hierarchical Clustering
- Dimensionality Reduction for Visualization: PCA, MDS, T-SNE
- Similarity: Cosine Similarity
- Classifier: LSTM with Convolution layer (Multi-class Classification)

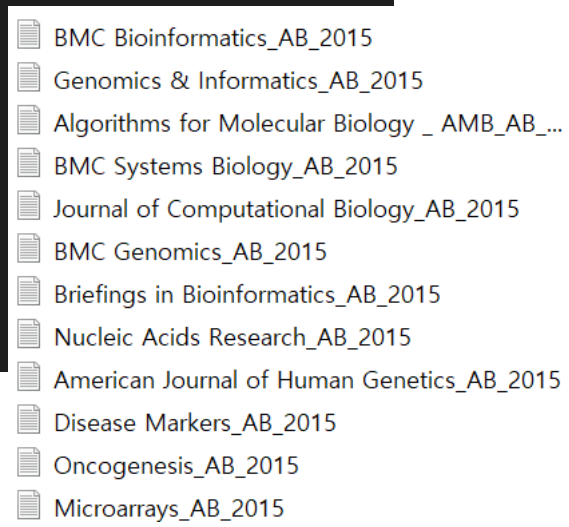
1. 데이터수집

Biopython을 이용하여,
Pubmed의 12개 저널에 대해 2015년 논문 추출

1. 데이터 수집

1) Abstract 추출

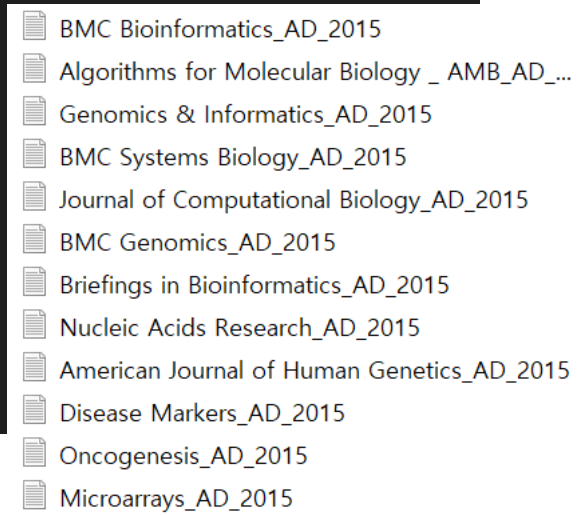
```
for journal in journalList:
    f = open(curdir + journal + "_AB_2015" + ".txt", "w", encoding="utf-8")
    keyword = "(₩"" + journal + "₩"[Journal]) AND (₩"2015/01/01₩"[Publication Date] : ₩"2015/12/31₩"[Publication Date]) "
    handle1 = Entrez.esearch(db='pmc', term=keyword, retmax=1000000)
    record = Entrez.read(handle1)
    idlist = record['IdList']
    handle1.close()

    handle2 = Entrez.efetch(db="pmc", id=idlist, rettype="medline",retmode="text")
    records = Medline.parse(handle2)
    records = list(records)

    i = 0
    if idlist:
        for record in records:
            i += 1
            f.write("" + ''.join(record.get("AB", "?"))+ "₩n")
    else:
        print(journal + ": list empty!!!")
    f.close()
```



1. 데이터 수집

2) Affiliation 추출

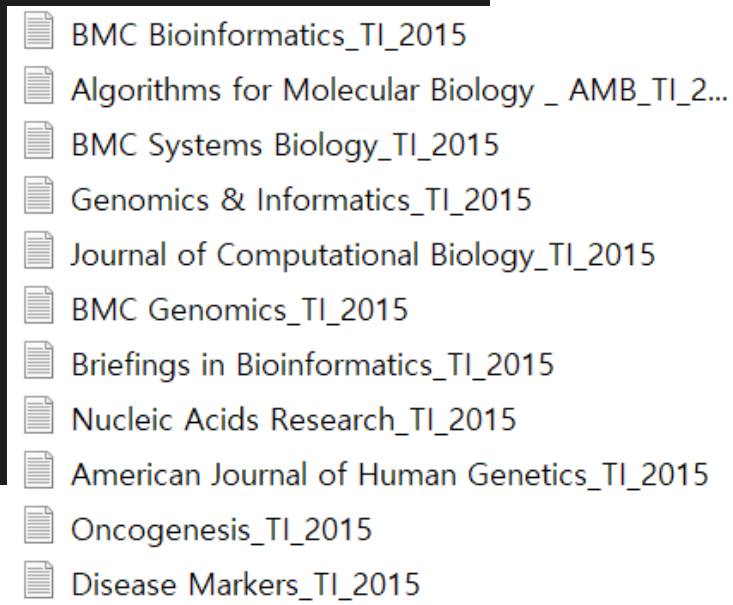
```
for journal in journalList:
    f = open(curdir + journal + "_AB_2015" + ".txt", "w", encoding="utf-8")
    keyword = "(₩"" + journal + "₩"[Journal]) AND (₩"2015/01/01₩"[Publication Date] : ₩"2015/12/31₩"[Publication Date]) "
    handle1 = Entrez.esearch(db='pmc', term=keyword, retmax=1000000)
    record = Entrez.read(handle1)
    idlist = record['IdList']
    handle1.close()

    handle2 = Entrez.efetch(db="pmc", id=idlist, rettype="medline",retmode="text")
    records = Medline.parse(handle2)
    records = list(records)

    i = 0
    if idlist:
        for record in records:
            i += 1
            f.write("" + ''.join(record.get("AB", "?"))+ "₩n")
    else:
        print(journal + ": list empty!!!")
    f.close()
```



1. 데이터 수집

3) Title 추출

```
for journal in journalList:
    f = open(curdir + journal + "_AB_2015" + ".txt", "w", encoding="utf-8")
    keyword = "(₩"" + journal + "₩"[Journal]) AND (₩"2015/01/01₩"[Publication Date] : ₩"2015/12/31₩"[Publication Date]) "
    handle1 = Entrez.esearch(db='pmc', term=keyword, retmax=1000000)
    record = Entrez.read(handle1)
    idlist = record['IdList']
    handle1.close()

    handle2 = Entrez.efetch(db="pmc", id=idlist, rettype="medline",retmode="text")
    records = Medline.parse(handle2)
    records = list(records)

    i = 0
    if idlist:
        for record in records:
            i += 1
            f.write("" + ''.join(record.get("AB", "?"))+ "₩n")
    else:
        print(journal + ": list empty!!!")
    f.close()
```



2. 데이터 전처리 및 토크나이징

corpus에서 정제가 필요한 부분을 찾아내고 제거,

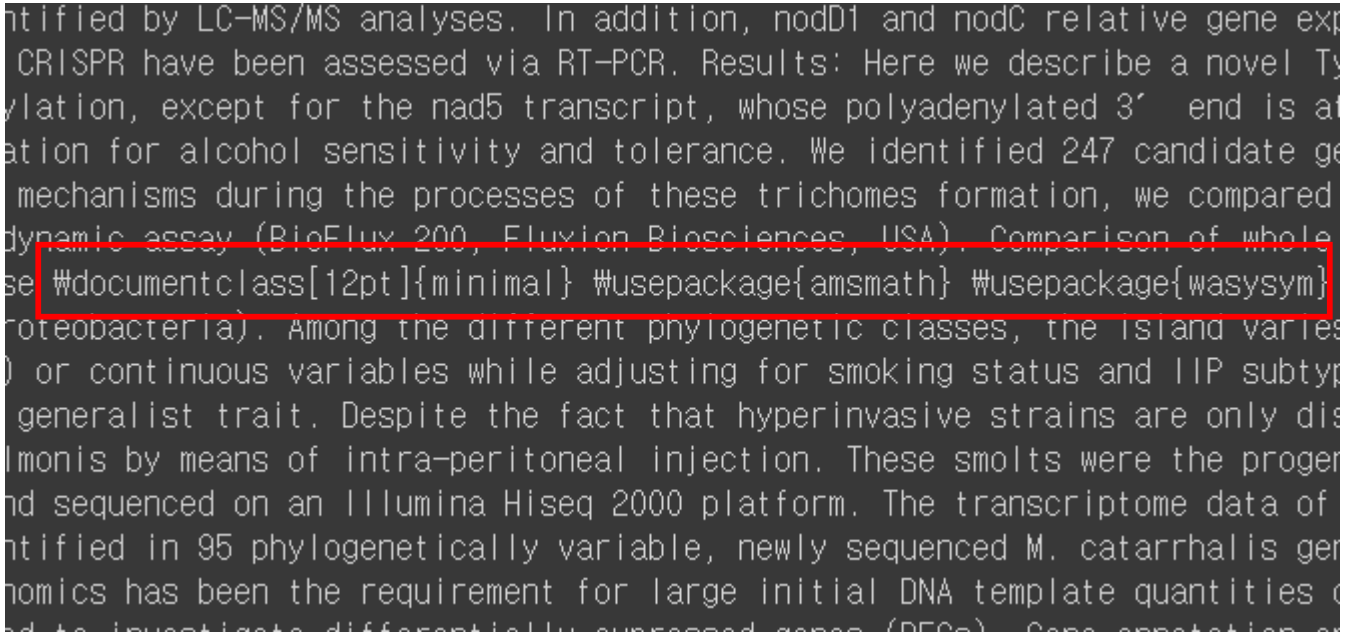
이후 토크나이징 후 최종적으로 prettify

2. 데이터 전처리 및 토크나이징

corpus에서 정제해야할 부분의 유형은 다음과 같았다.

- markdown 표현식으로 추정되는 패턴

: 빨간 박스 내의 표현식은 모든 경우에 ₩₩documentclass[12pt]{minimal}로 시작하여, ₩₩endofdocument{} 로 끝난다는 점을 발견할 수 있었다.



2. 데이터 전처리 및 토크나이징

- remove_markdown 함수

: ₩₩documentclass ~ 알파벳, 숫자, 모든 기호를 포함하는 [] block } 정규표현식 패턴을 활용하여 삭제

```
def remove_markdown(text):
  """
  markdown 같은 것들 지우기
  """
  pattern = "₩s₩₩₩₩documentclass₩[12pt₩]{minimal}[a-zA-Z₩s₩₩₩₩{}^_₩-0-9₩$₩+₩*,:@!#%&₩(₩)=₩|₩?/₩.<>]+}"
  return re.sub(pattern, "", text)
```

2. 데이터 전처리 및 토크나이징

- 하이퍼링크

using 0/1 values. Experimental results show that Gpos-ECC-mPLoc and Gneg-ECC-mPLoc can efficiently predict the subcellular locations of multi-label gram-positive and gram-negative bacterial proteins respectively. Conclusions: Gpos-ECC-mPLoc and Gneg-ECC-mPLoc can efficiently improve prediction accuracy of subcellular localization of multi-location gram-positive and gram-negative bacterial proteins respectively. The online web servers for Gpos-ECC-mPLoc and Gneg-ECC-mPLoc predictors are freely accessible at http://biomed.zzuli.edu.cn/bioinfo/gpos-ecc-mploc/ and http://biomed.zzuli.edu.cn/bioinfo/gneg-ecc-mploc/ respectively.₩nBackground: When biomolecules physically interact, natural selection operates on them jointly. Contacting positions in protein and RNA structures exhibit correlated patterns of sequence evolution due to constraints imposed by the interac

- remove_hyperlink 함수

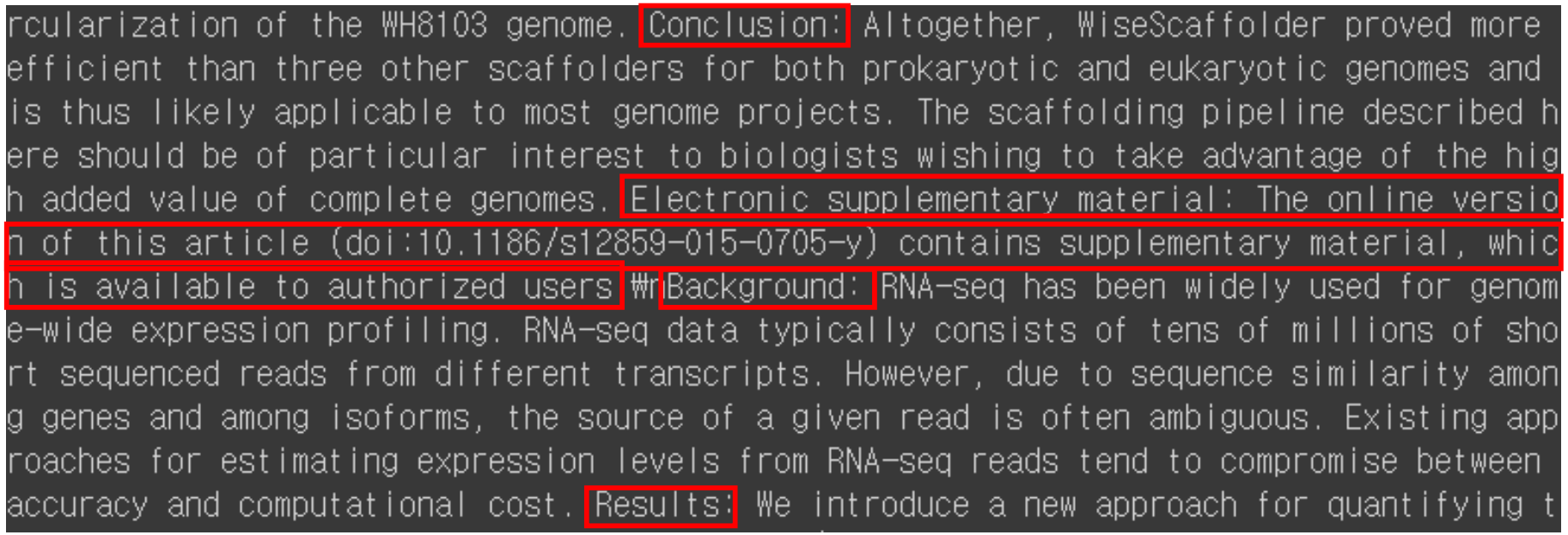
: “[a-z]+://[a-z0-9.-_]+” 정규표현식 패턴을 이용해 “”로 대체하는 함수

```
def remove_hyperlink(text):
  """
  하이퍼링크 지우기
  """
  regex = re.compile('[a-z]+://[a-z0-9.-_]+', flags = re.IGNORECASE)
  return regex.sub("", text)
```

2. 데이터 전처리 및 토크나이징

- 소제목 등 삭제

: Conclusion, Results, Background의 소제목이 대부분의 문서에 등장하므로 삭제, 또한 문서 끝에 등장하는 Electronic ~ users 부분 또한 document 내용에 큰 의미가 없으므로 삭제함.



2. 데이터 전처리 및 토크나이징

- remove_subheading 함수

: 정규표현식 패턴으로 처리, “Conclusions: | Results: |Background: |Electronic supplementary material.+” 패턴을 모두 빈칸으로 sub하는 함수

```
def remove_subheading(text):
  """
  subheading 지우기
  """
  return re.sub("Conclusions: |Background: |Results: |Electronic supplementary material: .+", "", text)
```

- preprocessing 함수

: 앞의 세 함수를 한꺼번에 적용하도록 하는 preprocessing 함수

```
def preprocessing(text):
  """
  텍스트 전처리 종합적으로 처리하는 함수
  """
  return remove_markdown(remove_hyperlink(remove_subheading(text)))
```

2. 데이터 전처리 및 토크나이징

- preprocessing 함수를 string 차원에서 적용한 후, tokenizing 후에 token 단위로 추가 전처리

1) 모든 토큰을 소문자로 변환

: 토큰을 소문자로 변환하면서 같은 단어의 중복을 방지하여 메모리 효율성 등을 제고함.

```
# 소문자로 변환
def get_lower_case(doc_tokens):
  """
  - input
  doc_tokens: tokenized documents
  - output
  tokenized documents with tokens lowered
  """
  return [token.lower() for token in doc_tokens]
```

2. 데이터 전처리 및 토크나이징

2) stop words 제거

: 의미 상 중요하지 않은 불용어 등을 삭제함. 이때, stopwords_set에 string 모듈의 punctuation을 추가함으로써 punctuation으로만 이루어진 토큰도 함께 삭제하도록 함.

```
# document로부터 stopwords 제거
def remove_stopwords_from_doc(doc_tokens, stopwords_set):
  """
  - input
  doc_tokens: tokenized document (in lower case)
  stopwords_set: a set of stopwords
  - output

  """
  return [token for token in doc_tokens if token not in stopwords_set]
```

```
## stopwords set
sw_eng = set(stopwords.words("english"))
sw_eng.update(list(string.punctuation))
```

2. 데이터 전처리 및 토크나이징

3) 숫자, 기호 등 삭제

: 숫자, 혹은 숫자 & 기호, 기호 등으로 이루어진 부분을 삭제함.

```
# get letters only
def get_letters_only(doc_tokens):
  """
  eliminate digits and marks

  - input
  doc_tokens: tokenized document (in lower case)
  - output
  document tokens w/o digits and mark
  """
  return [token for token in doc_tokens if (token.isalnum()) and (not token.isdigit())]
```

2. 데이터 전처리 및 토크나이징

4) lemmatizing

: nltk.stem.WordNetLemmatizer()를 이용하여 lemmatizing을 진행함. Stemmer를 사용하지 않은 이유는 사전에 존재하지 않는 단어 등을 리턴하기도 하므로 의미 해석이 불분명해지는 지점이 존재하기 때문.

```
# lemmatizing
def lemmatizing(doc_tokens):
  """
  - input
  doc_tokens: tokenized document (in lower case)
  - output
  lemmatized document tokens (list)
  """
  lemmatizer = nltk.stem.WordNetLemmatizer()
  return [lemmatizer.lemmatize(token) for token in doc_tokens]
```

2. 데이터 전처리 및 토크나이징

5) tokenize_only

: tokenize만 진행하는 함수.

```
def tokenize_and_stem(text):
  tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
  filtered_tokens = []
  stemmer = nltk.SnowballStemmer('english')

  for token in tokens:
      if re.search('[a-zA-Z]', token): # get_letters_only와 동일한 기능
          filtered_tokens.append(token)
  stems = [stemmer.stem(t) for t in filtered_tokens]
  return stems
```

2. 데이터 전처리 및 토크나이징

6) tokenize and stem

: tokenize하고 stemming하는 함수.

```
def tokenize_and_stem(text):
  tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
  filtered_tokens = []
  stemmer = nltk.SnowballStemmer('english')

  for token in tokens:
      if re.search('[a-zA-Z]', token): # get_letters_only와 동일한 기능
          filtered_tokens.append(token)
  stems = [stemmer.stem(t) for t in filtered_tokens]
  return stems
```

7) 한 글자로 이루어진 토큰 삭제

: 한 글자로 이루어진 토큰은 의미 파악이 불분명하므로 삭제함.

```
def remove_one_letter(doc_tokens):
  """
  - input
  doc_tokens: tokenized document (in lower case)
  - output
  document tokens over at least 2 letters
  """
  return [token for token in doc_tokens if len(token) > 1]
```

2. 데이터 전처리 및 토크나이징

8) comprehensive_tokenizing

: 종합적으로 토크나이징을 고려하는 함수. 3가지 mode가 있고, 해당 모드는 어근 추출을 어떻게 할 것인지에 대한 것이다. “stem”, “lemmatize”, “none”의 3가지 모드가 있으며, default는 어근 추출을 하지 않는 “none”이다.

```
# 토크나이징 관련 함수 종합
def comprehensive_tokenizing(document, stopwords_set, mode = "none"):
  """
  - input
  document: document (string)
  stopwords_set
  mode: lemmatize or stem or none (in string)
  - output
  prettified document tokens (list)
  """
  doc_tokens = word_tokenize(document)
  if mode == "stem":
    return remove_one_letter(remove_stopwords_from_doc(tokenize_and_stem(document), stopwords_set))
  elif mode == "lemmatize":
    return remove_one_letter(lemmatizing(get_letters_only(remove_stopwords_from_doc(get_lower_case(doc_tokens), stopwords_set))))
  else:
    return remove_one_letter(remove_stopwords_from_doc(tokenize_only(document), stopwords_set))
```

3. EDA

① 국적별
② 학과별

국적분포

국적분포 조사과정

1. pycountry를 이용하여 country_list를 구축

   1) 나라이름 불일치 문제 처리 : pycountry에는 “Korea, Republic of” 이라고 저장되어있으나, Affliation에는 Korea라고 저장되어있는 경우들이 존재하여 전처리

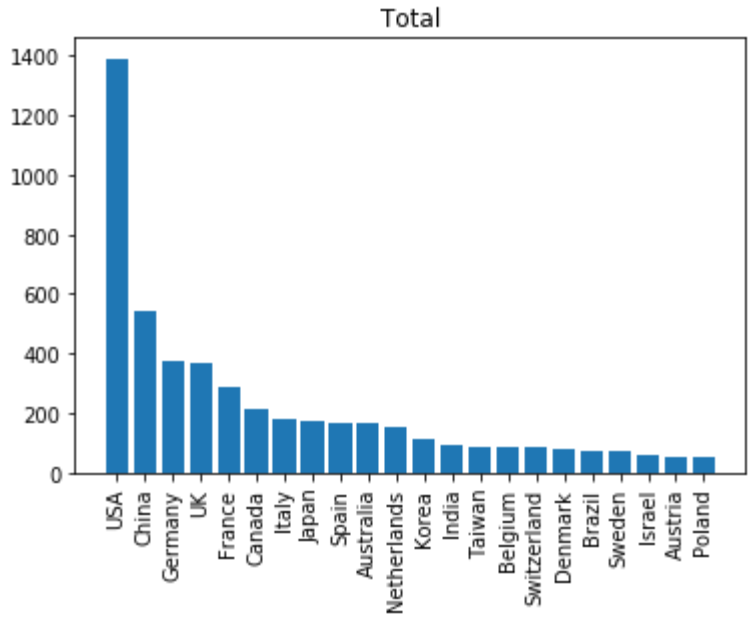
   2) 결측값 처리 : New York 등 나라 이름이 존재하지 않고 도시명으로 적혀있는 경우, 해당 도시가 속해있는 나라 추출

2. Affliation으로부터 나라이름 추출

3. 저널별 국적분포 시각화

국적분포

- 총 국적 분포

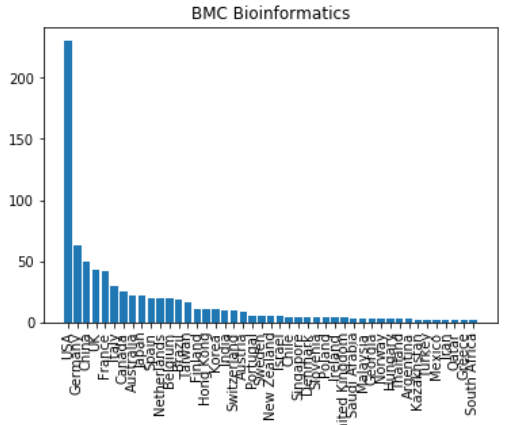


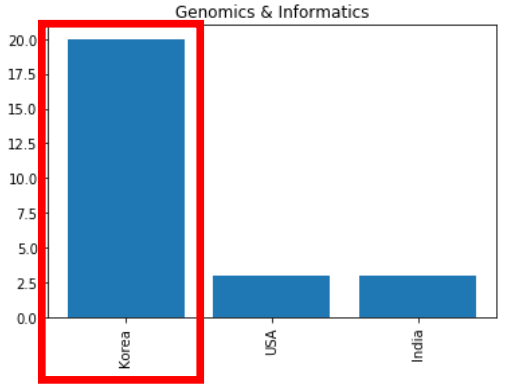
12개 저널의 총 국적 분포는 다음과 같다.

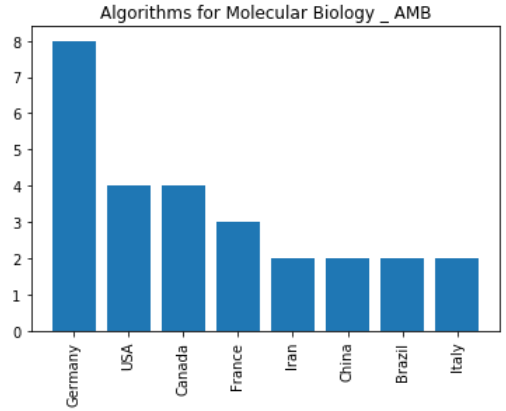
대부분의 저널에서 USA, China, Germany, UK 등이 많았으나몇몇 저널에서는 특이한 점이 발견되었다.
Genomics & Informatics에서는 한국의 논문수가 압도적으로 많았으며, Disease Markers에서는 중국이 많았다.

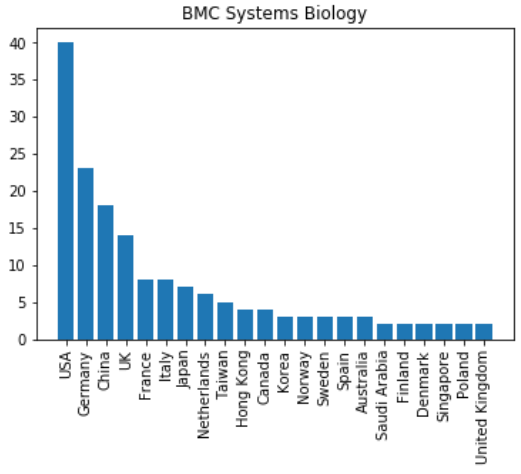
국적분포

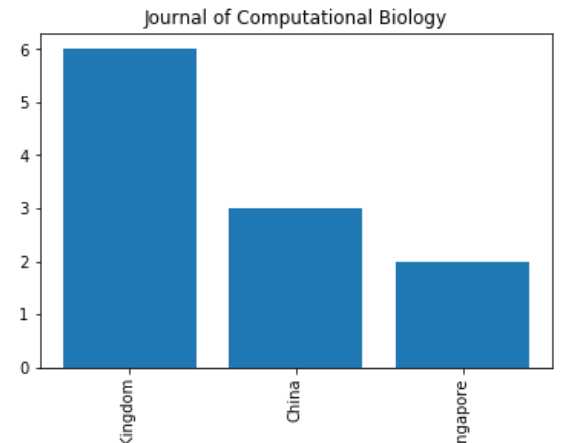
- 저널별 국적분포





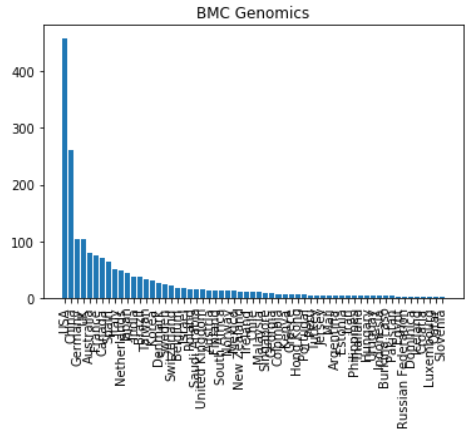


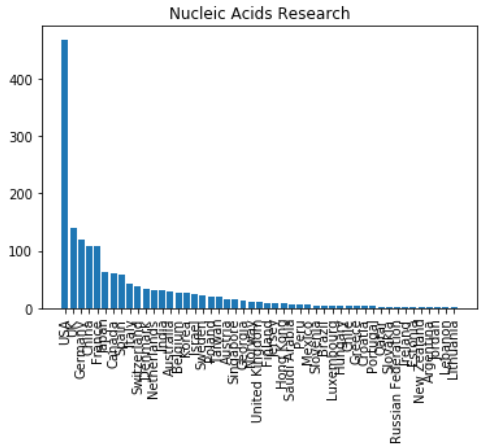


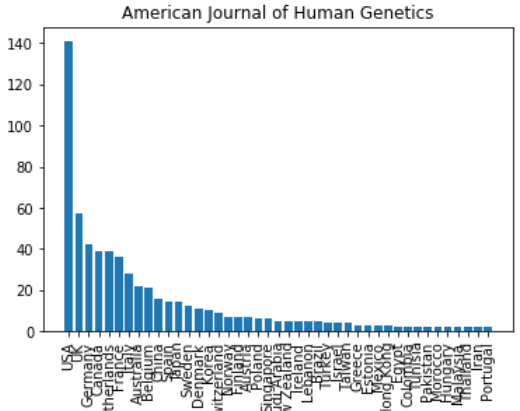


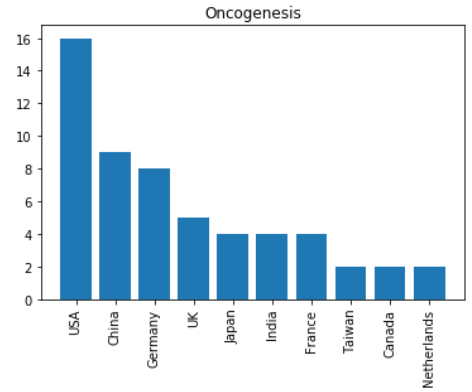
국적분포

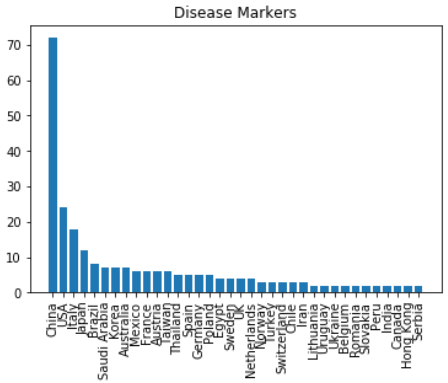
- 저널별 국적분포

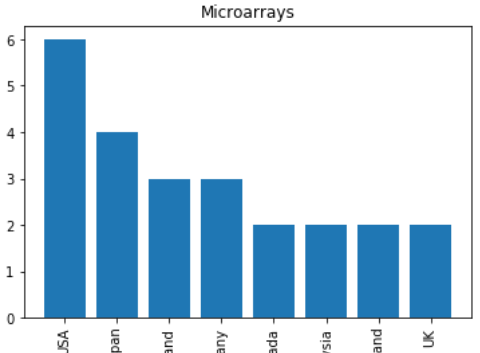








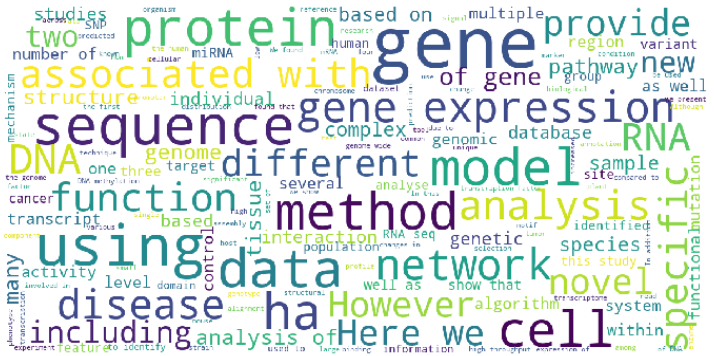




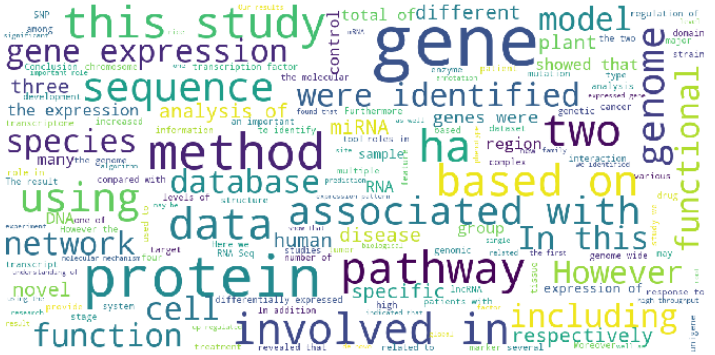
국적분포

- 나라별 워드클라우드

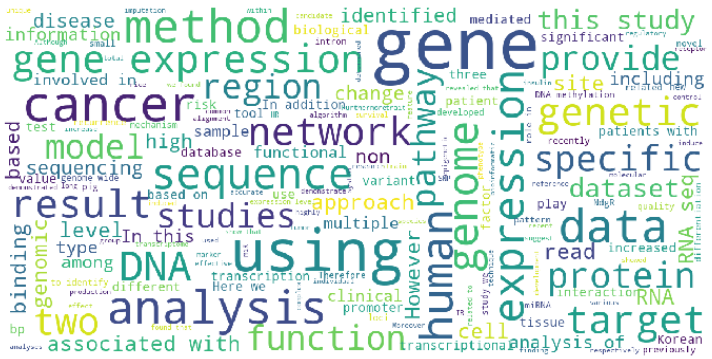
USA



China



Korea



Germany

-> 나라별로 워드클라우드 분포에는 큰 차이가 없었다.

학과별 분석

학과분포 조사과정

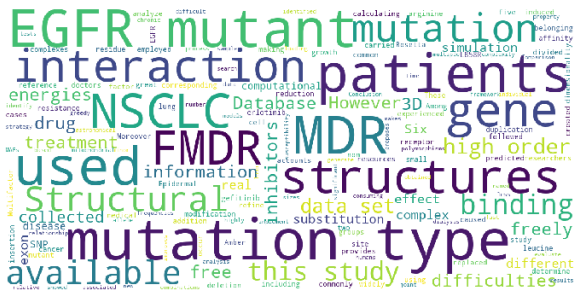
1. Affliation을 살펴본 결과, 다양한 학과들이 존재했는데 그중 주요 학과만 따로 dept_list를 만들어 살펴보았다.
2. Affiliation으로 부터 학과 추출
3. 학과별 워드클라우드

학과분포

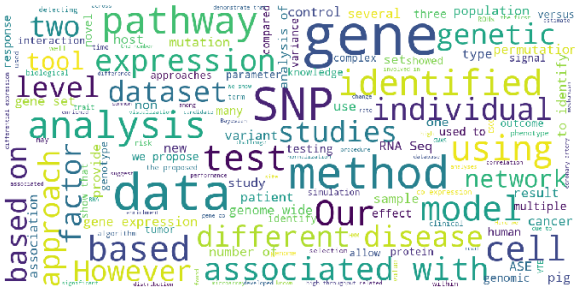
Computer Science

Life Science

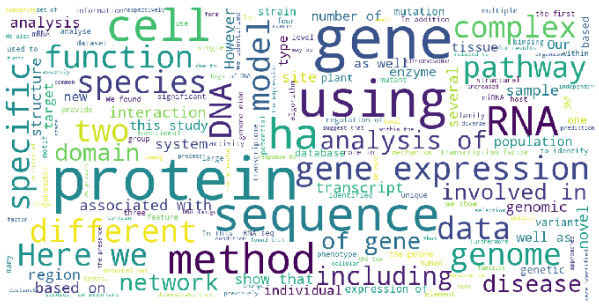
Electronic Engineering



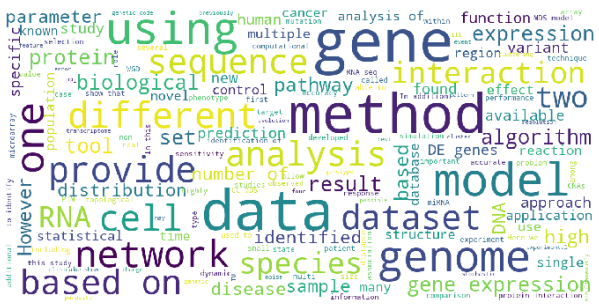
Statistics



Biology



Mathematics



4. 저널 내 분석

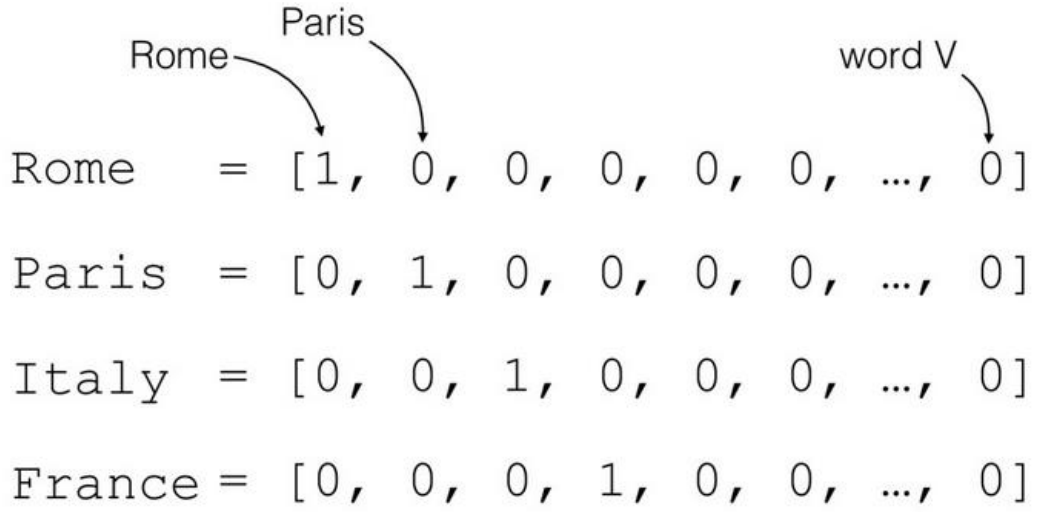
G&I, Nucleic Acids Research journal 내의 paper에 대해서
유사도 분석, keyword extraction, clustering을 진행함으로써
저널 내 paper에 대해 다각도로 분석하고자 한다.

참고)
Text Feature Representation 설명

Feature Representation for Text

- Sparse Representation: 해당 속성이 가질 수 있는 모든 경우의 수를 각각 독립적인 차원으로 표현하는 방식. 원하는 단어를 Dummy Variable로 표현하는 경우가 이에 해당한다고 볼 수 있다. 이 경우 Encoded text의 대부분의 요소가 0이게 되므로, 이러한 text representation 기법을 "Sparse"한 텍스트 표현방식이라고 한다.

ex.) one-hot encoding



https://medium.com/@athif.shaffy/one-hot-encoding-of-text-b69124bef0a7

Feature Representation for Text (ctd.)

- Dense Representation (Distributed Representation)

: 사전에 결정한 차원으로 텍스트를 대응시켜 표현한다. 앞선 one-hot encoding의 경우 대부분의 값이 0을 갖는 희소행렬로 텍스트가 표현되지만, dense representation 하에서는 대부분의 값이 실수 값을 갖게 된다. 단어의 개수인 N보다 작은 d차원에 좀 더 밀집하여 텍스트가 존재하게 되므로 dense representation이라고 하며, 이러한 표현 방법을 embedding이라고 한다.

	animal	fluffiness	dangerous	spooky
aardvark	0.97	0.03	0.15	0.04
black	0.07	0.01	0.20	0.95
cat	0.98	0.98	0.45	0.35
duvet	0.01	0.84	0.12	0.02
zombie	0.74	0.05	0.98	0.93

Document Embedding (Doc2Vec)

- Doc2Vec Concepts

: Word2Vec은 의미 공간에 word의 위치 좌표를 학습하는 방법이었다면, Doc2Vec은 이를 확장하여 의미 공간에 document의 위치 좌표를 학습하는 방법이다.

Doc2Vec은 각 document에 document tag(document id)를 붙여 이를 하나의 단어처럼 취급한 후, document id + window size 만큼의 이전 단어로 타겟 단어를 예측하는 과정에서 문맥이 비슷한 문서 벡터와 단어 벡터가 유사(in cosine similarity)하도록 업데이트된다.

Document Embedding (Doc2Vec)

- Doc2Vec 학습 방법

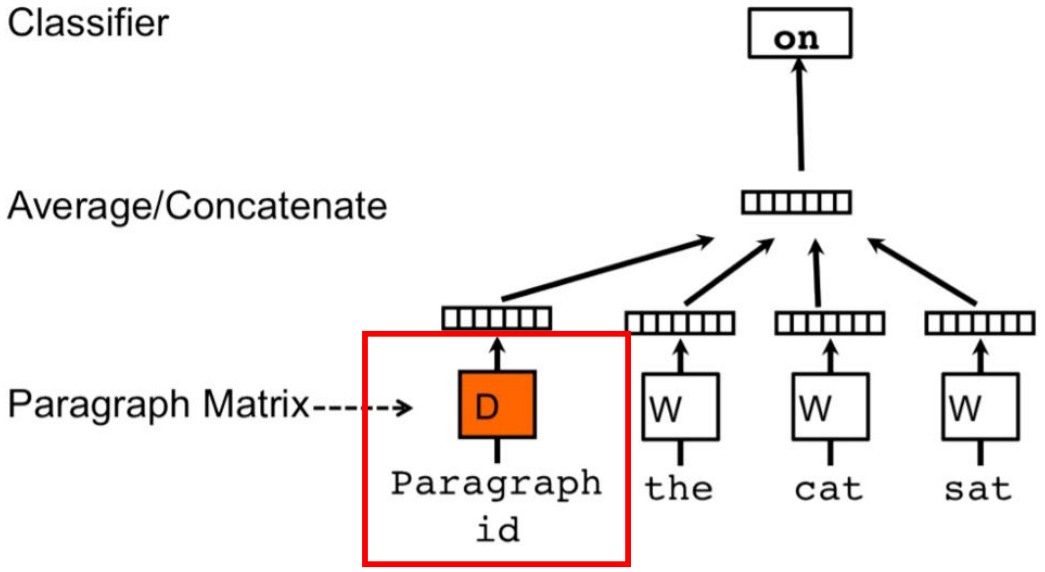
1) PV-DM

ex) doc1 (document id)이란 문서 내에서 the cat sat on the ecc 라는 문장이 있을 때, 학습 데이터는 다음과 같다.

- window size = 3
- [doc1, the, cat, sat] – 'on'
- [doc1, cat, sat, on] – 'the'
- [doc1, sat, on, the] – 'ecc'

→document에서 단어를 예측하며 로그 확률 평균을 최대화하는 과정에서 학습됨.

→document id가 인풋으로 들어가므로, 문맥이나 단어가 document matrix에 반영



N X d (N: 문서 개수)

Document Embedding (Doc2Vec)

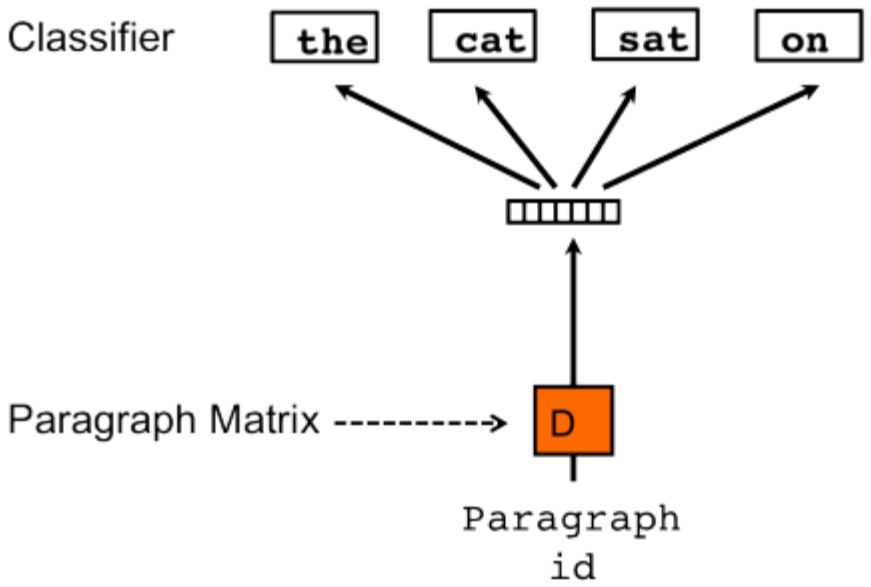
- Doc2Vec 학습 방법

2) PV-DBOW

: document vector를 input으로 넣고, 결과는 해당 document에 포함되어 있는 단어를 예측하는 방식으로 학습.

다만 이 경우에는 paragraph id만 input으로 들어가기 때문에 word embedding은 학습되지 않는다

반면, pv-dm은 학습데이터로 함께 들어가기 때문에 word vector도 함께 학습된다.



4-1)
Word Embedding &
Document Embedding

Document Embedding using D2V

```
## build doc2vec model and train
def train_doc2vec(tagged_document):
  max_epochs = 40
  vec_size = 100
  alpha = 0.025

  model = Doc2Vec(vector_size=vec_size,
                  alpha=alpha,
                  min_alpha=0.00025,
                  min_count=1,
                  window = 5,
                  dm =1)

  model.build_vocab(tagged_document)

  for epoch in range(max_epochs):
    model.train(tagged_document,
                  total_examples=model.corpus_count,
                  epochs=model.epochs)
    # decrease the learning rate
    model.alpha -= 0.0002
    # fix the learning rate, no decay
    model.min_alpha = model.alpha

    return model
```

- gensim 패키지에서 제공하는 Doc2Vec API를 활용해서 document embedding을 손쉽게 사용할 수 있다.
- train_doc2vec 함수를 정의

→ 100차원으로 임베딩 시키고, 학습 시 iteration 수는 40으로 설정함. learning rate는 0.025로 설정

→ window size = 5, pv-dm모델을 사용함. (word vector까지 같이 학습)

→ build_vocab을 통해서 model을 initialize하고 토큰을 모델에 올림

→ 40번 동안 학습을 진행함. 사용하는 example 수는 document 수로, 모델의 corpus_count를 사용함.

Document Embedding using D2V

- doc2vec을 사용하기 위해서 알아두어야 할 점은 TaggedDocument를 이용해서 document id를 명시해줘야 한다는 점이다.
- 저널 내 paper를 각각 임베딩해주어야 하기 때문에 각 journal을 담고 있는 리스트 -> 각 저널 -> 저널 내 paper 구조에서 Tagging을 해주어야 한다.
- org_doc 각각 요소는 저널 내에 paper 별로 tagging이 되어 있는 상태이다.
- d2v_list는 org_doc의 각 요소를 embedding한 결과이다.

```
### applying to original documents
org_doc = [[TaggedDocument(paper, [str(idx)]) for idx, paper in enumerate(journal)] for journal in papers_tokens]
d2v_list = list(map(lambda x: train_doc2vec(x), org_doc)) # 각 journal을 embedding한 모델이 요소인 리스트
```

4-2) Similarity

G&I 저널 내 paper에 대한 유사도 분석

- document 간 유사도
- gni journal 내의 29개의 document에 대해 cosine similarity를 계산함.
- 이후 numpy 모듈의 argmax를 사용하여 해당 문서와 가장 높은 유사도를 갖는 문서의 인덱스를 반환하도록 함

```
gni_embed = d2v_list[1] # gni corpus embedded
gni_embed_mat = gni_embed.docvecs.vectors_docs
```

```
similarities = np.triu(cosine_similarity(gni_embed_mat),k= 1)
which_is_similar = np.argmax(similarities, axis = 1) # row에 C
which_is_similar
```

← np.triu을 이용해, similarit의 upper triangular matrix를 return함. diagonal(자기 자신과의 유사도)는 무시

G&I 저널 내 paper에 대한 유사도 분석

- document 간 유사도

- 혹은 doc2vec.docvecs의 내장 메소드인 most_similar를 사용해도 유사한 결과를 얻을 수 있다.
- 다만, most_similar 메소드의 경우 vector의 mean을 사용하여 similarity를 계산하므로 앞선 결과와 완벽히 일치하지는 않는다.

```
[621] similarities = np.triu(cosine_similarity(gni_embed_mat),k= 1) # gni journal 내의 29개의 document 끼리의
      which_is_similar = np.argmax(similarities, axis = 1) # row에 대해서 max 값의 argument 리턴
      which_is_similar

      array([27, 21,  9,  8, 10, 10, 24, 21, 19, 12, 11, 14, 13, 18, 18, 20, 20,
             27, 19, 22, 22, 26, 28, 25, 27, 26, 27, 28,  0])

[622] # doc2vec 내장 메소드 most_similar 사용
      print(list(map(lambda x: d2v_list[1].docvecs.most_similar(x)[0][0], range(0,len(papers_tokens[1]))))) #

      ['27', '21', '9', '8', '10', '10', '24', '21', '3', '12', '5', '10', '9', '18', '18', '20', '20', '27',
```

NAR 저널 내 paper에 대한 유사도 분석

- document 간 유사도

- 아래는 NAR 저널의 1497개의 paper에 대해서 most_similar 메소드를 이용하여 문서 간 코사인 유사도가 가장 높은 문서들을 리턴하는 코드이다.

```
] # 저널 내 paper 간 유사도
  print(list(map(lambda x: d2v_list[7].docvecs.most_similar(x)[0][0], range(0,len(papers_tokens[7])))))
```

- 위의 코드의 결과에 따르면 각 문서에 가장 높은 유사도를 보이는 문서는 각각 [' 628 ' , ' 1244 ' , ' 361 ' , ' 1323 ' , ' 1244 ' , ' 1022 ' , ' 97 ' , ' 361', …] 이다. 즉, 0번째 paper의 경우 628번째 paper와 가장 유사했고, 1번째의 경우 1244번째와 가장 유사했다.

- 그러나 실제 원 텍스트를 비교해본 결과 1번째 paper는 protein, homologous recombination repair에 관한 내용이고, 1244번째 paper는 synthetic biology의 genetic device 디자인에 관한 내용으로 큰 내용 간 관련은 없어보였다.

4-3) Keyword Extraction

Keyword extraction

- document 벡터와 단어 벡터 간의 유사도를 계산하여 가장 유사한 단어를 k개 추출하여 document의 keyword extraction을 시도하였다.
- 첫번째 방법으로는 paper vector의 mean을 계산하여, 이를 단어 임베딩 벡터의 평균과 비교하는 방법을 사용하였다.
- 두번째 방법으로는 paper vector 전체에 대해 단어 벡터 전체와의 코사인 유사도를 계산하는 방법을 사용하였다.

Keyword extraction (G&I)

1) mean of document vector vs. mean of word vector

```
doc0_mean = gni_embed_mat[0].mean() # 0번째 paper의 임베딩 mean
words_mean = gni_embed.wv.vectors.mean(axis = 1) # 각 단어 임베딩 벡터의 평균값
abs_error = abs(doc0_mean - words_mean)
```

: 첫번째 paper의 embedding vector의 mean을 구한 후, gni embedding 모델의 wv.vectors에서 word vector를 가져온 다음 역시 mean을 구한다.

두 평균의 차이의 절댓값(absolute error)을 이용해 이 차이가 가장 적은 10개의 단어를 추출함.

```
] np.array(gni_embed.wv.index2word)[abs_error.argsort()[:10]] # words mean ve

array(['ladcs', 'technical', 'starting', 'captured', 'great', 'analogous',
       'eukaryote', 'selection', 'insertion', 'computationally'],
      dtype='<U16')
```

Keyword extraction (G&I)

1) mean of document vector vs. mean of word vector

```
] np.array(gni_embed.wv.index2word)[abs_error.argsort()[:10]] # words mean ve
```

```
array(['ladcs', 'technical', 'starting', 'captured', 'great', 'analogous',
       'eukaryote', 'selection', 'insertion', 'computationally'],
      dtype='<U16')
```

```
print(set(org_doc[1][0][0])) # gni의 첫번째 문서 확인
```

```
{'fragment', 'cleavage', 'disorder', 'review', 'product', 'biological', 'regulatory', 'class', 'trf', 'response', 'rna', 'context', 'cancer',
```

→ 실제 첫번째 문서의 토큰과 비교해본 결과 비슷하다고 판단하기는 어려웠다. 또한 keyword로 추출된 단어들이 다소 일반적인 단어이기에 문서를 대표하는 keyword로 보기엔 다소 어려움이 있다.

Keyword extraction (G&I)

2) Compute cosine similarity between two vectors

: get_most_similar_top_n → 비교 기준이 되는 문서 벡터와 단어 벡터 리스트를 받아 해당 문서와 가장 비슷한 단어를 n개 추출하는 함수 (코사인 유사도 기반)

```
def get_most_similar_top_n(embed_model, embed_vec_ref, embed_vec_list,n=5):
  """
  cosine similarity를 이용해서 가장 비슷한 단어 n개를 리턴하는 함수
  - input
  embed_model: 임베딩 모델
  embed_vec_ref: 임베드 벡터 (비교 대상)
  embed_vec_list: 임베드 벡터의 리스트 (iterable)
  n: 리턴할 단어의 개수 (defaulting to 5)
  - output
  n most similar words (in numpy array)
  """
  similarities = list(map(lambda x: float(cosine_similarity(x.reshape(1,-1), embed_vec_ref.reshape(1,-1))), embed_vec_list))
  #print(similarities)
  top_n = np.argsort(similarities)[::-1][:n]
  return np.array(embed_model.wv.index2word)[top_n]
```

Keyword extraction (G&I)

2) Compute cosine similarity between two vectors

: 첫번째 문서와 가장 비슷한 단어를 10개 추출한 결과 아래와 같았다.

```
get_most_similar_top_n(gni_embed, gni_embed_mat[0], gni_embed.wv.vectors, n = 10)
```

```
array(['layer', 'confirmed', 'captured', 'acinetobactin', 'consisting',
       'predict', 'long', 'mapped', 'purification', 'source'],
      dtype='<U16')
```

```
papers[1][0]
```

```
'tRNA-derived RNA fragments (tRFs) are an emerging class of non-coding RNAs (ncRNAs). A g
rowing number of reports have shown that tRFs are not random degradation products but are
functional ncRNAs made of specific tRNA cleavage. They play regulatory roles in several b
iological contexts such as cancer, innate immunity, stress responses, and neurological di
sorders. In this review, we summarize the biogenesis and functions of tRFs.'
```

그러나 실제 paper와의 비교를 해보았을 때 paper의 내용은 trfs의 cancer, immunity 등에서의 역할을 이야기 하고 있고 추출된 keyword의 내용은 이와 비슷하다고 보기는 힘들다.

Keyword extraction (NAR)

- cosine similarity 기반으로 mean이 아닌 벡터 전체를 비교

```
get_most_similar_top_n(nar_embed, nar_embed_mat[1067], nar_embed.wv.vectors, n = 20)
```

```
array(['strand', 'loss', 'upon', 'end', 'kinase', 'methyltransferase',
       'formation', 'molecule', 'particle', 'mutant', 'thus', 'relative',
       'ability', 'transfer', 'double', 'cause', 'work', 'nucleotide',
       'chromosomal', 'assay'], dtype='<U30')
```

→ 1067번째 문서에 대해 keyword extraction을 진행한 결과, 위와 같은 결과를 보였다. 다만 실제 paper를 확인했을 때 heterogeneity, tumor, deregulation 등이 중요 키워드로 파악되었으나 실제 추출된 키워드는 이와 일치하지 않음을 알 수 있다. 또한 upon ,end, double, cause, thus 등은 비교적 일반적인 의미로 document의 특징을 대변한다고 보기는 어렵다.

4-4) Clustering

Clustering on papers in a journal

- K-means clustering & Hierarchical clustering 을 진행함 .
- 클러스터링 결과를 plotting 할 때에 PCA 와 T-SNE 을 이용해 2차원으로 축소한 축 위에 시각화를 진행함
- G&I 는 2015 년 추출된 paper 개수가 29 개밖에 되지 않았으므로 클러스터링 결과가 좋지 못했다 .
- 반면 NAR 의 경우에는 데이터 개수가 다소 많았던 덕에 클러스터링 결과가 G&I 대비 깔끔했다 .
- 두 저널 모두 각 cluster 별로 mean function 으로 aggregation 하여 , keyword extraction 을 진행하였다 .

Clustering on papers in G&I journal

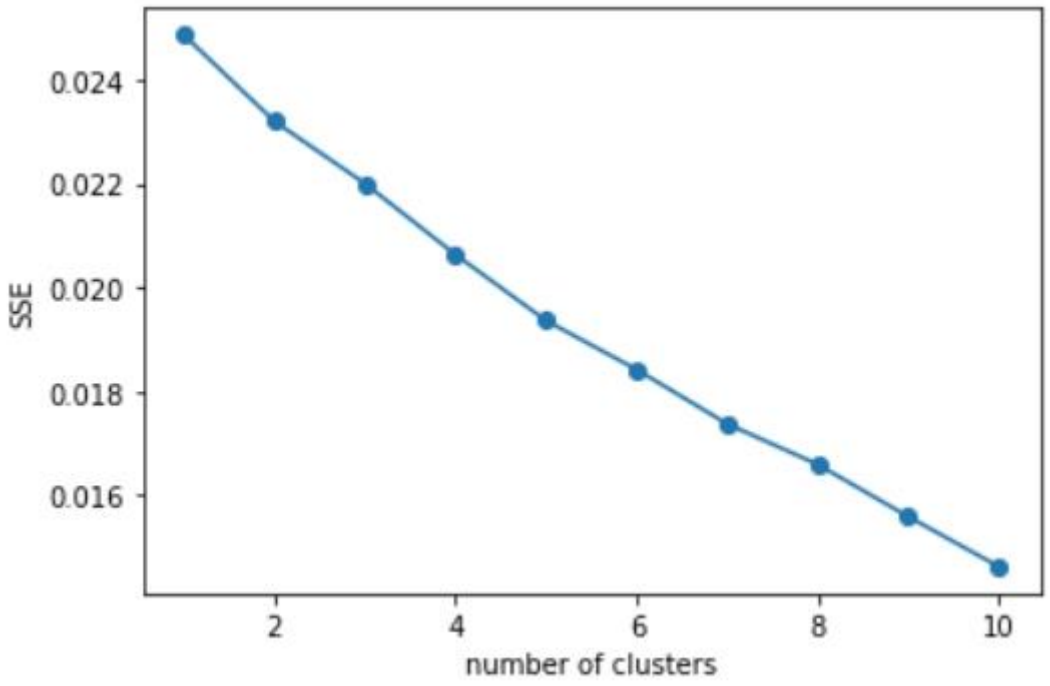
- 1 st try: K-means clustering

- 최적의 k 를 결정하기 위해 scree plot 을 그리고 , elbow point 를 최적의 k 로 선택한다 .
- scree plot 은 클러스터 개수가 증가함에 따라 클러스터 내의 SSE 의 패턴을 나타낸다 .
- 이를 위해 elbow 함수를 정의하였다 . (10 개의 클러스터까지 만들도록 함)

```
def elbow(X):
  """
  input X: 클러스터링할 데이터
  output: scree plot
  """
  sse = []
  for i in range(1,11):
    kmeans = KMeans(n_clusters = i, random_state = 0)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)

  plt.plot(range(1,11), sse, marker = "o")
  plt.xlabel("number of clusters")
  plt.ylabel("SSE")
  plt.show()

elbow(gni_embed_mat)
```

Clustering on papers in G&I journal

- 1 st try: K-means clustering

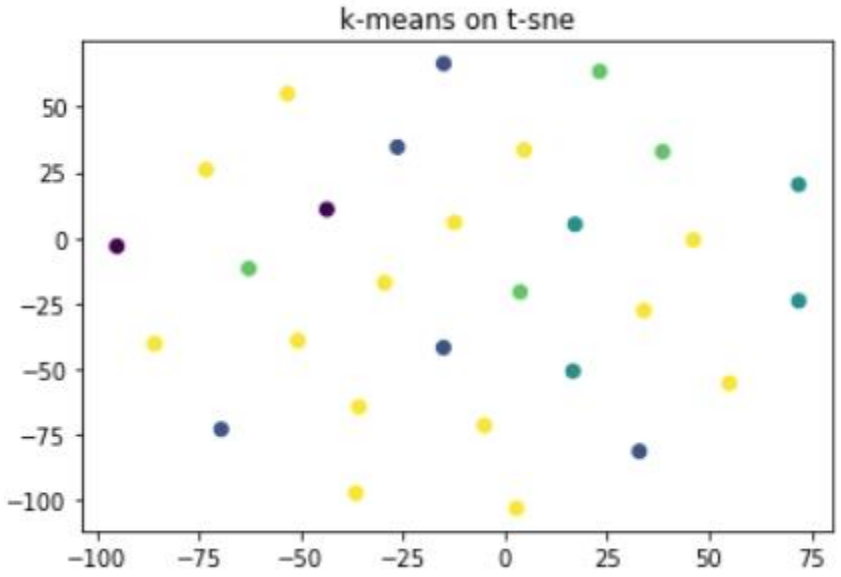


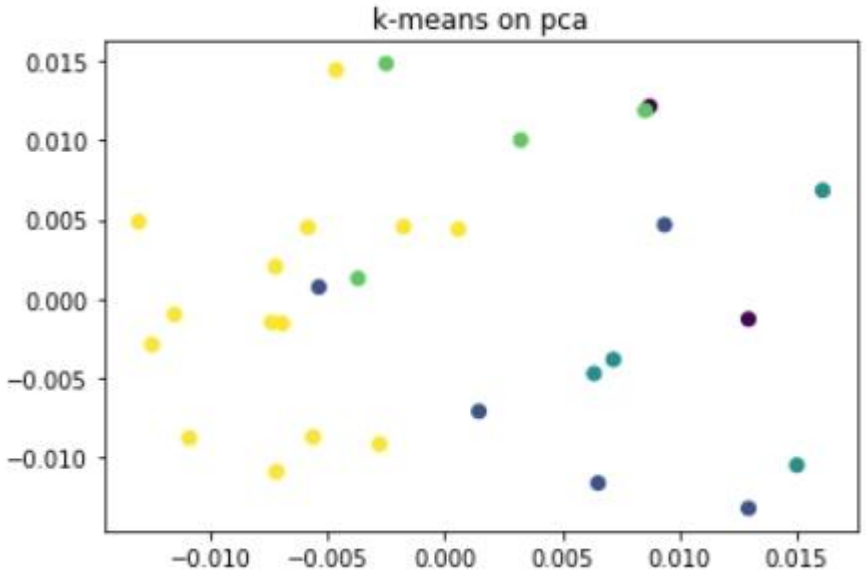
실제 scree plot 을 그린 결과 , scree plot 내에 elbow point 가 관찰되지 않았다 . 즉 G&I 의 경우 scree plot 을 기준으로 최적의 k 를 결정하긴 힘들다 .

따라서 임의로 중간 개수인 5 개 정도로 k 를 결정하였다 .

Clustering on papers in G&I journal

- 1 st try: K-means clustering





좌측의 plot 은 t-sne 을 이용해서 , 우측의 plot 은 pca 를 이용하여 2 차원으로 축소시킨 축에 클러스터 별로 색을 지정하여 paper 를 plotting 하였다 .
두 plot 모두 색깔이 혼재하고 있음으로 보아 clustering 결과가 좋지 못함을 알 수 있다 . 또한 두 축 위의 data 분포를 보았을 때 뚜렷한 클러스터링 현상이 관찰되지도 않았다 .

Clustering on papers in G&I journal

- cf) dimensionality reduction (code)

```
# pca로 차원축소
pca = PCA(n_components = 2)
gni_pca = pca.fit_transform(gni_embed_mat)

# tsne으로 차원축소
tsne = TSNE() # n_components default: 2
gni_reduced = tsne.fit_transform(gni_embed_mat)

## data frame화 해야함.

gni_df = pd.DataFrame({'x': gni_reduced[:,0], 'y': gni_reduced[:,1],'x2': gni_pca[:,0], 'y2': gni_pca[:,1], 'cluster_id' : clusters,
```

PCA 는 scikit learn 의 decomposition, tsne 은 scikit learn 의 manifold 모듈에 내장된 클래스이다 .
fit_transform 메소드를 이용하여 데이터를 2 차원으로 축소한다 .

Clustering on papers in G&I journal

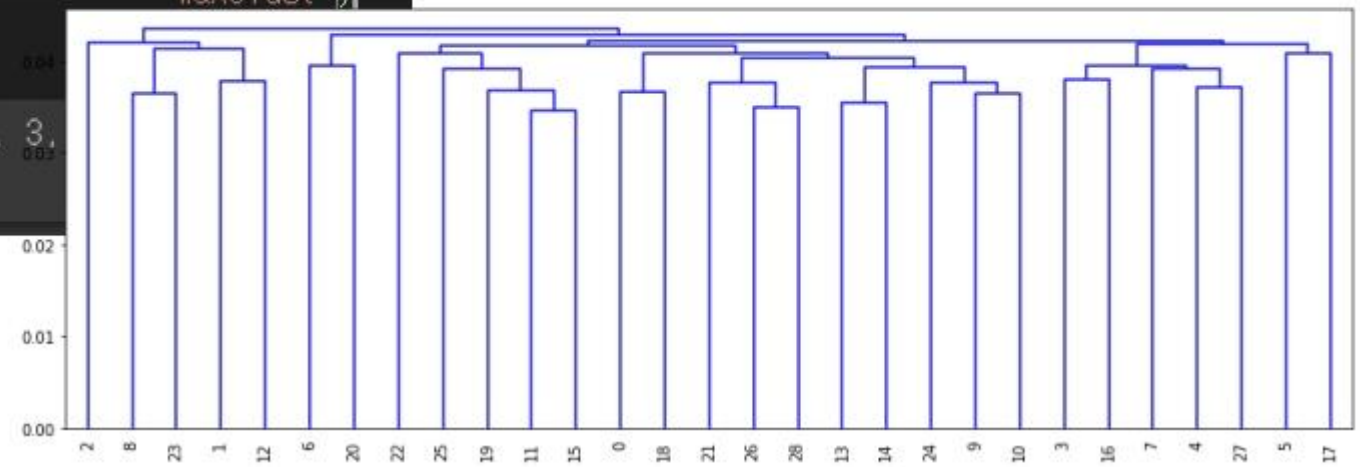
- 2nd try: Hierarchical Clustering (average link)

```
## 계층적 군집화
h_clusters = hierarchy.linkage(y = gni_embed_mat, method = 'average')
```

```
plt.figure(figsize = (15,5))
hierarchy.dendrogram(h_clusters, leaf_rotation = 90)
plt.show()
```

```
array([3, 1, 1, 3, 3, 3, 2, 3, 1, 3, 3, 3, 1, 3, 3, 3,
       3, 1, 3, 3, 3, 3, 3], dtype=int32)
```

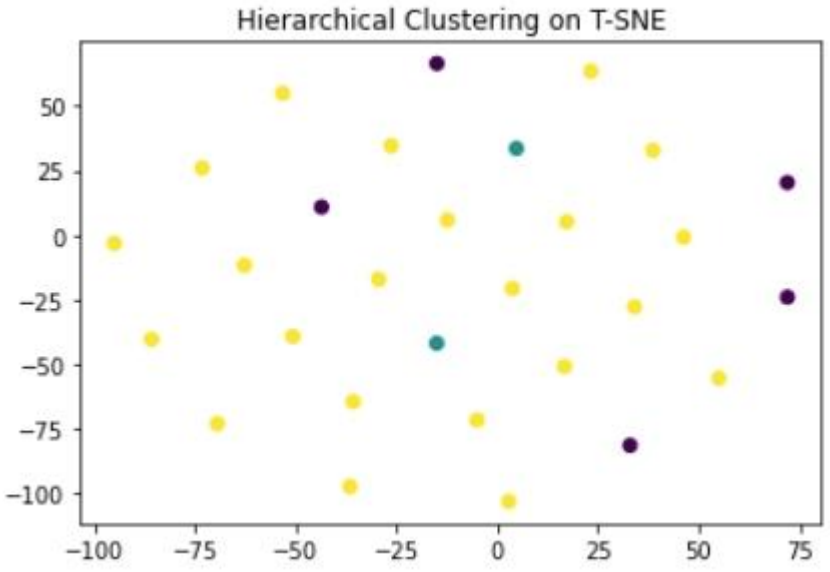
```
maxclust')
```

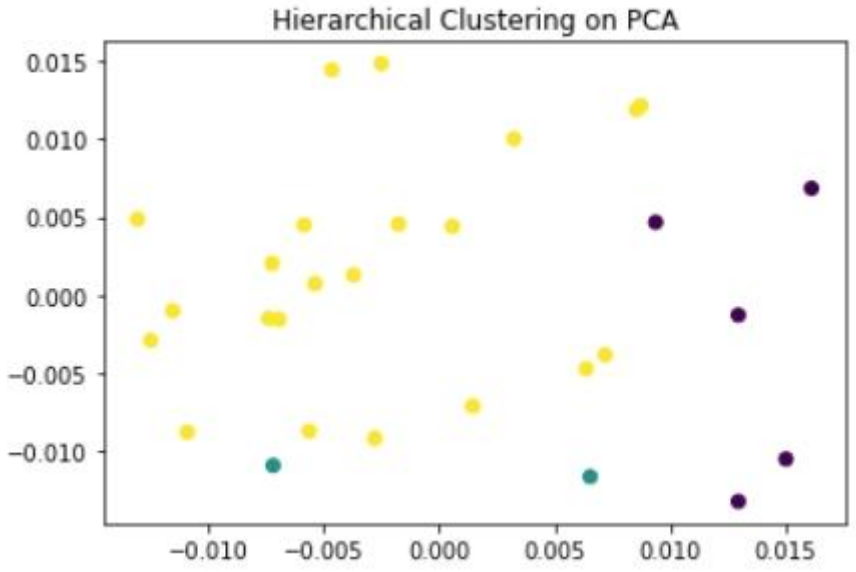


average link 로 설정하여 dendrogram 을 형성한 결과는 위와 같다 . 이 또한 클러스터가 뚜렷하게 나타나지는 않았으나 , 임의로 3 개로 클러스터 개수를 결정하여 tree 를 cut 했다 .

Clustering on papers in G&I journal

- 2nd try: Hierarchical Clustering (average link)





t-sne 과 pca 로 축소한 축에 data point 를 scatter plot 으로 나타낸 결과이다 . t-sne 에서는 역시 클러스터가 뚜렷이 관찰되지 않았지만 , PCA 의 경우에는 k-means 방법보다 조금 더 클러스터 간 경계가 뚜렷했다 .
그러나 여전히 노이즈가 존재함을 알 수 있다 .

Clustering on papers in G&I journal

- Interpretation of Clusters

  각 cluster 별로 groupby 하여 100 차원 임베딩 값을 평균을 취한 후에 get_most_similar_top_n 함수를 이용해 gni 내의 word vector 간의 코사인 유사도를 계산하여 가장 유사한 단어 10 개를 뽑았다 .
  즉 , cluster 별로 10 개의 keyword 를 추출하고자 하였다 .

```
cluster_mean = np.array(embed_df.drop(['cluster_h'], axis = 1).groupby('cluster').mean());
cluster_h_mean = np.array(embed_df.drop(['cluster'], axis = 1).groupby('cluster_h').mean(

# k-means cluster 별 키워드 뽑기
for i in range(len(cluster_mean)):
  print(f"₩nCluster {i}")
  print(get_most_similar_top_n(gni_embed, cluster_mean[i], gni_embed.wv.vectors, 10))
```

Clustering on papers in G&I journal

- Interpretation of Clusters (K-means)

K-means cluster 5 개에 대해 키워드를 추출한 결과는 다음과 같다 .

```
Cluster 0
['heterogeneity' 'copy' 'significance' 'antibiotic' 'argonaute'
 'characterize' 'interactively' 'predict' 'success' 'due']

Cluster 1
['investigation' 'protégé' 'domain' 'examined' 'technical' 'average'
 'holo' 'element' 'transcription' 'pdb']

Cluster 2
['developmental' 'likelihood' 'place' 'location' 'current' 'inspected'
 'robust' 'stored' 'light' 'facilitates']

Cluster 3
['healthy' 'could' 'holo' 'recently' 'rna' 'critical' 'survive' 'gene'
 'expertise' 'decoding']

Cluster 4
['ubiquitous' 'disease' 'long' 'mean' 'specific' 'hpv18' 'reportedly'
 'pkr' 'certain' 'reductase']
```

결과를 살펴보면 , cluster 4 의 경우에는 disease 와 관련된 경우라고 해석할 수 있었지만 다른 cluster 의 경우 holo 와 같이 cluster 간 겹치는 단어도 존재하고 , cluster 의 키워드로 추출된 단어 자체가 일반적인 의미를 가져 cluster 의 특징적인 부분을 설명하기엔 부족한 keyword 들도 있었다 .

Clustering on papers in G&I journal

- Interpretation of Clusters (Hierarchical)

Hierarchical cluster 3 개에 대해 키워드를 추출한 결과는 다음과 같다 .

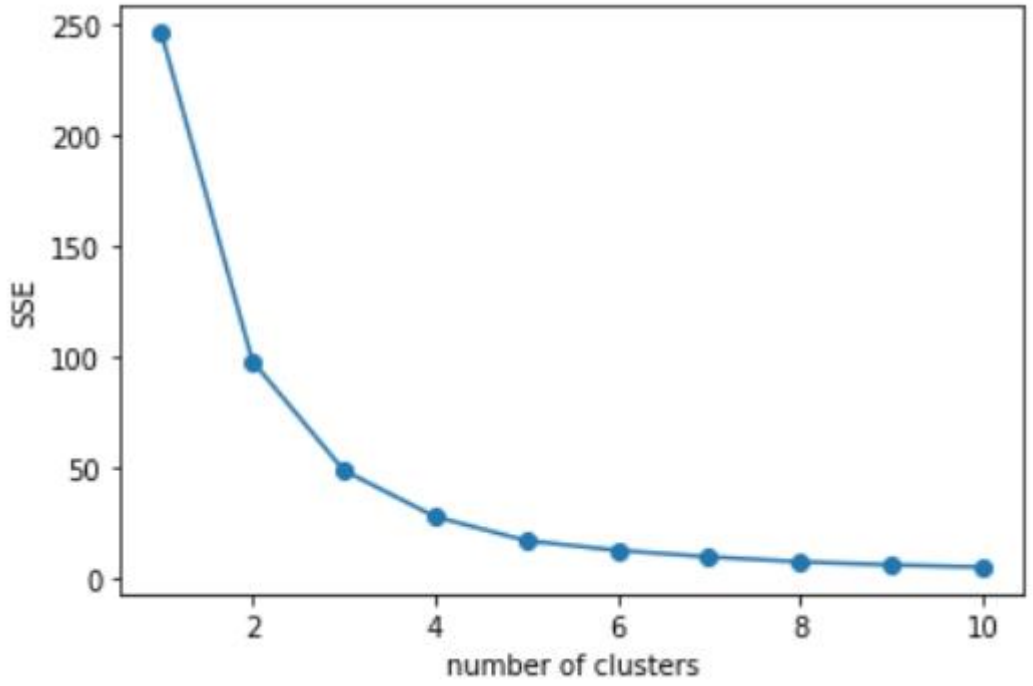
```
Cluster 0
['knowledge' 'need' 'residual' 'success' 'lineage' 'kinase' 'simulation'
 'report' 'area' 'outside']

Cluster 1
['structure' 'measured' 'table' 'including' 'biogenesis' 'researcher'
 'involved' 'arcgis' 'play' 'harvested']

Cluster 2
['disease' 'sample' 'mean' 'ubiquitous' 'hpv18' 'specific' 'landrace'
 'reportedly' 'long' 'metabolic']
```

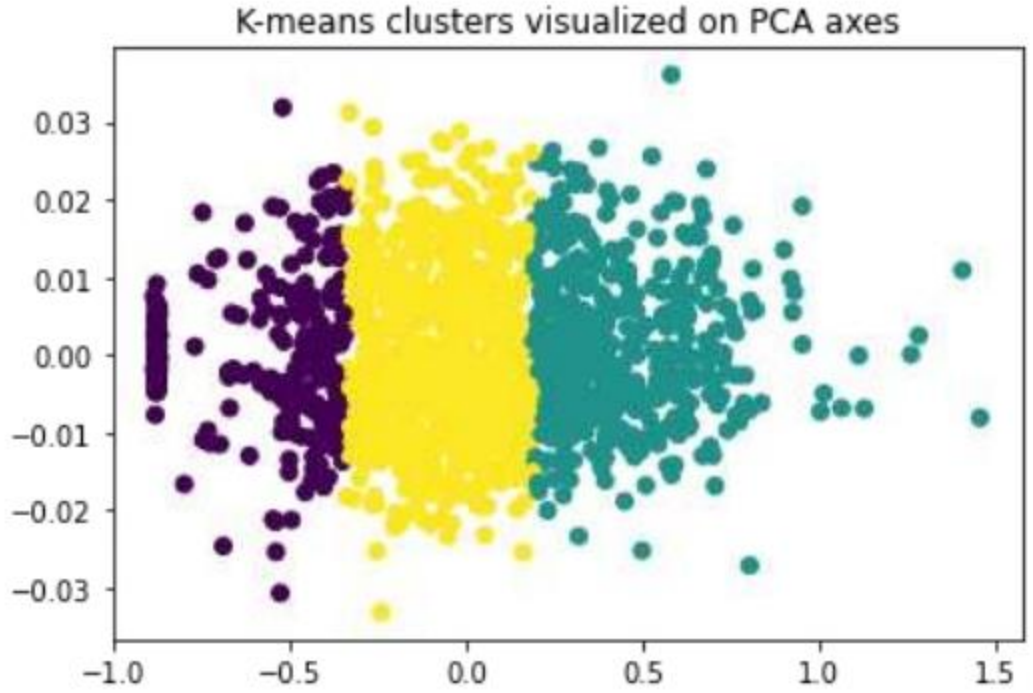
결과를 살펴보면 , cluster 2 의 경우에는 disease, hpv18 등의 키워드가 눈에 띄었다 . (K-means 의 cluster4 와 비슷한 군집으로 파악 된다 .) cluster1 의 경우 success, lineage 등의 비슷한 keyword 들이 존재하고 있음을 알 수 있었다 . cluster1 의 경우에는 다소 일반적인 의미의 단어들이 혼재해 있었다 .

Clustering on papers in NAR journal

- K-means Clustering

Scree plot 을 그려본 결과 최적의 클러스터 개수는 3 으로 결정할 수 있었다 .
또한 PCA 축에 클러스터 별로 색을 다르게 지정하여 scatter plot 을 그려본 결과 , 클러스터 경계가 뚜렷함을 알 수 있었다 .





Clustering on papers in NAR journal

- Interpretation of Clusters

```
cluster_mean = np.array(nar_embed_df.groupby('cluster_id').mean())

# k-means cluster 별 키워드 뽑기
for i in range(len(cluster_mean)):
  print(f"₩nCluster {i}")
  print(get_most_similar_top_n(nar_embed, cluster_mean[i], nar_emb
```

```
Cluster 0
['mutant' 'well' 'furthermore' 'group' 'one']

Cluster 1
['one' 'mutant' 'addition' 'furthermore' 'well']

Cluster 2
['well' 'including' 'class' 'virus' 'demonstrated']
```

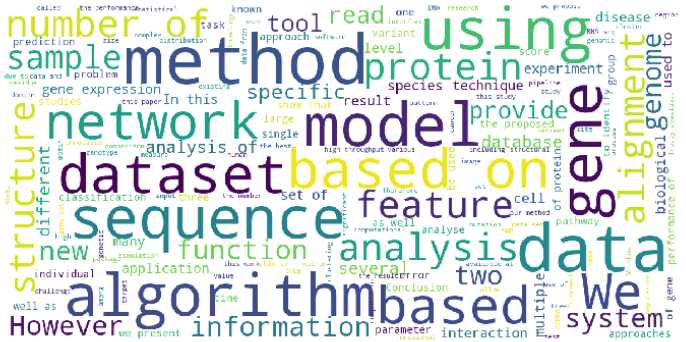
5 개씩 cluster 키워드를 추출해본 결과 ,
클러스터 간 특징이 뚜렷이 드러나진 않았다 .
cluster0 과 cluster1 의 경우 거의 비슷한 키워드 벡터가 리턴되었고 , well 등의 키워드는 모든 클러스터에 등장했다 .
다소 일반적인 단어들이 키워드를 이루고 있는 양상이다 .

5. 저널 별 분석

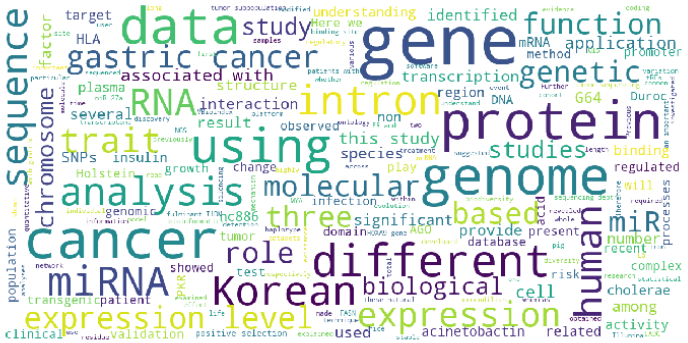
12개의 journal에 대해서 유사도 분석, clustering을 진행함으로써,
GNI와 유사한 journal을 찾아보도록한다.

5-1)
Journal 별 단어분포

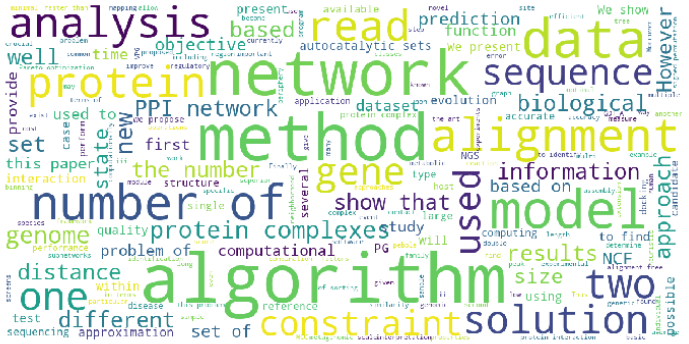
Journal 별 단어분포



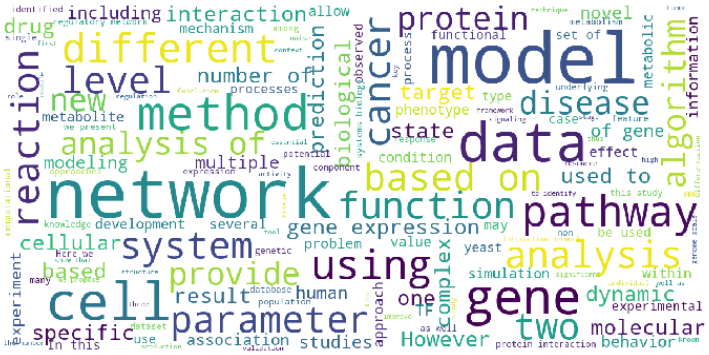
BMC Bioinformatics



Genomics & Informatics

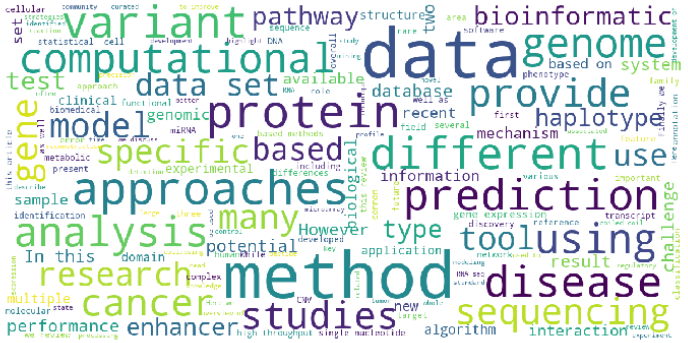


Algorithms for Molecular Biology : AMB



BMC Systems Biology

journal of Computational Biology

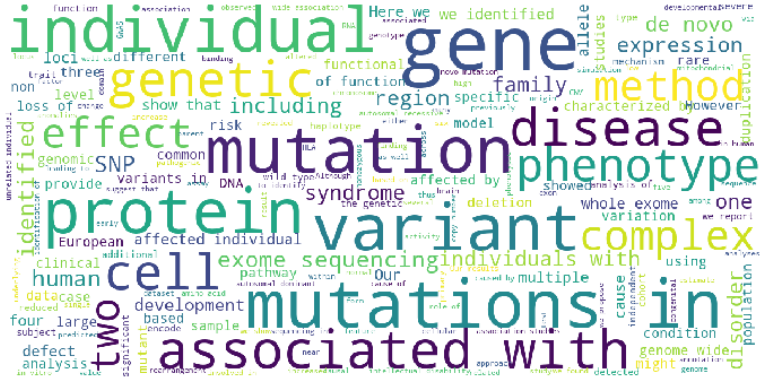


Briefings in Bioinformatics

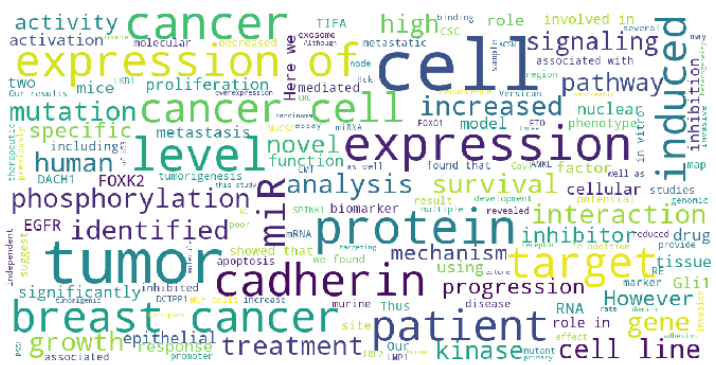
Journal 별 단어분포

BMC Genomics

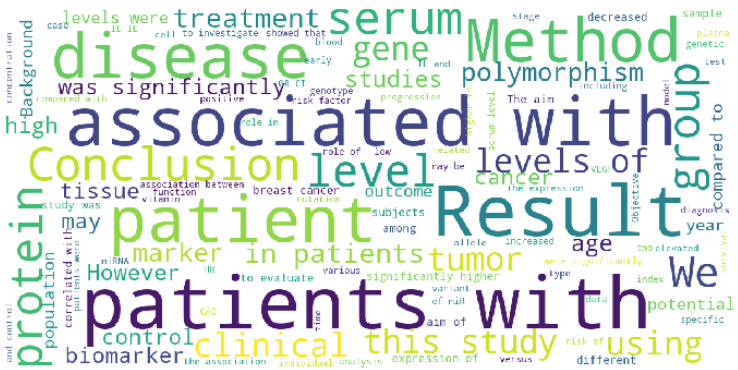
Nucleic Acids Research



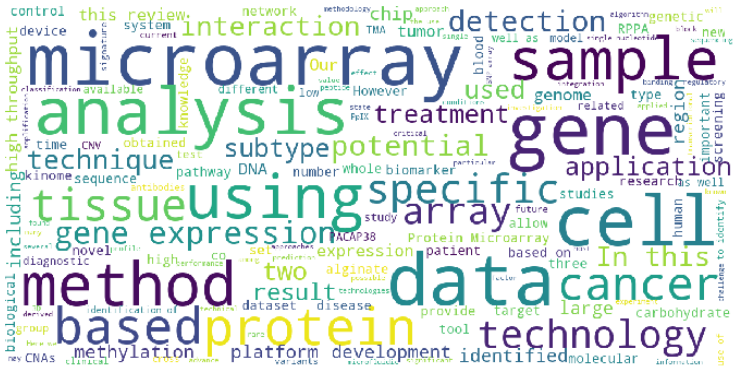
American Journal of Human Genetics



Oncogenesis



Disease Markers



Microarrays

5-2)
Word Embedding &
Document Embedding

Journal 별 임베딩

```
# journal 별 임베딩

tagged_doc = [TaggedDocument(journal, [str(idx)]) for idx, journal in enumerate(journal_tokens)]
journal_d2v = train_doc2vec(tagged_doc)
```

- 4-1에서와 마찬가지로 doc2vec을 사용하여 저널 별 paper를 임베딩한다.
각 journal을 담고 있는 리스트 -> 각 저널 구조로 Tagging을 해주어야 한다.
- tagged_doc 각각 요소는 저널 별로 tagging이 되어 있는 상태이다.
- journal_d2v는 tagged _doc의 각 요소를 embedding한 결과이다.

5-3) Similarity

Doc2Vec Similiarty

마찬가지로 most_similar 함수를 사용하여 GNI와 유사한 저널을 찾아본 결과이다.

```
# gni와 유사한 Journal 찾기
similar_doc = model.docvecs.most_similar('1')
print(similar_doc) # gni 와 유사한 저널 확인
```

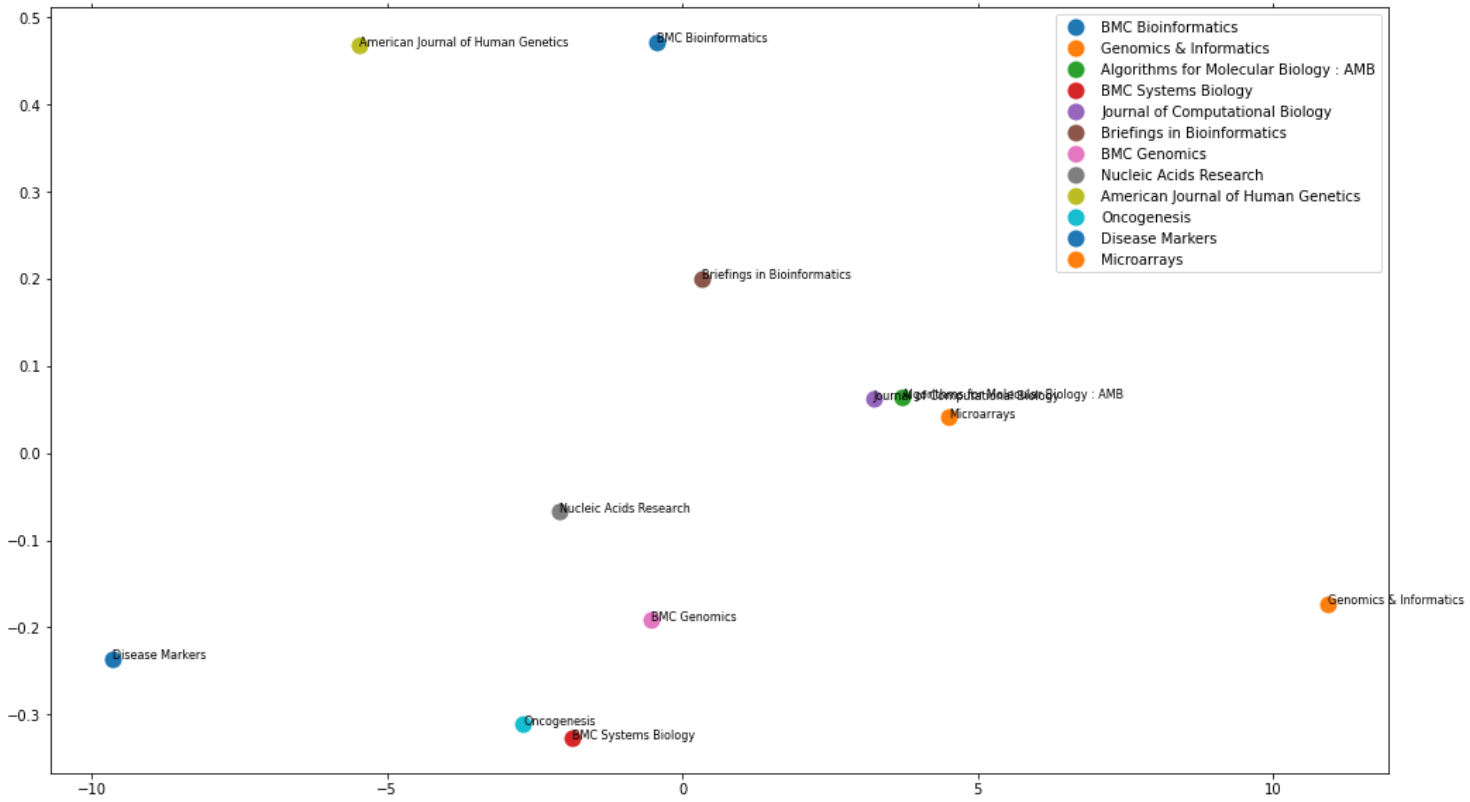
```
[('9', 0.5455480217933655), ('10', 0.5295531153678894), ('7', 0.5223298072814941), ('0', 0.5184515118598938), ('4', 0.5100185871124268), ('6', 0.5010871887207031),
```

11개의 저널은 GNI와 대체로 유사도가 높았으나, 그 중에서도 Disease Markers, Briefings in Bioinformatics, BMC Genomics, Nucleic Acids Research, American Joural of Human Genetics 순으로 유사도가 높았다. 이는 앞의 계층적 군집화 결과, tf-idf clustering 결과와 비슷하다.

5-4) clustering – part1

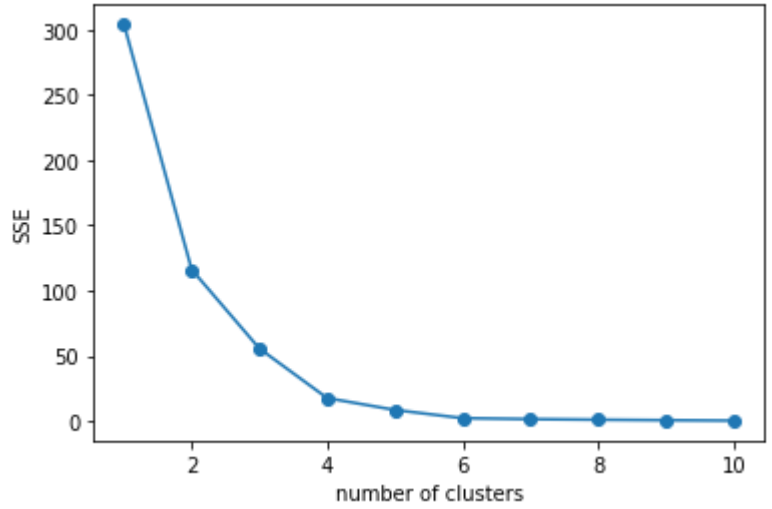
journal 별 클러스터링

```
# pca를 이용해서 2차원으로 축소한 데이터 공간에 각 document를 plotting

pca = PCA(n_components=2)
journal_pca = pca.fit_transform(journal_embed_mat)
```



-> PCA를 이용해서 2차원으로 축소한 이후, 저널 별로 Plotting 한 결과이다.
몇몇 document의 거리의 경우 매우 가까움을 알 수 있다.
clustering을 했을 때 유의미한 결과가 나올지 clustering을 해 보도록 한다.

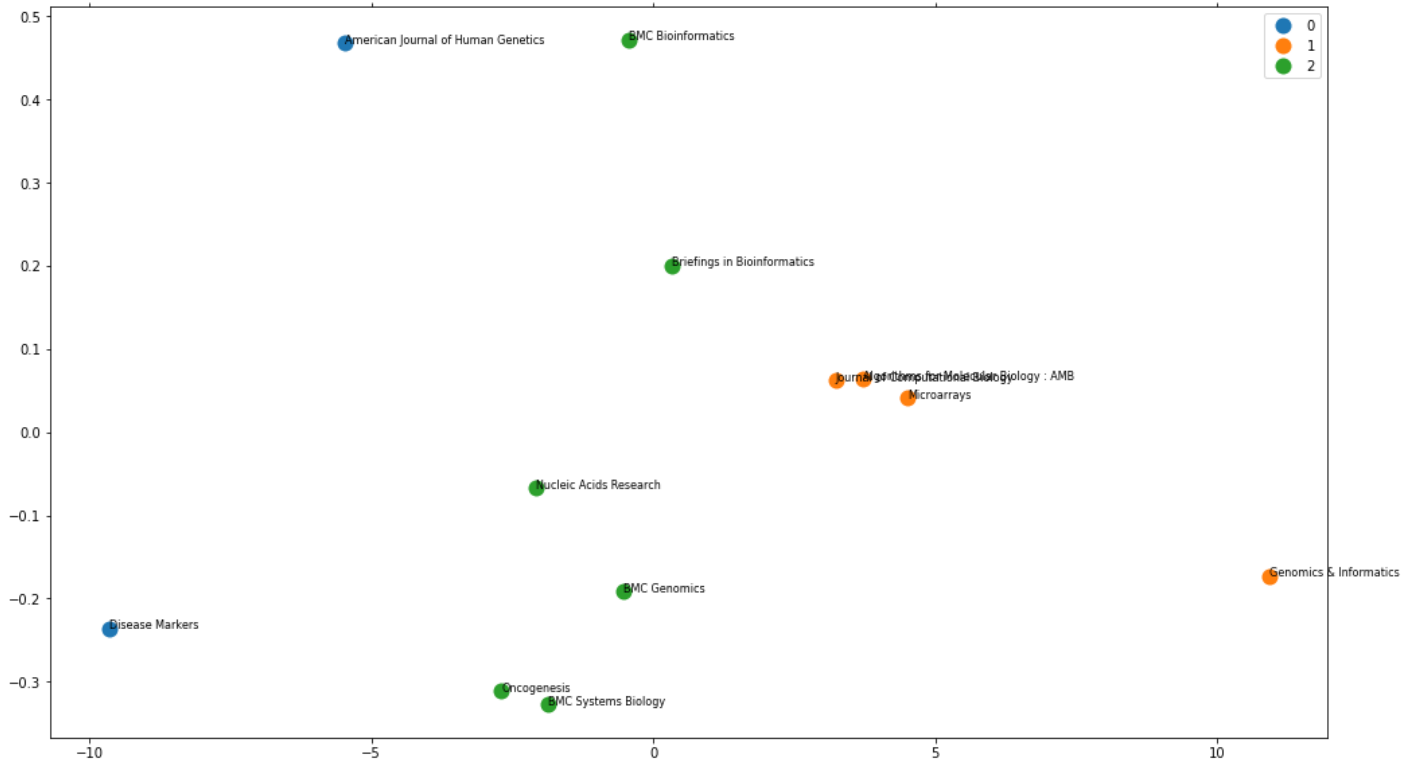
journal 별 클러스터링



elbow point를 그려본 결과이다.
-> 최적의 클러스터 개수는 역시 2~3개 정도로 결정 가능하다.
여기서는 3개로 결정하도록 한다.

```
# fitting k-means
kmeans = KMeans(n_clusters = 3)
cluster_k = kmeans.fit_predict(journal_embed_mat)
```

journal 별 클러스터링



-> doc2vec을 이용하여 Kmeans clustering 한 결과이다.

GNI는 Microarrays, AMB, Journal of Computational of Biology와 같은 클러스터로 구분되었다.

5-5) clustering - part2

앞서 doc2vec을 이용하여 clustering을 수행하였는데 본 Part 에서는 tf-idf를 이용하여 clustering을 해보도록한다.

TF-idf K-means Clustering

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=200000,
                                   min_df=0.2, stop_words='english',
                                   use_idf=True, tokenizer=tokenize_and_stem, ngram_range=(1,3))

%time tfidf_matrix = tfidf_vectorizer.fit_transform(pubmed)

print(tfidf_matrix.shape)
```

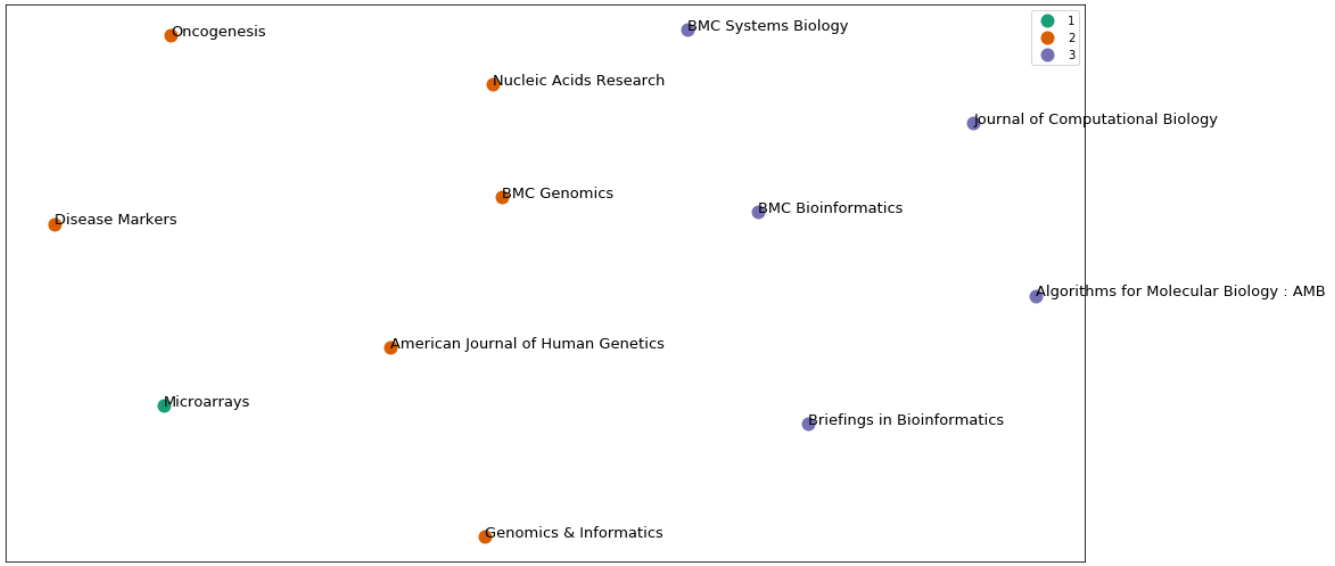
```
from sklearn.cluster import KMeans

num_clusters = 3

km = KMeans(n_clusters=num_clusters,random_state = 0)

%time km.fit(tfidf_matrix)

clusters = km.labels_.tolist()
```

TF-idf matrix를 정의하고 Kmeans 클러스터링을 수행한다.

TF-idf K-means Clustering

클러스터링 결과는 다음과 같다.

	journal	cluster
2	BMC Bioinformatics	2
1	Genomics & Informatics	1
2	Algorithms for Molecular Biology : AMB	2
2	BMC Systems Biology	2
2	Journal of Computational Biology	2
2	Briefings in Bioinformatics	2
1	BMC Genomics	1
1	Nucleic Acids Research	1
1	American Journal of Human Genetics	1
1	Oncogenesis	1
1	Disease Markers	1
0	Microarrays	0



GNI는 BMC Genomics, Nucleic Acids Research, American Jornal of Human Genetics, Diseases Markers와 같은 무리로 분류되었다.

TF-idf K-means Clustering

각 클러스터별로 주요 용어를 분석해본 결과, 다음과 같았다.

```
print("Top terms per cluster:")
print()
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
for i in range(num_clusters):
    print("Cluster %d words:" % i, end='')
    for ind in order_centroids[i, :6]:
        print(' %s' % vocab_frame.loc[terms[ind].split(' ')].values.tolist()[0][0].encode('utf-8', 'ignore'), end=',')
    print()
    print()
    print("Cluster %d journal:" % i, end='')
    for title in frame.loc[i]['journal'].values.tolist():
        print(' %s,' % title, end='')
    print()
    print()
```

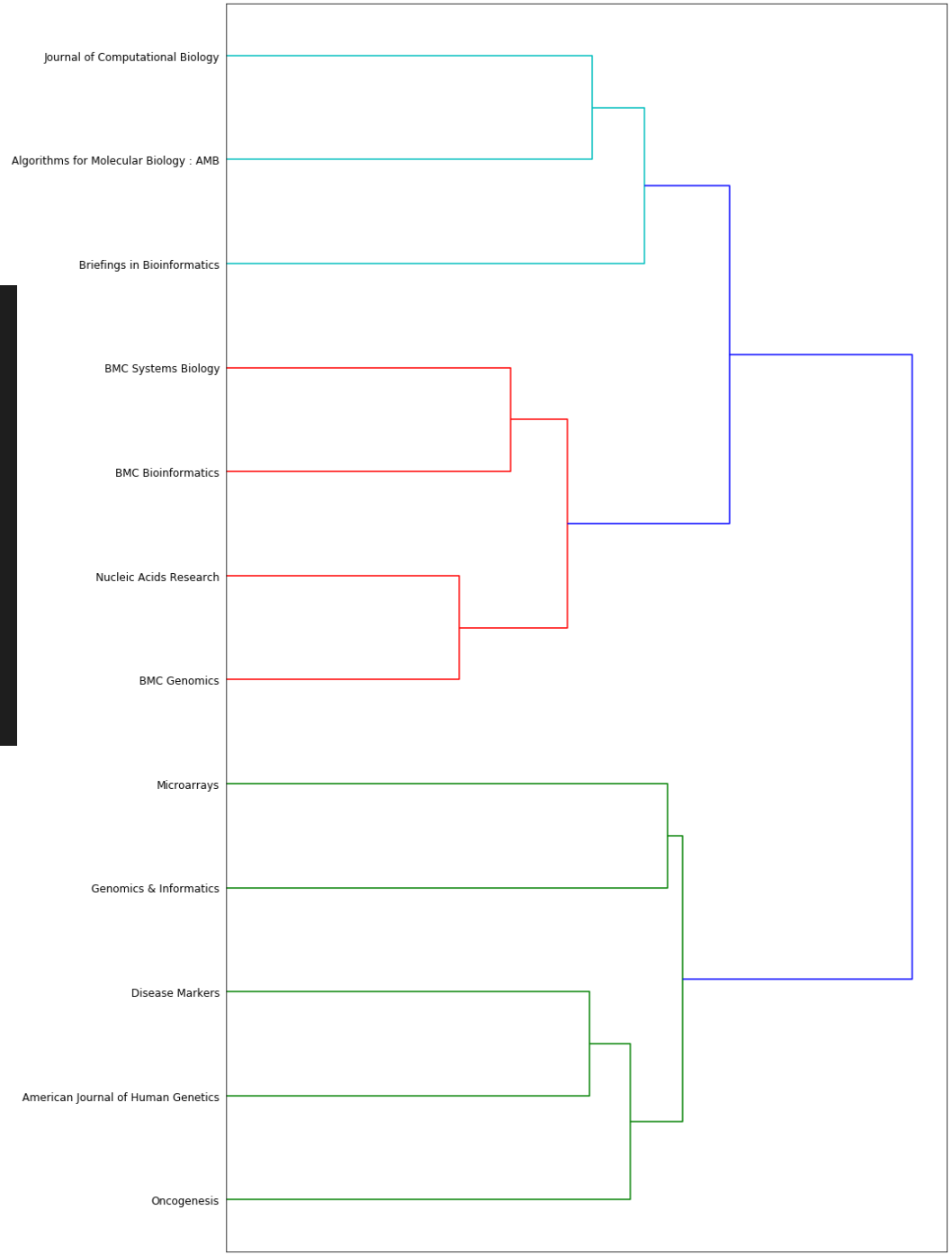
Cluster 0 : Microarrays
words: 'subtype', 'alginate', 'snp', 'device', 'microarray', 'osteoblastic'

Cluster 1 : Genomics & Informatics, BMC Genomics, Nucleic Acids Research, American Journal of Human Genetics, Oncogenesis, Disease Markers
words: 'mirnas', 'trait', 'mrna', 'serum', 'resistant'

Cluster 2 : BMC Bioinformatics, Algorithms for Molecular Biology : AMB, BMC Systems Biology, Journal of Computational Biology, Briefings in Bioinformatics,
words: 'alignment', 'problem', 'trees', 'inferred', 'data', 'task'

저널별 계층적 군집화

```
from scipy.cluster.hierarchy import ward, dendrogram

linkage_matrix = ward(dist) #define the linkage_matrix using ward clustering pre-computed distances

fig, ax = plt.subplots(figsize=(15, 20)) # set size
ax = dendrogram(linkage_matrix,orientation="right", labels=journalList);

plt.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')

plt.tight_layout() #show plot with tight layout
```



앞서 cosine 유사도로 구한 저널별 distance 를 사용하여 linkage_matri를 작성하고, 저널별 계층적 군집화를 해본 결과이다.
gni와 유사한 journal 로는 Oncogenesis, American journal of Human Genetics, Disease Markers 등이 있다.

6. Journal 분류 모델

Journal 을 label 로 하여 지도학습을 통해 ,
paper 를 인풋으로 하여
Journal 을 분류해내는 LSTM 모델을 구축함 .

data setting

• 분류 모델에 들어갈 data 를 setting

1) X

: 정수 인코딩된 papers. (정수 인코딩 : 토큰 대신 토큰의 인덱스가 들어간 sequence)

정수 인코딩한 시퀀스가 들어가야 하므로 , 정수 인코딩을 보다 용이하게 진행하기 위해 keras 의 내장 tokenizer 로 토크나이징을 진행함 .

```
# 정수 인코딩
keras_tokenizer = text.Tokenizer()
keras_tokenizer.fit_on_texts(X) # keras tokenizer를 이용하여 tokenize
sequences = keras_tokenizer.texts_to_sequences(X)
X = sequences # update X
```

2) y

: 각 paper 의 저널을 0 부터 12 까지의 정수로 encoding 한 값

단어 빈도 수 및 문장 분포 파악

- 몇 개의 단어를 input data 에 집어넣을지
(빈도수가 너무 적은 것은 생략해도 되므로),
- X 의 padding 을 길이 몇을 기준으로 진행할지 ,
에 대해 파악하기 위해 단어 빈도 수 및 문장 분포를 파악하고자 한다 .

단어 빈도 수 및 문장 분포 파악

- 단어 빈도 수 파악

```
## 단어 빈도 수에 대한 파악

threshold = 2
total_count = len(word_to_idx) # 총 단어 개수
rare_count = 0 # 빈도수가 threshold보다 작은 단어 개수
total_freq = 0 # 데이터 내에 전체 단어 빈도수 총합

for token, freq in keras_tokenizer.word_counts.items(): # word_counts.items(): (wo
  total_freq += freq

  if (freq < threshold): # 빈도수가 1
    rare_count += 1

print(f"num of rare({threshold-1} times) words in data: {rare_count}")
print(f"proportion of rare words in data: {rare_count/total_count*100}")
print(f"proprotion of rare freq from total freq: {rare_count / total_freq*100}")
```

```
num of rare(1 times) words in data: 14553
proportion of rare words in data: 40.15839289163608
proprotion of rare freq from total freq: 1.56959136482079O6
```

한 번만 등장하는 단어의 비율은 전체 단어 대비 40% 정도로 거의 반을 차지했다 . 그러나 , 전체 등장 빈도 수 대비로 했을 때의 비율은 약 1.6% 로 매우 낮았다 .

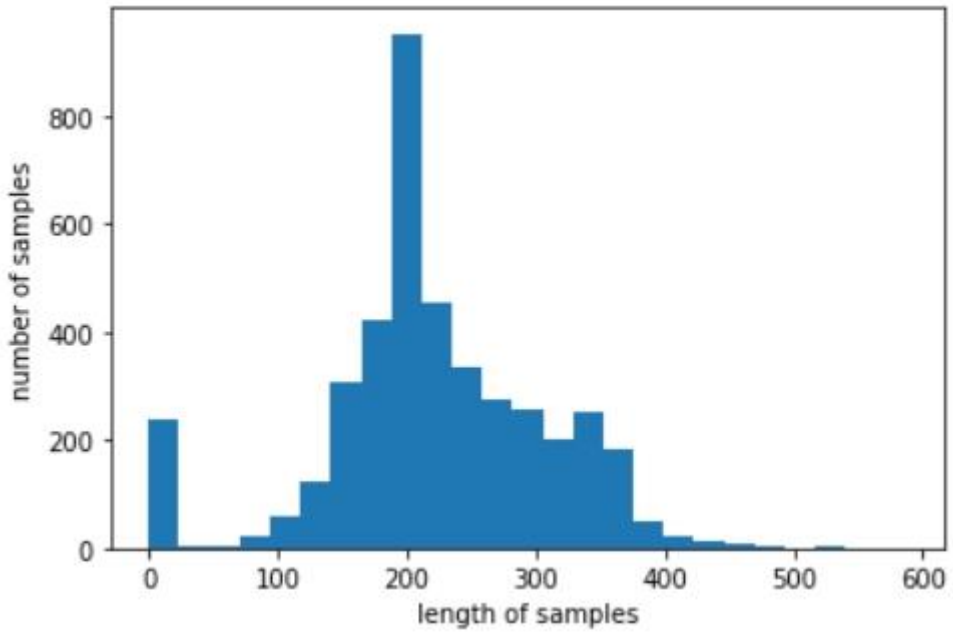
→ 다만 빈도수가 적은 용어들의 경우 특수 용어일 경우도 있으므로 모든 단어를 모델에 인풋으로 집어 넣기로 한다 .

단어 빈도 수 및 문장 분포 파악

- Document 길이 분포 파악

```
# Sequence 분포 확인
print("max length of document: {}".format(max(len(l) for l in X)))
print("average length of document: {}".format(sum(map(len, X))/len(X)))
```

```
max length of document: 587
average length of document: 221.4964166268514
```



Sequence, 즉 하나의 document (paper) 의 길이 분포를 보았을 때 ,
최대 길이는 587 이었고 ,
average length 는 221 이었다 .
길이에 대해 히스토그램을 그렸을 때
200 정도에 peak 을 갖고 있는 왼쪽으로 skewed 된 분포를 볼 수 있었다 .
또한 , 0 의 빈도가 꽤 높은 것으로 보아 결측치가 존재했음을 알 수 있다 .

data setting

- X padding & Y dummy 화

```
max_len = max(len(l) for l in X)

X_padded = sequence.pad_sequences(X, maxlen = max_len)
print("shape of X: ", X_padded.shape)
y = pd.get_dummies(y)
print("shape of y: ", y.shape)
```

```
shape of X:  (4186, 587)
shape of y:  (4186, 12)
```

Sequence 의 최대 길이에 맞춰 X를 패딩하고 , Y 는 12 개의 journal 즉 12 개의 label 이 있으므로 12 개의 binary 변수 칼럼을 얻도록 dummy 화 시킨다 .
(loss function 적절하게 사용하기 위해서는 dummy 화가 필수이다 .)

data setting

- train test split

```
# train test split (split 비율은 8:2)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_padded, y, test_size = .2,
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(3348, 587) (3348, 12)
(838, 587) (838, 12)
```

train set 과 test set 을 분리하여 , 학습에서 overfitting 이 일어나지는 않았는지 등에 대한 모델 평가를 추후에 진행하기 용이하게 함 . split 시 random_state = 0 으로 고정한다 . (항상 같은 dataset 으로 분리되도록)

Building Classifier

```
embedding_size = 100

model = Sequential([
                    Embedding(input_dim = vocab_size,
                              output_dim = embedding_size,
                              input_length = max_len),
                    Conv1D(filters = 50, kernel_size = 5, padding = 'same', activation = 'relu'),
                    MaxPooling1D(pool_size=2),
                    LSTM(50),
                    Dropout(0.25),
                    Dense(25, activation = 'sigmoid'),
                    Dense(12, activation = 'softmax')
])
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

① Embedding layer 를 구축한다 . (doc2vec 에서 구축한 pre trained embedding 을 사용할 수 있었지만 , 모든 단어가 다 build 되지 않으면서 invalid argument error 를 계속 raise 하여 embedding layer 에서 embedding 을 진행함)

Building Classifier

```
embedding_size = 100

model = Sequential([
                    Embedding(input_dim = vocab_size,
                              output_dim = embedding_size,
                              input_length = max_len),
                    Conv1D(filters = 50, kernel_size = 5, padding = 'same', activation = 'relu'),
                    MaxPooling1D(pool_size=2),
                    LSTM(50),
                    Dropout(0.25),
                    Dense(25, activation = 'sigmoid'),
                    Dense(12, activation = 'softmax')
])
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

① Embedding layer: input_dim 은 패딩 토큰을 포함해서 corpus 내 토큰 개수의 +1 로 넣어 준다 .
output 은 dim 은 임베딩 차원으로 doc2vec 과 같은 100 차원으로 지정해주었다 .
input_length 는 input sequence 의 길이로 아까 지정한 max_len 으로 넣어주었다 .

Building Classifier

```
embedding_size = 100

model = Sequential([
                    Embedding(input_dim = vocab_size,
                              output_dim = embedding_size,
                              input_length = max_len),
                    Conv1D(filters = 50, kernel_size = 5, padding = 'same', activation = 'relu'),
                    MaxPooling1D(pool_size=2),
                    LSTM(50),
                    Dropout(0.25),
                    Dense(25, activation = 'sigmoid'),
                    Dense(12, activation = 'softmax')
])
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

② Conv1D: 1-dimensional Convolution 을 통해서 구조적 특징을 추출함 . Convolution 연산을 진행하면서 텍스트 내에서 국지적인 특징을 좀 더 잘 잡아낼 수 있을 것이라는 기대를 해볼 수 있다 . 또한 각 sequence 는 1-dimensional 이므로 Conv1D layer 를 넣어주었다 .

Building Classifier

```
embedding_size = 100

model = Sequential([
                    Embedding(input_dim = vocab_size,
                              output_dim = embedding_size,
                              input_length = max_len),
                    Conv1D(filters = 50, kernel_size = 5, padding = 'same', activation = 'relu'),
                    MaxPooling1D(pool_size=2),
                    LSTM(50),
                    Dropout(0.25),
                    Dense(25, activation = 'sigmoid'),
                    Dense(12, activation = 'softmax')
])
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

② Conv1D: filter 개수를 50 개로 지정하였고 , 각 filter 의 사이즈는 5 로 지정하였다 . activation 함수는 활성화함수로 많이 쓰이는 relu 로 지정하였다 . padding 은 same 으로 지정함으로써 인풋과 아웃풋의 크기가 동일하게 패딩하도록 하였다 .

Building Classifier

```
embedding_size = 100

model = Sequential([
                    Embedding(input_dim = vocab_size,
                              output_dim = embedding_size,
                              input_length = max_len),
                    Conv1D(filters = 50, kernel_size = 5, padding = 'same', activation = 'relu'),
                    MaxPooling1D(pool_size=2),
                    LSTM(50),
                    Dropout(0.25),
                    Dense(25, activation = 'sigmoid'),
                    Dense(12, activation = 'softmax')
])
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

③ MaxPooling1D: 역시 Conv1D 와 동일하게 sequence 가 1D array 이므로 1D max pooling 을 진행하였다 . 이때 , pool_size 는 2 로 정의하여 주요 특징을 추출하였다 .

Building Classifier

```
embedding_size = 100

model = Sequential([
                    Embedding(input_dim = vocab_size,
                              output_dim = embedding_size,
                              input_length = max_len),
                    Conv1D(filters = 50, kernel_size = 5, padding = 'same', activation = 'relu'),
                    MaxPooling1D(pool_size=2),
                    LSTM(50),
                    Dropout(0.25),
                    Dense(25, activation = 'sigmoid'),
                    Dense(12, activation = 'softmax')
])
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

④ LSTM : 은닉 개수를 50 개로 설정하고 이외의 하이퍼 파라미터는 따로 설정하지 않고 디폴트 값을 사용하였다 . LSTM 을 통해 long-term dependency 문제를 방지하고 조금 더 이전의 단계의 정보를 잘 기억하기 위해 기존의 vanilla-RNN 대신 LSTM 레이어를 설정하였다 .

Building Classifier

```
embedding_size = 100

model = Sequential([
                    Embedding(input_dim = vocab_size,
                              output_dim = embedding_size,
                              input_length = max_len),
                    Conv1D(filters = 50, kernel_size = 5, padding = 'same', activation = 'relu'),
                    MaxPooling1D(pool_size=2),
                    LSTM(50),
                    Dropout(0.25),
                    Dense(25, activation = 'sigmoid'),
                    Dense(12, activation = 'softmax')
])
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

⑤ Dropout: overfitting 이 심하게 일어나 Dropout layer 를 이용하여 (drop_rate = 0.25) overfitting 을 방지하고자 하였다 .

Building Classifier

```
embedding_size = 100

model = Sequential([
                    Embedding(input_dim = vocab_size,
                              output_dim = embedding_size,
                              input_length = max_len),
                    Conv1D(filters = 50, kernel_size = 5, padding = 'same', activation = 'relu'),
                    MaxPooling1D(pool_size=2),
                    LSTM(50),
                    Dropout(0.25),
                    Dense(25, activation = 'sigmoid'),
                    Dense(12, activation = 'softmax')
])
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

⑥ Dense: 마지막에 fully connected layer 를 2 번 통과시켰다 . 처음에는 sigmoid 활성화함수를 통해 25 개의 아웃풋 노드가 형성되도록 dense layer 를 깔았고 , 마지막엔 다중 분류 태스크를 위해 softmax 활성화 함수를 사용한 12 개의 아웃풋 노드를 가진 dense layer 를 설정했다 . systematic 하게 아키텍처를 설계하진 않았고 하이퍼 파라미터 , 네트워크 구조 등은 임의로 설계했다 .

Model Fitting and Evaluation

: 배치 사이즈는 64, training iteration 수는 30 으로 하여 fitting

30 에폭에서 훈련 정확도가 93 프로이지만 , 테스트 어큐러시는 61% 로 매우 낮았다 .

```
print("Training...")
model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 30, batch_size = 64)
```

```
Epoch 30/30
3348/3348 [==============================] - 26s 8ms/step - loss: 0.2326 - accuracy: 0.9325 - val_loss: 1.7538 - val_accuracy: 0.6158
<keras.callbacks.callbacks.History at 0x7fe2c8812438>
```

```
model_accuracy = model.evaluate(X_test, y_test)[1]
print("Test Accuracy: {}%".format(model_accuracy*100))
```

```
838/838 [==============================] - 2s 2ms/step
Test Accuracy: 61.575180292129528%
```

결론

1) **국적별 분포** : 대부분의 저널에서 USA, China, Germany, UK 등이 많았으나 특히 Genomics & Informatics 에서는 한국의 논문수가 압도적으로 많았다 .

2) **학과별 분포** : 컴퓨터과학 , 통계학 , 수학과의 단어분포가 비슷하였으며 , 생명과학 , 생물학과의 단어분포가 비슷했다 .

결론

3) **저널 내 분석 :** doc2vec 을 이용해서 paper 간 유사도 , paper keyword extraction, clustering and cluster interpretation 등을 진행했으나 , embedding 결과가 좋지 못한 탓인지 원인은 정확히 파악하지 못했으나 embedding 값이 paper 간 유사도 , paper keyword 등을 제대로 반영하지 못했다 .

4) **저널 별 분석 :** 전처리를 통해 prettified 된 token 을 바탕으로 저널 별 분석을 시도했으나 유사도가 1 에 수렴하는 등 적절치 못한 결과가 나왔다 . 따라서 word_tokenize 만을 적용한 corpus 를 바탕으로 분석을 진행한 결과 gni 와 유사한 journal 을 확인할 수 있었다 .

결론

5) **분류 모델 평가 :** train accuracy 는 90% 정도로 높았으나 , test accuracy 는 60% 정도로 한참을 못 미쳤다 . 즉 overfitting 이 발생했음을 알 수 있다 . 원인에 대해 분석해보면 이는 저널 별로 데이터의 수가 매우 적은 저널이 존재하기에 데이터의 수가 부족했음에 기인했을 가능성이 있다 . 데이터의 개수를 좀 더 확보하거나 중간에 batch norm layer 혹은 dropout 을 좀 더 적용하는 것도 하나의 방법일 것이다 .