20기 MINI PROJECT (1) - NLP

# 청와대 청원 분류와 주제 분석


4조 김기수, 박진영, 서민진, 이은효

# 목차

## Contents


01 Introduction

1) 주제 선정 동기
2) 데이터셋 소개

02 EDA

1) 전처리
2) Topic Modeling

03 청와대 청원 분류

1) LightGBM
2) LSTM
3) GRU
4) KoBERT

04 군집화

1) 군집화
2) K-means

05 Conclusion

01

# Introduction

1) 주제 선정 동기

2) 데이터셋 소개

# 01 주제 선정 동기

01

NLP/추천시스템 희망

02

짧은 기간

03

제공된 데이터셋

주어진 시간 내에 자연어 처리의 전체 과정을 다루는 것을 목표로 삼음

# 02 데이터셋 소개

## 데이콘 – 청와대 청원 분류 경진대회

| index | category | data |
|---|---|---|
| 0 | 2 | 신혼부부위한 주택정책 보다 보육시설 늘려주세요.. 국민세금으로 |
| 1 | 0 | 학교이름에 '남자'도 붙여주세요. 울산여자중학교에 재학중인 학생 |
| 2 | 1 | 빙상연맹, 대한축구협회등 각종 체육협회의 비리를 철저하게 밝혀 |
| 3 | 1 | 티비 12세,15세 관람가도 연령확인 의무화 하자.. 제기 에전에 티비 |
| 4 | 1 | 무더운 여름철엔 남성들도 시원한 자율복장을 해야. 무더운 여름철 |
| 5 | 1 | 일간베스트 사이트 폐쇠를 청원합니다. 국익에 전혀 도움이 되지 |
| 6 | 0 | '초중고학교 페미니즘교육의무화'중복동의인원 무효처리해 주세요 |
| 7 | 2 | 수시 발표도 다 7일 미뤄주세요.. 수능 전 발표를 피하기 위해 수능 |

청원 주제/범주

청원 내용

0 : 인권/성평등
1 : 문화/예술/체육/언론
2 : 육아/교육

02

# EDA

1) 전처리

2) 토픽 모델링

# 01 전처리

## 1) 결측치, 특수문자 제거

[Train data 결측치]

```
np.sum(train.isnull())

index       0
category    0
data        8
dtype: int64
```

제거

[Test data 결측치]

```
np.sum(test.isnull())

index    0
data     0
dtype: int64
```

```
train['data'] = train['data'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
train['data'] = train['data'].str.replace(r'[-=+,#/₩?:^$.@*₩"※~&%・!」₩₩ '|₩(₩)₩[₩]₩<₩>`₩'…》₩₩n₩t]+', " ", regex=True)
train['data'] = train['data'].str.replace(r'₩t+', " ", regex=True)
train['data'] = train['data'].str.replace(r'[₩₩n]+'," ", regex=True)


test['data'] = test['data'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
test['data'] = test['data'].str.replace(r'[-=+,#/₩?:^$.@*₩"※~&%・!」₩₩ '|₩(₩)₩[₩]₩<₩>`₩'…》 ]', " ", regex=True)
test['data'] = test['data'].str.replace(r'₩t+', " ", regex=True)
test['data'] = test['data'].str.replace(r'[₩₩n]+'," ", regex=True)
```

# 01 전처리

## 2) 짧은 문장(필요 없는 문장) 제거

```
## 가장 짧은 문장은 삭제(길이=1)
pop_index = list(np.where(train['data'].apply(len)==1)[0])
train.drop(pop_index,inplace=True)
train.reset_index(inplace=True,drop=True)

## 그 다음으로 짧은 문장 삭제(길이=3)
pop_index = list(np.where(train['data'].apply(len)==3)[0])
train.drop(pop_index,inplace=True)
train.reset_index(inplace=True,drop=True)
```

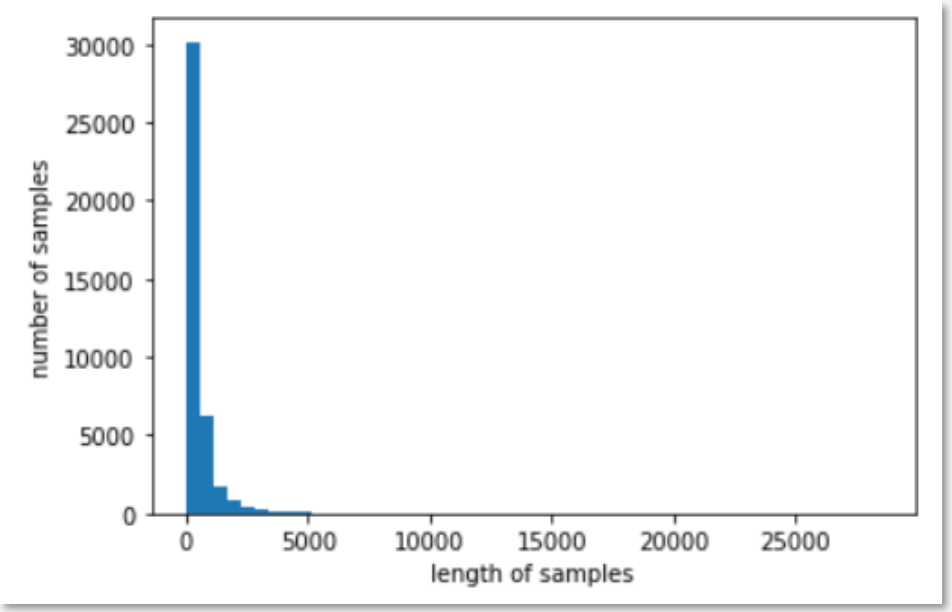
```
min_list = []

for line in train['data']:
  if len(line) == min(map(len, train['data'])):
    min_list.append(line)
```

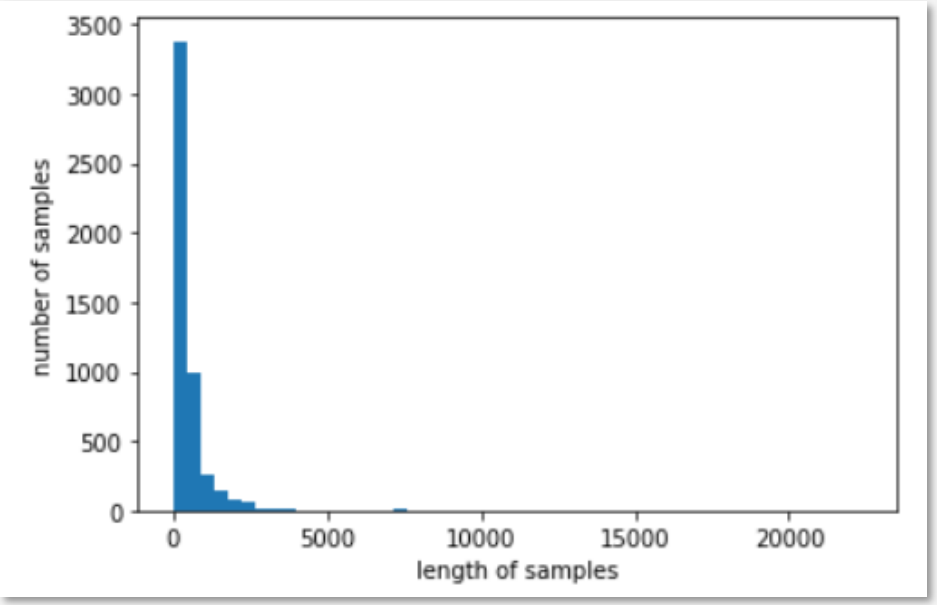
```
min_list
```

```
[' ', ' ', ' ', ' ', ' ']
```

# 01 전처리

## 3) 청원 내용의 길이의 분포 탐색




### Train data

① 최소 길이 : 5
② 최대 길이 : 28,481
③ 평균 길이 : 499.71

### Test data

① 최소 길이 : 8
② 최대 길이 : 22,394
③ 평균 길이 : 502.22

# 01 전처리

## 4) 워드클라우드

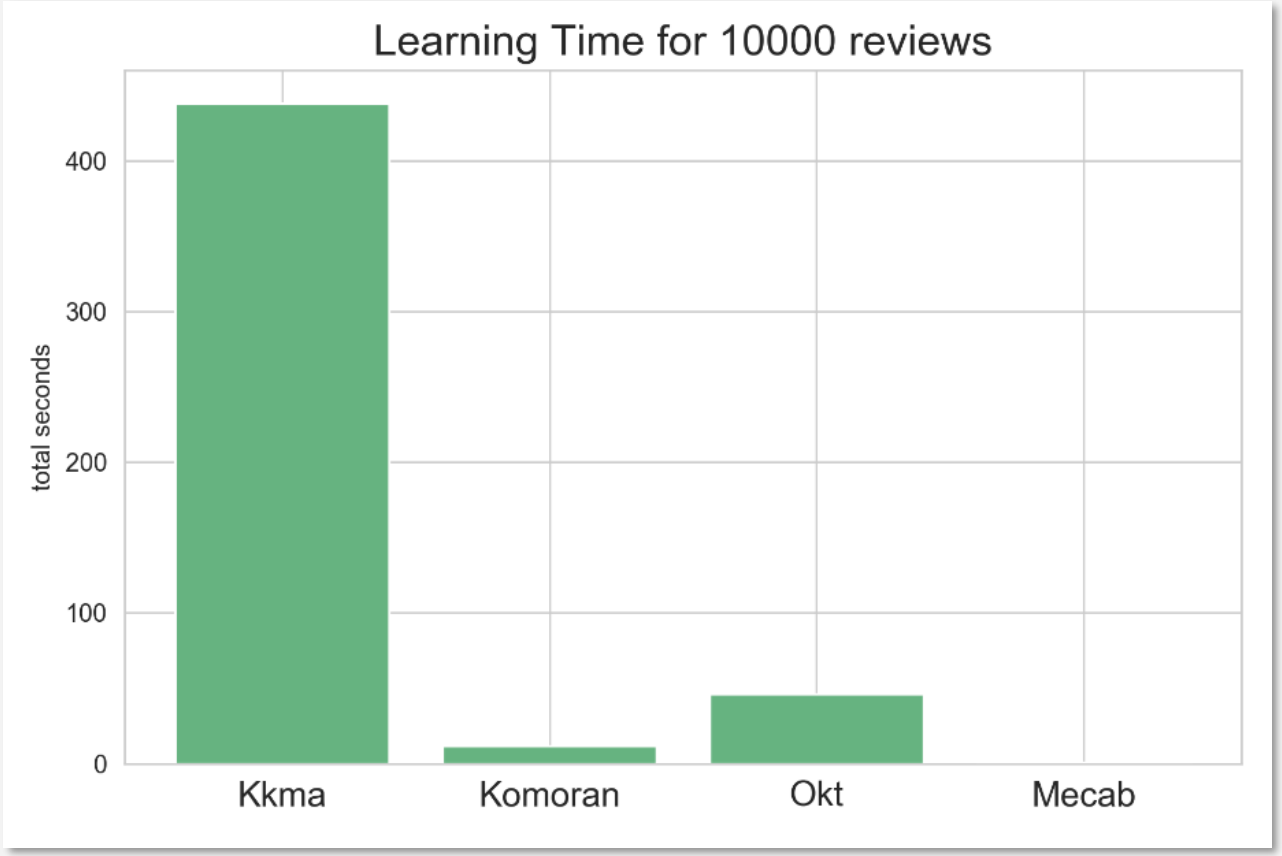
```
train2 = train[train['category']==2]
train1 = train[train['category']==1]
train0 = train[train['category']==0]

train2.reset_index(inplace=True,drop=True)
train1.reset_index(inplace=True,drop=True)
train0.reset_index(inplace=True,drop=True)


train0_word = []

for document in train0['data']:
    mecab = Mecab()
    word = mecab.nouns(document)
    train0_word.append(word)

train0_noun = []

for i in range(len(train0_word)):
  train0_noun += train0_word[i]

train0_word = train0_noun
```

① 카테고리별로 나눔(총 3개)

② mecab.nouns()를 이용해 명사인 단어만 추출

# 01 전처리

## [형태소 분석기 소요시간 비교]



# 01 전처리

## 4) 워드클라우드

```
c0 = Counter(train0_word)
c0.most_common(100)
```

```
[('것', 22735),
 ('수', 15302),
 ('여성', 13762),
 ('사람', 10931),
 ('일', 9692),
 ('생각', 9273),
 ('말', 8143),
 ('저', 8032),
 ('국민', 7891),
 ('법', 7883),
 ('년', 7529),
 ('처벌', 6974),
 ('사건', 6830),
 ('등', 6057),
 ('인권', 6056),
 ('남성', 6052),
 ('피해자', 5923),
 ('나라', 5890),
```

③ most_common()를 이용해 가장 많이 나온 단어 목록 파악

④ 불용어 제거 ex. '것,' '등'

```
remove_set = {'하','있','것','수','그','이','때','거','가','때문','왜','들','또','다'}
train0_word = [i for i in train0_word if i not in remove_set]
```

# 01 전처리

## 4) 워드클라우드 결과


category = 0

인권, 성평등


category = 1

문화, 예술, 체육, 언론


category = 2

육아, 교육

**각 카테고리를 대표하는 단어들이 추출됨**

# 02 Topic Modeling

## 1) LDA 토픽 모델링이란?

토픽 모델링은 문서의 집합에서 토픽을 찾아내는 통계적 모델로, 대표적으로 잠재 디리클레 할당(Latent Dirichlet Allocation, LDA)이 존재
LDA는 문서들은 토픽들의 혼합으로 구성되어져 있으며, 토픽들은 확률 분포에 기반하여 단어들을 생성한다고 가정

문서1 : 저는 사과랑 바나나를 먹어요
문서2 : 우리는 귀여운 강아지가 좋아요
문서3 : 저의 깜찍하고 귀여운 강아지가 바나나를 먹어요

LDA는 **각 문서의 토픽 분포**와 **각 토픽 내의 단어 분포**를 추정

**[각 문서의 토픽 분포]**
① 문서1 : 토픽 A 100%
② 문서2 : 토픽 B 100%
③ 문서3 : 토픽 B 60%, 토픽 A 40%

**[각 토픽의 단어 분포]**
① 토픽A : **사과 20%, 바나나 40%, 먹어요 40%**, 귀여운 0%, 강아지 0%, 깜찍하고 0%, 좋아요 0%
② 토픽B : 사과 0%, 바나나 0%, 먹어요 0%, **귀여운 33%, 강아지 33%, 깜찍하고 16%, 좋아요 16%**

두 토픽이 각각 과일에 대한 토픽과
강아지에 대한 토픽이라는 판단 가능

# 02 Topic Modeling

## 2) LDA 토픽 모델 학습

※ 빈도가 2 이상인 단어와 전체의 50% 이상 차지하는 단어는 필터링

| | data |
|---|---|
| 0 | [신혼, 부부, 주택, 정책, 보육, 시설, 국민, 세금, 일부, 정책, ㅂ |
| 1 | [학교, 이름, 남자, 울산, 여자, 중학교, 재학, 중, 학생, 최근, ㅇ |
| 2 | [빙상, 연맹, 축구, 협회, 등, 각종, 체육, 협회, 비리, 철저, 최근 |
| 3 | [티비, 세세, 관람, 연령, 확인, 의무, 전, 티비, 일, 동생, 리모컨 |
| 4 | [여름철, 남성, 자율, 복장, 여름철, 남성, 넥타이, 반바지, 등, ㅂ |

불용어 제거

| | data |
|---|---|
| 0 | [신혼, 부부, 주택, 정책, 보육, 시설, 국민, 세금, 일부, 정책, ㅂ |
| 1 | [학교, 이름, 남자, 울산, 여자, 중학교, 재학, 학생, 최근, 양성, |
| 2 | [빙상, 연맹, 축구, 협회, 각종, 체육, 협회, 비리, 철저, 최근, ㅇ |
| 3 | [티비, 세세, 관람, 연령, 확인, 의무, 티비, 일, 동생, 리모컨, ㅇ |
| 4 | [여름철, 남성, 자율, 복장, 여름철, 남성, 넥타이, 반바지, 복장, |

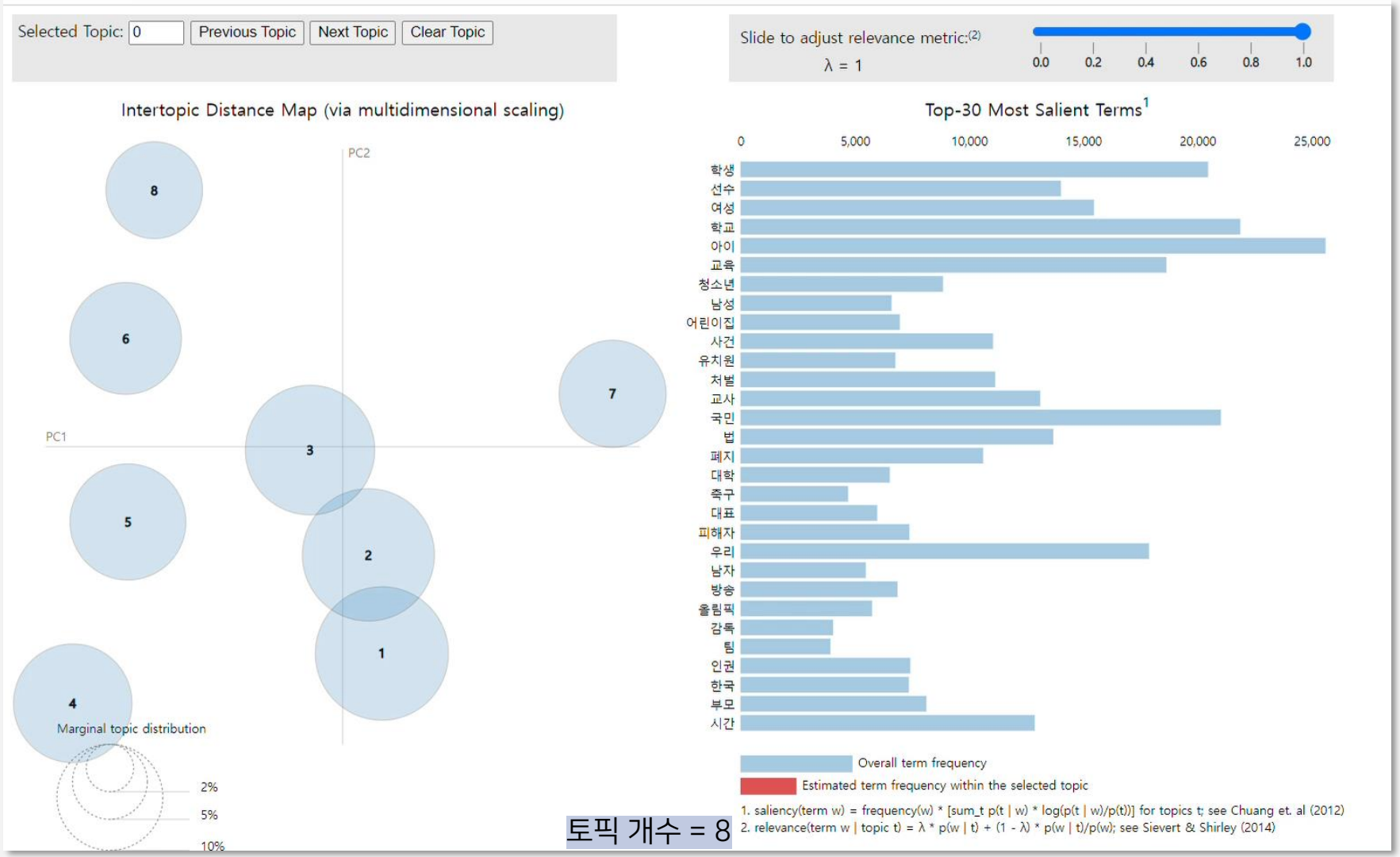
Mecab 이용해 명사만 추출

```
(0, '0.041*"학생" + 0.040*"학교" + 0.032*"교육" + 0.014*"생각"')
(1, '0.041*"여성" + 0.017*"남성" + 0.014*"남자" + 0.013*"사회"')
(2, '0.026*"국민" + 0.022*"우리" + 0.017*"나라" + 0.015*"대통령"')
(3, '0.053*"선수" + 0.018*"대표" + 0.018*"축구" + 0.015*"감독"')
(4, '0.013*"방송" + 0.009*"장애" + 0.008*"일" + 0.007*"공무원"')
(5, '0.009*"조사" + 0.008*"기사" + 0.008*"언론" + 0.008*"내용"')
(6, '0.022*"청소년" + 0.021*"사람" + 0.021*"법" + 0.018*"처벌"')
(7, '0.037*"아이" + 0.017*"일" + 0.016*"시간" + 0.015*"교사"')
```

8개의 토픽으로 구분이 되며, 각 단어 앞에 붙은 수치는 단어의 해당 토픽에 대한 기여도를 보여줌

# 02 Topic Modeling

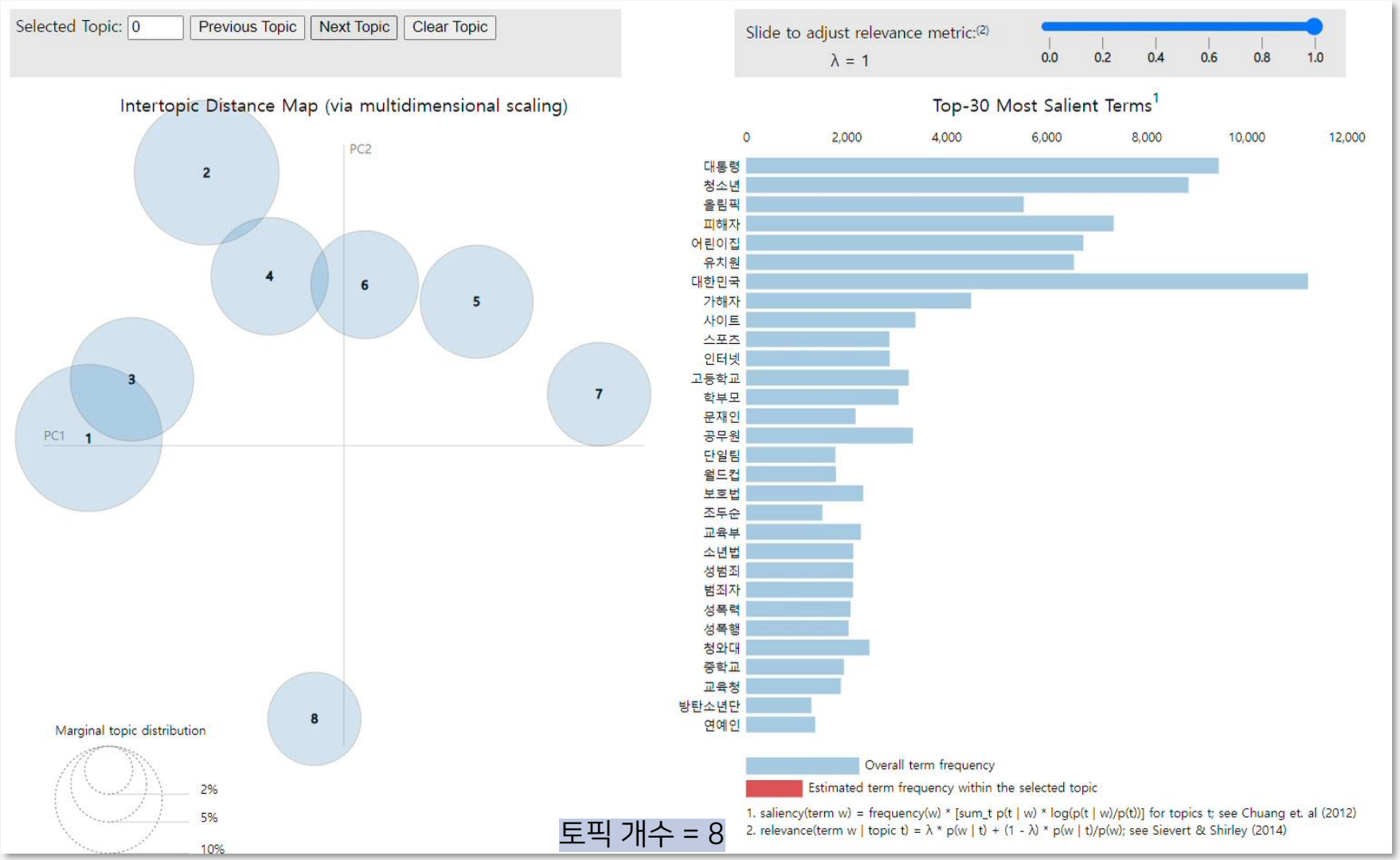
## 3) LDA 토픽 모델링 시각화 ① 명사




토픽 개수 = 8

# 02 Topic Modeling

## 3) LDA 토픽 모델링 시각화 ② 명사, 동사, 형용사 위주




토픽 개수 = 8

03

# 청와대 청원 분류

1) LightGBM

2) LSTM

3) GRU

4) KoBERT

# 01 TFIDF - LightGBM

## 1) TF-IDF(Term Frequency-Inverse Document Frequency)

① TF-IDF는 단어의 빈도(TF)와 역 문서 빈도(IDF)를 사용하여 DTM(Document Term Matrix) 내의 각 단어들마다 중요한 정도를 가중치로 두는 방법

② 단어의 빈도(TF), 역 문서 빈도(IDF), TF-IDF 값 구하는 방법

→ TF(d, t) : 특정 문서 d에 등장한 특정 단어 t의 개수

→ DF(d, t) : 특정 단어 t가 등장한 문서의 개수

→ IDF(d, t) : DF(d, t)에 반비례하는 수 (n : 총 문서의 수)

→ TF-IDF : TF(d, t) * IDF(d, t)

$$idf(d,t) = log(\frac{n}{1+df(t)})$$

# 01 TFIDF - LightGBM

## 2) GBM

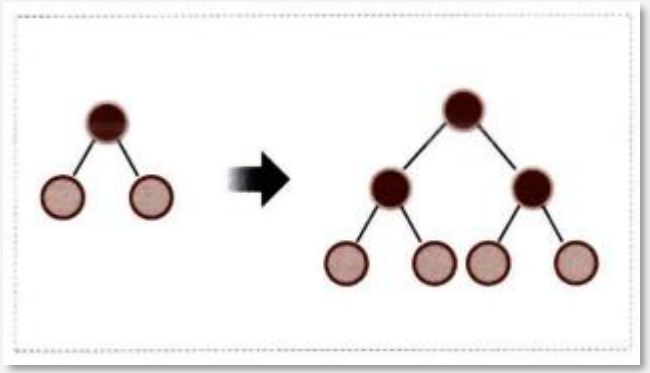
GBM은 tree를 기반으로 하는 학습 알고리즘으로 쉽게 말해 틀린 부분에 가중치를 더하면서 진행함

## 3) LightGBM

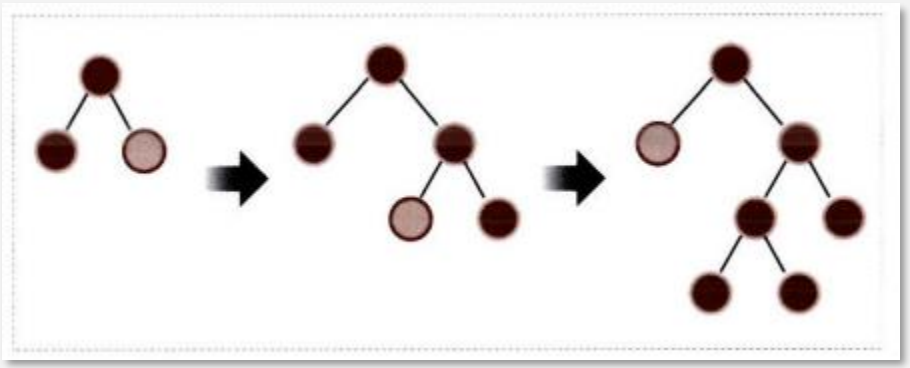
① XGBoost보다 학습에 걸리는 시간이 훨씬 적으면서 메모리 사용량도 적은 점이 특징

② 일반 GBM 계열의 (균형) 트리 분할 방법과 달리 **리프 중심 트리 분할 방식**을 사용

→ 리프 중심 트리 분할 방식은 최대 손실값을 가지는 리프노드를 지속적으로 분할해 학습을 반복할수록 **예측 오류 손실을 최소화**할 수 있음



균형 트리 분할 방법



리프 중심 트리 분할 방법

# 01 TFIDF - LightGBM

## 4) 실행 결과

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score

X_train, X_test, y_train, y_test = train_test_split(train['data'], train['category'],
                                                    test_size = 0.2, stratify = train['category'],
                                                    random_state = 42)

tfidfv = TfidfVectorizer(ngram_range = (1, 2))
tfidfv.fit(X_train)
X_train_transform = tfidfv.transform(X_train)
X_test_transform = tfidfv.transform(X_test)
```

← train, test data 분리

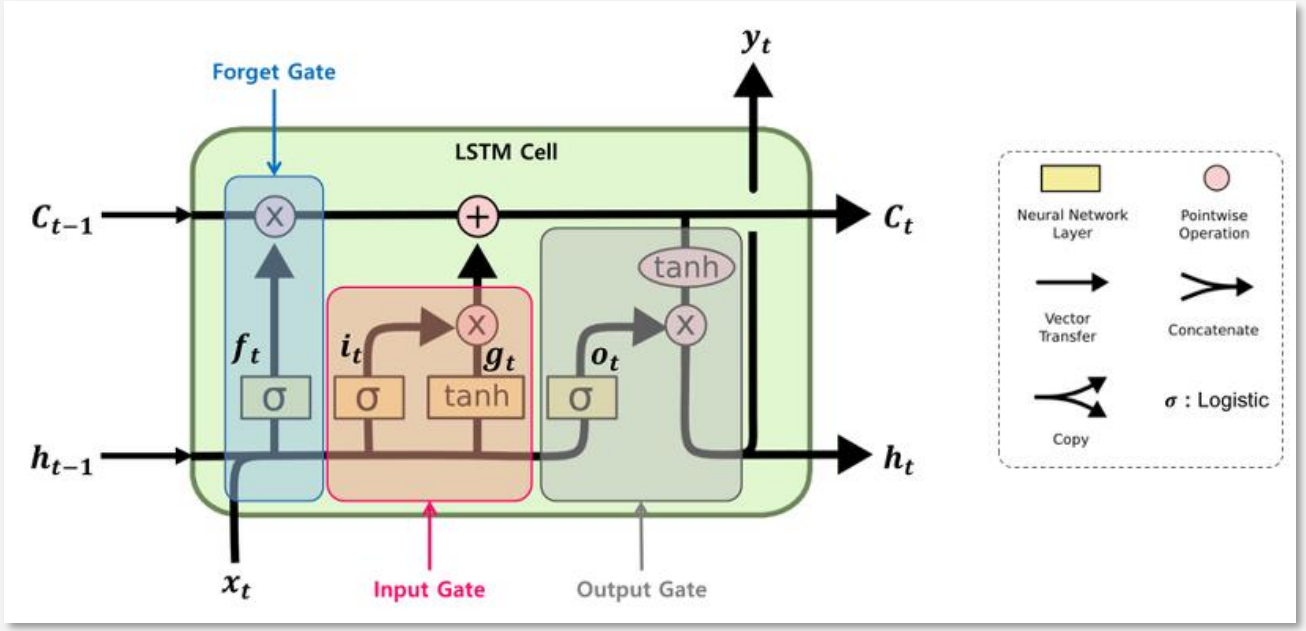
```
import lightgbm as lgb

lgb_clf = lgb.LGBMClassifier(random_state = 42)
lgb_clf.fit(X_train_transform, y_train)
lgb_pred = lgb_clf.predict(X_test_transform)

accuracy_score(y_test, lgb_pred), f1_score(y_test, lgb_pred, average = 'macro')
```

정확도 : 80.35%, f1 score : 80.17%

# 02 Tokenizer - LSTM

## 1) Model

LSTM은 Long Short-Term Memory의 약어로, 기울기 소실 문제를 해결하기 위해 고안된 신경망 모델




① Forget Gate Layer : cell state의 기존 정보를 얼마나 잊어버릴지를 결정하는 gate
② Input Gate Layer : 새롭게 만들어진 cell state를 기존 cell state에 얼마나 반영할지 를 결정하는 gate
③ Output Gate Layer : 새롭게 만들어진 cell state를 새로운 hidden state에 얼마나 반영할지를 결정하는 gate

# 02 Tokenizer - LSTM

## 2) 실행 결과 - Epoch = 2, 15

① 하이퍼파라미터인 임베딩 벡터의 차원은 120, 은닉 상태의 크기는 120으로 설정

```
embedding_dim = 120
hidden_units = 120

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(LSTM(hidden_units))
model.add(Dense(3, activation='softmax')) ← 다중클래스 분류

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

② model.fit() 을 이용해서 train accuracy 확인

```
history = model.fit(X_train, y_train, batch_size=128, epochs=2)
```

```
Epoch 1/2
313/313 [==============================] - 66s 187ms/step - loss: 0.6577 - acc: 0.7381
Epoch 2/2
313/313 [==============================] - 39s 123ms/step - loss: 0.3893 - acc: 0.8680
```

③ model.predict() 을 이용해서 test accuracy 확인

```
y_pred = model.predict(X_test)
```

위 과정을 epoch이 15일 때도 반복하여 accuracy 확인

Epoch = 2

| (단위 : %) | Train accuracy | Test accuracy |
|---|---|---|
| Embedding=200, hidden=120 | 86.80 | 85.02 |
| Embedding=120, hidden=120 | 89.72 | 84.14 |

# 02 Tokenizer - LSTM

## 3) 실행 결과 정리

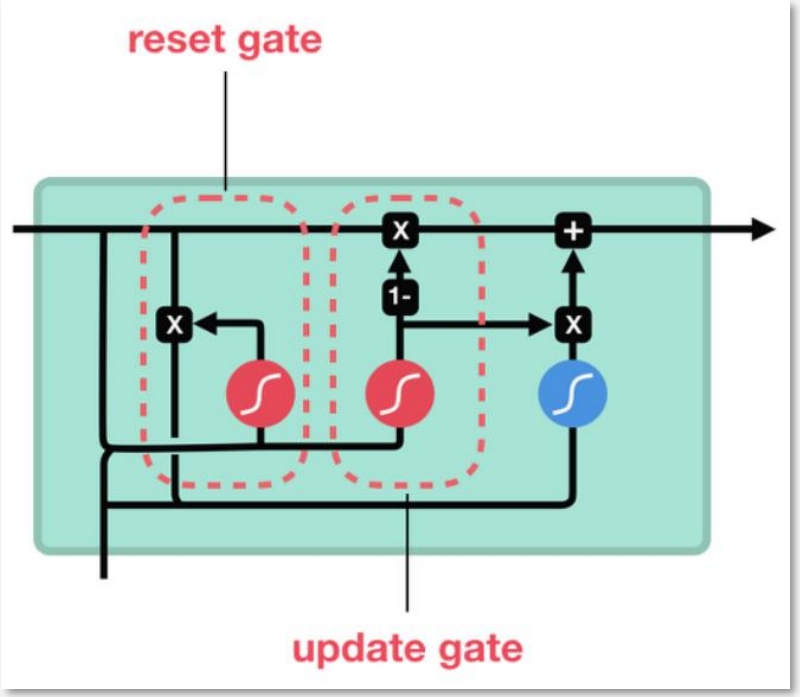
| Epoch | Embedding dim | Hidden units | Train accuracy | Test accuracy |
|---|---|---|---|---|
| 2 | 120 | 120 | 86.80% | 85.02% |
| | 200 | 120 | 89.72% | 84.14% |
| 15 | 120 | 120 | 98.10% | 92.64% |
| | 200 | 120 | 97.96% | 90.42% |

Epoch이 2에서 15로 증가 시 모델의 정확도 높아짐

# 03 Tokenizer - GRU

## 1) GRU(Gated Recurrent Unit)

LSTM의 장기 의존성 문제에 대한 해결책을 유지 + 은닉 상태를 업데이트하는 계산을 줄임
⇒ LSTM과 유사한 성능이지만, 구조를 더 간단히 만듦




**Reset Gate**

- 이전 상태의 hidden state와 현재 상태의 x를 받아 sigmoid 처리
- 이전 hidden state의 값을 얼마나 활용할 것인지에 대한 정보 제공

**Update Gate**

- 이전 상태의 hidden state와 현재 상태의 x를 받아 sigmoid 처리
- LSTM의 forget gate, input gate와 비슷한 역할을 하며 이전 정보와 현재 정보를 각각 얼마나 반영할 것인지에 대한 비율을 구하는 것이 핵심
  → Update Gate의 계산 한 번으로 LSTM의 forget gate + input gate의 역할을 대신함
- 최종 결과는 다음 상태의 hidden state로 보내짐

# 03 Tokenizer - GRU

## 2) 실행 결과 - Epoch = 2, 15

① 하이퍼파라미터인 임베딩 벡터의 차원은 120, 은닉 상태의 크기는 120으로 설정

```
embedding_dim = 120
hidden_units = 120
```

```
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(GRU(hidden_units))
model.add(Dense(3, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])

history = model.fit(X_train, y_train, batch_size=128, epochs=2)
```

② model.fit() 을 이용해서 train accuracy 확인

```
history = model.fit(X_train, y_train, batch_size=128, epochs=2)
```

```
Epoch 1/2
313/313 [==============================] - 45s 137ms/step - loss: 0.5438 - acc: 0.7705
Epoch 2/2
313/313 [==============================] - 28s 88ms/step - loss: 0.2611 - acc: 0.9021
```

③ model.predict() 을 이용해서 test accuracy 확인

```
y_pred = model.predict(X_test)
```

위 과정을 epoch이 15일 때도 반복하여 accuracy 확인

Epoch = 2

| (단위 : %) | Train accuracy | Test accuracy |
|---|---|---|
| Embedding=200, hidden=120 | 90.21 | 70.34 |
| Embedding=120, hidden=120 | 90.93 | 77.80 |

# 03 Tokenizer - GRU

## 3) 실행 결과 정리

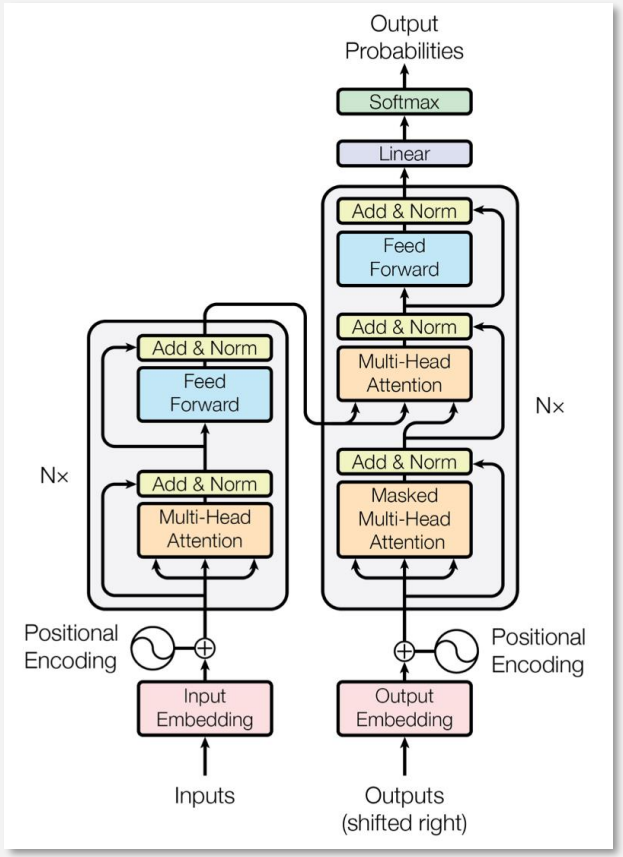
| Epoch | Embedding dim | Hidden units | Train accuracy | Test accuracy |
|---|---|---|---|---|
| 2 | 120 | 120 | 90.21% | 70.34% |
| | 200 | 120 | 90.93% | 77.80% |
| 15 | 120 | 120 | 98.83% | 80.72% |
| | 200 | 120 | 98.96% | 81.38% |

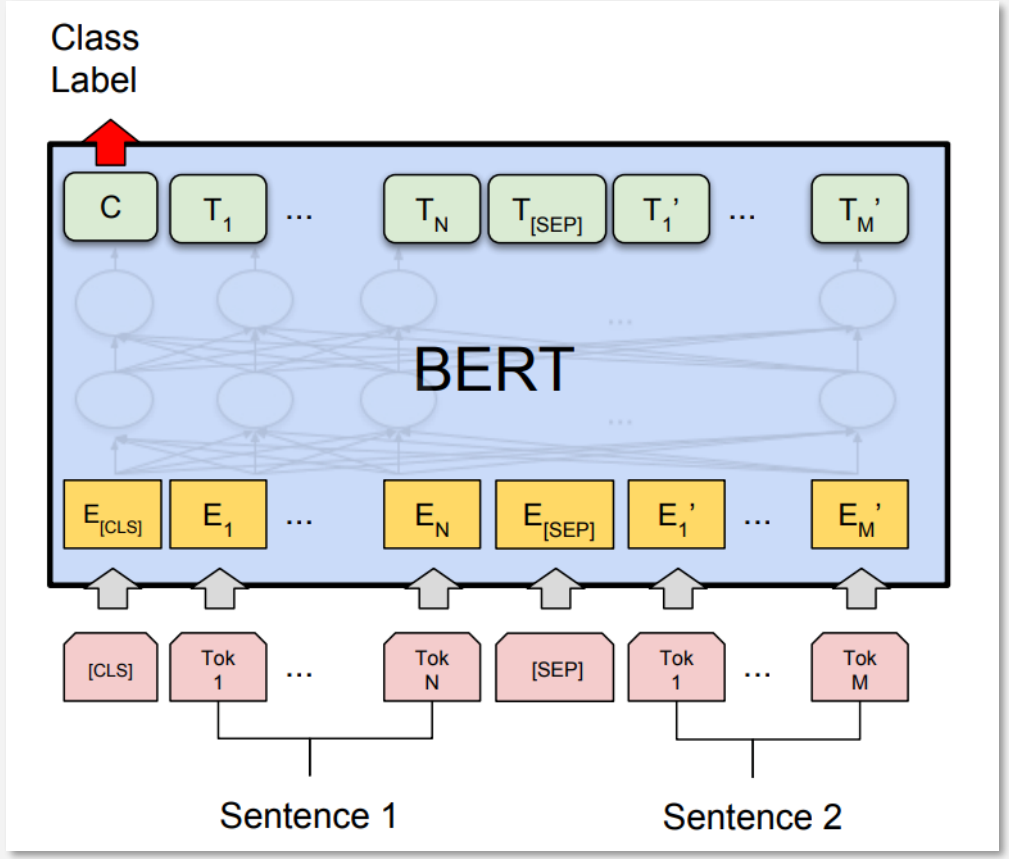
Epoch이 2에서 15로 증가 시 모델의 정확도 높아짐

# 04 KoBERT

## 1) model

국립국어원과 SKT가 협업하여 개발한 한국어 언어 모델로, 기존 BERT 모델을 한국어에 더 적합하게끔 개발



# 04 KoBERT

## 2) 실행 결과

```
#kobert
from kobert.utils import get_tokenizer
from kobert.pytorch_kobert import get_pytorch_kobert_model

#GPU 사용
device = torch.device("cuda:0") ← 인덱스가 0인 GPU를 지정

#BERT 모델, Vocabulary 불러오기 필수
bertmodel, vocab = get_pytorch_kobert_model()
```

### ① KoBERT 모델에 사용될 데이터셋 정리

```
class BERTDataset(Dataset):
    def __init__(self, dataset, sent_idx, label_idx, bert_tokenizer, max_len,
                 pad, pair):
        transform = nlp.data.BERTSentenceTransform(
            bert_tokenizer, max_seq_length=max_len, pad=pad, pair=pair)

        self.sentences = [transform([i[sent_idx]]) for i in dataset]
        self.labels = [np.int32(i[label_idx]) for i in dataset]

    def __getitem__(self, i):
        return (self.sentences[i] + (self.labels[i], ))

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

[모델 학습 시 설정해야 하는 파라미터]

```
max_len = 64
batch_size = 120
warmup_ratio = 0.1
num_epochs = 2
max_grad_norm = 1
log_interval = 200
learning_rate =  1e-5
```

**num_epochs 수치를 바꿔주며 진행 및 비교**

※ Batch_size : 모델 학습 중 파라미터를 업데이트할 때 사용할 데이터 개수
※ Num_epochs : 전체 데이터셋을 학습한 횟수

# 04 KoBERT

## ② KoBERT 모델 정의

```
class BERTClassifier(nn.Module): ## 클래스를 상속
    def __init__(self,
                 bert,
                 hidden_size = 768,
                 num_classes=3,   ##클래스 수 조정##
                 dr_rate=None,
                 params=None):
        super(BERTClassifier, self).__init__()
        self.bert = bert
        self.dr_rate = dr_rate

        self.classifier = nn.Linear(hidden_size , num_classes)
        if dr_rate:
            self.dropout = nn.Dropout(p=dr_rate)

    def gen_attention_mask(self, token_ids, valid_length):
        attention_mask = torch.zeros_like(token_ids)
        for i, v in enumerate(valid_length):
            attention_mask[i][:v] = 1
        return attention_mask.float()

    def forward(self, token_ids, valid_length, segment_ids):
        attention_mask = self.gen_attention_mask(token_ids, valid_length)

        _, pooler = self.bert(input_ids = token_ids, token_type_ids = segment_ids.long(), attention_mask = attention_mask.float().to(token_ids.device))
        if self.dr_rate:
            out = self.dropout(pooler)
        return self.classifier(out)
```

# 04 KoBERT

③ 토큰화 진행

```
tokenizer = get_tokenizer()
tok = nlp.data.BERTSPTokenizer(tokenizer, vocab, lower=False)
```

④ train data, test data 분리

```
from sklearn.model_selection import train_test_split

dataset_train, dataset_test = train_test_split(data_list, test_size = 0.2, random_state = 2022, shuffle = True)
data_train = BERTDataset(dataset_train, 0, 1, tok, max_len, True, False)
data_test = BERTDataset(dataset_test, 0, 1, tok, max_len, True, False)

train_dataloader = torch.utils.data.DataLoader(data_train, batch_size=batch_size)
test_dataloader = torch.utils.data.DataLoader(data_test, batch_size=batch_size)
```

⑤ model 불러오기

```
model = BERTClassifier(bertmodel, dr_rate = 0.5).to(device)
```

# 04 KoBERT

## ⑥ optimizer와 scheduler 설정

```
no_decay = ['bias', 'LayerNorm.weight']
optimizer_grouped_parameters = [
    {'params': [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)], 'weight_decay': 0.01},
    {'params': [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)], 'weight_decay': 0.0}
]

optimizer = AdamW(optimizer_grouped_parameters, lr=learning_rate)
loss_fn = nn.CrossEntropyLoss()

t_total = len(train_dataloader) * num_epochs
warmup_step = int(t_total * warmup_ratio)

scheduler = get_cosine_schedule_with_warmup(optimizer, num_warmup_steps=warmup_step, num_training_steps=t_total)
```

## ⑦ 정확도 측정을 위한 함수 정의

```
def calc_accuracy(X,Y):
    max_vals, max_indices = torch.max(X, 1)
    train_acc = (max_indices == Y).sum().data.cpu().numpy()/max_indices.size()[0]
    return train_acc

def predict_label_num(X):
    max_vals, max_indices = torch.max(X, 1)
    return max_vals
```

**※ Adam W**
: 가중치 증가 제한을 고려한 Adam 알고리즘을 적용할 수 있는 옵티마이저

**※ get_cosine_schedule_with_warmup**
: 0과 초기 lr 사이에서 선형적으로 증가하는 워밍업 기간 후에 옵티마이저에서 0으로 설정된 초기 lr 사이의 코사인 함수 값에 따라 감소하는 lr로 스케줄을 생성

# 04 KoBERT

## ⑧ 모델 학습

```
max_acc = 0.0
max_train_acc = 0.0
models = []
optimizers = []

for e in range(num_epochs):
    train_acc = 0.0
    test_acc = 0.0

    model.train()
    for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm(train_dataloader)):
        optimizer.zero_grad()
        token_ids = token_ids.long().to(device)
        segment_ids = segment_ids.long().to(device)
        valid_length= valid_length
        label = label.long().to(device)
        out = model(token_ids, valid_length, segment_ids)
        loss = loss_fn(out, label)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
        optimizer.step()
        scheduler.step()  # Update learning rate schedule
        train_acc += calc_accuracy(out, label)
        if batch_id % log_interval == batch_id:
            print("epoch {} batch id {} loss {} train acc {}".format(e+1, batch_id+1, loss.data.cpu().numpy(), train_acc / (batch_id+1)))

    print("epoch {} train acc {}".format(e+1, train_acc / (batch_id+1)))

    model.eval()
    for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm(test_dataloader)):
        token_ids = token_ids.long().to(device)
        segment_ids = segment_ids.long().to(device)
        valid_length= valid_length
        label = label.long().to(device)
        out = model(token_ids, valid_length, segment_ids)

        test_acc += calc_accuracy(out, label)
        predict = predict_label_num(out)

        max_vals, max_indices = torch.max(out, 1)

    print("epoch {} test acc {}".format(e+1, test_acc / (batch_id+1)))

    models.append(model)
    optimizers.append(optimizer)
```

KoBERT 모델에서 학습시킬 수 있도록 입력 데이터셋을 처리 후
파라미터를 모두 지정하였으므로 다음과 같이 모델 학습

# 04 KoBERT

## 3) 실행 결과 정리

| # of epochs | Batch size | Train accuracy | Test accuracy |
|---|---|---|---|
| 2 | 32 | 89.32% | 85.02% |
| | 64 | 88.35% | 84.14% |
| 10 | 32 | 97.58% | 87.27% |
| 15 | 32 | 98.33% | 86.32% |

04

# 군집화

1) 군집화

2) K-means

# 01 군집화

## 1) 군집화(clustering)

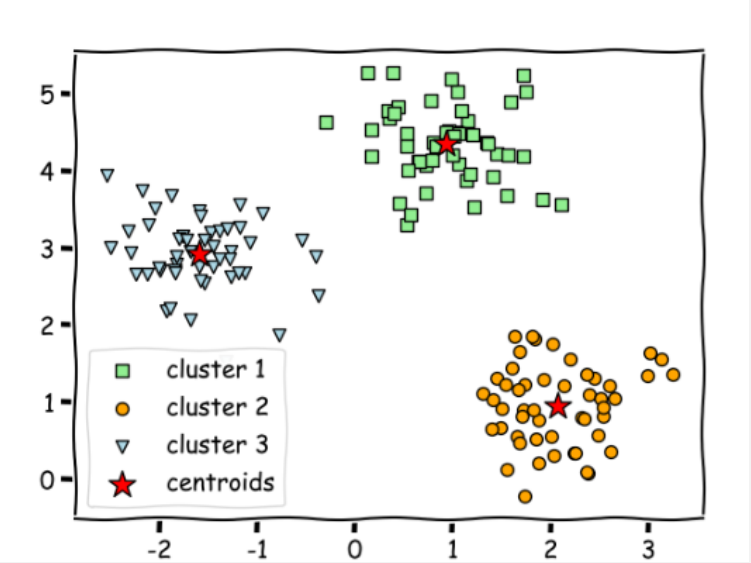
비슷한 샘플을 구별해 하나의 클러스터 또는 비슷한 샘플의 그룹으로 할당하는 비지도학습의 기술로, 종류로는 K-Means, Mean Shift, Gaussian Mixture Model 등이 있음

## 2) K-Means

임의의 지점을 선택해 중심점에 가장 가까운 포인트들을 선택하여 K개의 군집으로 묶음

**각 카테고리안에서 같은 주제끼리 묶을 수 있을까?**

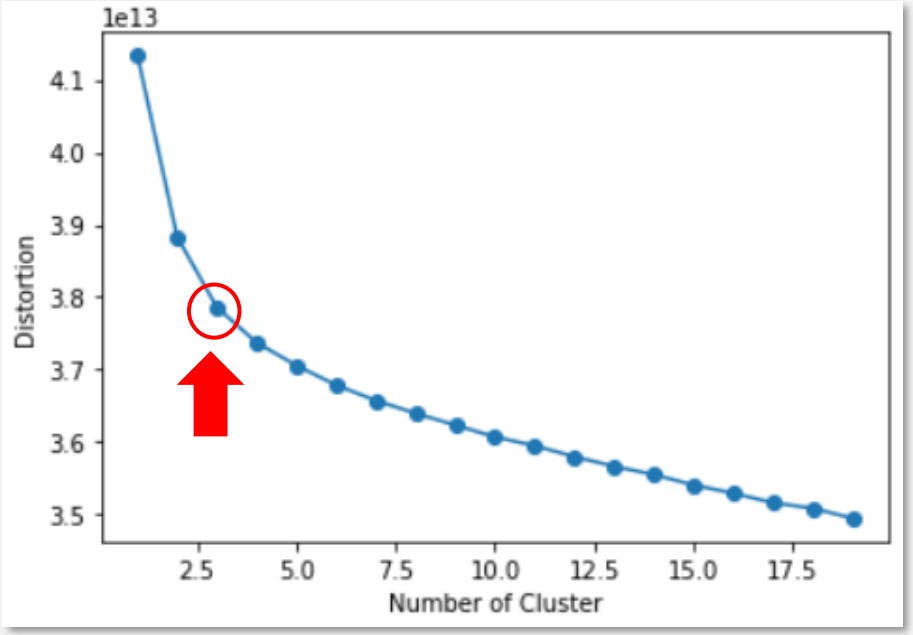
0 : 인권/성평등
1 : 문화/예술/체육/언론
2 : 육아/교육



# 02 K-means

## 3) K-means 결과

Scree plot을 활용하여 적절한 클러스터의 개수를 3개로 설정




```
from sklearn.cluster import KMeans

def visualize_elbowmethod(data, param_init='random', param_n_init=10, param_max_iter=300):
    distortions = []
    for i in range(1, 20):
        km = KMeans(n_clusters=i, init=param_init, n_init=param_n_init, max_iter=param_max_iter, random_state=0)
        km.fit(data)
        distortions.append(km.inertia_)

    plt.plot(range(1, 20), distortions, marker='o')
    plt.xlabel('Number of Cluster')
    plt.ylabel('Distortion')
    plt.show()

visualize_elbowmethod(pad_X, param_init = 'k-means++', param_n_init = 20, param_max_iter = 300)
```

# 02 K-means

## 3) K-means 결과 – 실루엣 계수(Silhouette Index)

- 실루엣 계수 : 각 데이터별로 그 데이터가 속한 군 내의 (거리 기반) 유사도와 인접한 군의 유사도를 비교하는 지표
- **-1에서 1사이의 값**을 가지며 1로 가까울수록 근처의 군집과 더 멀리 떨어져 있다, 0에 가까울수록 근처의 군집과 가까움

→ 즉, **1에 가까울수록 군집화가 잘 되었음을 의미**

$$S(i) = \begin{cases} \dfrac{b(i) - a(i)}{\max\{a(i), b(i)\}}, & if\ |C_I| > 1 \\ 0\ , & if\ |C_I| = 1 \end{cases}$$

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, j \neq i} d(i,j)$$

$$b(i) = min_{J \neq I} \frac{1}{|C_J| - 1} \sum_{j \in C_J} d(i,j)$$

$C_I$ : 클러스터 $C_I$에 속하는 개수
$d(i,j)$ : $C_I$에 속하는 i,j번째 데이터 간의 거리
$a(i)$ : 자신이 속하는 군의 모든 데이터와의 평균 거리
$b(i)$ : 와는 다른 인접한 클러스터와의 유사도 측정


클러스터가 3일 때 실루엣 계수가 0.206으로 좋지 않은 결과가 나옴

# 02 K-means

## 3) K-means 결과 - 워드클라우드


category = 0

인권, 성평등


category = 1

문화, 예술, 체육, 언론


category = 2

육아, 교육

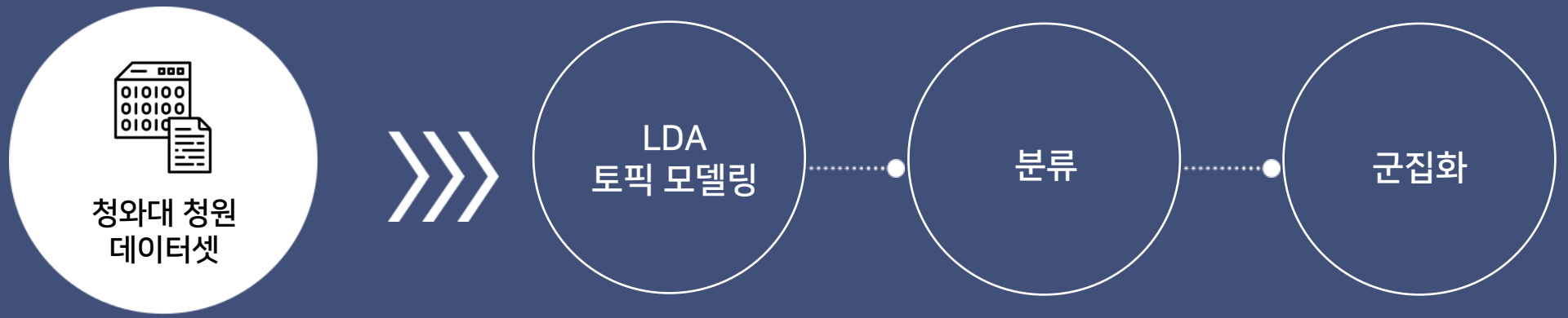
① '사람', '생각'이라는 단어가 반복 등장

② 세심한 전처리가 부족된 것으로 추측

05

# Conclusion

# Conclusion




| (단위 : %) | Lightgbm | LSTM | GRU | KoBERT |
|---|---|---|---|---|
| Train accuracy | 80.35 | 97.96 | 98.96 | 97.58 |
| Test accuracy | 80.52 | 90.42 | 81.38 | 86.318 |

※ accuracy는 가장 높은 수치로 기록

# 참고문헌

[1] 딥러닝을 이용한 자연어 처리 입문, https://wikidocs.net/book/2155

[2] 핸즈온 머신러닝 2판

[3] 파이썬 머신러닝 가이드

[3] https://bangu4.tistory.com/98

[4] https://zephyrus1111.tistory.com/193

[5] https://ariz1623.tistory.com/224

[6] https://3months.tistory.com/66

[7] https://wooono.tistory.com/242

[8] https://conanmoon.medium.com/

[9] https://wikidocs.net/44249

[10] https://pytorch.org/docs/stable/generated/torch.zeros_like.html

[11] https://pytorch.org/docs/stable/tensor_attributes.html

[12] https://www.youthdaily.co.kr/news/article.html?no=80345

[13] https://velog.io/@yuns_u/LSTMLong-Short-Term-Memory%EA%B3%BC-GRUgated-Recurrent-Unit

[14] https://soohee410.github.io/compare_tagger

# 감사합니다

## 청와대 청원 분류와 주제 파악 분석

4조 | 김기수, 박진영, 서민진, 이은효