**Machine Learning (Autumn 2023)**

**Homework 8th week-2 (November 4)**

**Student ID 2022712151**

**Name 정지원**

**Instruction: This report is a LightGBM-based Legal Document Classification Model for Claims/Facts/Judgments/Conclusions**

**(1)** 목표

1. AIHub에서 텍스트 데이터 수집을 진행한다.
2. 데이터에 대한 탐색적 분석을 수행하고 활용 목적에 알맞는 전처리를 수행한다.
3. TF-IDF를 활용해 텍스트 데이터를 임베딩한다.
4. 임베딩된 데이터로 LightGBM모델을 학습시켜 판결문의 주장/사실/판단/결론에 대한 분류를 수행한다.­

**(2)** 데이터 소개 및 수집

이 데이터셋은 Aihub의 법률/규정 (판결서, 약관등) 텍스트 분석 데이터로 1만 건 이상의 판결문을 대상으로 기초사실, 주장 등을 가공한 데이터와 판례 내용을 기반으로 판결문 분석 데이터 구축, 1만건 이상의 약관의 유/불리 조항 판단. 위법성과 유리 판단 이유 태길 및 레이블링을 통해 소비자 입장에서의 유/불리 확인을 위한 법률 텍스트 분석 데이터셋이다. AiHub의 (<https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100&aihubDataSe=realm&dataSetSn=580>) 사이트를 통해 다운로드를 받았다.[그림 1]

텍스트, 폰트, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[그림 1] 데이터 다운로드

이번 과제의 목적은 위 데이터 중 판결문 데이터(TL\_1. 판결문)를 가지고 주장, 사실, 판단, 결론이라는 4가지의 레이블을 분류하는 과정을 거친다.

(3) EDA 및데이터 전처리

**(3.1)** 라이브러리 import 및 초기설정

* 시각화에 사용할 ml-things을 설치한다.
* 사용하고자 하는 분석에 필요한 라이브러리들을 불러온다.

**Code:**

|  |
| --- |
| import pandas as pd  import numpy as np  import re  import io  import os  from tqdm.notebook import tqdm  tqdm.pandas()  from torch.utils.data import Dataset, DataLoader  from ml\_things import plot\_dict, plot\_confusion\_matrix, fix\_text  from sklearn.metrics import classification\_report, accuracy\_score  # GPU 메모리 및 최대 시퀀스 길이에 따른 배치 수 설정이 필요  # 시퀀스 길이가 짧다면 32 이상의 배치 사이즈를 사용하는 것이 가능  batch\_size = 32  data\_path ='./라벨링데이터/TL\_1.판결문'  # 분류 라벨  labels\_ids = {'01\_주장':0, '02\_사실':1, '03\_판단':2, '04\_결론':3}  # How many labels are we using in training.  # This is used to decide size of classification head.  n\_labels = len(labels\_ids) |

**(3.2)** EDA 수행 및 데이터 전처리

* 사용하고자 하는 데이터 샘플을 불러와 살펴본다.
* Json 확장자 파일에 9개의 key(info, concerned, org, disposal, mentionedltems, assrs, facts, dcss, close)와 각 key에 해당하는 내용들이 포함되어 있다.

**Code:**

|  |
| --- |
|  |

* 읽어온 JSON 객체에 특정 Key의 존재 여부를 확인하기 위한 is\_json\_key\_present 함수를 정의한다.
* 설정한 경로에서 데이터를 불러와 딕셔너리로 반환하는 JsonDataet 클래스를 정의한다.

**Code:**

|  |
| --- |
| import json  def is\_json\_key\_present(json, key):  try:  buf = json[key]  except KeyError:  return False  return True  class JsonDataset(Dataset):  def \_\_init\_\_(self, path):  # Check if path exists.  if not os.path.isdir(path):  # Raise error if path is invalid.  raise ValueError('Invalid `path` variable! Needs to be a directory')  self.texts = []  self.labels = []  for label in ['01.민사', '02.형사', '03.행정']:  sentiment\_path = os.path.join(path, label)  # Get all files from path.  for (root, directories, files) in os.walk(sentiment\_path):  # Go through each file and read its content.  for file\_name in tqdm(files, desc=f'{root} files'):  file\_path = os.path.join(root, file\_name)  with open(file\_path, "r") as json\_file:  json\_data = json.load(json\_file)  # print('json\_data',json\_data)  if(is\_json\_key\_present(json\_data,"assrs") and is\_json\_key\_present(json\_data['assrs'],"acusrAssrs")) :  for data in json\_data['assrs']['acusrAssrs'] :  self.texts.append(fix\_text(data))  self.labels.append('01\_주장')  if(is\_json\_key\_present(json\_data,"assrs") and is\_json\_key\_present(json\_data['assrs'],"dedatAssrs")) :  for data in json\_data['assrs']['dedatAssrs'] :  self.texts.append(fix\_text(data))  self.labels.append('01\_주장')  if(is\_json\_key\_present(json\_data,"facts") and is\_json\_key\_present(json\_data['facts'],"bsisFacts")) :  for data in json\_data['facts']['bsisFacts'] :  self.texts.append(fix\_text(data))  self.labels.append('02\_사실')  if(is\_json\_key\_present(json\_data,"dcss") and is\_json\_key\_present(json\_data['dcss'],"courtDcss")) :  for data in json\_data['dcss']['courtDcss'] :  self.texts.append(fix\_text(data))  self.labels.append('03\_판단')  if(is\_json\_key\_present(json\_data,"close") and is\_json\_key\_present(json\_data['close'],"cnclsns")) :  for data in json\_data['close']['cnclsns'] :  self.texts.append(fix\_text(data))  self.labels.append('04\_결론')  # Number of exmaples.  self.n\_examples = len(self.labels)  return  def \_\_len\_\_(self):  return self.n\_examples  def \_\_getitem\_\_(self, item):  return {'text':self.texts[item],  'label':self.labels[item]} |

* Train, test 데이터 분리를 진행해준다.
  + 사전 정의한 Jsondataset 클래스로 데이터를 불러온다.
  + 불러온 데이터에 pd.DataFrame.from\_records를 적용해 데이터프레임으로 만든다.
  + Train\_test\_split()을 통해 데이터프레임을 9:1의 비율로 나눈다.
  + Label 별로 나누어져 있는 데이터프레임을 합쳐 훈련 데이터와 테스트 데이터를 만든다.

**Code :**

|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  # JSON Data  dataset = JsonDataset(path=data\_path)  df = pd.DataFrame.from\_records(dataset)  df.groupby('label').size()  df = df.sample(frac=0.05, random\_state=123)  df1 = df[df['label'].str.startswith('01')]  df2 = df[df['label'].str.startswith('02')]  df3 = df[df['label'].str.startswith('03')]  df4 = df[df['label'].str.startswith('04')]  # 전체 데이터에서 train, test 분리 (9 : 1)  df1\_train, df1\_test = train\_test\_split(df1, test\_size=0.1, random\_state=123)  df2\_train, df2\_test = train\_test\_split(df2, test\_size=0.1, random\_state=123)  df3\_train, df3\_test = train\_test\_split(df3, test\_size=0.1, random\_state=123)  df4\_train, df4\_test = train\_test\_split(df4, test\_size=0.1, random\_state=123)  df\_train = pd.concat([df1\_train,df2\_train,df3\_train,df4\_train])  df\_test = pd.concat([df1\_test,df2\_test,df3\_test,df4\_test]) |

* StopWords 처리
  + Konlpy 라이브러리에서 한국어 형태소 분석기인 Okt를 import한다.
  + 한국어에서 중요한 의미를 가지지 않는 단어들을 불용어 처리하기 위한 set을 만든다.
  + Okt와 stop\_words를 활용하여 텍스트 데이터를 전처리하는 함수를 정의한다.
  + 데이터에 전처리를 수행하고 새로운 column prepro\_data로 추가한다.
  + 전처리 과정에서 모든 문장이 삭제되어 공백이 되는 데이터가 존재할 수 있으므로 해당 데이터를 제거한다.
  + 데이터를 정렬하고 데이터를 shuffle한다.

**Code :**

|  |
| --- |
| # 한국어 형태소 분석기 Okt를 사용하기 위해 import  from konlpy.tag import Okt  # Okt의 객체 생성  okt = Okt()  # 한국어에서 의미가 거의 없는 단어들을 불용어 처리하기 위해 집합을 만듦  stop\_words = set(['은', '는', '이', '가', '아', '하', '들', '것', '의', '있', '되', '수', '보', '주', '등', '한', '에'])  # 전처리 함수  def preprocessing(okt, review):  # 한글이 아닌 단어는 지우고 " "으로 대체  review\_text = re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣ]", " ", review)  # 형태소 분석을 하며 stemming 기법을 사용  token\_li = okt.morphs(review\_text, stem=True)  # 이전 stop\_words에 포함되지 않는 단어만 clean\_review에 들어갈 수 있게 함  clean\_review = [token for token in token\_li if not token in stop\_words]  # 단어들을 ' '기준으로 이어 붙혀서 하나의 string 타입의 변수를 생성함  clean\_review = ' '.join(clean\_review)  # 전처리 완료된 문장을 return  return clean\_review.strip()  # preprocessing 함수를 적용  df\_train["prepro\_data"] = df\_train["text"].progress\_apply(lambda x: preprocessing(okt, x))  df\_test["prepro\_data"] = df\_test["text"].progress\_apply(lambda x: preprocessing(okt, x))  # 전처리 과정에서 모든 문장이 삭제될 가능성이 있으므로, 해당 데이터 제거 |

**(4)** 모델링 및 결과 확인

1. TFidfVectorizer를 통해 훈련, 테스트 데이터 embedding

**Code :**

|  |
| --- |
| # tfidf embedding  from sklearn.feature\_extraction.text import TfidfVectorizer  # TfidfVectorizer를 초기화하여 사용할 수 있다.  tfidf\_vectorizer = TfidfVectorizer(max\_features=5000)  # 훈련 데이터에 대해 fit\_transform을 수행하여 TF-IDF 행렬을 생성한다.  x\_train\_tfidf = tfidf\_vectorizer.fit\_transform(x\_train)  # 테스트 데이터에 대해 transform을 수행하여 TF-IDF 행렬을 생성한다.  x\_test\_tfidf = tfidf\_vectorizer.transform(x\_test) |

1. LightGBM 모델을 사용하며 GridSearch를 통해 5-fold cross validation 방식으로 모델을 학습시킨다.

**Code :**

|  |
| --- |
| # modeling  from lightgbm import LGBMClassifier  # LightGBM 분류기를 초기화한다.  lgb\_classifier = LGBMClassifier()  # 사용하고자 하는 파라미터 조합을 설정한다.  param\_grid = {  'n\_estimators': [50, 100, 200],  'learning\_rate': [0.01, 0.05, 0.1],  'num\_leaves': [31, 63, 127]  }  # GridSearchCV를 사용하여 5-fold cross validation을 수행하며 모델을 학습시킨다.  from sklearn.model\_selection import GridSearchCV  grid\_search = GridSearchCV(lgb\_classifier, param\_grid, cv=5, scoring='accuracy',n\_jobs=-1,verbose=2)  grid\_search.fit(x\_train\_tfidf, y\_train)  # 최적의 파라미터를 출력한다.  print("Best Parameters:", grid\_search.best\_params\_)  Best Parameters: {'learning\_rate': 0.05, 'n\_estimators': 200, 'num\_leaves': 31} |

3. 훈련된 모델을 적용하여 4진 분류(0:주장, 1:사실, 2:판단, 3:결론)를 수행하고 모델의 성능을 확인한다.

텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진

자동 생성된 설명

**Results :**

|  |
| --- |
| Classification Report:  precision recall f1-score support  01\_주장 0.66 0.43 0.52 177  02\_사실 0.76 0.75 0.75 327  03\_판단 0.78 0.87 0.82 591  04\_결론 0.95 0.93 0.94 111  accuracy 0.78 1206  macro avg 0.79 0.74 0.76 1206  weighted avg 0.77 0.78 0.77 1206 |