주제: 악성 댓글 자동 감지 및 카테고리별 분류 프로그램 개발

12223550 서보성 인공지능공학과 선택과제 (A 형)

서론

최근 온라인 커뮤니케이션의 확산과 함께 댓글, SNS 게시글 등 사용자 생성 콘텐츠(User-Generated Content)의 양이 방대해지고 있다. 이는 정보공유, 자유로운 소통 등을 가능하게 하지만, 동시에 악성 댓글과 같은 유해 콘텐츠가 사회적 문제로 대두되고 있다. 악성 댓글은 욕설, 차별, 혐오, 비하, 편향된 사고, 범죄 조장 등의 형태로 나타나며, 사회 전반의 건전한 소통 환경을 해치는 심각한 요인이다. 이에 따라, 악성 댓글을 자동 감지하고, 문장의 유형을 구체적으로 분류할 수 있는 지능형 필터링 시스템의 필요성이 증가하고 있다. 기존의 키워드 기반 필터링 방식은 표현 방식이 점점 교묘해짐에 따라 한계점이 있으며, 이를 극복하기 위해 본 프로젝트에서는 딥러닝 기반 자연어 처리기술(NLP)을 활용해 악성 댓글 자동감지 및 카테고리별 분류 프로그램을 개발하고자 한다. 본 프로젝트는 악성 댓글의 유무를 판단하는데 그치지 않고, 여성/가족, 남성 차별, 성소수자에 대한 편견, 인종 및 국적 비하, 노인 비하, 지역 비하, 종교 비하, 기타 혐오 표현, 욕설, 개인 지칭 여부, 그리고 악플의 유무까지 11가지의 레이블로 문장을 상세하게 분류해내는 것이 최종 목표이다. 이를 통해 다양한 플랫폼에서 악성 댓글로 부터 사용자를 보호하고, 보다 안전하고, 건강한 디지털 커뮤니케이션 환경 조성에 기여하는 것을 궁극적인 목표로 한다.

본론

본 프로젝트에서 사용하는 학습 데이터 셋은 'Smilegate AI'에서 공개한 한국어 혐오 표현 "UnSmile" 데이터 셋이다. 사전 학습된 모델을 fine-tuning하여 한국어 문장을 11개의 레이블로 분류하는 모델을 만들고자 한다.

항목	여성/가족	남성	성소수자	인종/국적	연령	지역	종교	기타혐오	악플/욕설	clean	개인지칭
Train	1,599	1,347	1,141	1,728	603	1,052	1,181	569	3,143	3,739	315
Validation	394	334	280	426	146	260	290	134	786	935	74
Total	1,993	1,681	1,421	2,154	749	1,312	1,471	703	3,932	4,674	389

모델 훈련 시 데이터 셋의 Train과 Validation을 모두 합치고, 이를 8:1:1 비율로 훈련, 검증, 테스트 세트로 나누어 사용하였다.

또한 모델의 성능을 확인하기 위해 악성 댓글 유무로 이진 분류 되어있는 공개 데이터 셋을 추가로 사용한다.

악성 댓글 O	악성 댓글 X
5,000개	5,000개

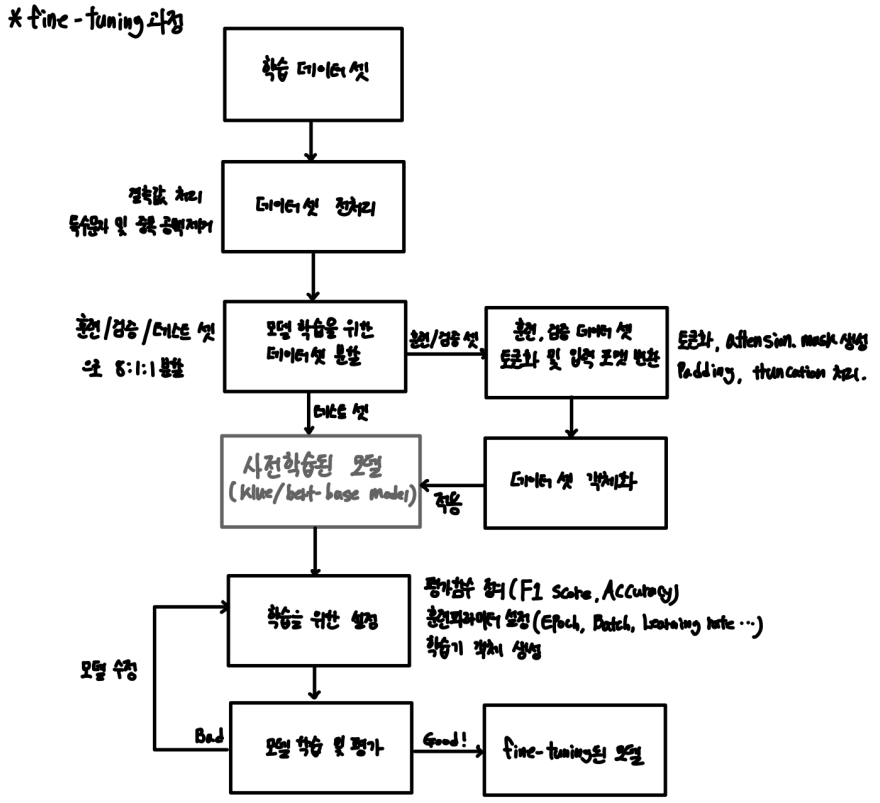
해당 데이터 셋을 사용해 최종적으로 모델이 이진분류되어있는 문장을 정확히 11가지 레이블로 분류하도록 하고, 시각화하는 것을 최종 목표로 본 프로젝트를 진행한다.

본 프로젝트에서 사용하는 모델은 **klue/bert-base**이다. 이는 한국어 자연어 처리(NLP) 문제를 해결하기 위해 사전 학습된(pre-training)된 BERT 기반의 모델이다.
BERT란 Google AI Language가 발표한 사전 학습 기반의 문맥 이해모델이다.
기존의 자연어 처리 모델은 단방향 정보(좌 -> 우 or 우 -> 좌)만을 문맥 이해를 위해 사용한 모델이었지만, BERT 모델은 양방향 문맥 정보를 이해하고, 이를 다양한 자연어 처리 task에 fine-tuning하여 적용 가능하도록 하는 범용 언어 표현 모델이다.
klue/bert-base model은 입력 문장의 일부를 MASK로 가리고, 이를 예측하거나 두 문장이 연속된 문장인지 예측하는 task를 통해 사전 학습된 모델이다.
간단히 말해, 한국어 문장의 문맥을 이해하고, 예측하도록 사전 설계된 모델이다.
이와 같은 사전 학습된 모델을 11가지 레이블을 가진 한국에 데이터 셋으로 학습시켜, fine-tuning할 것이다. 이 fine-tuning시킨 모델을 통해 레이블링 되어있지 않은 악플 및 선플 데이터 셋을 11가지 레이블로 분류 시키고, 분류 결과를 시각화 하여, 여러 유형의 악플들을 세분화할 것이다. 이를 통해 욕과 같은 키워드 기반의 필터링 시스템과 달리 다양한 유형의 악플들을 캐치해낼 것이다.

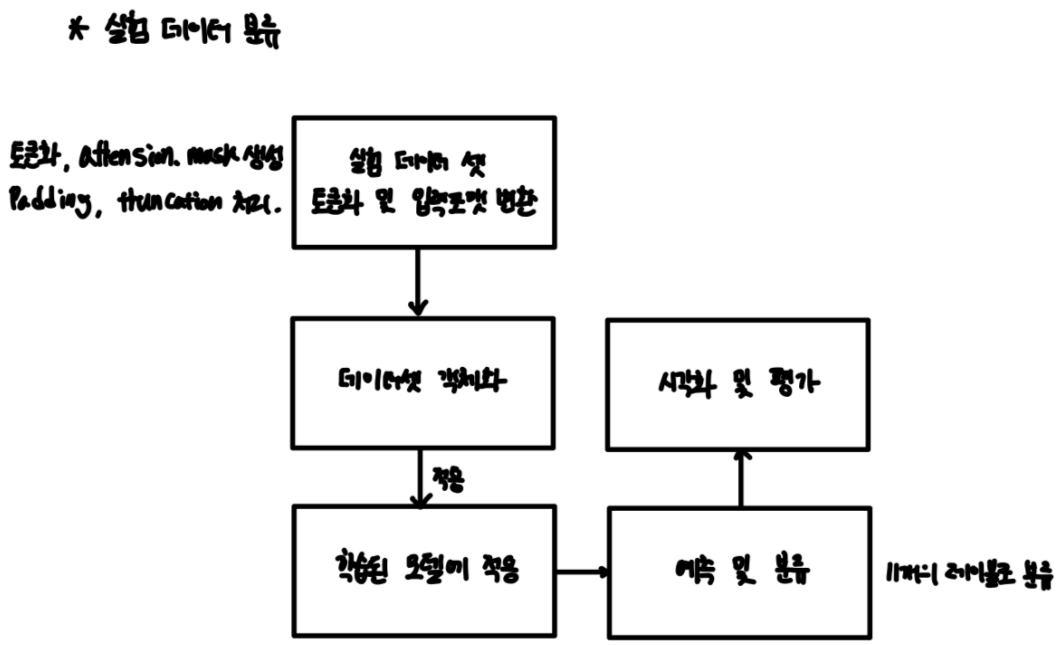
문제 해결 방법에 대한 알고리즘을 설명하도록 하겠다.
문제 해결은 크게 두 가지로 나뉜다. 1) fine-tuning 2) 실험 데이터 분류

먼저, 간단한 요약도/순서도를 보도록 하자.

1) fine-tuning 과정



2) 실험 데이터 분류



이제 해당 과정에 대한 중요 코드를 보며, 자세히 설명하도록 하겠다.

fine-tuning 과정의 순서도 부터 살펴보자.”
첫번째 과정은 **데이터 셋을 준비하고, 학습에 적용할 수 있도록 전처리하는 과정**이다. 학습 데이터로 사용하기 위해 수행해야 하는 전처리 과정에는, 결측 값 처리, 특수문자 제거, 중복 공백 제거의 과정이 필요하다. 학습 데이터로 사용할 때, 필요 없는 데이터이기 때문이다. 또한 한국어 데이터셋을 학습할 때 불용어를 처리하는 과정이 포함될 수 있다. 불용어란 ‘이었다’, ‘이와 같은’, ‘일지라도’, ‘하는 것도’ 등 단어 단위로 끊어 학습하는 자연어 처리 모델의 학습에 도움이 되지 않는 단어 및 문장을 의미한다. 보통 BERT 기반 모델에서는 양방향의 단어를 통해 문맥을 동적으로 이해하기 때문에 불용어를 처리하지 않는다. 본 프로젝트에서 또한 문맥을 이해하는 것이 문제 해결에 큰 도움이 되며, klue/bert-base model은 불용어 포함 문장으로 사전 학습된 모델이므로 불용어 처리과정을 제외하였다.

```
input_path = 'unsmile_train_v1.0.tsv'
df = pd.read_csv(input_path, sep='\t')

df.dropna(subset=['문장'], inplace=True)
```

위 과정은 데이터 셋의 원본 파일을 불러오고, 결측 값을 처리하는 코드이다.
결측 값(missing data)이란, 측정 되지 않은 값이나, 존재하지 않는 값을 말한다.(None)
subset=[‘문장’]은 데이터 셋의 ‘문장’이라는 이름을 가진 열(column)만을 살펴보겠다는

의미이다.
또한 inplace = True는 쉽게 말해 df라는 변수에 바로 수정된 값을 저장해 반영시키겠다는 의미이다.

```
def clean_text(text):
  text = re.sub(r'[^가-힣ㄱ-ㅎㅏ-ㅣa-zA-Z0-9\s]', '', text)
  text = re.sub(r'\s+', ' ', text).strip()
  return text

df['clean_text'] = df['문장'].apply(clean_text)
```

결측 값 처리를 제외한 나머지 전처리는 함수로 제작하였다.
먼저 re.sub(r'[^가-힣ㄱ-ㅎㅏ-ㅣa-zA-Z0-9₩s]', '', text)는 가~힣, ㄱ~ㅎ, ㅏ~ㅣ, a~z, A~Z, 0~9, 공백을 제외한 나머지 특수문자를 제외한 언어만 추출하는 과정이다.
re.sub(r'₩s+', ' ', text).strip()는 중복 공백을 처리하는 과정이다.
마지막 줄 코드는 '문장' 열에 있는 모든 문장에 앞에서 만든 전처리 함수를 적용하고, 이를 clean_text라는 새로운 열을 만들어, 추가로 삽입해준다는 의미이다.
이것으로 데이터 전처리 과정은 끝이 났다.

다음으로는 **모델 학습을 위한 데이터 셋 분할**을 진행하도록 해보자.

```
clean_df = df[['clean_text']]
clean_text_path = 'unsmile_clean_text.tsv'
clean_df.to_csv(clean_text_path, sep = '\n', index = False, header = False)
```

이는 clean_text라는 열에 새로 저장한 전처리된 문장들 만을 추출하여, 새로운 파일에 저장하는 과정이다.

```
label_columns =
['여성/가족', '남성', '성소수자', '인종/국적', '연령', '지역',
'종교', '기타혐오', '악플/욕설', 'clean', '개인지칭']

label_df = df[label_columns]

label_save_path = 'unsmile_labels.tsv'
label_df.to_csv(label_save_path, sep='\t', index=False)
```

해당 데이터 셋에 대한 11가지 레이블을 변수에 따로 저장하고, 이 변수를 이용해 데이터 셋에서 정답 데이터(레이블 값)만을 추출하는 과정이다.
데이터 셋을 살펴보면, 나열된 레이블 순서로 정답 데이터가 나열되어 있기 때문에, 각 레이블에 맞는 열의 데이터들을 추출하는 과정이다.
이후 추출한 데이터를 또 다른 새로운 파일에 저장한다.

```
from sklearn.model_selection import train_test_split

clean_text_path = "unsmile_clean_text.tsv"
X = pd.read_csv(clean_text_path, header=None)[0].tolist() # series -> list 변환

label_path = "unsmile_labels.tsv"
Y = pd.read_csv(label_path, sep='\t').values
```

이 과정은, 위에서 저장한 전처리된 문장(=INPUT)과 정답 레이블(=OUTPUT)을 각각 X, Y라는 변수에 저장하는 과정이다. 새롭게 저장한 파일에 각각 접근하여, 불러온다.

```
X_train, X_temp, Y_train, Y_temp = train_test_split(X, Y, test_size=0.2,
random_state=42)

X_valid, X_test, Y_valid, Y_test = train_test_split(X_temp, Y_temp, test_size=0.5,
random_state=42)
```

위 과정은 데이터 셋을 학습 데이터와 나머지 데이터 셋으로 8:2 비율 분할을 하는 과정과 나머지 데이터 셋을 검증 데이터 셋과 테스트 데이터 셋으로 1:1 비율 분할하는 과정이다. 기본적으로 본 프로젝트에서 사용하는 모델은 딥러닝 모델이며, 딥러닝은 머신러닝 분야에 속한다. 머신러닝에서는 보통 모델을 훈련시키는 학습 데이터와 모델의 성능을 검증하는 검증 데이터 셋을 나눠서 학습한다. 모델을 학습 데이터로 모델의 성능을 측정할 경우, 학습 데이터에 적응한 모델이므로 다른 데이터에서의 성능과는 차이를 보일 수 있다. 그렇기 때문에 검증 데이터를 사용한다. 또한 모델의 학습이 끝난 이후 모델을 평가할 때, 검증 데이터를 통해 모델을 평가할 경우, 이미 학습 과정에서 훈련 데이터와 검증 데이터에 맞게 훈련된 모델이므로 실제 모델의 성능과는 거리가 있을 수 있다. 그러므로 테스트 데이터 또한 따로 만들어주는 것이다. 본 프로젝트에서는 학습/검증/테스트 데이터를 8:1:1 비율로 나눠 사용하였다.

데이터를 분할하기 전에 비슷한 데이터끼리 뭉쳐 있는 경우를 방지하고자, 데이터를 임의로 섞는 과정이 포함된다. 이는 random_state 변수로 섞는 패턴을 고정할 수 있다. 하지만 이는 선택적 사항이다. Random_state 변수를 포함시켜주지 않아도 된다.

다음으로는 **사전 학습된 모델**을 불러오도록 하자.

```
from transformers import AutoModelForSequenceClassification, AutoTokenizer
model_name = "klue/bert-base"

tokenizer = AutoTokenizer.from_pretrained(model_name)

model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=len(label_columns),
    problem_type="multi_label_classification"
)
```

프로젝트에서 사용할 사전 학습 모델인 “klue/bert-base”를 지정해준다. (model_name)
자연어 처리 모델은 각 모델에 맞는 토큰화 함수를 가지고 있는데, 현재 사용하는 모델에 맞는 토큰화 함수를 호출해 저장한다. (tokenizer)
현재 사용할 모델을 정의한다. 모델 정의에는 모델의 이름, 해결하고자 하는 문제의 유형, 모델 출력층(Output Layer)의 노드 수(예측할 레이블 수)를 지정해준다.

본 프로젝트에서는 11가지의 레이블을 사용하고, 이를 통해 다중 레이블 분류를 하는 것이 목적이기에 맞게 설정해주었다.

다음으로는 **훈련 데이터 및 검증 데이터 토큰화 및 입력 포맷 변환 과정**이다.

```
X_train = [str(x) for x in X_train]
X_valid = [str(x) for x in X_valid]

train_encodings = tokenizer(X_train, truncation=True, padding=True,
max_length = 128)

valid_encodings = tokenizer(X_valid, truncation=True, padding=True,
max_length = 128)
```

X_train, X_valid는 일부 데이터의 값이 숫자(정수형, 실수형…)이거나 None 값일 수 있기 때문에 토큰화 과정에서 오류가 나지 않도록 전부 문자열로 통일 시켜주는 것이다. 오류를 방지하기 위한 안전 장치라고 생각하면 좋다.

다음으로는 X_train과 X_valid를 **토큰화** 하는 작업이다. 이는 텍스트와 같은 입력데이터를 모델이 이해할 수 있는 숫자 형태로 바꿔주는 encoding 과정이다.
이 작업은 문장을 작은 단어로 쪼개는 과정을 의미한다.
예를 들어, “이 영화 정말 재미 있어요”라는 문장이 있다고 하자.
토큰화는 이 문장을 [”이”, “영화”, “정말”, “재미”, “있어요”](단어 기반 토큰화), [”이”, “영”, “화”, “정”, “말”, “재”, “미”, “있”, “었”, “어”, “요”](음절 기반 토큰화)와 같이 쪼개는 과정이다. 또한 쪼갠 단어들을 숫자로 변환해 입력 포맷으로 변환 시켜주는 작업이 포함된다. 숫자 변환은 각 모델이 가지고 있는 단어 사전을 참고한다.
[“이”, “영화”, “정말”] => [101, 300, 1317]처럼 토큰을 숫자 ID로 변환하는 과정이다.

다음은 **padding**과정이다. Padding과정은 토큰화 과정에서 숫자로 변환된 여러 문장들의 길이를 맞추기 위한 과정이다. 빈공간이 남는 경우, 0을 채워 문장의 길이를 맞춘다.
[101, 300, 1317] => [101, 300, 1317, 0 ,0]
Transformer는 여러 문장을 encoding하여, 동시에 병렬 연산을 수행하는 모델이다.
(우리가 사용하는 klue/bert-base model이 transformer 모델이며, transformer 모델이란, encoder 구조를 가지고 있는 모델이라고만 알고 넘어가자.)
여러 문장들을 동시에 병렬 연산하기 위해서는 입력 길이가 일정해야 하므로 padding 과정을 거친다.

Truncation 과정은 문장이 너무 길 경우, 지정한 길이를 초과한 만큼 잘라내는 것을 말한다. **max_length** 변수가 최대 길이를 지정하는 변수이다.

다음으로는 **데이터 셋 객체를 생성하는 과정**이다.
데이터 셋을 직접 사용하지 않고, 데이터 셋 객체를 만들어 사용하는 이유를 설명하겠다. 우리는 딥러닝 모델을 만들고, 학습시키기 위한 전체 파이프라인을 제공하는 프레임워크인 PyTorch를 사용하게 된다. 모델을 학습 시킬 때, 전처리한 데이터를 일정 묶음 단위(batch)로 모델에 전달해주게 되는데, 이때 사용하는 것이 PyTorch의 DataLoader라는 함수이다. DataLoader에 맞게 데이터를 관리하고, 학습에 효율적으로 사용하기 위해 데이터 셋 객체를 사용한다.

데이터 셋 클래스를 정의해보자.

```
import torch

class ToxicDataset(torch.utils.data.Dataset):
  def __init__(self, encodings, labels=None):
    self.encodings = encodings
    if labels is not None:
      self.labels = torch.tensor(np.array(labels, dtype=np.float32))
    else:
      self.labels = None

  def __getitem__(self, idx):
    item = {
        'input_ids': torch.tensor(self.encodings['input_ids'][idx]),
        'attention_mask': torch.tensor(self.encodings['attention_mask'][idx])
    }
    if self.labels is not None:
      item['labels'] = self.labels[idx]
    return item

  def __len__(self):
    return len(self.encodings['input_ids'])
```

__init__: ToxicDataset 클래스에서 받는 인자는 encoding한 데이터와 레이블 정보이다. 실험 데이터 셋의 경우 레이블링 되어있지 않은 데이터이고, 이 데이터 또한 객체화가 필요하기 때문에, label 값이 입력되지 않아도 오류가 나지 않도록 설정하였다.

__getitem__: 데이터 셋의 실제 데이터에 접근할때, 다음과 같은 정보를 제공하며, 이는 PyTorch 모델에 바로 입력 가능한 형태이다.
i번째 데이터에 접근한 경우, i번째 데이터의 숫자 벡터 값과 attention_mask와 선택적으로 레이블을 반환한다.
attention_mask란 padding된 정보를 0, padding되지 않은 정보를 1로 표기하는 실제 단어와 padding을 구분하는 마스크이다.

__len__: 데이터 셋의 전체 샘플 개수를 반환하여 DataLoader가 몇 개의 데이터 묶음을 만들지 판단할 수 있게 한다.

다음으로는 **학습을 위한 설정** 과정이다.
먼저, **평가함수 정의**에 대해 살펴보자.

```python
from sklearn.metrics import f1_score, accuracy_score, classification_report

def compute_metrics(pred):
  logits, labels = pred
  if isinstance(logits, tuple):
    logits = logits[0]
  preds = (logits > 0.5).astype(int)
  macro_f1 = f1_score(labels, preds, average="macro")
  accuracy = accuracy_score(labels, preds)
  return {
      "macro_f1": macro_f1,
      'accuracy': accuracy}
```

본 프로젝트에서 사용하는 모델 평가함수는 accuracy와 macro F1 score이다.
해당 평가함수에 대해 설명하기 전에, 기본 개념부터 짚고 넘어가도록 하자.
TP: Positive라고 예측했을때, 정답(True)인 경우
TN: Negative라고 예측했을때, 정답(True)인 경우
FP: Positive라고 예측했을때, 오류(False)인 경우
FN: Negative라고 예측했을때, 오류(False)인 경우

Accuracy(정확도): (TP + TN) / (TP + TN + FP + FN)
Recall(재현율): TP / (TP + FN), 실제로 Positive인 샘플 중 정답을 맞춘 비율
Precision(정밀도): TP / (TP + FP): Positive라고 예측한 샘플 중 정답을 맞춘 비율
F1_score(조화평균): (2 x Precision x Recall) / (Precision x Recall)
Macro F1 score: 각 레이블마다의 F1 score 계산하고, 평균 낸 것.

$$\text{Macro-F1} = \frac{1}{N}\sum_{i=1}^{N}\text{F1}_i$$

멀티 분류 문제에서 Accuracy는 높은 신뢰도를 가지는 평가 지표는 아니다.
멀티 분류에서는 예측 결과가 정답과 완전히 일치하지 않으면, 샘플을 완전히 틀렸다고 처리하기 때문에 상대적으로 정확도가 낮게 측정될 수 있다.
쉽게 설명하면, 정답이 [1, 0, 1, 0]일 때, 예측이 [1, 0, 0, 0]이면, 부분정답을 인정하는 것이 아니라 이 샘플에 대한 정확도를 0이라고 매핑한다.
그러므로 macro F1 score 평가 지표를 더 집중적으로 보도록 하겠다.

다음은 훈련에 필요한 하이퍼파라미터들을 설정하는 과정을 보자.
(하이퍼파라미터는 학습 전 사용자가 직접 설정해주는 파라미터를 의미한다.

```python
from transformers import Trainer, TrainingArguments, EarlyStoppingCallback
import numpy as np

training_args = TrainingArguments(
    output_dir = "/content/drive/MyDrive/toxic_comment_project/outputs",
    num_train_epochs = 7,
    per_device_train_batch_size = 16,
    per_device_eval_batch_size = 16,
    learning_rate=1e-5,
    weight_decay=0.05,
    warmup_ratio=0.05,
    eval_strategy='epoch',
    save_strategy='epoch',
    logging_dir='/content/drive/MyDrive/toxic_comment_project/logs',
    logging_steps=50,
    load_best_model_at_end=True,
    metric_for_best_model='macro_f1',
)
```

output_dir: 학습 중 저장되는 모델의 파라미터를 저장할 폴더 경로이다.
num_train_epoch: 전체 데이터셋을 몇 번 반복해서 학습할지를 지정한다.
per_device_train_batch_size: 훈련 시 GPU 당 사용할 배치(batch) 크기를 지정한다.
-> batch란 훈련 시 데이터를 몇 개씩 묶어 처리할지를 의미한다.
per_device_eval_batch_size: 모델 평가 시 사용되는 배치(batch) 크기를 지정한다.
learning_rate: 학습 시 모델의 파라미터들을 얼마나 빠르게 업데이트할지 지정한다.
weight_decay: 학습 시 과적합 방지를 위한 regularization의 정도를 지정한다.
-> requlrization은 학습 시 훈련 데이터에 모델이 과도하게 fitting되는 것을 막기 위해 모델의 파라미터가 너무 커지는 것에 패널티를 부여하는 과정이다.
warmup_ratio: 학습 초기에 학습률을 천천히 증가 시켜 안정적 학습을 돕는 비율 지정
eval_strategy: 모델을 어떤 주기로 평가할지 지정한다.
save_strategy: 모델을 어떤 주기로 저장할지 지정한다.
logging_steps: 학습 중 몇 step마다 로그를 출력할지 설정한다.
load_best_model_at_end: 학습 종료 시 가장 성능이 좋은 모델이 최종 모델로 선택되게 하는 하이퍼파라미터
metric_for_best_model: 가장 좋은 모델을 판단할 때 사용하는 지표
(현재 설정된 하이퍼파라미터는 5번의 모델 학습 중 가장 좋은 성능을 보인 파라미터)

다음은 **학습기 객체를 정의**해보도록 하자..

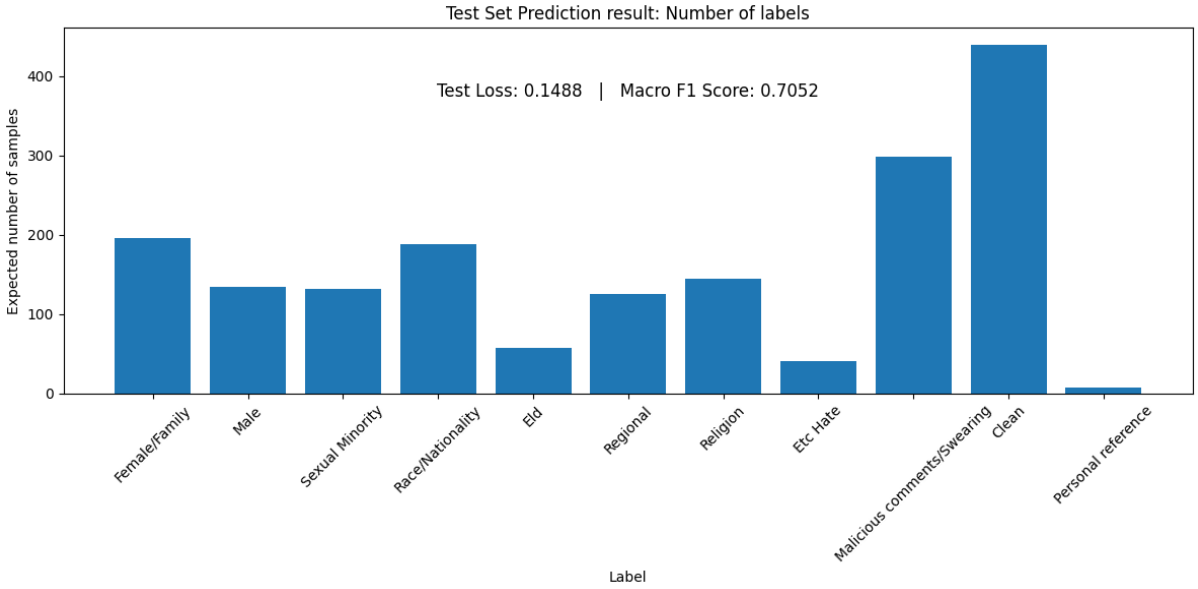
```
trainer = Trainer(
    model = model,
    args = training_args,
    train_dataset = ToxicDataset(train_encodings, Y_train),
    eval_dataset = ToxicDataset(valid_encodings, Y_valid),
    compute_metrics = compute_metrics,
    tokenizer = tokenizer,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2,
    early_stopping_threshold=0.002)]
)
```

model: 본 프로젝트에서는 “klue/bert-base” model 사용
args: 이전 과정에서 설정한 하이퍼파라미터 사용
train_dataset: 토큰화 및 입력 포맷 변환한 훈련 데이터셋 사용
eval_dataset: 토큰화 및 입력 포맷 변환한 검증 데이터셋 사용
compute_metrics: 이전 과정에서 정의한 평가 지표 사용
tokenizer: “klue/bert-base” model의 tokenizer 사용
call_back: 두 번의 epoch 동안 성능 개선이 없다면, 학습 조기종료, 평가지표 수치가 0.002 이상 상승하지 않는다면, 학습 조기종료

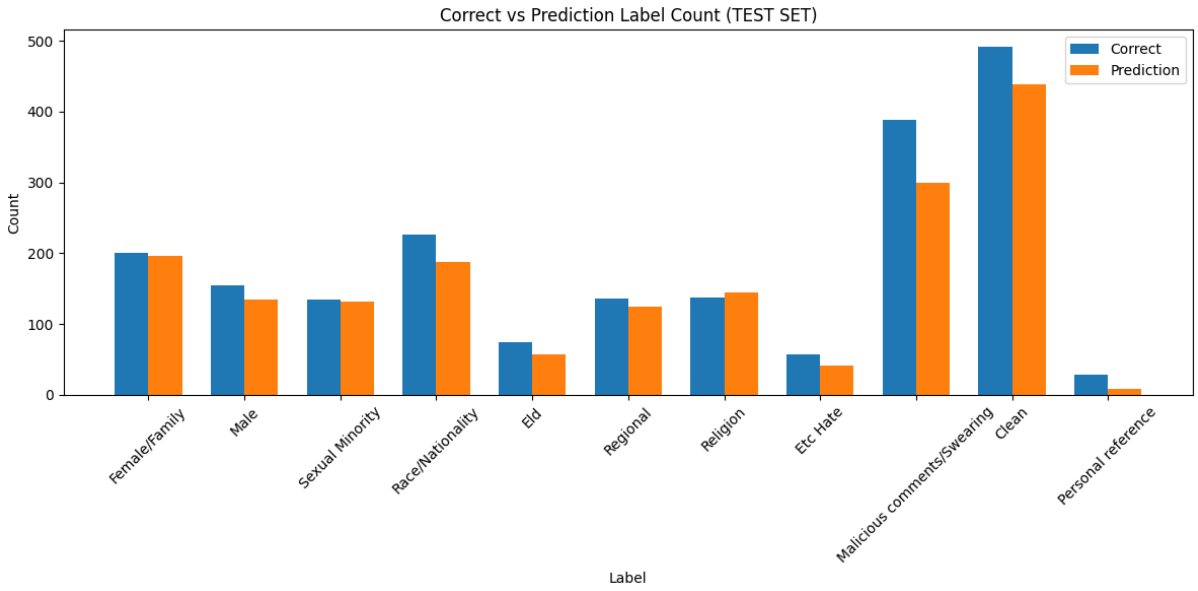
모든 학습 준비가 끝났다.
다음으로는 학습 결과 및 결과를 시각화 해보도록 하자.

Epoch	Training Loss	Validation Loss	Macro F1	Accuracy
1	0.170800	0.172523	0.549734	0.458378
2	0.143100	0.142203	0.617509	0.591249
3	0.115400	0.139803	0.649734	0.614194
4	0.096900	0.138655	0.674183	0.630203
5	0.075200	0.143980	0.684165	0.645144
6	0.073700	0.148417	0.703930	0.655816
7	0.071300	0.151978	0.696566	0.651547

총 7번의 epoch 마다의 훈련 loss, 검증 loss, Macro F1, Accuracy 계산 결과이다.
평가지표 수치가 가장 좋은 6번째 모델이 실제 모델로 채택되었다.
이렇게 모든 fine-tuning과정은 끝이 났다.
훈련 데이터에 대한 결과를 시각화해서 보도록 하자.

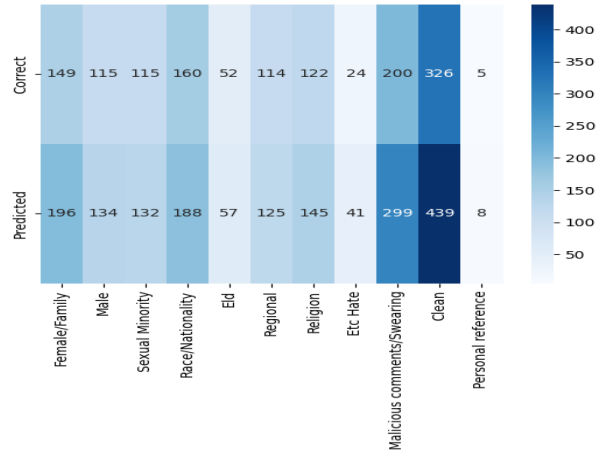


Test loss는 약 0.15로 성능이 나쁘지 않게 나왔다. Training loss를 보면, 과적합이 발생하고 있다는 것을 확인할 수 있지만, 5번의 학습 중 가장 성능이 좋은 모델이다.
5번의 학습 과정에서 4번의 학습에서는 모두 과적합이 심하게 발생하였지만, 평가지표 수치는 조금씩 계속 올라가 학습이 조기종료 되지 않는 어려움이 있었다.
이전 학습 과정에서 Test loss가 0.3 이상을 보인 것에 비하면, 좋은 모델을 얻었다고 판단하였다.
다음은 **실제 정답과 모델의 예측을 시각화해 비교**해보도록 하자.



그래프를 보면, 특정 레이블에서는 좋지 않은 성능을 보이기도 한다.
레이블 중 “개인지칭여부”를 제외하고는 전부 정답 비율이 80% 이상이다.
레이블 마다 성능이 차이가 있으므로 **각 레이블마다의 평가지표**를 살표보도록 하자.

	precision	recall	f1-score
Female/Family	0.76	0.74	0.75
Male	0.86	0.74	0.80
Sexual Minority	0.87	0.85	0.86
Race/Nationality	0.85	0.71	0.77
Eld	0.91	0.69	0.79
Regional	0.91	0.84	0.87
Religion	0.84	0.88	0.86
Etc Hate	0.59	0.42	0.49
Malicious comments/Swearing	0.67	0.52	0.58
Clean	0.74	0.66	0.70
Personal reference	0.62	0.18	0.28
micro avg	0.78	0.68	0.73
macro avg	0.78	0.66	0.71
weighted avg	0.78	0.68	0.72
samples avg	0.71	0.69	0.69



이를 통해 “기타혐오”, “악플/욕설”, “개인지칭여부” 레이블에 대해서 좋지 않은 성능을 보인다는 것을 알았다. 하지만, 딥러닝 학습의 경우 시간이 오래 걸린다는 것이 본 프로젝트의 한계점이었다. 총 5번의 학습을 하는 동안 다양한 하이퍼파라미터 조합을 설정해 시도해보았지만, 훈련 데이터에 대한 과적합을 완전히 막을 수는 없었다.

훈련 데이터를 증강시키거나, 학습률을 낮추거나, Reguralization의 정도를 높이거나 epoch를 줄여가는 등 많은 방법을 시도해 성능을 개선시키기 위해 노력하였다.

그 결과 현재 모델을 사용하게 되었다.

이후 마주하게 된 또 다른 한계점은 실험 데이터로 실제 뉴스 기사 또는 SNS 댓글을 크롤링해와 실험데이터로 사용하도록 계획하였지만, 대부분의 뉴스 기사 또는 SNS는 댓글의 크롤링을 직접적으로 막아놓아 REST API 호출을 이용하는 크롤링 방식, request + BeautifulSoup을 이용한 크롤링 방식 모두 실패하였다.
그래서 실험 데이터로 또 다른 악성 댓글 데이터 셋을 사용하게 되었다.

다음으로는 **실험 데이터 분류** 과정에 대해 설명해보도록 하겠다.

```
df = pd.read_csv("experimentDataset.tsv", sep='\t')
df['content'].to_csv("new_inputs.tsv", index=False, header=False)
```

실험 데이터 셋을 읽고. 데이터 셋에서 ‘content’ 열(column)에 해당하는 데이터만 추출하는 과정이다. (사용할 실험 데이터는 레이블링 되어있지 않은 문장 데이터 이므로)

다음은 **실험 데이터 셋 토큰화 및 입력 포맷 변환 과정 및 데이터 객체화 과정**이다.

```
experiment_encodings = tokenizer(new_texts, truncation=True, padding=True,
max_length=128)
experiment_dataset = ToxicDataset(experiment_encodings)
```

fine-tuning 과정에서와 동일하게, 실험 데이터 셋을 토큰화하고, 입력 포맷으로 변환해 준다. 마찬가지로 encoding된 실험 데이터를 객체화한다.
이때, 앞에서 사용한 ToxicDataset을 재사용한다. 레이블이 없는 실험 데이터에서 이를 재사용하기 위해 클래스 정의에서 label 변수를 None으로 두었던 것이다.

다음은 준비된 encoding된 **실험데이터를 학습된 모델에 적용하고, 예측 및 분류 결과를 얻는 과정이다.**

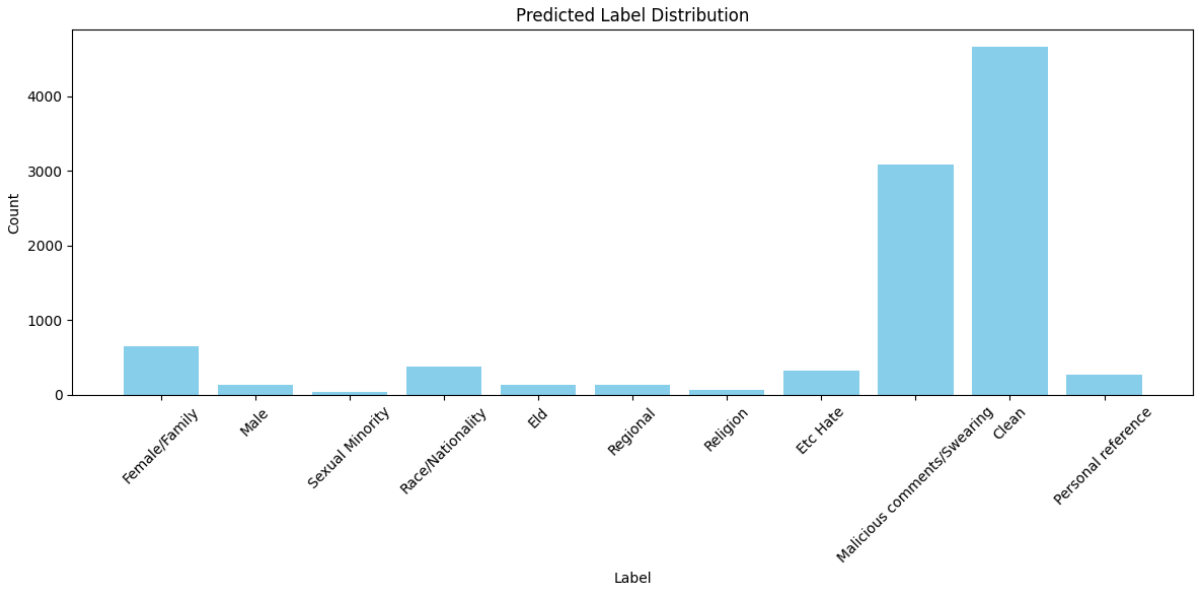
```
predictions = trainer.predict(experiment_dataset)
logits = predictions.predictions
probs = torch.sigmoid(torch.tensor(logits)).numpy()
preds = (probs >= 0.5).astype(int)
```

Fine-tuning된 모델인 “trainer”를 통해 실험데이터를 예측한다.
예측 결과는 실수 벡터로 저장되는데, 이를 확률 값으로 변환한다.
이후 확률 값이 0.5 이상인 레이블을 예측 결과로 저장한다.

모든 예측을 종료한 이후에는 예측 값이 모든 레이블에 대해서 0인 경우
예측 값을 “None”으로 바꿔주는 작업을 수행한다.

최종적으로, 모든 예측 결과를 별도의 파일에 저장하면, 예측 및 분류는 끝이 난다.

다음은 **실험데이터를 분류한 결과를 시각화하고, 평가** 해보도록 하겠다.




```
=== 예측된 레이블별 개수 ===
Female/Family (여성/가족): 652
Male          (남성): 128
Sexual Minority (성소수자): 42
Race/Nationality (인종/국적): 382
Eld           (연령): 136
Regional      (지역): 132
Religion      (종교): 59
Etc Hate      (기타혐오): 319
Malicious comments/Swearing (악플/욕설): 3084
Clean         (clean): 4655
Personal reference (개인지칭): 274
```

실험 데이터 셋은 11가지 레이블에 대한 정답 데이터가 없기 때문에 정확한 성능은 파악할 수 없다.

분류된 “Clean” 레이블 데이터가 4655개이고, “Un-Clean”에 속하는 데이터 (“Clean” 이외의 모든 레이블)가 5208개이고, 그리고 None으로 분류된 것이 137개이다.

악성 댓글 O	악성 댓글 X
5,000개	5,000개

실제 실험 데이터가 다음과 같은 것을 레이블 비율을 보이는 것을 봐서는 예측 및 분류 결과가 큰 오차를 보이지 않은 것으로 예측된다.

또한 총 5000개의 악성 댓글 데이터 중 실제 욕설 문장이 들어간 데이터가 2032개이다. 실험 데이터 셋에 “Korean Haste speech dataset”의 혐오 표현 데이터 1818개가 포함되어있으며,

이 데이터에는 욕설이 대부분 포함 되어있는 것을 감안했을때, 욕설 레이블로 분류된 데이터가 상대적으로 많은 비율을 차지하는 것이 정상적이라고 판단하였다.

다른 레이블의 정확도를 살펴보기 위해 임의의 10개의 문장을 반복적으로 출력해 결과를 직접 확인해보도록 하자.

```
1. 씨바알..노무노무 술프노... 오늘 저녁은 꽃등심이다ㅠㅜ  악플/욕설
2. 오로지 냄비가 무기인 오징어썩은 냄새나는 세균덩어리김 치 년 들  여성/가족
3. 베트콩들 왜이리 나댑니까 인종/국적
4. 차은우는 아스트로 탈퇴하고 배우길 걸으면 안돼나요? clean
5. 역시 한국인은 남 잘되는 꼴을 못 봐  인종/국적
6. 하여튼 전라도놈들은 뒷배가 좋아 ㅉㅉ  지역
7. 진영아 원더걸스 재결합 가자 clean
8. 기사가 더 자극적이고 지랄이다 기레기야 기타혐오
9. 정두홍 인성 드럽기로 유명하지  개인지칭
10. 틀딱이네 --- 먹을 나이 됐다  연령
```

전체적으로 잘 분류한 결과도 있었지만,

```
1. 이건 무슨 소리? 악플/욕설
2. 박나래 저사각턱이?? 방송사고급 표현 아닌가 악플/욕설
3. 씨밸년=틀딱용어ㅁㅈㅎ  연령
4. 그냥 결혼하지 왠 기사 헐~~~  악플/욕설
5. 솔직히 클라라 비주얼 대한민국 최상위급이지  clean
6. 전 그냥 조보아 땜에 봐요...  여성/가족
7. 혼전임신은 미리 조심하지 못한 여자 잘못이 크다  clean
8. 핀란드..?ㅈ |라 ㄹ하네 clean
9. 이름한번 ㅈ같네  악플/욕설
10. 페미들아 제발 죽어줘 여성/가족
```

잘 분류가 되지 않은 결과도 있었다.
(총 10번의 추출결과, 즉 100개의 문장 중 86개의 문장이 정확하게 분류가 되었다.)

특정 악성 댓글 레이블 중 하나로 예측 되어야 하지만, “Clean” 레이블로 예측되어 오류가 난 것도 있고, “개인지칭” 레이블은 특히 오류가 많았다.
이는 테스트 데이터에서도 발생한 문제로 개선할 필요가 있다.
“개인지칭” 레이블의 학습 데이터가 적었던 것이 이러한 문제를 발생시켰다고 예상한다.
다른 데이터와 같이 “개인지칭” 데이터를 많이 확보한다면, 이 문제가 충분히 개선될 것이라고 생각한다.

또한, 문제 해결을 위해 계획했던 최종목표는 다중 레이블 분류이다.
이는 예측이 여러 레이블에 속할 수 있는 분류 문제이다.
하지만, 학습 데이터가 다중 레이블 분류가 되어있지 않은, 하나의 레이블에 One-hot encoding되어있는 데이터인 것의 영향이 커 다중 레이블 분류가 잘 수행되지 않았다고 생각한다.

이는 학습 데이터의 문장이 One-hot encoding으로 레이블링 되어있지 않고, Multi-label encoding으로 레이블링 되어있었다면, 성공적으로 다중 레이블 분류를 수행할 수 있었을 것이라고 예상한다.

결론

본 프로젝트에서는 11개의 악플 유형으로 레이블링 되어있는 약 18,000개의 데이터로 사전 학습된 한국어 자연어 처리 모델 “klue/bert-base”을 fine-tuning하였다.

이후, 레이블링 되지 않은 실험 데이터를 통해 11개의 악플 유형 레이블로 잘 분류하는지 fine-tuning한 모델을 통해 예측하고, 분류 결과를 시각화해보았다.

전체 데이터가 정확하게 분류되었는지, 확인할 수는 없었지만, 10번의 표본 추출 후 직접 문장과 예측된 레이블을 확인해본 결과 약 85%의 정확도를 보였다.

하지만, 처음 문제해결의 목표였던, 다중 레이블 분류는 성공하지 못하였으며, 결과적으로 하나의 정답 클래스를 예측하는 다중 클래스 분류를 성공적으로 수행하였다.

본 프로젝트에서 생겼던 여러가지 문제점은 대부분 데이터 셋이 한정적이라는 점에서 생겼다고 생각한다. 모든 레이블이 균형 있게 데이터가 존재하고, 데이터의 양의 많았더라면, 현재보다 성능이 개선되었을 것이라고 생각한다.

훈련 데이터 학습 및 테스트 데이터 예측 과정에서 matplotlib를 통해 다양한 그래프로 데이터를 시각화 하였다. 이 과정에서 다양한 방식으로 데이터를 표현하고, 이를 분석할 수 있었다. 실제로 이러한 시각화가 모델을 개선하는 데에 있어서 많은 도움이 되었다.

또한, 레이블링 되어있지 않은 데이터를 fine-tuning한 모델을 통해 분류하고, 시각화 하여, 데이터 전체의 악플 유형 별 비율을 한 눈에 확인할 수 있었다.

본 프로젝트에서는 최근 증가하는 온라인 사용자 생성 콘텐츠 속에서 사회적 문제로 대두되고 있는 악성 댓글을 효과적으로 식별하고, 그 유형을 분류하기 위한 딥러닝 기반 자연어 처리 시스템을 제안하였다. 본 시스템은 총 11가지의 레이블을 기준으로 문장을 정밀하게 분류함으로써 다중 레이블 분류 문제를 해결하는 데 초점을 맞췄다.

실험 결과, fine-tuning한 모델은 단순 악성 여부 판별 뿐만 아니라 구체적인 악성 댓글 표현의 유형까지 높은 정확도로 분류하는 결과를 보였으며, 이는 기존 방식보다 더 풍부하고, 세밀한 대응이 가능함을 시사한다. 이는 디지털 커뮤니케이션 공간의 안전성과 포용성 확보에 실질적으로 기여할 수 있는 기반을 마련하였다.

초기 목표인 다중 레이블 분류는 성공적으로 수행하지 못하였지만, 다중 클래스 분류는 성공적으로 수행할 수 있었고, 이는 향후에 데이터 셋의 보강으로 다중 레이블 분류로 발전 시킬 수 있음을 예상한다.

향후에는 다양한 플랫폼에서 활용될 수 있는 악성 댓글 대응 시스템으로 발전될 수 있도록 고도화할 예정이다.

참고문헌

https://www.inews24.com/view/1444614

https://github.com/ZIZUN/korean-malicious-comments-dataset

https://github.com/kocohub/korean-hate-speech

https://github.com/smilegate-ai/korean_unsmile_dataset

https://wikidocs.net/31379

https://wikidocs.net/152922

https://wikidocs.net/166796

https://huggingface.co/klue/bert-base