# 기계학습을 활용한 문서 군집화 구현 및 실험 1

저작권디지털포렌식전공 석사과정

2021021635 박재하

# Abstract


- 피의자를 대상으로 디지털 증거를 압수수색할 경우, 증거로 식별해내고자 하는 압수수색의 범위가 사건과 관련성이 있는가 또는 과도한가가 큰 쟁점이 된다.
  (윤경, "형사<디지털증거 압수수색 : 범죄관련 정보와 무관 정보 혼재된 경우>", https://yklawyer.tistory.com/6123, 2019)
  (박병민, "디지털 증거 압수수색 개선방안에 관한 연구", 사법정책연구원, 2021)
- 범죄 관련 정보와 무관 정보가 혼재되어 있는 경우, 나아가 범죄 관련 파일 또는 그 메타데이터가 변조될 가능성을 고려한다면 적법한 절차로 압수수색을 하는 것은 더욱 어려운 문제가 된다.

- 문제 해결을 돕고자 디지털 포렌식과 최신 머신러닝 기술을 활용하여 방대하고 혼재되어 있는 디스크 이미지 내에서 문서 파일을 자동으로 수집하고 그 텍스트 정보를 추출하여, 텍스트의 내용 유사성에 기반하여 범죄와 관련된 정보를 빠르게 식별하는 데 도움을 줄 수 있는 알고리즘과 응용 프로그램을 개발하는 것을 본 연구의 목적으로 한다.
- (결론) **확보한 디스크 내 문서파일을 모두 수집하여 문서 내용별로 자동으로 군집화 하여 그 결과를 제공하는 방법론 및 도구 개발** (디지털증거 수집에 활용)

# Previous Work

1. 문서파일 확장자 텍스트 추출
   - 참조 논문 : 유병영, 이상진, “디지털 포렌식 조사를 위한 문서 필터 도구 개발”, 고려대학교 석사논문, 2011

2. BERT를 활용한 주제별 문서 군집화
   - 참조 논문 : Alvin Subakti etal, “The performance of BERT as data representation of text clustering”, Journal of Big Data, 2022

3. 기계학습을 활용한 문서 군집화 구현 및 실험
   - 참조 1 : SCKIT-LEARN, "Clustering text documents using k-means" (https://scikit-learn.org/stable/auto_examples/text/plot_document_clustering.html)
   - 참조 2 : HUGGINGFACE, "Docs > Transformers > Task Guides > Natural Language Processing > Text Classification" (https://huggingface.co/docs/transformers/tasks/sequence_classification#text-classification)
   - 참조 3 : jaehyeong(velog), "[Basic NLP] Transformers와 Tensorflow를 활용한 BERT Fine-tuning" (https://velog.io/@jaehyeong/Fine-tuning-Bert-using-Transformers-and-TensorFlow)

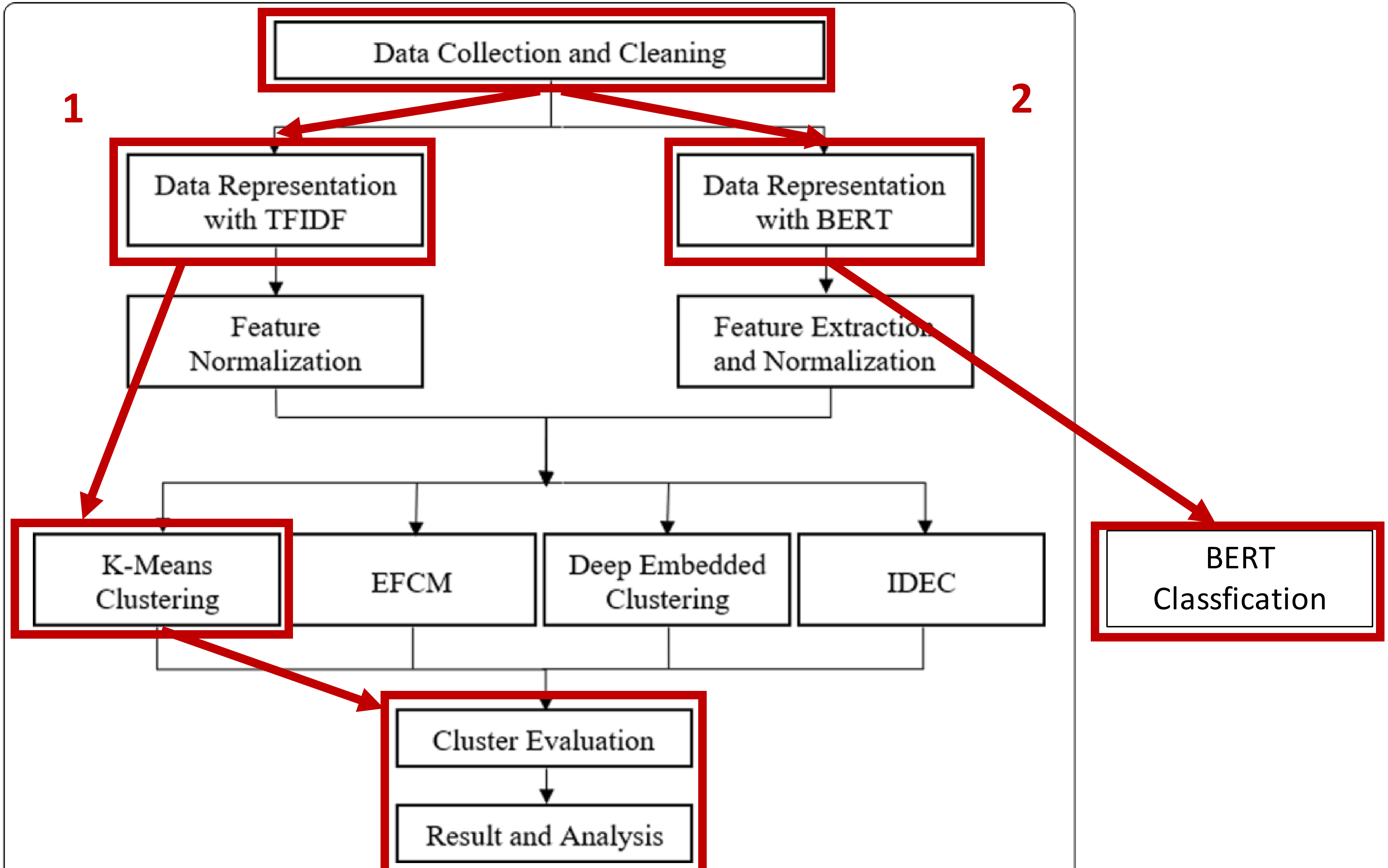



**Fig. 1** Flowchart of Simulation

# 1. TF-IDF + KMeans Clustering

- 데이터셋 로드 (20NewsGroups)

In [117…

```python
# 전체 다 받는거 테스트
import numpy as np
from sklearn.datasets import fetch_20newsgroups

dataset_all = fetch_20newsgroups(
    remove=("headers", "footers", "quotes"),
    subset='all',
    shuffle=True,
)
print(list(dataset_all.target_names))

labels = dataset_all.target
unique_labels, category_sizes = np.unique(labels, return_counts=True)
true_k = unique_labels.shape[0]

print(f"{len(dataset_all.data)} documents - {true_k} categories")

dataset_old = dataset
dataset = dataset_all
```

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.au
tos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'tal
k.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']
18846 documents - 20 categories
```

# 1. TF-IDF + KMeans Clustering

- TF-IDF 특징 추출

In [120…

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(
    max_df=0.5,
    min_df=5,
    stop_words="english",
)
t0 = time()
X_tfidf = vectorizer.fit_transform(dataset.data)

print(f"vectorization done in {time() - t0:.3f} s")
print(f"n_samples: {X_tfidf.shape[0]}, n_features: {X_tfidf.shape[1]}")
```

```
vectorization done in 1.110 s
n_samples: 18846, n_features: 24164
```

# 1. TF-IDF + KMeans Clustering

- Fit and Evaluate 함수 구성 (w ACC, ARI, NMI 등 + Silhouette Coeff.)

**Table 2** Cluster evaluation on AG News dataset

| Method | AG news | | |
|---|---|---|---|
| | ACC | NMI | ARI |
| TFIDF + KM | 0.5019 ± 0.0718 | 0.2559 ± 0.0802 | 0.2552 ± 0.0803 |
| BERT + Max + I + KM | 0.7674 ± 0.0018 | 0.4872 ± 0.0021 | 0.4868 ± 0.0021 |
| BERT + Max + LN + KM | 0.7913 ± 0.0040 | 0.5199 ± 0.0050 | 0.5195 ± 0.0050 |
| BERT + Max + N + KM | 0.7858 ± 0.0017 | 0.5136 ± 0.0025 | 0.5132 ± 0.0025 |
| BERT + Max + MM + KM | 0.4408 ± 0.0012 | 0.1986 ± 0.0014 | 0.1979 ± 0.0014 |
| BERT + Mean + I + KM | 0.6491 ± 0.0016 | 0.4196 ± 0.0010 | 0.4191 ± 0.0010 |

```
In [115]:
from collections import defaultdict
from sklearn import metrics
from time import time

evaluations = []
evaluations_std = []

def fit_and_evaluate(km, X, name=None, n_runs=5):
    name = km.__class__.__name__ if name is None else name

    train_times = []
    scores = defaultdict(list)
    for seed in range(n_runs):
        km.set_params(random_state=seed)
        t0 = time()
        km.fit(X)
        train_times.append(time() - t0)
        scores["Accuracy"].append(metrics.accuracy_score(labels, km.labels_))
        scores["Homogeneity"].append(metrics.homogeneity_score(labels, km.labels_))
        scores["Completeness"].append(metrics.completeness_score(labels, km.labels_))
        scores["V-measure"].append(metrics.v_measure_score(labels, km.labels_))
        scores["Normalized Mutual Information"].append(
            metrics.normalized_mutual_info_score(labels, km.labels_)
        )
        scores["Adjusted Rand-Index"].append(
            metrics.adjusted_rand_score(labels, km.labels_)
        )
        scores["Silhouette Coefficient"].append(
            metrics.silhouette_score(X, km.labels_, sample_size=2000)
        )
    train_times = np.asarray(train_times)

nt(f"clustering done in {train_times.mean():.2f} ± {train_times.std():.2f} s ")
luation = {
 "estimator": name,
 "train_time": train_times.mean(),

luation_std = {
 "estimator": name,
 "train_time": train_times.std(),

 score_name, score_values in scores.items():
 mean_score, std_score = np.mean(score_values), np.std(score_values)
 print(f"{score_name}: {mean_score:.3f} ± {std_score:.3f}")
 evaluation[score_name] = mean_score
 evaluation_std[score_name] = std_score
luations.append(evaluation)
luations_std.append(evaluation_std)
```

### 2.3.10.5. Silhouette Coefficient

If the ground truth labels are not known, evaluation must be performed using the model itself. The Silhouette Coefficient (`sklearn.metrics.silhouette_score`) is an example of such an evaluation, where a higher Silhouette Coefficient score relates to a model with better defined clusters. The Silhouette Coefficient is defined for each sample and is composed of two scores:

- **a**: The mean distance between a sample and all other points in the same class.
- **b**: The mean distance between a sample and all other points in the *next nearest cluster*.

The Silhouette Coefficient *s* for a single sample is then given as:

$$s = \frac{b - a}{\ldots}$$

# 1. TF-IDF + KMeans Clustering

- KMeans 클러스터링 w TF-IDF Vector

In [123…

```
kmeans = KMeans(
    n_clusters=true_k,
    max_iter=100,
    n_init=5,
)

fit_and_evaluate(kmeans, X_tfidf, name="KMeans\non tf-idf vectors")
```

```
clustering done in 3.59 ± 1.18 s
Accuracy: 0.044 ± 0.023
Homogeneity: 0.208 ± 0.092
Completeness: 0.396 ± 0.046
V-measure: 0.253 ± 0.079
Normalized Mutual Information: 0.253 ± 0.079
Adjusted Rand-Index: 0.052 ± 0.020
Silhouette Coefficient: 0.005 ± 0.002
```

# 1. TF-IDF + KMeans Clustering

- LSA 차원 축소 적용(TruncatedSVD)

In [124…

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

lsa = make_pipeline(TruncatedSVD(n_components=100), Normalizer(copy=False))
t0 = time()
X_lsa = lsa.fit_transform(X_tfidf)
explained_variance = lsa[0].explained_variance_ratio_.sum()

print(f"LSA done in {time() - t0:.3f} s")
print(f"Explained variance of the SVD step: {explained_variance * 100:.1f}%")
```

```
LSA done in 2.092 s
Explained variance of the SVD step: 10.5%
```

Using a single initialization means the processing time will be reduced for both **KMeans** and **MiniBatchKMeans**.

In [125…

```
kmeans = KMeans(
    n_clusters=true_k,
    max_iter=100,
    n_init=1,
)

fit_and_evaluate(kmeans, X_lsa, name="KMeans\nwith LSA on tf-idf vectors")
```

```
clustering done in 0.59 ± 0.09 s
Accuracy: 0.060 ± 0.027
Homogeneity: 0.338 ± 0.013
Completeness: 0.367 ± 0.016
V-measure: 0.351 ± 0.014
Normalized Mutual Information: 0.351 ± 0.014
Adjusted Rand-Index: 0.164 ± 0.014
Silhouette Coefficient: 0.061 ± 0.002
```

# 1. TF-IDF + KMeans Clustering

- KMeans 대신 MiniBatchKMeans 적용

In [126…

```
from sklearn.cluster import MiniBatchKMeans

minibatch_kmeans = MiniBatchKMeans(
    n_clusters=true_k,
    n_init=1,
    init_size=1000,
    batch_size=1000,
)

fit_and_evaluate(
    minibatch_kmeans,
    X_lsa,
    name="MiniBatchKMeans\nwith LSA on tf-idf vectors",
)
```

```
clustering done in 0.43 ± 0.04 s
Accuracy: 0.048 ± 0.014
Homogeneity: 0.309 ± 0.028
Completeness: 0.337 ± 0.022
V-measure: 0.322 ± 0.024
Normalized Mutual Information: 0.322 ± 0.024
Adjusted Rand-Index: 0.148 ± 0.044
Silhouette Coefficient: 0.045 ± 0.005
```

# 1. TF-IDF + KMeans Clustering

- 결과

```
df_merge.rename(index={
    'MiniBatchKMeans\n with LSA on hashed vectors':'HV+LSA+mKM',
    'KMeans\nwith LSA on hashed vectors':'HV+LSA+KM',
    'MiniBatchKMeans\nwith LSA on tf-idf vectors':'TFIDF+LSA+mKM',
    'KMeans\nwith LSA on tf-idf vectors':'TFIDF+LSA+KM',
    'KMeans\non tf-idf vectors':'TFIDF+KM',
}, columns={
    'Accuracy':'ACC',
    'Normalized Mutual Information':'NMI',
    'Adjusted Rand-Index':'ARI',
    'Silhouette Coefficient':'SC',
}, inplace=True)
df_merge.index.names=['Method']

df_merge
```

Out[132]:

| Method | ACC | NMI | ARI | SC |
|---|---|---|---|---|
| HV+LSA+mKM | 0.0553±0.0236 | 0.3219±0.02 | 0.1229±0.0255 | 0.049±0.0065 |
| HV+LSA+KM | 0.0575±0.017 | 0.3476±0.0118 | 0.1446±0.0113 | 0.0604±0.0028 |
| TFIDF+LSA+mKM | 0.0484±0.0138 | 0.322±0.0237 | 0.1481±0.0443 | 0.0452±0.0048 |
| TFIDF+LSA+KM | 0.0598±0.0273 | 0.3515±0.0138 | 0.1636±0.0136 | 0.061±0.0022 |
| TFIDF+KM | 0.0439±0.0229 | 0.253±0.0788 | 0.0522±0.0202 | 0.0053±0.0023 |

# 1. TF-IDF + KMeans Clustering

- 분석 : Terms per 20 Clusters

In [127…

```
original_space_centroids = lsa[0].inverse_transform(kmeans.cluster_centers_)
order_centroids = original_space_centroids.argsort()[:, ::-1]
terms = vectorizer.get_feature_names_out()

for i in range(true_k):
    print(f"Cluster {i}: ", end="")
    for ind in order_centroids[i, :10]:
        print(f"{terms[ind]} ", end="")
    print()
```

| | |
|---|---|
| alt.atheism | Cluster 0: did didn just think say let know got way don |
| comp.graphics | Cluster 1: right people left government rights just make think law amendment |
| comp.os.ms-windows.misc | Cluster 2: just don like know think time good people sure really |
| comp.sys.ibm.pc.hardware | Cluster 3: card monitor video bus board apple memory bit ram mac |
| comp.sys.mac.hardware | Cluster 4: people think time evidence say true don point believe life |
| comp.windows.x | Cluster 5: israel jews israeli armenian arab jewish armenians people arabs turkish |
| misc.forsale | Cluster 6: sale 00 offer shipping price 10 condition new asking interested |
| rec.autos | Cluster 7: thanks advance hi know mail does email info looking help |
| rec.motorcycles | Cluster 8: list mailing subscribe com send mail price faq know address |
| rec.sport.baseball | Cluster 9: does know anybody like mean just work use new info |
| rec.sport.hockey | Cluster 10: key chip encryption clipper government keys escrow algorithm use nsa |
| sci.crypt | Cluster 11: windows file program window use files dos using problem server |
| sci.electronics | Cluster 12: space nasa shuttle launch orbit earth moon cost mission station |
| sci.med | Cluster 13: gun fbi batf koresh guns people law children government police |
| sci.space | Cluster 14: ve got seen heard just like good don used years |
| soc.religion.christian | Cluster 15: drive scsi hard disk drives ide controller floppy cd software |
| talk.politics.guns | Cluster 16: car bike cars engine new miles just like good speed |
| talk.politics.mideast | Cluster 17: game team year games players hockey season play win baseball |
| talk.politics.misc | Cluster 18: edu com mail computer new read good phone article use |
| talk.religion.misc | Cluster 19: god jesus bible christ believe faith sin christian christians people |

# 1. TF-IDF + KMeans Clustering

- 분석 : Terms per 20 Clusters

In [127…

```
original_space_centroids = lsa[0].inverse_transform(kmeans.cluster_centers_)
order_centroids = original_space_centroids.argsort()[:, ::-1]
terms = vectorizer.get_feature_names_out()

for i in range(true_k):
    print(f"Cluster {i}: ", end="")
    for ind in order_centroids[i, :10]:
        print(f"{terms[ind]} ", end="")
    print()
```

| | |
|---|---|
| alt.atheism | Cluster 0: did didn just think say let know got way don |
| comp.graphics | Cluster 1: right people left government rights just make think law amendment |
| comp.os.ms-windows.misc | Cluster 2: just don like know think time good people sure really |
| comp.sys.ibm.pc.hardware | Cluster 3: card monitor video bus board apple memory bit ram mac |
| comp.sys.mac.hardware | Cluster 4: people think time evidence say true don point believe life |
| comp.windows.x | Cluster 5: israel jews israeli armenian arab jewish armenians people arabs turkish |
| misc.forsale | Cluster 6: sale 00 offer shipping price 10 condition new asking interested |
| rec.autos | Cluster 7: thanks advance hi know mail does email info looking help |
| rec.motorcycles | Cluster 8: list mailing subscribe com send mail price faq know address |
| rec.sport.baseball | Cluster 9: does know anybody like mean just work use new info |
| rec.sport.hockey | Cluster 10: key chip encryption clipper government keys escrow algorithm use nsa |
| sci.crypt | Cluster 11: windows file program window use files dos using problem server |
| sci.electronics | Cluster 12: space nasa shuttle launch orbit earth moon cost mission station |
| sci.med | Cluster 13: gun fbi batf koresh guns people law children government police |
| sci.space | Cluster 14: ve got seen heard just like good don used years |
| soc.religion.christian | Cluster 15: drive scsi hard disk drives ide controller floppy cd software |
| talk.politics.guns | Cluster 16: car bike cars engine new miles just like good speed |
| talk.politics.mideast | Cluster 17: game team year games players hockey season play win baseball |
| talk.politics.misc | Cluster 18: edu com mail computer new read good phone article use |
| talk.religion.misc | Cluster 19: god jesus bible christ believe faith sin christian christians people |

# 1. TF-IDF + KMeans Clustering

- 분석 : **Tokenizing** 해서 무의미한 word 제거, 일반형으로 변환 필요

**Text rerpesentation**

First, text representation from TFIDF is extracted. Tokenization with the help of the natural language toolkit (NLTK), where each word in a sentence is separated, is carried out beforehand. Next, the tokenized text data representation is taken by calculating the weight as described in Eq. 1. Then, to be used in DEC and IDEC models, normalization is applied to the text data representation generated by TFIDF. The representation is multiplied by the root of the feature dimension so that for an $i$-th text data representation vector, $x_i$ with the dimension $D$, we get $\frac{1}{D}||x_i||_2^2 = 1$

## tf-idf with scikit-learn - Code

Here is the code not much changed from the original: Document Similarity using NLTK and Scikit-Learn . The input files are from Steinbeck's Pearl ch1-6.

```
import nltk
import string
import os

from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.stem.porter import PorterStemmer

path = './tf-idf'
token_dict = {}


def tokenize(text):
    tokens = nltk.word_tokenize(text)
```

<--
https://www.bogotobogo.com/python/NLTK/tf_idf_with_scikit-learn_NLTK.php

## sklearn.feature_extraction.text.TfidfVectorizer

*class* `sklearn.feature_extraction.text.TfidfVectorizer`(**, input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, analyzer='word', stop_words=None, token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.float64'>, norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False*) [source]

Convert a collection of raw documents to a matrix of TF-IDF features.

Equivalent to `CountVectorizer` followed by `TfidfTransformer`.

Read more in the User Guide.

**Parameters:**

**input : {'filename', 'file', 'content'}, default='content'**

- If `'filename'`, the sequence passed as an argument to fit is expected to be a list of filenames that need reading to fetch the raw content to analyze.
- If `'file'`, the sequence items must have a 'read' method (file-like object) that is called to fetch the bytes in memory.
- If `'content'`, the input is expected to be a sequence of items that can be of type string or byte.

**encoding : *str, default='utf-8'***

If bytes or files are given to analyze, this encoding is used to decode.

**decode_error : *{'strict', 'ignore', 'replace'}, default='strict'***

Instruction on what to do if a byte sequence is given to analyze that contains characters not of the given `encoding`. By default, it is 'strict', meaning that a UnicodeDecodeError will be raised. Other values are 'ignore' and 'replace'.

**strip_accents : *{'ascii', 'unicode'} or callable, default=None***

Remove accents and perform other character normalization during the preprocessing step. 'ascii' is a fast method that only works on characters that have a direct ASCII mapping. 'unicode' is a slightly slower method that works on any characters. None (default) does nothing.

Both 'ascii' and 'unicode' use NFKD normalization from `unicodedata.normalize`.

**lowercase : *bool, default=True***

Convert all characters to lowercase before tokenizing.

**preprocessor : *callable, default=None***

Override the preprocessing (string transformation) stage while preserving the tokenizing and n-grams generation steps. Only applies if `analyzer` is not callable.

**tokenizer : *callable, default=None***

Override the string tokenization step while preserving the preprocessing and n-grams generation steps. Only applies if `analyzer == 'word'`.

# 2. BERT Classification

- Load Train/Test Dataset (나눠서)

In [1]:
```
import numpy as np
from sklearn.datasets import fetch_20newsgroups

dataset_train = fetch_20newsgroups(
    remove=("headers", "footers", "quotes"),
    subset='train',
    shuffle=True,
)

labels = dataset_train.target
unique_labels, category_sizes = np.unique(labels, return_counts=True)
true_k = unique_labels.shape[0]

print(f"{len(dataset_train.data)} documents - {true_k} categories")
print(list(dataset_train.target_names))
print()

dataset_test = fetch_20newsgroups(
    remove=("headers", "footers", "quotes"),
    subset='test',
    shuffle=True,
)

labels = dataset_test.target
unique_labels, category_sizes = np.unique(labels, return_counts=True)
true_k = unique_labels.shape[0]

print(f"{len(dataset_test.data)} documents - {true_k} categories")
print(list(dataset_test.target_names))
```

```
11314 documents - 20 categories
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.au
tos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'tal
k.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']

7532 documents - 20 categories
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.au
tos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'tal
k.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']
```

# 2. BERT Classification

- Train Set은 다시 Train / Validation Set으로 나눈다(8:2)

In [6]:
```
train_texts = train_df["text"].to_list()
train_labels = train_df["encoded_label"].to_list()
```

In [7]:
```
from sklearn.model_selection import train_test_split

# Split Train_Set to Actual Training and Validating Data
train_texts, val_texts, train_labels, val_labels = train_test_split(train_texts, train_labels, test_size=0.2, random_state=0)
```

In [8]:
```
print(len(train_texts), len(val_texts), len(train_labels), len(val_labels))
```

```
9051 2263 9051 2263
```

# 2. BERT Classification

- Load Tokenizer and Tokenizing

```
HUGGINGFACE_MODEL_PATH = "bert-base-uncased"
```

In [12]:
```
from transformers import BertTokenizer

# Load Tokenizer
tokenizer = BertTokenizer.from_pretrained(HUGGINGFACE_MODEL_PATH)

# Tokenizing
train_encodings = tokenizer(train_texts, truncation=True, padding=True)
val_encodings = tokenizer(val_texts, truncation=True, padding=True)
```

In [13]:
```
type(train_encodings)
```

Out[13]:
```
transformers.tokenization_utils_base.BatchEncoding
```

In [17]:
```
dict_train_encodings = dict(train_encodings)
print(type(dict_train_encodings))
```

```
<class 'dict'>
```

# 2. BERT Classification

- Creating Dataset object for PyTorch

이거랑 똑같이, torch.utils.data.Dataset 하위 클래스 하나 만들어서 넣어주면됨. torch.utils.data.Dataset 요건에 맞게 `__init__`, `__getitem__`, `__len__` 만들어줘야됨.

In [23]:

```
import torch

class TNGDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

# trainset-set
train_dataset = TNGDataset(train_encodings, train_labels)

# validation-set
val_dataset = TNGDataset(val_encodings, val_labels)
```

In [26]:

```
# 되나?!?!?!?!?

print(train_dataset.__len__)
```

```
<bound method TNGDataset.__len__ of <__main__.TNGDataset object at 0x12cf3ebe0>>
```

# 2. BERT Classification

- Fine-Tuning BERT(bert-base-uncased)

In [27]:
```
from transformers import BertForSequenceClassification, Trainer, TrainingArguments

num_labels = len(label_encoder_classes_)
print("num_labels: ", num_labels)

training_args = TrainingArguments(
    output_dir='./230315BC/results',  # output directory
    num_train_epochs=5,  # total number of training epochs
    per_device_train_batch_size=16,  # batch size per device during training
    per_device_eval_batch_size=64,  # batch size for evaluation
    warmup_steps=500,  # number of warmup steps for learning rate scheduler
    weight_decay=0.01,  # strength of weight decay
    logging_dir='./230315BC/logs'  # directory for storing logs
)

trainer_model = BertForSequenceClassification.from_pretrained(HUGGINGFACE_MODEL_PATH, num_labels=num_labels)

trainer = Trainer(
    model = trainer_model,  # the instantiated HugginFace Transformers model to be trained
    args=training_args,  # training arguments, defined above
    train_dataset=train_dataset,  # training dataset
    eval_dataset=val_dataset  # evaluation dataset
)
```

```
num_labels:  20
```

```
HUGGINGFACE_MODEL_PATH = "bert-base-uncased"
```

```
trainer.train()
```

```
/Users/jaeha/opt/anaconda3/lib/python3.9/site-packages/transformers/optimization.py:306: FutureWarning: This implementation of AdamW is deprecated and wil
l be removed in a future version. Use the PyTorch implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to disable this warning
  warnings.warn(
***** Running training *****
  Num examples = 9051
  Num Epochs = 5
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 16
  Gradient Accumulation steps = 1
  Total optimization steps = 2830
  Number of trainable parameters = 109497620
```

[2830/2830 11:01:42, Epoch 5/5]

| Step | Training Loss |
|---|---|
| 500 | 1.986500 |
| 1000 | 0.883200 |
| 1500 | 0.575500 |
| 2000 | 0.348500 |
| 2500 | 0.216000 |

```
Saving model checkpoint to ./230315BC/results/checkpoint-500
Configuration saved in ./230315BC/results/checkpoint-500/config.json
Model weights saved in ./230315BC/results/checkpoint-500/pytorch_model.bin
Saving model checkpoint to ./230315BC/results/checkpoint-1000
Configuration saved in ./230315BC/results/checkpoint-1000/config.json
Model weights saved in ./230315BC/results/checkpoint-1000/pytorch_model.bin
Saving model checkpoint to ./230315BC/results/checkpoint-1500
Configuration saved in ./230315BC/results/checkpoint-1500/config.json
Model weights saved in ./230315BC/results/checkpoint-1500/pytorch_model.bin
Saving model checkpoint to ./230315BC/results/checkpoint-2000
Configuration saved in ./230315BC/results/checkpoint-2000/config.json
Model weights saved in ./230315BC/results/checkpoint-2000/pytorch_model.bin
Saving model checkpoint to ./230315BC/results/checkpoint-2500
Configuration saved in ./230315BC/results/checkpoint-2500/config.json
Model weights saved in ./230315BC/results/checkpoint-2500/pytorch_model.bin


Training completed. Do not forget to share your model on huggingface.co/models =)
```

```
TrainOutput(global_step=2830, training_loss=0.7269764256561603, metrics={'train_runtime': 39717.5859, 'train_samples_per_second': 1.139, 'train_steps_per_
second': 0.071, 'total_flos': 1.1909015171665592e+16, 'train_loss': 0.7269764256561603, 'epoch': 5.0})
```

# 2. BERT Classification

- LABEL 이름 수정

```
model = trainer_model
```

```
model
```

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (1): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
```

```
import re

for label, id_ in trainer_model.config.label2id.items():
    print(id_, label, label_encoder_classes_[int(re.sub('LABEL_','',label))])
```

```
0 LABEL_0 alt.atheism
1 LABEL_1 comp.graphics
2 LABEL_2 comp.os.ms-windows.misc
3 LABEL_3 comp.sys.ibm.pc.hardware
4 LABEL_4 comp.sys.mac.hardware
5 LABEL_5 comp.windows.x
6 LABEL_6 misc.forsale
7 LABEL_7 rec.autos
8 LABEL_8 rec.motorcycles
9 LABEL_9 rec.sport.baseball
10 LABEL_10 rec.sport.hockey
11 LABEL_11 sci.crypt
12 LABEL_12 sci.electronics
13 LABEL_13 sci.med
14 LABEL_14 sci.space
15 LABEL_15 soc.religion.christian
16 LABEL_16 talk.politics.guns
17 LABEL_17 talk.politics.mideast
18 LABEL_18 talk.politics.misc
19 LABEL_19 talk.religion.misc
```

```
id2label = model.config.id2label
fixed_id2label = {id : label_encoder_classes_[int(re.sub('LABEL_', '', label))]  for id, label in id2label.items()}
fixed_id2label
```

```
{0: 'alt.atheism',
 1: 'comp.graphics',
 2: 'comp.os.ms-windows.misc',
 3: 'comp.sys.ibm.pc.hardware',
 4: 'comp.sys.mac.hardware',
 5: 'comp.windows.x',
 6: 'misc.forsale',
 7: 'rec.autos',
 8: 'rec.motorcycles',
 9: 'rec.sport.baseball',
 10: 'rec.sport.hockey',
 11: 'sci.crypt',
 12: 'sci.electronics',
 13: 'sci.med',
 14: 'sci.space',
 15: 'soc.religion.christian',
 16: 'talk.politics.guns',
 17: 'talk.politics.mideast',
 18: 'talk.politics.misc',
 19: 'talk.religion.misc'}
```

```
label2id = model.config.label2id
fixed_label2id = {label_encoder_classes_[int(re.sub('LABEL_', '', label))] : id   for id, label in id2label.items()}
fixed_label2id
```

# 2. BERT Classification

- Predict 준비, Predict (TOP-1)

```python
from transformers import TextClassificationPipeline

# Load Fine-tuned model
loaded_tokenizer = BertTokenizer.from_pretrained(MODEL_SAVE_PATH)
loaded_model = BertForSequenceClassification.from_pretrained(MODEL_SAVE_PATH)

text_classifier = TextClassificationPipeline(
    tokenizer=loaded_tokenizer,
    model=loaded_model,
    framework="pt",  # PyTorch:"pt", TensorFlow:"tf"
    return_all_scores=True,  # test case당 모든 레이블에 대한 결과값 다 받고,
                             # 이따 최상위 하나 고를거임
    truncation=True  # 이거 넣어주니까 길이때매 에러 안난다 → 512토큰으로 자르는거.
                     # BERT max-length가 512자너
)
```

```
loading file vocab.txt
loading file added_tokens.json
loading file special_tokens_map.json
loading file tokenizer_config.json
loading configuration file 230315BC/soddokayo/bert-base-uncased-20newsgroups/config.j
Model config BertConfig {
  "_name_or_path": "bert-base-uncased",
  "architectures": [
    "BertForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "alt.atheism",
    "1": "comp.graphics",
    "2": "comp.os.ms-windows.misc",
    "3": "comp.sys.ibm.pc.hardware",
    "4": "comp.sys.mac.hardware",
    "5": "comp.windows.x",
    "6": "misc.forsale",
```

```python
# 참고 블로그랑 변수 맞춰주자
import pandas as pd

dt_labels = [dataset_test.target_names[_] for _ in dataset_test.target]
#print(dataset_test_labels[:10])
test_dataset_list = [{'text':text, 'label':label, 'encoded_label':target} ₩
                     for text, label, target ₩
                     in zip(dataset_test.data, dt_labels, dataset_test.target)
test_df = pd.DataFrame(test_dataset_list)
test_df.head()

#print(type(dataset_test.data[0]), type(dt_labels[0]))
```

| | text | label | encoded_label |
|---|---|---|---|
| 0 | I am a little confused on all of the models of... | rec.autos | 7 |
| 1 | I'm not familiar at all with the format of the... | comp.windows.x | 5 |
| 2 | \nIn a word, yes.\n | alt.atheism | 0 |
| 3 | \nThey were attacking the Iraqis to drive them... | talk.politics.mideast | 17 |
| 4 | \nI've just spent two solid months arguing tha... | talk.religion.misc | 19 |

```python
predicted_label_list = []
predicted_score_list = []

for text in test_df['text']:
    # predict
    preds_list = text_classifier(text)[0]
    #print(preds_list)

    sorted_preds_list = sorted(preds_list, key=lambda x: x['score'], reverse=True)
    predicted_label_list.append(sorted_preds_list[0]['label']) # label
    predicted_score_list.append(sorted_preds_list[0]['score']) # score
    print(sorted_preds_list[0]['label'], sorted_preds_list[0]['score'])
```

```
rec.autos 0.995415449142456
comp.windows.x 0.971951007843017 6
talk.politics.misc 0.288884669542312 6
alt.atheism 0.967523634433746 3
alt.atheism 0.728478670120239 3
sci.med 0.997557878494262 7
talk.religion.misc 0.482773840427398 7
comp.os.ms-windows.misc 0.758078813552856 4
comp.windows.x 0.996731162071228
comp.graphics 0.993117690086364 7
comp.os.ms-windows.misc 0.994273722171783 4
comp.windows.x 0.996301293373107 9
talk.politics.mideast 0.997228443622589 1
talk.politics.misc 0.371242731809616 1
soc.religion.christian 0.978166699409484 9
comp.sys.ibm.pc.hardware 0.985935688018798 8
comp.sys.mac.hardware 0.993780791759491
comp.sys.mac.hardware 0.514266431331634 5
misc.forsale 0.969426572322845 5
talk.politics.guns 0.993849456310272 2
sci.med 0.940613448619842 5
misc.forsale 0.756355166435241 7
talk.politics.mideast 0.965000092983245 8
sci.space 0.997312068939209
comp.sys.ibm.pc.hardware 0.990989029407501 2
sci.med 0.996659398078918 5
sci.crypt 0.566331326961517 3
rec.autos 0.995601058006286 6
rec.autos 0.996310770511627 2
soc.religion.christian 0.897425055503845 2
comp.windows.x 0.996439158916473 4
comp.windows.x 0.996763706207275 4
comp.sys.mac.hardware 0.993170499801635 7
comp.sys.ibm.pc.hardware 0.987447202205658
sci.space 0.997376203536987 3
comp.graphics 0.967552185058593 8
rec.sport.baseball 0.997227847576141 4
comp.sys.mac.hardware 0.991850197315216 1
misc.forsale 0.790114104747772 2
comp.graphics 0.995283424854278 6
talk.politics.mideast 0.997053384780883 8
comp.os.ms-windows.misc 0.516814768314361 6
rec.motorcycles 0.997046291828155 5
comp.graphics 0.993740439414978
talk.politics.guns 0.980272114276886
```

# 2. BERT Classification

- 결과

```
test_df['pred'] = predicted_label_list
test_df['score'] = predicted_score_list
test_df.head()
```

| | text | label | encoded_label | pred | score |
|---|---|---|---|---|---|
| 0 | I am a little confused on all of the models of... | rec.autos | 7 | rec.autos | 0.995415 |
| 1 | I'm not familiar at all with the format of the... | comp.windows.x | 5 | comp.windows.x | 0.971951 |
| 2 | \nIn a word, yes.\n | alt.atheism | 0 | talk.politics.misc | 0.288885 |
| 3 | \nThey were attacking the Iraqis to drive them... | talk.politics.mideast | 17 | alt.atheism | 0.967524 |
| 4 | \nI've just spent two solid months arguing tha... | talk.religion.misc | 19 | alt.atheism | 0.728479 |

```
from sklearn.metrics import classification_report

print(classification_report(y_true=test_df['label'], y_pred=test_df['pred']))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| alt.atheism | 0.50 | 0.44 | 0.47 | 319 |
| comp.graphics | 0.69 | 0.74 | 0.71 | 389 |
| comp.os.ms-windows.misc | 0.69 | 0.69 | 0.69 | 394 |
| comp.sys.ibm.pc.hardware | 0.66 | 0.69 | 0.68 | 392 |
| comp.sys.mac.hardware | 0.79 | 0.73 | 0.76 | 385 |
| comp.windows.x | 0.86 | 0.76 | 0.81 | 395 |
| misc.forsale | 0.83 | 0.83 | 0.83 | 390 |
| rec.autos | 0.81 | 0.74 | 0.77 | 396 |
| rec.motorcycles | 0.75 | 0.77 | 0.76 | 398 |
| rec.sport.baseball | 0.60 | 0.86 | 0.71 | 397 |
| rec.sport.hockey | 0.90 | 0.90 | 0.90 | 399 |
| sci.crypt | 0.81 | 0.73 | 0.76 | 396 |
| sci.electronics | 0.64 | 0.63 | 0.63 | 393 |
| sci.med | 0.81 | 0.83 | 0.82 | 396 |
| sci.space | 0.83 | 0.77 | 0.80 | 394 |
| soc.religion.christian | 0.75 | 0.72 | 0.73 | 398 |
| talk.politics.guns | 0.60 | 0.68 | 0.64 | 364 |
| talk.politics.mideast | 0.90 | 0.75 | 0.82 | 376 |
| talk.politics.misc | 0.56 | 0.45 | 0.50 | 310 |
| talk.religion.misc | 0.28 | 0.37 | 0.32 | 251 |
| accuracy | | | 0.72 | 7532 |
| macro avg | 0.71 | 0.70 | 0.71 | 7532 |
| weighted avg | 0.73 | 0.72 | 0.72 | 7532 |

# Future Work

1. TF-IDF 전처리 (word tokenize, extracting nouns)

Module contents

NLTK Tokenizer Package

Tokenizers divide strings into lists of substrings. For example, tokenizers can be used to find the words and punctuation in a string:

```
>>> from nltk.tokenize import word_tokenize
>>> s = '''Good muffins cost $3.88\nin New York.  Please buy me
... two of them.\n\nThanks.'''
>>> word_tokenize(s)
['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.',
'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.']
```

This particular tokenizer requires the Punkt sentence tokenization models to be installed. NLTK also provides a simpler, regular-expression based tokenizer, which splits text on whitespace and punctuation:

```
>>> from nltk.tokenize import wordpunct_tokenize
>>> wordpunct_tokenize(s)
['Good', 'muffins', 'cost', '$', '3', '.', '88', 'in', 'New', 'York', '.',
'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.']
```

We can also operate at the level of sentences, using the sentence tokenizer directly as follows:

```
>>> from nltk.tokenize import sent_tokenize, word_tokenize
>>> sent_tokenize(s)
['Good muffins cost $3.88\nin New York.', 'Please buy me\ntwo of them.', 'Thanks.']
>>> [word_tokenize(t) for t in sent_tokenize(s)]
[['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.'],
['Please', 'buy', 'me', 'two', 'of', 'them', '.'], ['Thanks', '.']]
```

```
import nltk

lines = 'lines is some string of words'
# function to test if something is a noun
is_noun = lambda pos: pos[:2] == 'NN'
# do the nlp stuff
tokenized = nltk.word_tokenize(lines)
nouns = [word for (word, pos) in nltk.pos_tag(tokenized) if is_noun(pos)]

print nouns
>>> ['lines', 'string', 'words']
```

# Future Work

## 2. IDEC, DEC 클러스터링 적용 + α

github.com/XifengGuo/IDEC

| | | |
|---|---|---|
| data | add data folder and fix error in README.md | 6 years ago |
| .gitignore | add data folder and fix error in README.md | 6 years ago |
| DEC.py | remove dims default value | 6 years ago |
| IDEC.py | release | 6 years ago |
| README.md | add data folder and fix error in README.md | 6 years ago |
| datasets.py | release | 6 years ago |
| dec_model.png | release | 6 years ago |
| idec_model.png | release | 6 years ago |

README.md

### Improved Deep Embedded Clustering (IDEC)

Keras implementation for our IJCAI-17 paper:

- Xifeng Guo, Long Gao, Xinwang Liu, Jianping Yin. Improved Deep Embedded Clustering with Local Structure Preservation. IJCAI 2017.

and re-implementation for paper:

- Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. ICML 2016.

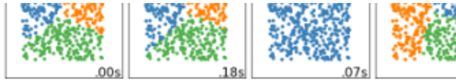
This code requires pretrained autoencoder weights provided. Use IDEC-toy code for a quick start.

Usage

**scikit-learn 1.2.2**
Other versions

Please **cite us** if you use the software.






A compariso

| Method name | Parameters | Scalabi |
|---|---|---|
| K-Means | number of clusters | Very lar medium MiniBat |
| Affinity propagation | damping, sample preference | Not sca n_samp |
| Mean-shift | bandwidth | Not sca n_sampl |
| Spectral clustering | number of clusters | Medium small n_ |
| Ward hierarchical clustering | number of clusters or distance threshold | Large n n_clust |
| Agglomerative clustering | number of clusters or distance thresh-old, linkage type, distance | Large n n_clust |
| DBSCAN | neighborhood size | Very lar medium |
| OPTICS | minimum cluster membership | Very lar large n_ |
| Gaussian mixtures | many | Not sca |
| BIRCH | branching factor, threshold, optional global clusterer. | Large n n_sampl |
| Bisecting K-Means | number of clusters | Very lar medium |

# Future Work

## 3. WordPiece Tokenization before BERT , Feature Extraction & Normalization after BERT (for Clustering)

The process of taking text representation using the feature-based approach of BERT is done by feeding a text input into BERT. The text input is tokenized using WordPiece Model before being fed into BERT. For a document containing n tokens, the text representation obtained is $n$ numeric vectors with dimension 768. The output vector of all words in the document can be arranged into a matrix of size $n \times 768$.

**Feature extraction and normalization strategies**

A feature extraction strategy is necessary to convert high-dimensional representation from BERT into a fixed-sized feature vector with lower dimensions. Two feature extraction strategies were implemented, namely max pooling and mean pooling. Max pooling assumes that the highest value contains the most important features. Suppose that there are n tokens in a document and the $i$-th token has a vector representation as $h_i = [h_{i1}, h_{i2}, \ldots, h_{id}]$ with dimension $d$. The max pooling strategy can be represented by the following equation [8]:

```python
model.embeddings.position_embeddings
```

[실행 결과]
Embedding(512, 768)

여기서 512는 BERT가 한 번에 받아들일 수 있는 최대 token 수를 의미하고, 768은 마찬가지로 `hidden_size`(출력되는 embedding의 차원)를 의미한다. line 39, 40을 보면, `position_ids`가 직접 입력되지 않은 경우 `past_key_values_length`(default: 0)부터 `seq_length + past_key_values_length`까지 token의 길이(`seq_length`)개의 정수를 `position_ids`로 삼는 것을 볼 수 있다. 그리고 line 59에서 `position_ids`를 인덱스로 하여 `model.embeddings.position_embeddings` layer로 `position_+embeddings`를 계산하는 것을 볼 수 있다. 이때 성능을 위해 line 16에서 `position_ids`를 BERT가 한 번에 받아들일 수 있는 최대 token 수까지 `torch.nn.Module.register_buffer`로 미리 만들어 놓은 것을 볼 수 있다.

<-- https://heekangpark.github.io/nlp/huggingface-bert

# Future Work

4. 다른 데이터셋에 적용(AG_NEWS, Yahoo!Answers 등)

**IMDB movie review sentiment classification dataset**

- load_data function
- get_word_index function

**Reuters newswire classification dataset**

- load_data function
- get_word_index function

Datasets

- Text Classification
  - AG_NEWS
  - AmazonReviewFull
  - AmazonReviewPolarity
  - CoLA
  - DBpedia
  - IMDb
  - MNLI
  - MRPC
  - QNLI
  - QQP
  - RTE
  - SogouNews
  - SST2
  - STSB
  - WNLI
  - YahooAnswers
  - YelpReviewFull
  - YelpReviewPolarity

`fetch_20newsgroups(*, data_home, subset, ...])` Load the filenames and data from the 20 newsgroups dataset (classification).

Table 1: Task Overview

| Name | Type | Format | Eval. Metric | # Class | (\|Train\|, \|Dev\|, \|Test\|) | Source | Style |
|---|---|---|---|---|---|---|---|
| KLUE-TC (YNAT) | Topic Classification | Single Sentence Classification | Macro F1 | 7 | 45k, 9k, 9k | News (Headline) | Formal |
| KLUE-STS | Semantic Textual Similarity | Sentence Pair Regression | Pearson's $r$, F1 | [0, 5] 2 | 11k, 0.5k, 1k | News, Review, Query | Colloquial, Formal |
| KLUE-NLI | Natural Language Inference | Sentence Pair Classification | Accuracy | 3 | 25k, 3k, 3k | News, Wikipedia, Review | Colloquial, Formal |
| KLUE-NER | Named Entity Recognition | Sequence Tagging | Entity-level Macro F1 Character-level Macro F1 | 6, 12 | 21k, 5k, 5k | News, Review | Colloquial, Formal |
| KLUE-RE | Relation Extraction | Single Sentence Classification (+2 Entity Spans) | Micro F1 (without *no_relation*), AUPRC | 30 | 32k, 8k, 8k | Wikipedia, News | Formal |
| KLUE-DP | Dependency Parsing | Sequence Tagging (+ POS Tags) | Unlabeled Attachment Score, Labeled Attachment Score | # Words, 38 | 10k, 2k, 2.5k | News, Review | Colloquial, Formal |
| KLUE-MRC | Machine Reading Comprehension | Span Prediction | Exact Match, ROUGE-W (LCCS-based F1) | 2 | 12k, 8k, 9k | Wikipedia, News | Formal |
| KLUE-DST (WoS) | Dialogue State Tracking | Slot-Value Prediction | Joint Goal Accuracy Slot Micro F1 | (45) | 8k, 1k, 1k | Task Oriented Dialogue | Colloquial |

Text classification

- ag_news_subset
- bool_q
- imdb_reviews
- natural_instructions
- paws_wiki
- paws_x_wiki
- sentiment140
- trec

# Future Work

5. 기타 성능 향상을 위한 preprocessing 고려, 한글 지원 등

Table 1: Task Overview

| Name | Type | Format | Eval. Metric | # Class | {\|Train\|, \|Dev\|, \|Test\|} | Source | Style |
|---|---|---|---|---|---|---|---|
| KLUE-TC (YNAT) | Topic Classification | Single Sentence Classification | Macro F1 | 7 | 45k, 9k, 9k | News (Headline) | Formal |
| KLUE-STS | Semantic Textual Similarity | Sentence Pair Regression | Pearson's $r$, F1 | [0, 5] 2 | 11k, 0.5k, 1k | News, Review, Query | Colloquial, Formal |
| KLUE-NLI | Natural Language Inference | Sentence Pair Classification | Accuracy | 3 | 25k, 3k, 3k | News, Wikipedia, Review | Colloquial, Formal |
| KLUE-NER | Named Entity Recognition | Sequence Tagging | Entity-level Macro F1 Character-level Macro F1 | 6, 12 | 21k, 5k, 5k | News, Review | Colloquial, Formal |
| KLUE-RE | Relation Extraction | Single Sentence Classification (+2 Entity Spans) | Micro F1 (without *no_relation*), AUPRC | 30 | 32k, 8k, 8k | Wikipedia, News | Formal |
| KLUE-DP | Dependency Parsing | Sequence Tagging (+ POS Tags) | Unlabeled Attachment Score, Labeled Attachment Score | # Words, 38 | 10k, 2k, 2.5k | News, Review | Colloquial, Formal |
| KLUE-MRC | Machine Reading Comprehension | Span Prediction | Exact Match, ROUGE-W (LCCS-based F1) | 2 | 12k, 8k, 9k | Wikipedia, News | Formal |
| KLUE-DST (WoS) | Dialogue State Tracking | Slot-Value Prediction | Joint Goal Accuracy Slot Micro F1 | (45) | 8k, 1k, 1k | Task Oriented Dialogue | Colloquial |