

# 악플 탐지 프로그램



## 목차

1. 개요
2. 데이터 소개
3. 방법론 및 모델링
4. 결과

# 1. 개요

---



## 프로젝트 배경

### [전문] 보아, 국내외 대규모 고소 진행 “악플로 힘들어”

SM 엔터테인먼트 공식 입장

현재 여러 SNS 및 동영상 공유 플랫폼, 온라인 커뮤니티 등을 중심으로 아티스트에 대한 허위 사실 유포, 악의적 인신공격, 모욕, 비방이 지속적으로 발생함에 따라, 당사 아티스트 권리 침해관련 법적 대응을 담당하고 있는 법무법인(유한) 세종을 비롯해, 추가적으로 외부 법률 자문기관과도 공조하여 국내외로 대규모 고소를 진행하고 있습니다.

이전 공지 이후, 당사는 이미 다수의 게시물에 대해 충분한 자료를 수집하고, 사이버 렉카 등 특정 사례에 대한 조사 및 고소를 진행 중입니다. 또한 지속적인 정보 수집을 통해 법적 책임을 끝까지 물을 계획입니다.

위와 같은 행위가 **한 사람으로서 감당하기 어려울 정도로 심각한 수준에 이르렀으며, 이로 인해 아티스트가 큰 정신적 고통을 겪으며 매우 힘든 시간**을 보내고 있습니다. 당사는 선처나 합의 없이 관련 행위자들을 법적으로 처벌받도록 할 방침이며, 지금 이 순간에도 악의적인 게시물을 게재하는 모든 행위가 처벌 대상이 될 수 있음을 인지하시기 바랍니다.

다시 한번 이와 같은 행위가 개인의 명예와 존엄성을 해치는 범죄 행위임을 분명히 하며, 관련 법령에 따라 이러한 행위자들은 강력히 처벌받을 수 있음을 엄중히 경고합니다. 아티스트에 대한 무분별한 공격을 중단해 주시길 간곡하게 부탁드립니다.



# 프로젝트 목표

악플/선플 분류를 통한 댓글 필터링

## 2. 데이터 소개

---



## 데이터

- Smilegate AI - Korean Unsmile Dataset
- Korean HateSpeech Dataset
- Curse-detection-data

욕설, 혐오, 지역, 연령, 젠더갈등, 비하발언 등을 전부 악플로 취급하여 데이터셋 구성

-> 악플 17737개, 깨끗한 댓글 10367개



## 데이터

- ㅋㅋㅋㅋㅋ 너 용접공이냐? 존나 웃기네. 잘가 병신아.
- <https://www.youtube.com/watch?v=EZtENB3qA4Y> 6:30 부터봐라
- ㅂㄷㅂㄷ 대지말고
- A: 마스크쓰면 이쁘네!
- 얼굴의 조화가 뭔가 아쉬운형 ——
- 클라라는 사랑입니다♥♥♥♥♥
- 화장안하면 디게모생격내



분석에 필요없는 링크, 숫자, 문장 부호, 영어, 이모지 등

오타, 문법 오류, 신조어, 초성만 이루어진 단어들이 존재





## 전처리

- PyKoSpacing을 이용해 띄어쓰기 교정
- py-hanspell을 이용해 맞춤법, 띄어쓰기 교정
- j5ng/et5-typus-corrector을 이용해 맞춤법, 띄어쓰기 교정
- mecab 토큰라이저를 이용
  - 문장을 토큰별로 분리
  - 유효 태그 선별
  - 표제어 추출
  - 불용어 제거
- 이후 길이 5넘는 문장만 선택

## 전처리

```
'ㄱㄱ': '고고',  
'ㄲ': '고고',  
'ㄱㄱㅅ': '고고싱',  
'ㄱㄱㅆ': '고고싱',  
'ㄱㄴ': '가능',  
'ㄱㄴㅇ': '가나요',  
'ㄱㄷ': '기다려',  
'ㄲ': '감사',  
'ㄱㅅ': '감사',  
'ㄱㅇㅇ': '귀여워',  
'ㄱㅌ': '개추',  
'ㄱㅎ': '극혐',  
'ㄲㅈ': '아깝다',
```

초성단어 변환시도

```
convert_to_korean_word1('ㄱㅌㅈ')
```

'미친놈 '

## 전처리

고참이 딸러들어간 스티로폼 씹을 때마다 희열을 느끼는 거야 ?

"['고참/NNG', '딸리/VV', '들어가/VV', '스티로폼/NNG', '씹/NNG', '때/NNG', '희열/NNG', '느끼/VV']"

문장 태깅 후 어간추출

"['고참/NNG', '딸리다/VV', '들어가다/VV', '스티로폼/NNG', '씹/NNG', '때/NNG', '희열/NNG', '느끼다/VV']"

표제어 추출

"['고참/NNG', '딸리다/VV', '들어가다/VV', '스티로폼/NNG', '씹/NNG', '때/NNG', '희열/NNG', '느끼다/VV']"

"['고참', '딸리다', '들어가다', '스티로폼', '씹', '때', '희열', '느끼다']"



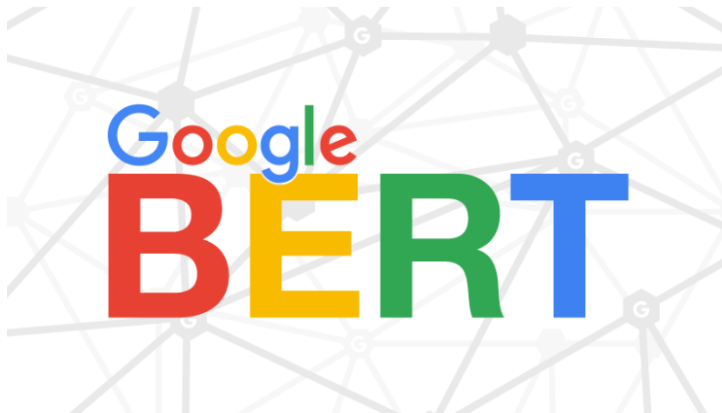
## 전처리

- 중복문장 제거
- E-mail 제거
- URL 제거
- 모음으로만 된 글자 제거
- HTML 태그 제거
- 특수기호(이모지) 제거
- 정규화(3번 이상 같은 글자로 반복되는 글자 2개로 제한)
- 10글자 이내 문장 제거
- 자음으로된 단어 적절한 뜻으로 변환
- 이후 남은 자음 제거

### 3. 방법론 및 모델링

---

## 분석 방법



## google-research/ electra



ELECTRA: Pre-training Text Encoders as  
Discriminators Rather Than Generators

5

Contributors

53

Issues

2k

Stars

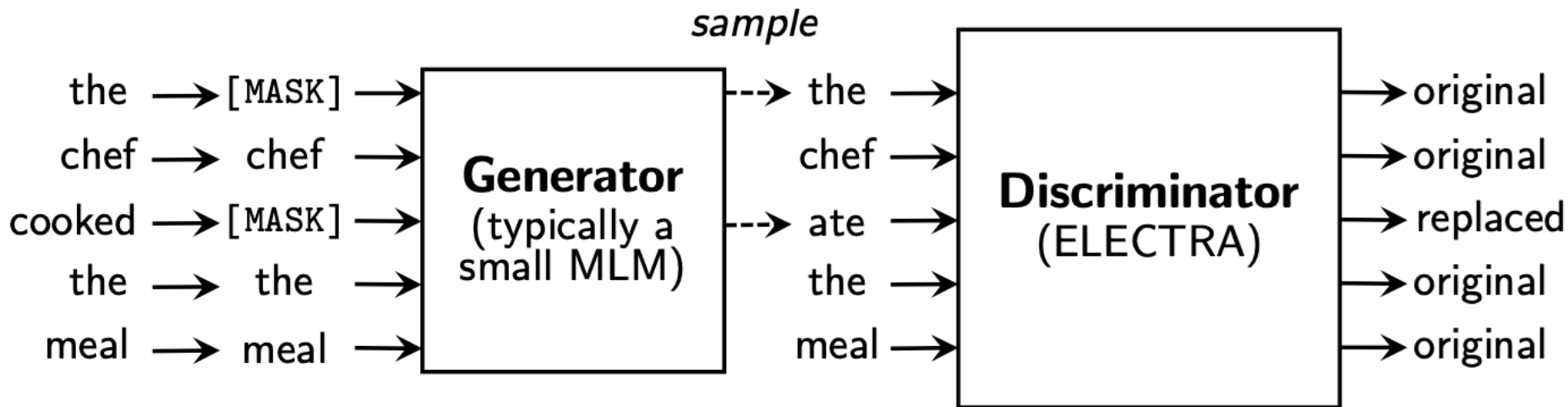
350

Forks





# ELECTRA





## 토큰화 예시

4년동안 몇천 뿌리고 한다는짓이 ㅋㅋ

```
tensor([ 3754, 1649, 17075, 16315, 4253, 11644, 188, 188, 3, 3,  
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3,  
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3,  
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3,  
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3,  
        3, 3, 3, 3])
```



## 모델 설계 및 구현

AdamW

CrossEntropy

gelu

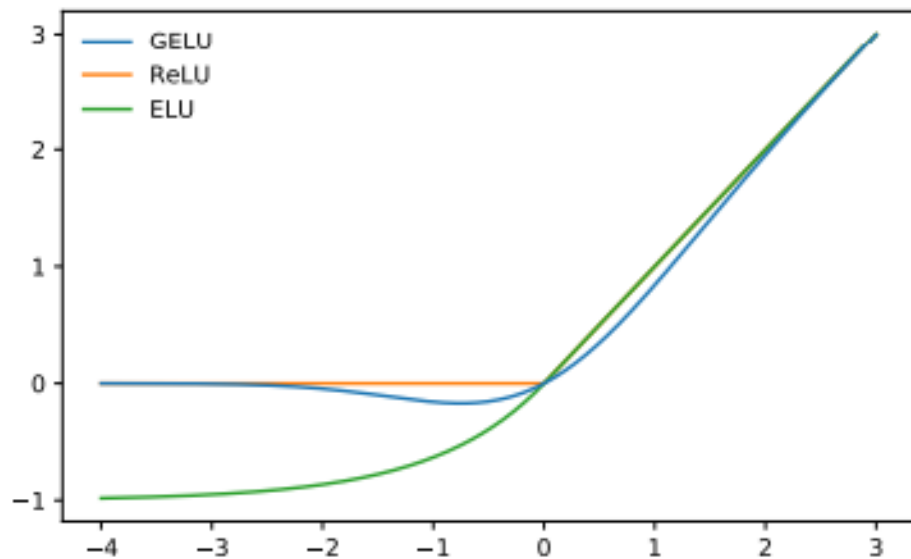


Figure 1: The GELU ( $\mu = 0, \sigma = 1$ ), ReLU, and ELU ( $\alpha = 1$ ).

## 모델 설계 및 구현

Model	Train FLOPs	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	WNLI	Avg.*	Score
BERT	1.9e20 (0.06x)	60.5	94.9	85.4	86.5	89.3	86.7	92.7	70.1	65.1	79.8	80.5
RoBERTa	3.2e21 (1.02x)	67.8	96.7	89.8	91.9	90.2	90.8	95.4	88.2	89.0	88.1	88.1
ALBERT	3.1e22 (10x)	69.1	<b>97.1</b>	<b>91.2</b>	92.0	90.5	<b>91.3</b>	–	89.2	91.8	89.0	–
XLNet	3.9e21 (1.26x)	70.2	<b>97.1</b>	90.5	<b>92.6</b>	90.4	90.9	–	88.5	<b>92.5</b>	89.1	–
ELECTRA	3.1e21 (1x)	<b>71.7</b>	<b>97.1</b>	90.7	92.5	<b>90.8</b>	<b>91.3</b>	<b>95.8</b>	<b>89.8</b>	<b>92.5</b>	<b>89.5</b>	<b>89.4</b>

Table 3: GLUE test-set results for large models. Models in this table incorporate additional tricks such as ensembling to improve scores (see Appendix B for details). Some models do not have QNLI scores because they treat QNLI as a ranking task, which has recently been disallowed by the GLUE benchmark. To compare against these models, we report the average score excluding QNLI (Avg.\*) in addition to the GLUE leaderboard score (Score). “ELECTRA” and “RoBERTa” refer to the fully-trained ELECTRA-1.75M and RoBERTa-500K models.

- ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators:

<https://openreview.net/pdf?id=r1xMH1BtvB>



# 모델 설계 및 구현

## 임베딩

`word_embeddings`: 입력 토큰을 임베딩. 사전에 정의된 단어 사용 30000개의 임베딩 행렬 사용

`position_embeddings`: 입력 토큰의 위치 정보를 포함하는 임베딩.

`token_type_embeddings`: 입력 토큰의 타입을 나타내는 임베딩

## 인코더

입력된 임베딩을 인코딩하여 출력 생성. `self-attention` 매커니즘 (`dropout = 0.1`)

## 출력(분류)

출력을 선형변환, `gelu`, 이진분류를 위한 출력층 클래스 개수 설정

## self-Attention

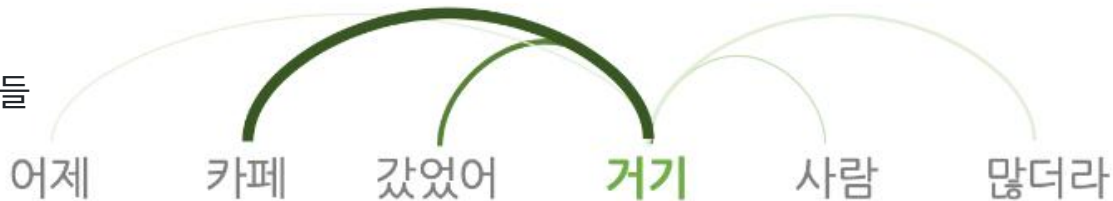
Attention

이 시퀀스 전체 단어 - 타겟 시  
퀀스 단어 1개 연결



self-Attention

소스(입력) 시퀀스의 전체 단어들  
사이 연결



# 모델 설계 및 구현

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

tokenizer = AutoTokenizer.from_pretrained("beomi/KcELECTRA-base")
model = AutoModelForSequenceClassification.from_pretrained("beomi/KcELECTRA-base", num_labels=2)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

X_list = X.values.tolist()
y = y.values
sequences = tokenizer(X_list, padding=True, truncation=True, return_tensors="pt")

X_train, X_test, y_train, y_test = train_test_split(sequences['input_ids'], y, test_size=0.2, random_state=42)
X_train_mask, X_test_mask, _, _ = train_test_split(sequences['attention_mask'], y, test_size=0.2, random_state=42)

train_dataset = TensorDataset(X_train, X_train_mask, torch.tensor(y_train, dtype=torch.long))
test_dataset = TensorDataset(X_test, X_test_mask, torch.tensor(y_test, dtype=torch.long))

train_loader = DataLoader(dataset=train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=16, shuffle=False)

optimizer = AdamW(model.parameters(), lr=5e-5)
criterion = nn.CrossEntropyLoss()
```

```
EPOCHS = 1
for epoch in range(EPOCHS):
    model.train()
    for input_ids, attention_mask, labels in train_loader:
        input_ids = input_ids.to(device)
        attention_mask = attention_mask.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        logits = outputs.logits
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()

    model.eval()
    val_losses = []
    val_accs = []
    for input_ids, attention_mask, labels in test_loader:
        input_ids = input_ids.to(device)
        attention_mask = attention_mask.to(device)
        labels = labels.to(device)

        with torch.no_grad():
            outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
            val_losses.append(outputs.loss.item())
            logits = outputs.logits
            preds = torch.argmax(logits, dim=1)
            acc = (preds == labels).float().mean().item()
            val_accs.append(acc)

    val_loss = np.mean(val_losses)
    val_acc = np.mean(val_accs)
    print(f"Epoch {epoch + 1}/{EPOCHS}, Validation Loss: {val_loss}, Validation Accuracy: {val_acc}")
```

```
torch.save(model, 'mymodel.pth')
```



## 훈련 및 평가

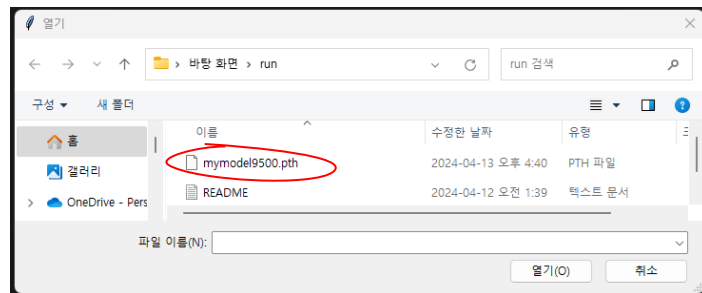
	GPT 3	LSTM	LSTM-CNN	KcElectra	Bert
Loss	0.7060	0.6829	0.5653	0.3898	0.7541
Accuracy	0.7548	0.5962	0.7788	0.8702	0.8077

## 4. 결과

---

# 결과

1.



2.



실행파일 링크 : <https://drive.google.com/file/d/1NrGoBwfFkAiFuiAhD-u2ztLxzcSQdQ7h/view?usp=sharing>






## 결론 및 제안

### 결론/보완

기존 악성댓글을 구분하는 기능을 유지하고, 욕설/비하하는 단어만을 필터링하는 기능을 추가  
더 많은 데이터를 수집하고 라벨링하여 학습을 시키면서 품질 향상

### 제안

- 본 모델을 인스타그램, 트위터, 스레드 등에 적용하여, 악성 댓글 필터링이 잘 이루어지지 않는 곳에서 발생하는 문제를 해결할 수 있다.
- 이를 통해 사용자들이 악의적인 댓글로 인해 상처받거나 눈살을 찌푸리는 일을 방지하고, 건강한 온라인 커뮤니티 문화를 조성할 수 있다.



## 참고 문헌

- Smilegate AI ([https://github.com/smilegate-ai/korean\\_unsmile\\_dataset](https://github.com/smilegate-ai/korean_unsmile_dataset))
- Korean HateSpeech Dataset (<https://github.com/kocohub/korean-hate-speech>)
- 욕설 감지 데이터셋 (<https://github.com/2runo/Curse-detection-data>)
- /beomi/KcELECTRA-base (<https://huggingface.co/beomi/KcELECTRA-base>)
- CrossEntropyLoss (<https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>)
- AdamW (<https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html>).
- [https://velog.io/@judy\\_choi/NLP-Transformer-3-Self-Attention-Multi-Head-Attention](https://velog.io/@judy_choi/NLP-Transformer-3-Self-Attention-Multi-Head-Attention)

# 실행파일 소스코드

```
import tkinter as tk
from tkinter import filedialog, messagebox
import torch
from transformers import AutoTokenizer

PATH = None

def browse_file():
    global PATH
    PATH = filedialog.askopenfilename()
    if PATH:
        path_label.config(text=f"파일 경로: {PATH}")
        messagebox.showinfo("알림", f"파일 경로가 설정되었습니다: {PATH}")
    else:
        messagebox.showinfo("알림", "파일을 선택하지 않았습니다.")

def classify_comment():
    if PATH is None:
        messagebox.showinfo("알림", "모델 파일을 선택하세요.")
        return

    model = torch.load(f'{PATH}')
    tokenizer = AutoTokenizer.from_pretrained("beomi/KcELECTRA-base")
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    input_text = text_input.get().strip()
    if input_text:
        sequences = tokenizer(input_text, padding=True, truncation=True, return_tensors="pt")
        input_ids = sequences['input_ids'].to(device)
        attention_mask = sequences['attention_mask'].to(device)
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=1)

        if preds[0].item() == 0:
            result_text.config(text="정상", fg="green", font=("Helvetica", 14, "bold"))
        else:
            result_text.config(text="악플", fg="red", font=("Helvetica", 14, "bold"))
    else:
        messagebox.showinfo("알림", "입력이 비어 있습니다.")

window = tk.Tk()
window.title("악플 분류기")

browse_button = tk.Button(window, text="파일 찾기", command=browse_file, font=("Helvetica", 12))
browse_button.grid(row=0, column=0, padx=10, pady=5)

path_label = tk.Label(window, text="", font=("Helvetica", 10), anchor="e")
path_label.grid(row=0, column=1, padx=10, pady=5)

input_label = tk.Label(window, text="텍스트를 입력하세요:", font=("Helvetica", 12))
input_label.grid(row=1, column=0, padx=10, pady=5)

text_input = tk.Entry(window, width=50, font=("Helvetica", 12))
text_input.grid(row=1, column=1, padx=10, pady=5, sticky="ew")

classify_button = tk.Button(window, text="분류하기", command=classify_comment, font=("Helvetica", 12))
classify_button.grid(row=1, column=2, padx=10, pady=5)

result_frame = tk.Frame(window, relief=tk.GROOVE, borderwidth=5)
result_frame.grid(row=2, column=1, columnspan=2, padx=10, pady=5, sticky='w')

result_text = tk.Label(result_frame, text="", font=("Helvetica", 14), justify="center")
result_text.pack()

result_label = tk.Label(window, text="결과 :", font=("Helvetica", 12))
result_label.grid(row=2, column=0, padx=10, pady=5, sticky="e")

window.mainloop()
```