

BERT&GPT 파인튜닝 및 파이프라이닝을 활용한 감성 대화 챗봇

김형준, 박정민, 유재형, 이성주

요약

최근 사용자와 **Interactive**하게 소통할 수 있는 챗봇 서비스의 발달로 상담 및 연애회 등 활용 부문이 점점 다양해지고 있다. 따라서 본 문서는 **Fine-tuned BERT** 모델과 **GPT-3.5-turbo** 모델을 활용하여 **BERT** 모델로 사용자가 입력한 문장의 감정을 분석하고 **GPT** 모델로 해당 감정을 적절하게 응답 문장에 반영하는 챗봇 서비스를 제안하고 구현한다.

1. 서론

1.1. 연구배경

감성 대화 챗봇은 인공지능 기술을 활용하여 사용자와 자연스럽게 대화하며, 사용자의 감정을 이해하고 그에 맞는 응답을 제공하는 서비스이다. 이는 현대 사회에서 더 나은 사용자 경험과 상호작용을 위해 매우 중요한 요소로 간주되고 있다. 과거의 챗봇은 단순히 특정 질문에 대해 정형화된 답변을 제공하는 데에 그쳤지만 인공지능 기술의 발전과 자연어 처리 기술의 진보로 인해 감성 대화 챗봇은 사용자의 감정을 인식하고 공감하는 능력을 갖추게 되었다. 이는 사용자와 보다 깊은 수준의

상호작용을 가능하게 하고, 대화의 흐름에 맞게 적절한 응답을 제공함으로써 사용자의 감정적인 요구에 부합하는 서비스를 제공할 수 있게 되었다. 감성 대화 챗봇은 사용자의 감정에 맞는 위로와 조언을 제공하여 효과적인 상담을 진행하고, 사용자의 감정과 취향을 파악하여 개인 맞춤형 추천을 제공함으로써 고객 만족도를 높이고, 사용자들이 더 나은 자기 이해와 감정적인 안정을 얻을 수 있도록 돕는 역할을 할 수 있다. 이러한 감성 대화 챗봇은 **BERT&GPT** 파인튜닝 및 파이프라이닝과 같은 기술을 활용하여 구현될 수 있다. **BERT**와 **GPT**는 각각 언어 이해와 생성에 탁월한 성능을 보이는 모델로, 이들을 조합하고 파인튜닝하여 감성 대화 챗봇의 정확성과 자연스러움을 향상시킬 수 있다. 또한 파이프라이닝은 대화 흐름을 관리하고 응답의 일관성을 유지하는 데 도움이 되며, 챗봇의 대화 품질을 높이는 데 중요한 역할을 한다. 따라서 본 연구는 챗봇 서비스가 사용자의 감정을 정확하게 해석하고 해당 감정을 담은 응답 문장을 반환하여 부자연스러움을 줄이고 서비스의 효과성을 높이기 위해 **BERT&GPT** 파인튜닝 및 파이프라이닝을 활용한 감성 대화 챗봇을 제안하고 구현한다. 이를 통해 챗봇 서비스를 활용하는 사용자의 긍정적인 경험을 유도하여 챗봇을 이용한 긍정적인 상호작용으로 사용자에게 감정적인 도움을 주고자 한다.

1.2. 연구목표

본 프로젝트에서 달성하고자 하는 주요 목표는 ‘자연어처리 관련 자유주제 개발’이라는 주제에 맞추어 자연어처리 기술을 활용하여 **BERT&GPT** 파이프라이닝 감성 대화 챗봇을 개발하는 것이다. 해당 주요 목표를 달성하기 위한 세부 목표는 다음과 같다. 먼저 자연어처리(NLP) 기술 중 **BERT** 모델을 일상대화를 약 60가지의 감정으로 분류한 데이터로 **Fine-tuning**하여 사용자의 입력 문장으로부터 감정을 분석하는 모델을 개발한다. 다음으로 **BERT** 모델이 파악한 사용자의 감정에 대해 해당 감정을 출력 문장에 반영할 수 있도록 자연어처리(NLP) 기술 중 **GPT-3.5-turbo** 모델을 활용하여 응답 문장을 생성한다. 마지막으로 사용자의 **Input Sentence**와 동일한 감정이 담긴 **Output Sentence**를 **GPT** 모델로 생성하여 사용자에게 전달할 수

있도록 GPT 모델을 튜닝한다. 이에 따라서 사용자에게 자신과 챗봇이 같은 맥락에서 대화를 나누고 있음을 인지하게 한다. 더 나아가, 어디서든 자신의 감정에 공감해 주며 대화를 할 수 있는 챗봇 서비스를 사용자에게 제공하여 감정 부문에서 소외되는 사람의 수를 줄이고, 결과적으로 사회 전체적인 행복을 증진시킨다.

2. 관련연구

2.1. NLP

자연어 처리(NLP)는 기계가 인간의 언어를 이해, 해석 및 생성할 수 있도록 하는 모델 및 알고리즘 개발에 중점을 둔 연구 분야이다. 인터넷에서 사용할 수 있는 텍스트 데이터의 양이 증가함에 따라, NLP는 기업과 연구원 모두에게 필수적인 도구가 되었다. 최근 몇 년 동안 NLP 연구에서 가장 중요한 혁신은 대규모 사전 훈련 방법의 개발이다. 예를 들어 BERT(Bidirectional Encoder Representations from Transformers) 모델은 언어 모델링에서 질문 응답 및 감정 분석에 이르기까지 광범위한 NLP 작업에서 놀라운 결과를 보여주었다. 사전 학습 방법을 사용하면, 모델이 방대한 양의 텍스트 데이터에서 학습하여 언어의 맥락과 뉘앙스를 더 잘 포착할 수 있다. 그러나, NLP 연구자들이 직면한 과제는 여전히 많다. 가장 큰 장애물은 언어 데이터의 다양성 부족이다. 대부분의 NLP 모델은 영어 데이터로 학습되며, 다른 언어로 학습된 모델은 리소스와 데이터가 제한적인 경우가 많다. 따라서, 다른 언어, 방언 및 악센트의 의미와 뉘앙스를 정확하게 캡처할 수 있는 모델을 만드는 것이 어렵다. NLP의 또 다른 과제는 언어의 빈정거림과 아이러니를 이해하는 것이다. 빈정거림과 아이러니가 소셜 미디어에 널리 퍼져 있으며, 기계가 그러한 언어의 의도를 이해하기 어려운 경우가 많다. 예를 들어 “좋아요, 또 다른 월요일!”과 같은 문장은 긍정적으로 보일 수 있지만, 반대 감정을 전달하기 위해 종종 냉소적으로 사용된다. 이러한 문제에도 불구하고 NLP는 chatbot 및 가상 비서에서

자동 번역 및 소셜 미디어의 감정 분석에 이르기까지 많은 잠재적 응용 프로그램을 가지고 있다. **NLP**는 이미 고객 서비스 경험을 개선하고, 온라인 남용 및 증오심 표현을 식별 및 완화하고, 검색 엔진 결과를 개선하는 데 사용되고 있다. 지속적인 연구 개발을 통해 **NLP**는 앞으로 우리 일상 생활에서 훨씬 더 중요한 역할을 할 것이다. 결론적으로 **NLP**는 최근 몇 년 동안 상당한 발전을 이루었지만, 아직 해야 할 일이 많다. 연구자는 **NLP** 모델이 인간 언어의 뉘앙스를 정확하게 캡처할 수 있도록 데이터 다양성 및 풍자 이해와 같은 문제를 해결해야 한다. 이러한 어려움에도 불구하고, **NLP**는 우리가 기계와 상호 작용하는 방식을 혁신하고 삶을 더 쉽고 효율적으로 만들 수 있는 잠재력을 가지고 있다.

2.2. Transformer

Transformer 모델은 최근 몇 년 동안 자연어 처리(NLP) 분야에서 엄청난 인기를 얻은 일종의 **neural network architecture**이다. 이러한 모델은 **language translation**, **text summarization**, **sentiment analysis**와 같은 다양한 애플리케이션에서 사용되었다. Transformer의 가장 중요한 장점은 **input sequence**에서 **long-range dependency**를 모델링 하는 기능이다. 기존의 순환 신경망(RNN)은 **vanishing gradient problem**으로 인해 **long sequence**를 처리하는 능력이 제한된다. Transformers는 다양한 중요도 수준으로 **input sequence**의 다른

부분에 주의를 기울일 수 있는 **self-attention mechanism**을 사용하여 이 문제를 해결한다. 다양한 **NLP** 작업에서 최첨단 결과를 달성한 여러 **transformer-based**

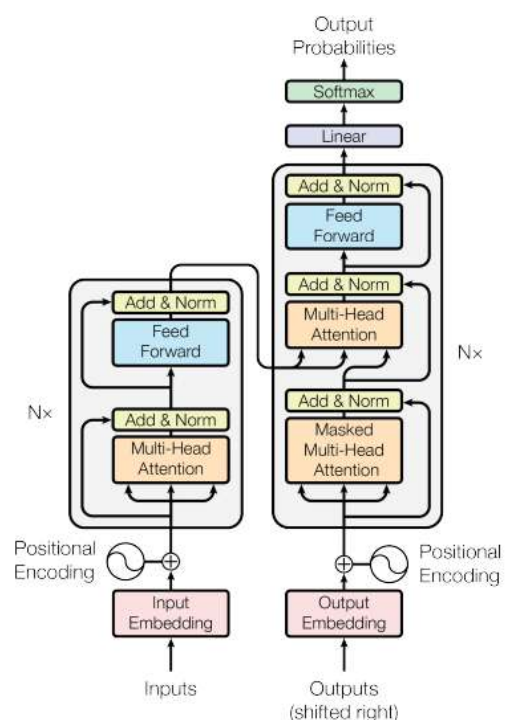
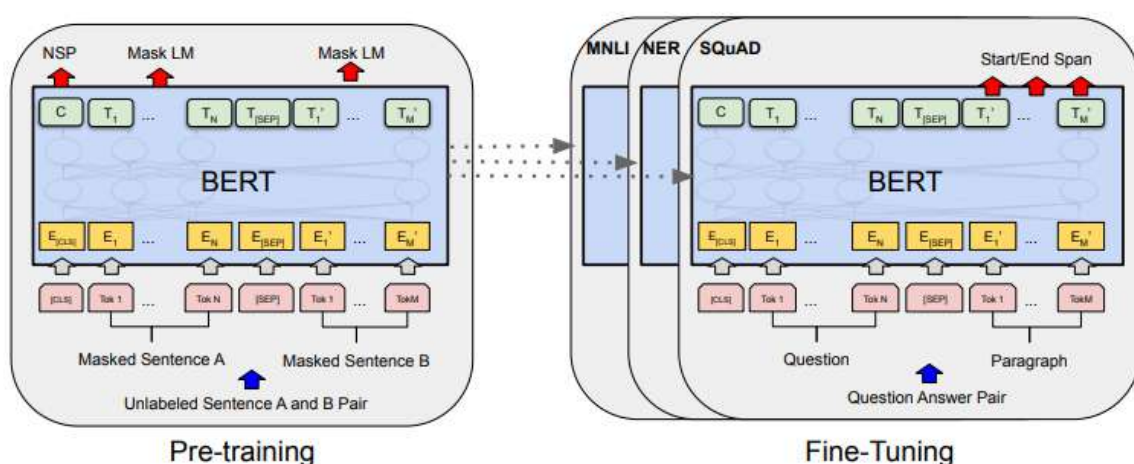


Figure 1: The Transformer - model architecture.

model이 개발되었다. 예를 들어 OpenAI에서 개발한 GPT-3 모델은 놀라운 language generation 기능을 보여줬고, Google에서 개발한 BERT 모델은 question-answering, sentiment analysis, natural language inference을 포함한 다양한 NLP 작업에서 뛰어난 성능을 보여주었다. 이러한 큰 pre-trained model 외에도 연구자들은 제한된 양의 training data로 특정 작업에 대해 이러한 모델을 fine-tuning 하는 방법을 개발했다. 이 transfer learning approach 방식은 다양한 NLP 작업의 성능을 향상시키는 동시에 training에 필요한 데이터의 양을 줄이는 데 매우 효과적인 것으로 나타났다. Transformers는 NLP 분야에 혁명을 일으켰고, 다양한 응용 분야에서 상당한 발전을 이루었다. 새로운 transformer-based model의 개발과 기존 모델의 개선은 활발한 연구 분야이며, 앞으로 이 분야에서 더 많은 흥미로운 발전을 기대할 수 있다.

2.3. BERT

BERT(Bidirectional Encoder Representations from Transformers)는 Google AI Language에서 개발한 pre-trained language model이다. 일반적인 언어 표현을 학습하기 위해 대규모 텍스트 corpus에서 훈련된 transformer model을 기반으로 하는 deep neural network architecture를 사용한다.



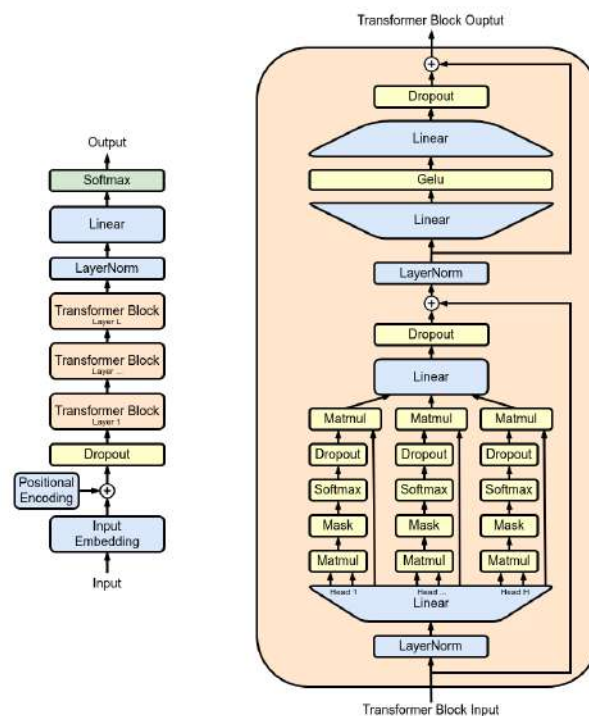
2018년 출시 이후 BERT는 text classification, question answering, named entity recognition과 같은 자연어 처리(NLP) 작업에서 가장 인기 있고 널리 사용되는 언어

모델 중 하나가 되었다. BERT의 핵심 혁신은 왼쪽 및 오른쪽 context를 모두 고려하여, 문장에서 단어의 context와 의미를 capture하는 기능이다. 여러 연구에 따르면, BERT는 다양한 NLP 작업에서 이전의 최신 모델보다 훨씬 뛰어난 성능을 보인다. 예를 들어 Stanford Question Answering Dataset(SQuAD)에서 BERT는 93.2%의 점수를 달성하여 이전 최고 점수인 90.9%를 넘어섰다. 마찬가지로 다양한 NLP 작업에 대한 모델의 성능을 측정하는 GLUE benchmark에서 BERT는 9개 작업 모두에서 최첨단 결과를 달성했다. 또한 BERT는 biomedical text mining, sentiment analysis, text summarization과 같은 특정 도메인 및 작업에 맞게 adapted, fine-tuned 되었다. 이러한 적응성과 다양성 덕분에 BERT는 NLP의 다양한 분야에서 일하는 연구자와 실무자를 위한 강력한 도구가 되었다. 전반적으로 BERT 모델은 언어 이해 및 처리를 위한 강력하고 유연한 도구를 제공함으로써 NLP 분야에 혁명을 일으켰다. 그 영향은 다양한 NLP 작업에서 최첨단 결과를 달성하기 위해 그 기능을 활용할 수많은 응용 프로그램 및 연구 연구에서 분명하다.

2.4. GPT-3.5-turbo

GPT(Generative Pre-trained Transformer)는 OpenAI에서 개발한 자연어 처리 모델로 트랜스포머 구조를 기반으로 한다. GPT는 대량의 텍스트 데이터를 사전학습시켜 문맥과 의미를 이해해야 하는 다양한 자연어 처리 작업에 사용될 수 있는 다목적 언어 모델이다. 그 중 본 프로젝트에서 활용하는 GPT-3.5는 GPT-3 모델을 기반으로 한 최신 버전이다. 대규모의 텍스트 데이터로 사전 훈련되었으며, 매우 다양한 영역과 주제에 대해 일반적인 언어 이해와 생성 능력을 갖추고 있다. 예를 들어, 질문 답변, 자동 번역, 자연어 이해, 콘텐츠 생성, 대화 시스템 등 다양한 자연어 처리 작업에 활용될 수 있다. 이전 버전인 GPT-3과 비교하여 GPT-3.5가 더 발전한 점은 더 큰 규모와 매개 변수의 수를 갖추고 있다는 점이다. 이는 GPT-3.5가 더 많은 양의 데이터로 사전훈련 되었고 그 결과로 더 정교하고 다양한 언어 작업에 대한 지식을 보유하고 있음을 의미한다. GPT-3.5는 이전 모델보다 더욱 자연스러운

문장 생성, 더 복잡한 질문에 대한 대답, 상세한 문맥 이해 등 다양한 **NLP** 작업에서 좋은 성능을 발휘한다. 더하여 특정 작업이나 도메인에 맞춰 세부 조정과 추가적인 훈련을 **GPT-3.5** 모델에 실시하면 모델을 더욱 정확하고 원하는 작업에 유용하게 만들 수 있다.



2.5. 챗봇

기존 챗봇 서비스가 제공하는 기능은 다음과 같다.

- (1) 대화 이해와 의도 파악: 기존 챗봇은 사용자의 입력을 이해하고 의도를 파악하는 능력이 중요하다. 자연어 처리 기술과 머신러닝 알고리즘을 활용하여 사용자의 질문이나 요청에 대해 적절한 응답을 생성하기 위해 연구되고 개발된다.
- (2) 대화 생성: 기존 챗봇은 자연스러운 대화를 생성하는 능력을 갖추어야 한다. 이를 위해 자연어 생성 모델, 대화 트리 기반 모델, 템플릿 기반 모델 등 다양한

방식을 통해 대화를 생성한다. 적절한 응답을 생성하기 위해 문맥 이해, 문체 조절, 상황 인식 등의 기술을 활용한다.

- (3) 사용자 모델링과 개인화: 기존 챗봇은 사용자의 특성과 선호도를 파악하여 맞춤형 서비스를 제공하는 능력을 갖추고자 한다. 사용자 모델링은 사용자의 프로필, 행동 기록, 피드백 등을 분석하여 사용자에게 맞춤형 정보나 추천을 제공하는 기술이다. 개인화 기술은 대화의 맥락에서 사용자에게 관련 정보를 제공하거나 대화 스타일을 반영하여 자연스러운 대화를 구현한다.

2.6. 기존 연구의 문제점 및 해결 방안

기존 연구들도 많은 노력과 시간이 투입되었지만, 몇 가지 문제점이 있었다. 여기에는 몇 가지 문제점과 해당 문제점을 해결하기 위한 해결 방안을 소개한다.

- (1) 감정 분석의 정확도 부족: 기존 연구에서 사용된 감정 분석 모델은 정확도 측면에서 한계가 존재했다. 이로 인해 사용자 입력 문장의 감정을 정확하게 이해하지 못하거나 잘못된 감정을 응답 문장에 부여하는 경우가 종종 발생했다. 사용자의 감정을 정확하게 파악하는 것이 중요하므로, 이를 개선할 필요가 있다.
- (2) 응답 문장에서의 감정 반영 부족: 기존 연구에서는 감성 분석 결과를 활용하여 응답을 생성하는 과정에서 감정을 충분히 반영하지 못하는 문제가 있다. 챗봇의 응답은 사용자의 감정을 고려하여 자연스럽게 적절하게 이루어져야 하는데, 그러나 기존 연구에서는 감정 분석 결과를 응답에 적절하게 반영하지 못하여 사용자와의 상호작용에서 부자연스러움을 초래하는 문제점이 존재한다.
- (3) 다양한 감정 처리의 어려움: 사용자의 감정은 다양한 형태와 표현으로 나타날 수 있는데, 기존 연구에서는 주로 긍정, 부정, 중립 등과 같은 단순한 감정 범주만을 다루는 경우가 많았다. 이로 인해 사용자의 복잡한 감정 상태나 세부적인 감정을 제대로 이해하고 처리하는 점에 어려움이 존재한다.

- (4) 사회적 소외 감정의 무시: 일부 기존 연구는 일상적인 감정 범주에만 초점을 맞추고 사회적으로 소외된 감정에 대한 고려가 부족했다. 예를 들어, 우울감, 불안감, 외로움 등의 감정은 챗봇을 통해 표현 및 공유될 수 있는 중요한 감정이지만, 이러한 감정을 충분히 다루지 못한 기존 연구의 한계가 존재한다.

따라서 이러한 문제점을 극복하고자 본 프로젝트를 해결 방안으로써 제시한다. 본 프로젝트는 **BERT** 모델을 **Fine-tuning**하여 사용자의 감정을 더욱 정확하게 분석할 수 있는 모델을 개발한다. 데이터의 다양성과 균형을 고려하여 모델을 학습시키고 성능을 향상시킨다. 또한 감정을 적절히 반영한 응답을 생성하기 위해 **GPT-3.5-Turbo** 모델과 파라미터들을 활용한다. 위의 해결 방안을 통해 본 프로젝트는 감성 분석 기술을 향상시키고, 감정을 적절하게 반영하는 응답을 생성하며, 다양한 감정을 다루고 사회적으로 소외된 감정에 대한 이해와 지원을 강화한다. 이를 통해 챗봇 기술의 감정 인식 및 상호작용 측면에서 기존 연구의 한계를 극복할 수 있다.

3. 제안하는 프로젝트 소개

3.1. 시나리오

- (1) 사용자가 감성 대화 챗봇 서비스에 접속한다.
- (2) 사용자는 입력란에 대화를 시작할 문장을 입력하고, 해당 문장은 서버에 있는 **BERT** 모델로 전달되어 입력 문장의 감정을 분류하고 해당 감정을 판단하는 감정 분석이 수행된다.
- (3) 분석된 감정 정보가 **GPT** 모델에 전달되어 감정을 고려한 응답 문장을 생성한다. **GPT** 모델은 분석된 감정을 바탕으로 사용자에게 자연스러운 응답을 제공한다.

- (4) 생성된 응답 문장이 사용자에게 출력되고, 사용자는 해당 생성된 문장에 대한 반응을 보낸다. 사용자의 반응은 다시 **BERT** 모델로 전달되어 감정 분석이 수행된다. 분석된 감정 정보가 **GPT** 모델에 전달되어 새로운 응답 문장을 생성한다.
- (5) 위 과정이 반복되며, 사용자와의 대화가 계속 진행된다. 사용자가 대화를 마치고 서비스를 종료하면, 챗봇과의 상호작용이 종료된다.

위 시나리오에서 감성 대화 챗봇은 사용자의 입력 문장을 받아 **fine-tuned BERT** 모델을 사용하여 감정을 분석하고, 분석된 감정에 따라 **GPT** 모델을 사용하여 응답 문장을 생성한다. 이를 통해 챗봇은 사용자의 감정을 인식하고 그에 맞는 자연스러운 대화를 제공할 수 있다.

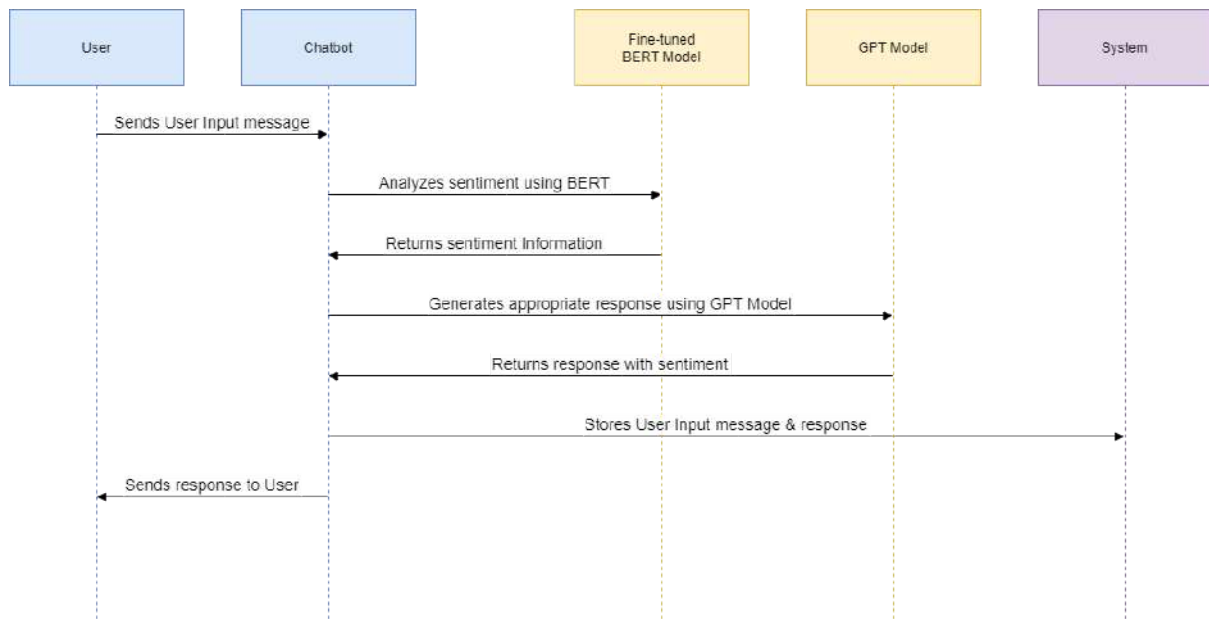
3.2. 요구사항

- (1) **BERT model training**: 수집된 **dataset**에 적절한 **training algorithm**을 사용하여 **BERT model**을 **training**해야 한다. 이 단계에서는 **pre-trained BERT model**을 **fine-tuning**하여 감정분석이라는 특정 작업에 맞게 조정한다.
- (2) **감정 분석 기능**: **Fine-tuned BERT** 모델을 통해 사용자 입력 문장의 감정을 분석할 수 있어야 한다. 입력 문장의 감정을 정확하게 분류하고 판단할 수 있어야 한다.
- (3) **응답 생성 기능**: **GPT** 모델을 활용하여 **BERT** 모델이 분석한 감정을 고려한 자연스러운 응답 문장을 생성할 수 있어야 한다. 생성된 응답은 사용자의 감정과 상황에 맞게 적절하고 의미 있는 내용을 담고 있어야 한다.
- (4) **대화 흐름 관리**: 사용자와의 대화 흐름을 관리할 수 있어야 한다. 이전 대화 내용과 사용자의 감정 분석 결과를 모두 고려하여 상황에 맞는 적절하고 의미 있는 내용을 가진 응답을 생성해야 한다.

- (5) 사용자 인터페이스: 사용자가 쉽게 서비스에 접근하고 대화할 수 있도록 직관적이고 **User-friendly**한 **interface**를 만들어야 한다. **Interface**는 사용자의 입력을 받을 수 있는 입력란과 서비스로부터의 응답을 표시할 수 있는 출력 영역이 있어야 한다.
- (6) **Quality assurance**: 정확한 감정 분석과 자연스러운 응답 생성을 위해 품질 보증 절차를 수립해야 한다. 시스템에서 생성된 감정을 담은 대화가 적절하며 유용한지를 확인해야 한다. 감성 대화 챗봇의 정확성과 효과를 개선하기 위해 정기적인 품질 점검을 실시하고 사용자로부터 피드백을 구해야 한다.
- (7) **Data privacy**: 고객으로부터 수집된 데이터가 데이터 보호법에 따라 안전하게 처리되도록 해야 한다. 적절한 데이터 보안 조치를 구현하고, 데이터 수집 및 처리에 대해 고객으로부터 정보에 입각한 동의를 얻는다.
- (8) 성능: 실시간 대화에 최적화된 성능을 갖추어야 한다. 응답 시간이 빠르고, 대용량 처리에도 효율적으로 대응할 수 있어야 한다.
- (9) 확장성: 모델 업데이트 및 추가 학습을 위한 확장성을 갖추어야 한다. 새로운 데이터를 추가하여 모델을 학습시킬 수 있는 기능이 있어야 한다.

3.3. Sequence Diagram

챗봇 서비스의 **Sequence Diagram**은 다음과 같다. 사용자로부터 **Input message**를 입력으로 받아서, **BERT** 모델로 전달하여 감정을 분석한 결과를 받고, 해당 내용을 **GPT** 모델에 전달하여 감정이 담긴 응답 문장을 반환 받는다. 매 생성마다 **User Input message**와 **response**를 시스템에 저장하고, 생성한 응답을 사용자에게 전달한다. 해당 과정은 사용자가 명시적으로 챗봇과의 대화를 종료할 때까지 계속 반복된다.



4. BERT 파인튜닝

4.1. 모델 소개

이 파트에서는 BERT를 활용한 감정 분석 모델의 구현과 평가 결과를 제시한다. BERT(Bidirectional Encoder Representations from Transformers)는 자연어 처리를 위한 사전 학습(pre-trained) 모델 중 하나이다. 이 모델은 Google이 개발한 전이 학습 방법으로, Transformer라는 양방향 인코더를 기반으로 한다. BERT는 언어의 다양한 문맥과 의미를 이해하기 위해 양방향으로 문장을 인코딩하는 능력을 갖추고 있다. BERT는 사전 학습과 후속 작업(task-specific fine-tuning)의 두 단계로 구성된다. 사전 학습 단계에서는 대규모의 텍스트 데이터를 사용하여 모델의 파라미터가 사전 학습된다. 후속 작업 단계에서는 사전 학습된 BERT 모델을 다양한 자연어 처리 작업에 맞게 세부적으로 조정(fine-tuning)한다. 이를 위해 해당 작업에 맞는 데이터셋으로 모델을 추가로 학습시키는 과정을 거친다. 감정 분석, 문장 분류, 질문-답변 등 다양한 자연어 처리 작업에 BERT를 적용할 수 있다. BERT는 사전

학습된 언어 모델의 일종으로, 큰 양의 텍스트 데이터를 기반으로 다양한 자연어 처리 작업에 적용할 수 있는 강력한 기능을 제공한다. 이 모델은 언어의 문맥을 이해하고, 문장의 의미를 추론하며, 문장 간의 관계를 파악하는 데에 매우 유용하다. 따라서 BERT를 활용한 감정 분석 모델은 효과적으로 감정을 인식하고 분류하는 데에 활용될 수 있다.

4.2. 데이터 전처리

4.2.1. 데이터 불러오기

BERT를 사용한 감정 분석 **fine-tuning**을 위해, 문장과 문장의 감정에 대한 많은 데이터가 필요했다. 따라서, 이를 충족하는 AI-Hub의 ‘감성 대화 말뭉치’ 데이터(Excel 파일)를 사용했다. 원래 데이터 구조는 [연령, 성별, 상황키워드, 신체질환, 감정_대분류, 감정_소분류, 사람문장1, 시스템문장1, 사람문장2, 시스템문장2, 사람문장3, 시스템문장3]로 이루어져 있다. 필요한 데이터는 ‘문장’과 그 문장의 상세한 ‘감정’ 두 개 이므로, [감정_소분류, 사람문장 1~3]를 선택했다. 사람문장을 ‘sentence’로, 감정_소분류를 ‘sub_sentiment’로 지정하였다. (만드는 과정에서 감정_대분류도 테스트하며 참고하였기 때문에, sentiment로 추출되어 있다.) 아래는 Excel 데이터를 Pandas의 ‘read_excel’ 함수를 통해 Dataframe 형태로 바꾼 데이터셋의 일부이다.

	sentence	sentiment	sub_sentiment
	아빠에게 나를 믿어달라고 말할 거야.	슬픔	낙담한
	뿌듯하고 기쁜 마음을 오래 기억하고 싶어.	기쁨	기쁨
	매장 내부도 다시 보고 재료를 확인할 거야. 청결하게 관리해야지.	기쁨	신이 난
	아무래도 가족들과 대화를 좀 해 봐야겠어. 왜 그러는지.	분노	분노
	맞아. 날 갑자기 날 괴롭히기 시작해서 괴로워.	당황	당황
	저녁 먹고 아내랑 산책하는 게 요즘 사는 낙이야.	기쁨	편안한
	맛있는 저녁을 먹고 바람 좀 쐬려고 해.	슬픔	낙담한
	조금씩 배웠어도 이 지경까지 되진 않았을 텐데 너무 후회돼.	상처	괴로워하는
	얼마 전에 암 수술을 했어. 몸이 안 좋아서 일할 수가 없으니 생활이 너무 어렵네.	상처	버려진
	나 오래 연애했던 여자친구와 헤어졌어. 버려진 기분이 들어서 슬퍼.	상처	버려진

4.2.2. 결측값 및 중복 샘플 제거

```
# 결측값 및 중복 샘플 제거
def drop_na_and_duplicates(df, col):
    df = df.dropna(how='any')
    df = df.drop_duplicates(subset=[col])
    df = df.reset_index(drop=True)
    return df

dataset = drop_na_and_duplicates(dataset, dataset.columns[0])
```

불러온 데이터에서 결측값(null)을 가진 행을 제거하기 위해 'dropna' 함수를 사용하여 처리하였다. 이렇게 하여 데이터의 누락을 방지하고, 분석 작업에서의 오류를 예방했다. 또한, 'drop_duplicates' 함수를 활용하여 'sentence' 열에서 중복된 샘플을 찾아 제거했다. 이를 통해, 중복으로 인한 데이터 왜곡을 방지하였다. 이렇게 정제한 후의 결과로는 총 144,723개의 데이터가 남았다. 이는 충분한 양의 데이터로, 신뢰성 있는 감정 분석 모델의 학습에 적합하다고 할 수 있다. 데이터의 품질을 향상시키고 정확한 분석 결과를 얻기 위해, 결측값과 중복 샘플을 처리한 과정은 매우 중요했다.

```
print(f'총 데이터 개수: {len(dataset)}')
```

총 데이터 개수: 144723

4.2.3. 라벨링 데이터와의 조인

sub_sentiment, 즉, 감정 소분류는 총 58개의 감정을 포함하고 있다. 이 58개의 감정은 아래 사진에 나와 있는 것과 동일하다. 각각의 감정은 1,500~2,500개의 데이터로 골고루 분포되어 있다.

```
sub_sentiment_cnt = dataset['sub_sentiment'].nunique()
print(f'감정 라벨 개수: {sub_sentiment_cnt}')
```

감정 라벨 개수: 58

가난한, 불우한	느긋	불안	안도	짜증내는	후회되는
감사하는	당혹스러운	비통한	억울한	초조한	흥분
걱정스러운	당황	상처	열등감	충격 받은	희생된
고립된	두려운	성가신	염세적인	취약한	
괴로워하는	마비된	스트레스 받는	외로운	툴툴대는	
구역질 나는	만족스러운	슬픔	우울한	편안한	
기쁨	방어적인	신뢰하는	자신하는	한심한	
낙담한	배신당한	신이 난	조심스러운	혐오스러운	
남의 시선을 의식하는	버려진	실망한	좌절한	혼란스러운	
노여워하는	부끄러운	악의적인	죄책감의	환멸을 느끼는	
눈물이 나는	분노	안달하는	질투하는	회의적인	

BERT Classification을 위해 분류를 숫자로 변환해야 했다. 따라서, 감정을 숫자로 라벨링하기 위해, 0부터 57까지의 라벨로 지정한 labeling Dataframe과 앞서 만든 dataset Dataframe을 Join 했다. dataset의 sub_sentiment 열과 labeling의 sentiment 열을 기준으로 조인해서, [sentiment, label] 데이터프레임을 완성시켰다. 아래는 데이터셋의 일부이다.

	sentence	label
	난 병원 신세라 언제 떠날지 모르는데 빛과 함께 남을 아들을 생각하니 안타까워.	18
	일도 못하고 이렇게 병원신세만 지다 죽을까 두려워. 모든 게 후회스러워.	13
	오늘 딸이 내 발에 걸려 넘어지는 바람에 울어서 놀랐어.	13
	코딩 개념 자체를 아무리 설명을 들어도 머리만 아프고 이해를 못하겠어.	52
	지저분하고 더러워서 못 살겠어.	5
	그래서 요즘 근래 모임도 같이 나가고 자주 보기는 해.	2
	남편이 같이 점심 먹자고 직장으로 찾아온 적은 처음이라 당황스러워.	13
	그래야지. 먹는 양 줄이고 운동 시작할 거야. 처방약도 꾸준히 복용해야지.	42
	내가 돈이 없으니 자식들도 내 말을 듣지 않고 나를 괘시하는 게 느껴져서 환멸이 나.	53
	어머니 십 주기 추도식에 다녀왔어. 벌써 시간이 그렇게나 흘렀네.	34

4.2.4. 최대 문장 길이 설정


```

for index, value in dataset['sentence'].items():
    mylist[len(value)] += 1
    max_seq_len = max(max_seq_len, len(value))
    if len(value) >= 64 and len(value) < 128:
        len_64_128 += 1
    elif len(value) >= 128 and len(value) < 256:
        len_128_256 += 1

print(f'가장 긴 문장 길이: {max_seq_len}')
print(f'64과 128 사이 길이 문장 개수: {len_64_128}')
print(f'128과 256 사이 길이 문장 개수: {len_128_256}')

```

가장 긴 문장 길이: 156
 64과 128 사이 길이 문장 개수: 2374
 128과 256 사이 길이 문장 개수: 3

BERT에서는 최대 문장 길이(max sequence length: max_seq_len)를 일반적으로 64, 128, 256 등으로 설정한다. 이 데이터셋에서 가장 긴 문장의 길이는 156이다. 그러나, 길이가 128 초과 256 미만인 문장은 세 개로 매우 적다. 이에 따라, max_seq_len을 256으로 설정하면 padding이 많아져 메모리 사용량이 증가하고 성능도 저하될 수 있다. 그래서 적절하게 128로 설정했다. 길이가 128을 초과하는 문장은 자동으로 잘려서 입력에 들어가게 된다. BERT 모델은 문장의 전반적인 의미를 이해하고 학습하기 때문에, 일반적으로 문장의 앞부분이 더 중요한 정보를 담고 있다. 따라서, 길이가 128을 넘어가는 부분은 잘려도 문장의 핵심 내용은 유지될 수 있다. 이를 고려하여 적절한 최대 문장 길이를 설정하여 메모리 사용량을 효율적으로 관리하고 성능을 유지하는 것이 중요하다.

4.2.5. train/validation/test 데이터 분할

훈련 집합(train set), 검증 집합(validation set), 테스트 집합(test set)은 머신 러닝 모델을 학습하고 평가하기 위해 데이터를 분할하는 과정이다. 훈련 집합(train set)은

모델을 학습하는 데 사용된다. 모델은 훈련 집합의 예제와 레이블을 사용하여 패턴을 학습하고 파라미터를 조정한다. 손실 함수를 최소화하고 최적의 가중치를 찾기 위해 훈련 집합을 사용한다. 검증 집합(validation set)은 모델의 성능을 평가하고 최적의 하이퍼파라미터를 선택하는 데 사용된다. 모델은 학습 중에 검증 집합의 예제와 레이블을 사용하여 성능을 평가한다. 이를 통해 과적합(overfitting)을 방지하고 일반화 성능을 추정한다. 모델의 학습 과정에서 검증 집합의 성능을 기준으로 조기 종료(early stopping) 등의 결정을 내릴 수 있다. 테스트 집합(test set)은 최종 모델의 성능을 평가하기 위해 사용된다. 훈련과 검증을 마친 후, 모델의 성능을 신뢰성 있게 측정하기 위해 테스트 집합을 사용한다. 이는 모델이 이전에 보지 못한 데이터에 대해 얼마나 잘 일반화되는지 평가하는데 사용된다. 데이터셋을 분할할 때 일반적인 규칙은 train, validation, test 데이터셋을 비율로 나누는 것이다. 일반적으로는 6:2:2의 비율로 분할되며, 훈련 집합은 전체 데이터의 약 60-80%를 차지하고, 검증 집합과 테스트 집합은 각각 전체 데이터의 약 10-20%를 차지한다. 이러한 비율은 일반화 성능을 최적화하기 위한 권장 비율이지만, 실제로는 데이터의 특성과 상황에 따라 조정될 수 있다.

```
from sklearn.model_selection import train_test_split

# train/validation/test dataset 분할
# train:val:test = 6:2:2
train_X, temp_X, train_y, temp_y = train_test_split(dataset_X, dataset_y,
test_size=0.4, random_state=0)
val_X, test_X, val_y, test_y = train_test_split(temp_X, temp_y,
test_size=0.5, random_state=0)
```

BERT fine-tuning에 사용할 데이터셋은 총 144,723개의 데이터로 구성되어 있다. 위에서 언급한 권장 분할 비율 6:2:2에 따라 데이터셋을 분할했다. 데이터 셋을 분할하기 위해 'sklearn.model_selection' 모듈의 'train_test_split' 함수를 사용했다. 이를 통해 train_X, train_y, val_X, val_y, test_X, test_y로 분할된 데이터를 얻을 수 있었다. 또한, 분할 과정에서 'random_state' 파라미터를 사용하여 난수 발생기의

시드(seed)를 설정할 수 있다. 시드를 설정하면, 동일한 시드로 실행할 때마다 항상 동일한 분할 결과를 얻을 수 있어, 실험의 재현성을 보장할 수 있다. 데이터가 감정별로 정렬되어 있었기 때문에 데이터를 무작위로 섞는 작업이 필요했다. 따라서, 특정 하이퍼파라미터 값을 변경하면서 최적의 값을 찾는 실험을 수행하고 일관성 있는 실험 결과를 얻기 위해 'random_state' 값을 0으로 설정했다.

```
print(f'train set length: {len(train_X)}')
print(f'valid set length: {len(val_X)}')
print(f'test set length: {len(test_X)}')
```

```
train set length: 86833
valid set length: 28945
test set length: 28945
```

4.2.6. 데이터셋 토큰나이저

```
from transformers import BertTokenizerFast

tokenizer = BertTokenizerFast.from_pretrained("klue/bert-base",
max_len=max_seq_len, truncation=True, padding=True)

train_X = tokenizer(train_X, truncation=True, padding=True)
val_X = tokenizer(val_X, truncation=True, padding=True)
```

BERT 모델에 입력으로 사용될 데이터를 tokenize하기 위해 'transformers' 라이브러리의 'BertTokenizerFast'를 활용했다. 이 과정에서 'klue/bert-base'와 같은 사전 훈련된 BERT 모델의 tokenizer를 사용했다. 해당 tokenizer를 활용하여 앞서 분할한 훈련 데이터셋의 입력 데이터(train_X)와 검증 데이터셋의 입력 데이터(val_X)를 전처리했다. 이를 통해 문장을 토큰화하고 특수 토큰(시작, 종료, 패딩)을 추가하여 BERT 모델의 입력 형식에 맞게 변환했다. 또한, 훈련 데이터셋과 검증 데이터셋의 예측값(train_y, val_y)과 함께 (train_X, train_y), (val_X, val_y)를

쌍으로 묶고, 이를 **dictionary** 형태로 변환하여 데이터셋에 전달했다. 이렇게 구성된 데이터셋은 BERT 모델의 입력으로 사용될 수 있다.

4.3. 모델 파인 튜닝

```
from transformers import TFBertForSequenceClassification

optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)

model = TFBertForSequenceClassification.from_pretrained("klue/bert-base",
num_labels=58, from_pt=True)
model.compile(optimizer=optimizer, loss=model.hf_compute_loss, metrics=
['accuracy'])
```

BERT를 기반으로 한 **sequence classification** 모델을 학습하기 위해 ‘transformers’ 라이브러리의 ‘TFBertForSequenceClassification’ 모델을 활용했다. 이 과정에서 사전 훈련된 BERT 모델 중 하나인 ‘klue/bert-base’의 모델을 불러왔다. **tokenize**에 사용한 모델과 동일한 모델을 불러와야 호환성이 유지된다. 모델 학습에는 ‘Adam’이라는 현재 가장 많이 사용되는 최적화 알고리즘(**optimizer**)를 사용하였고, 학습률(**learning rate**)로는 **2e-5(0.00002)**를 설정했다. 후술할 결과 분석에서 학습률에 따른 결과를 분석할 예정이다. ‘**model.compile()**’을 사용하여 모델을 컴파일할 때, 위에서 정의한 **optimizer**를 전달하고, 손실 함수(**loss function**)로는 모델의 ‘**hf_compute_loss**’를 사용했으며, 정확도를 측정하기 위해 ‘**accuracy**’를 설정했다.

```
callback_earlystop = tf.keras.callbacks.EarlyStopping(
    monitor="val_accuracy",
    min_delta=0.001,
    patience=0
)
```

검증 세트의 정확도를 모니터링하고, 변화량(min_delta)이 0.001 이상 없을 경우 (patience) 조기 종료하는 'EarlyStopping' callback을 생성했다. 변화량이 0.001 이상 없는 경우가 관찰되지 않았기 때문에, patience를 0으로 설정했다. (patience를 n(n≠0)으로 설정하면, 변화량이 0.001 이상 없는 경우가 n번 발생할 때까지 학습을 종료하지 않는다.) 훈련 데이터셋과 검증 데이터셋은 'shuffle()' 함수를 사용하여 데이터를 섞은 후, 'batch()' 함수를 사용하여 배치 크기를 64로 설정했다.

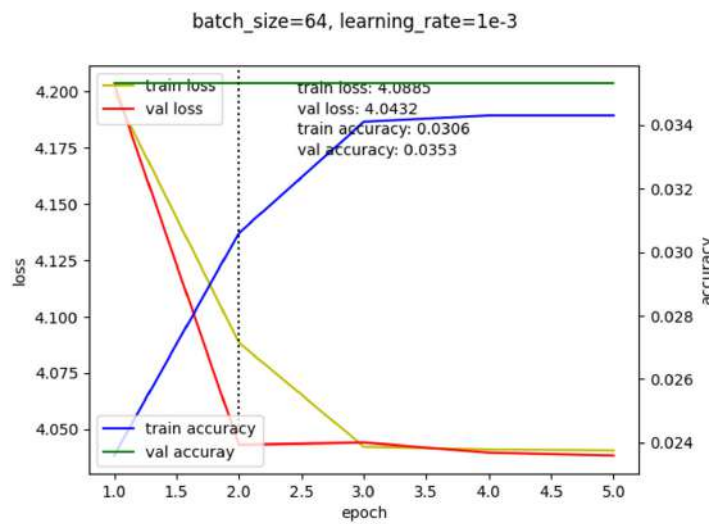
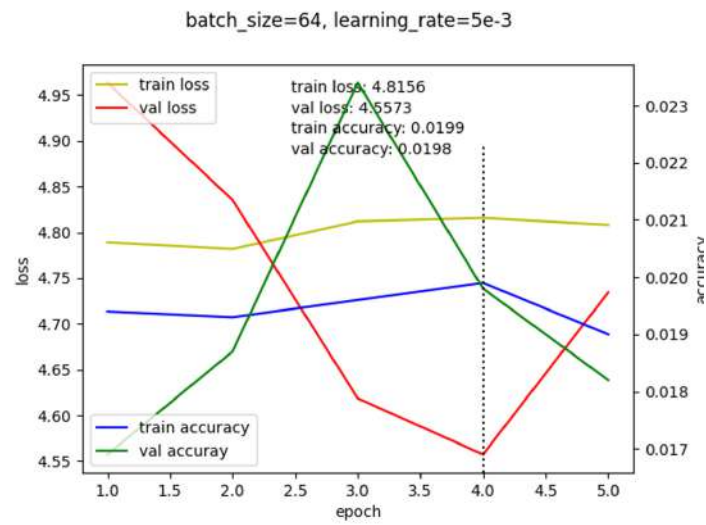
```
model.fit(
    train_dataset, epochs=3, batch_size=64,
    validation_data = val_dataset,
    callbacks = [callback_earlystop]
)
```

최종적으로 전처리를 거친 train_dataset과 val_dataset, 불러온 모델, 정의한 callback을 활용하여 'model.fit()'으로 모델을 훈련했다. epoch은 3으로 설정하고, batch size는 64로 설정했다. epoch와 batch에 따른 결과 분석은 후술할 예정이다. 아래는 모델 훈련 결과 중 하나의 예시이다.

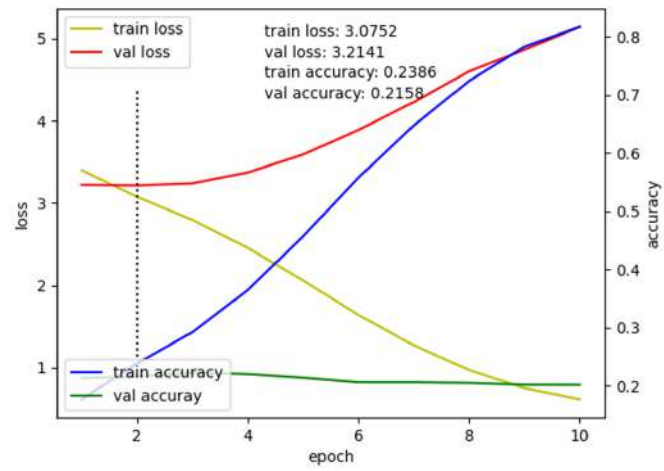
```
Epoch 1/3
1357/1357 [=====] - 264s 157ms/step - loss: 3.4731 - accuracy: 0.1651
- val_loss: 3.2163 - val_accuracy: 0.2168
Epoch 2/3
1357/1357 [=====] - 194s 143ms/step - loss: 3.0988 - accuracy: 0.2361
- val_loss: 3.1422 - val_accuracy: 0.2283
Epoch 3/3
1357/1357 [=====] - 193s 142ms/step - loss: 2.8891 - accuracy: 0.2742
- val_loss: 3.1452 - val_accuracy: 0.2327
```

아래 내용은 최적의 하이퍼파라미터 설정을 위해 값을 조정하면서 모델을 학습시켰을 때의 분석 결과이다. validation loss가 감소하다가 증가하기 시작하는 시점에서 overfitting이 발생하므로, 해당 지점의 validation accuracy를 결과값으로 설정했다.

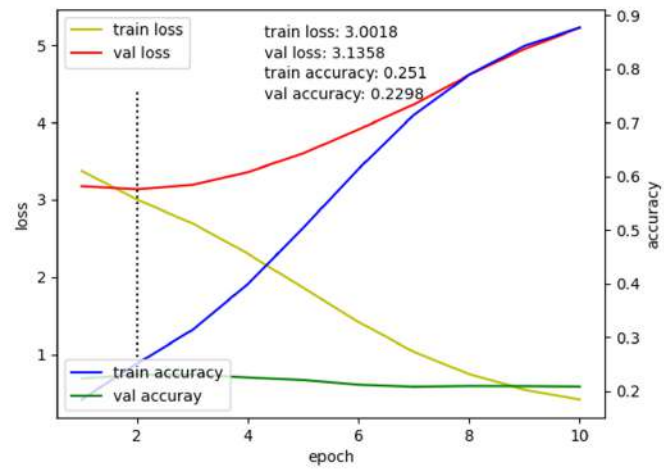
4.3.1. 학습률 설정



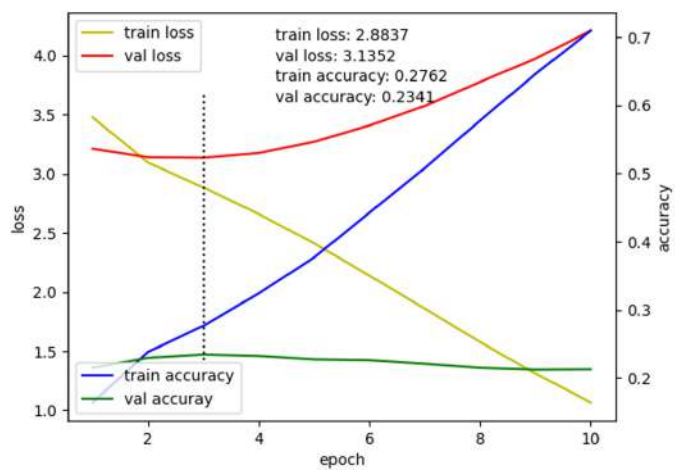
batch_size=64, learning_rate=1e-4



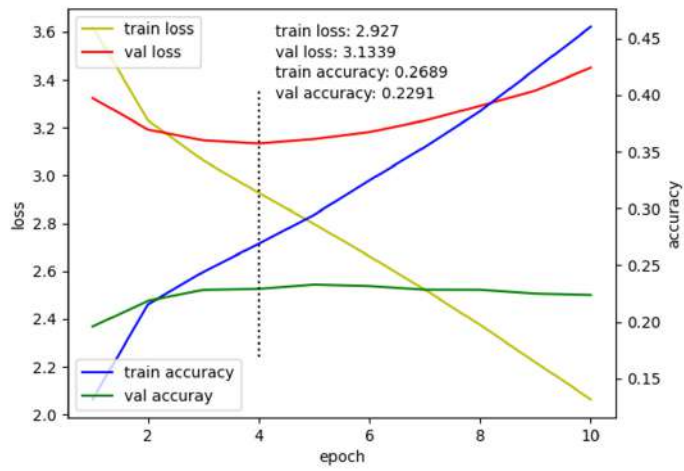
batch_size=64, learning_rate=5e-5



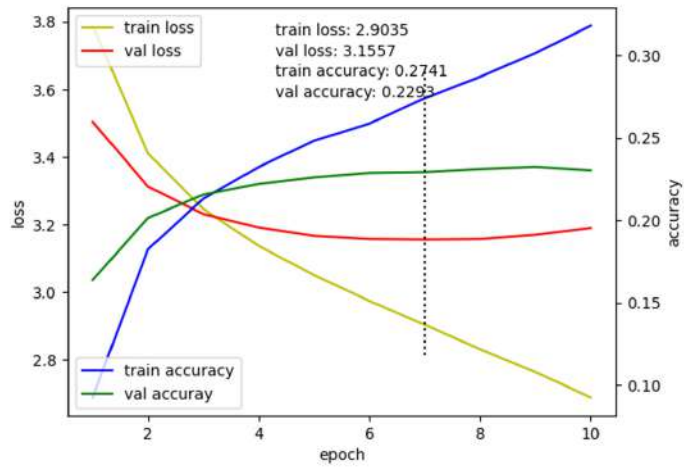
batch_size=64, learning_rate=2e-5



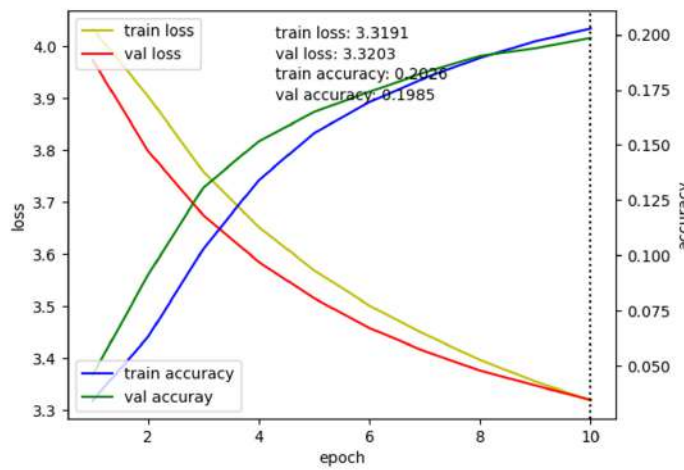
batch_size=64, learning_rate=1e-5

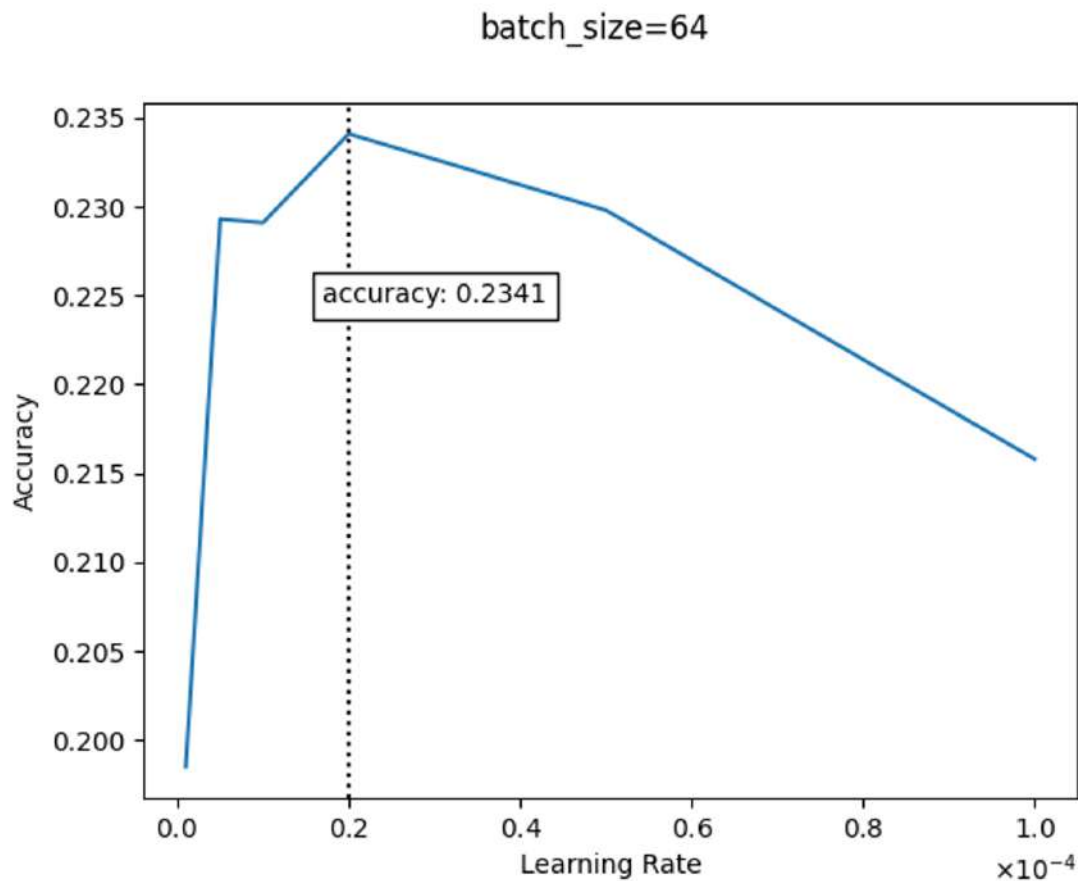


batch_size=64, learning_rate=5e-6



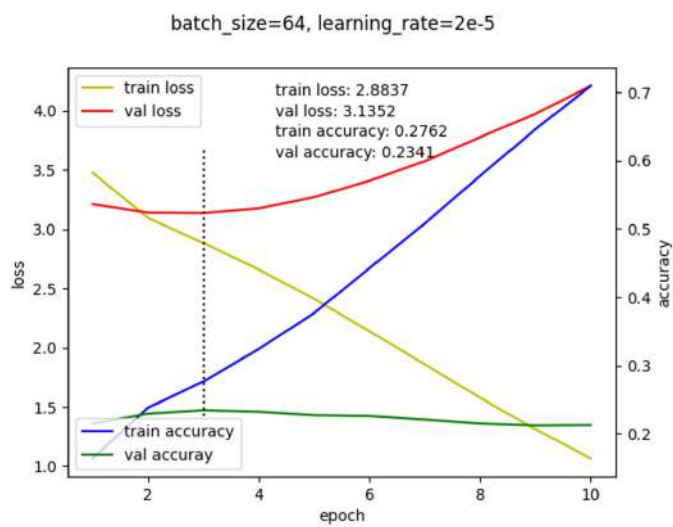
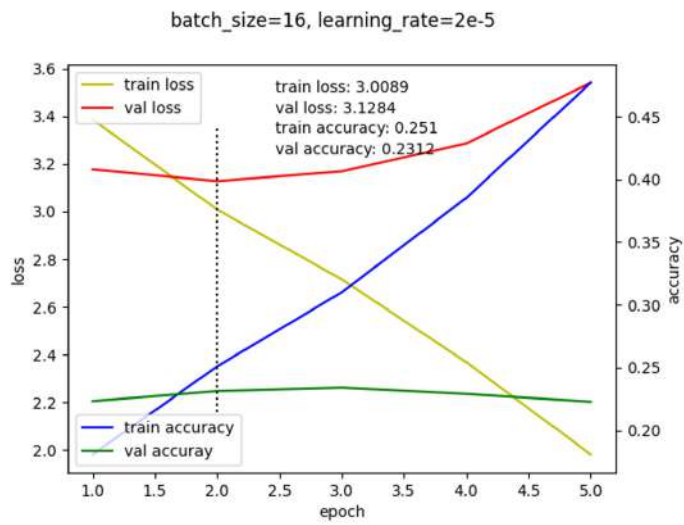
batch_size=64, learning_rate=1e-6

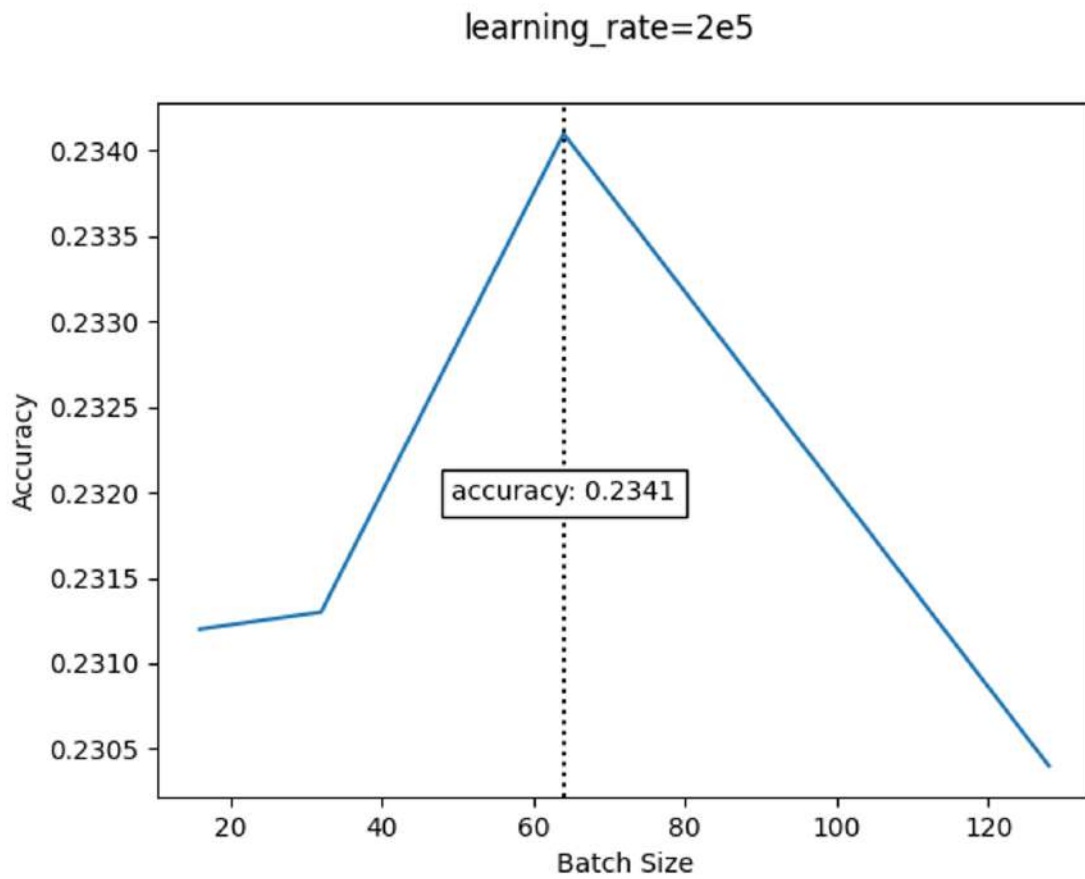
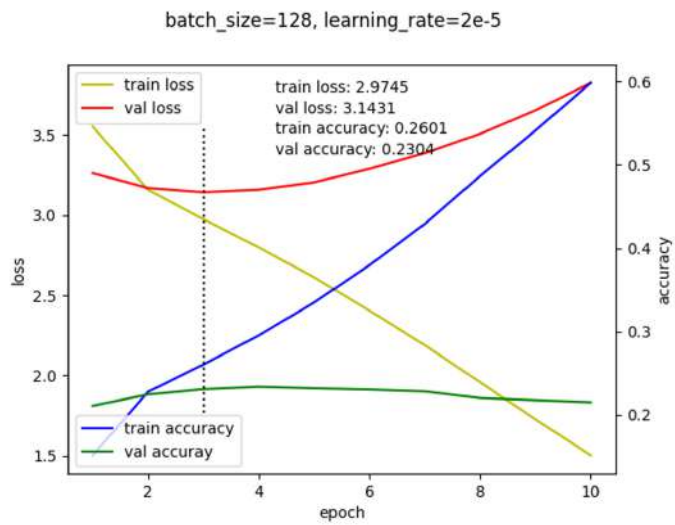




다른 모든 파라미터를 고정시킨채로 **learning rate**만 변화시켜서 **epoch=10**으로 모델을 학습시켰다. 결과적으로 **learning rate**가 **2e-5**일 때 **accuracy**가 **0.2341**로 가장 높음을 확인할 수 있었다. 이를 토대로 최종적으로 **learning rate** 값을 **2e-5**로 설정했다. 추가적으로, 10 이후의 **epoch**에서는 유의미한 변화가 발견되지 않았으므로, 최대 **epoch**을 10으로 설정했다.

4.3.2. 배치 사이즈 설정





다른 모든 파라미터를 고정시킨채로 **batch size**만 변화시켜서 **epoch=10**으로 모델을 학습시켰다. 결과적으로 **batch size**가 64일 때 **accuracy**가 0.2341로 가장 높음을 확인할 수 있었다. 이를 바탕으로 최종적으로 **batch size** 값을 64로 설정했다.

추가적으로, 10 이후의 epoch에서는 유의미한 변화가 발견되지 않았으므로, 최대 epoch을 10으로 설정했다.

4.3.3. 모델 정확도 분석

sentiment classification을 위해 fine-tuning된 BERT 모델의 최대 정확도는 약 23.41%로 측정됐다. 반면, fine-tuning 이전의 pre-trained 모델 'klue/bert-base'는 task classification(TC)에서 85.73%의 정확도를 보여준다. 이 TC 작업은 두 개의 라벨(0과 1)만을 가지고 있기 때문에, 50%의 랜덤한 분류보다 우수한 성능을 보인다고 볼 수 있다. 그러나, 우리가 개발한 모델은 총 58개의 라벨을 가지고 있다. 따라서, $1/58 \approx 0.0172$ (약 1.72%)보다 훨씬 우수한 23.41%의 성능을 보여주고 있다. 실제로 58개의 감정을 사람이 정확하게 구분하는 기준은 없으며, 개인마다 주관적인 감정 분류가 다를 수 있다. 따라서, BERT 모델의 출력 결과가 개인이 생각하는 정답(감정) 중에 포함된다면 맞다고 간주할 수 있다. 이에 따라, 실제로 결과 값을 확인해본 결과, BERT 모델이 상당히 합리적인 감정 출력을 제공함을 확인할 수 있었다. 아래는 최종 모델을 사용하여 도출해낸 결과 값인, 문장에서 추출한 감정의 예시이다.


조용히 해주세요 => ['짜증내는']
일 더하기 일은 귀요미 => ['신이 난']
아 수업 가기 싫다 => ['툭툭대는']
그래서 님 티어가? => ['열등감']
내 어린 시절 우연히 들었던 믿지 못할 한 마디 => ['배신당한']
늦었다고 생각할 때가 진짜 너무 늦었다 => ['후회되는']
면접에서 떨어졌어 => ['좌절하는']
중요한건 꺾이지 않는 마음 => ['자신하는']
자라 보고 놀란 가슴 솥뚜껑 보고 놀란다 => ['충격 받은']
과제가 너무 많아 => ['스트레스 받는']
끝렸죠? => ['신이 난']
아버지 이제야 깨달아요 어찌 그렇게 사셨나요 => ['비통한']
난 그렇게 생각하지 않아 => ['회의적인']

위 내용을 종합해보면, 우리가 fine-tuning한 BERT 모델이 다양한 감정 분류 작업에서 상당히 신뢰할만한 성능을 보여주었다고 할 수 있다.

4.4. 모델 저장

```
model.save_pretrained(MODEL_SAVE_PATH)
tokenizer.save_pretrained(MODEL_SAVE_PATH)
```

모델을 저장하기 위해 'model.save_pretrained()'와 'tokenizer.save_pretrained()' 함수를 사용했다. 저장한 모델은 다음과 같은 형태로 저장되었다.

이름	유형	크기
 vocab	텍스트 문서	243KB
 tokenizer_config	JSON 원본 파일	1KB
 tokenizer	JSON 원본 파일	735KB
 tf_model.h5	H5 파일	432,555KB
 special_tokens_map	JSON 원본 파일	1KB
 config	JSON 원본 파일	3KB

- (1) vocab.txt: 이 파일은 BERT 모델이 이해하고 처리할 수 있는 모든 단어와 토큰들의 목록을 담고 있다. 모델 학습 과정에서 사용된 어휘와 인덱스 매핑 정보가 포함되어 있다.
- (2) tokenizer_config.json: 이 파일은 문장을 단어 또는 토큰 단위로 분리하는 역할을 담당하는 BERT 토크나이저의 구성(config)을 저장한다. 여기에는 토큰화 방법, 특수 토큰 처리 방식 등과 같은 config가 포함되어 있다.
- (3) tokenizer.json: 이 파일은 BERT 토크나이저의 내부 상태를 저장하며, 토크나이저 객체를 로드하고 사용하기 위한 정보를 담고 있다. 따라서, 모델을 훈련한 토크나이저 상태를 저장하고 있으며, 새로운 문장을 토큰화할 때 필요한 정보를 제공한다.

- (4) `tf_model.h5`: 이 파일은 BERT 모델의 가중치(weight)를 저장한 HDF5 파일이다. 모델의 신경망 가중치는 훈련된 모델의 핵심 요소로, 문장의 의미를 이해하고 예측을 수행하는데 사용된다.
- (5) `special_tokens_map.json`: 이 파일은 문장의 시작, 끝, 패딩 등과 같이 특별한 의미를 가지는 토큰인 특수 토큰(special token)에 대한 매핑 정보를 담고 있다.
- (6) `config.json`: 이 파일은 BERT 모델의 구성 정보를 저장한다. 모델의 아키텍처, 레이어 수, 히든 유닛 수 등과 같은 구성 정보가 포함되어 있다. 따라서, 모델을 로드하고 구성 정보를 확인하는데 사용된다.

4.5. 모델 불러오기

```
# text classification을 위한 pipeline class import
from transformers import TextClassificationPipeline

# 저장했던 tokenizer, model을 불러옴
loaded_tokenizer = BertTokenizerFast.from_pretrained(MODEL_SAVE_PATH)
loaded_model = TFBertForSequenceClassification.from_pretrained(MODEL_SAVE_PATH)

# tokenizer, model을 pipeline에 할당
text_classifier = TextClassificationPipeline(
    tokenizer=loaded_tokenizer,
    model=loaded_model,
    framework='tf'
)

# pipeline 사용 예시
result = text_classifier("This is an example sentence")
predicted_label = result[0]['label']
predicted_score = result[0]['score']
```

모델을 불러오기 위해 'transformers' 라이브러리에서 텍스트 분류를 위한 파이프라인 클래스 'TextClassificationPipeline'을 import한다. 이 파이프라인을 사용하면, 텍스트 분류 모델을 쉽게 로드하고, 텍스트 입력에 대한 예측을 생성할 수 있다. 'BertTokenizerFast.from_pretrained()', 'TFBertForSequenceClassification.from_pretrained()' 함수를 사용하여 저장한 모델을 불러왔다. 이때, 파이프라인 클래스에는

불러온 `tokenizer`와 `model`을 할당하고, `framework`를 `'tf(tensorflow)'`로 설정했다. 이렇게 설정된 텍스트 분류 파이프라인(`TextClassificationPipeline`)은 입력 문장을 전달하여 해당 문장의 클래스 레이블 및 점수를 예측하는데 사용됐다.

4.6. 모델 예측

앞서 분리해서 만든 테스트 데이터셋(28945개)을 입력으로 사용해서, 이전에서 생성한 파이프라인을 사용하여 모델의 예측을 수행한다. 아래는 일부 예시이다.

	sentence	label	pred	score
	어학 성적도 있어야 하고 관련 분야 자격증들도 있어야지. 그런데 자격증들이 너무 많...	25	25	0.174657
	솔직하게 나의 단점을 미리 말해주는 것도 하나의 방법이겠지.	14	47	0.167723
	머칠 전에 무릎이 아파서 정형외과에 갔더니 퇴행성 관절염이라네.	52	52	0.060358
	응. 줄곧 그러지 말라고 얘기했었는데 이제는 스트레스받을 지경이야.	26	26	0.418709
	부모님이 돌아가시고 고생하고 있을 형제들을 생각하니 슬퍼.	38	10	0.254380
	내 체력에서 가능한 운동을 할 거야. 조금씩 하다 보면 건강도 좋아질 거야.	15	35	0.046843
	평소에도 나를 본체만체하는데 앞으로는 더 무시하겠지.	36	17	0.190723
	응. 이번 결혼기념일에 남편이랑 해외여행 가기로 했거든. 이십 주년 기념으로 가는데...	29	29	0.599084
	맞아 부모님께서 지금까지 나를 도와주시지 않았다면 난 취업에 성공하지 못 했을 거야.	1	1	0.535330
	우울증으로 인해 흥미도 없고 뭘 하든 힘들어.	26	40	0.586288

위와 같이 텍스트를 입력으로 주고, 모델은 해당 문장의 클래스 레이블과 예측 점수를 출력한다. 이를 통해, 모델이 주어진 문장에 대해 어떤 감정을 예측하는지 확인할 수 있다.

4.7. 모델 평가

모델을 평가하기 위해 `'sklearn.metric'` 라이브러리의 `'classification_report'` 함수를 사용했다. 이 함수를 사용하여 이전에서 도출된 결과를 각각 `'y_true'`와 `'y_pred'`에 대입시켜 평가 결과를 측정했다. 측정값은 다음과 같다.

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.22	0.27	0.24	528	30	0.23	0.26	0.25	478
1	0.37	0.52	0.43	352	31	0.25	0.25	0.25	496
2	0.16	0.26	0.20	646	32	0.19	0.13	0.15	505
3	0.19	0.30	0.24	979	33	0.33	0.43	0.37	366
4	0.25	0.18	0.21	534	34	0.35	0.27	0.30	574
5	0.26	0.20	0.23	500	35	0.21	0.15	0.18	483
6	0.21	0.23	0.22	383	36	0.18	0.11	0.13	480
7	0.21	0.12	0.15	536	37	0.31	0.28	0.29	539
8	0.25	0.26	0.26	459	38	0.26	0.26	0.26	577
9	0.12	0.09	0.11	548	39	0.27	0.39	0.32	305
10	0.28	0.25	0.26	552	40	0.23	0.26	0.24	438
11	0.24	0.31	0.27	298	41	0.15	0.09	0.11	508
12	0.43	0.18	0.25	491	42	0.21	0.40	0.27	492
13	0.28	0.18	0.22	461	43	0.26	0.34	0.30	504
14	0.23	0.24	0.24	599	44	0.15	0.30	0.21	555
15	0.14	0.13	0.14	490	45	0.24	0.17	0.20	485
16	0.29	0.23	0.26	350	46	0.18	0.24	0.21	503
17	0.16	0.10	0.12	427	47	0.12	0.14	0.12	457
18	0.27	0.26	0.26	562	48	0.19	0.11	0.14	512
19	0.13	0.17	0.15	466	49	0.34	0.33	0.33	315
20	0.36	0.28	0.31	485	50	0.32	0.22	0.26	481
21	0.19	0.20	0.19	502	51	0.28	0.16	0.20	458
22	0.22	0.16	0.19	477	52	0.19	0.26	0.22	1005
23	0.18	0.11	0.13	551	53	0.21	0.16	0.18	506
24	0.16	0.17	0.17	511	54	0.28	0.20	0.23	513
25	0.33	0.29	0.31	565	55	0.25	0.32	0.28	525
26	0.20	0.28	0.23	555	56	0.30	0.20	0.24	329
27	0.21	0.17	0.19	563	57	0.21	0.18	0.19	478
28	0.46	0.38	0.41	343					
29	0.33	0.31	0.32	365					
					accuracy			0.23	28945
					macro avg	0.24	0.23	0.23	28945
					weighted avg	0.24	0.23	0.23	28945

- (1) **precision**(정밀도): 정밀도는 “양성”으로 예측한 샘플 중에서 실제로 “양성”인 샘플의 비율을 나타낸다. 즉, 모델이 “양성” 클래스를 얼마나 정확하게 예측하는지를 나타내는 지표이다. 정밀도 공식은 다음과 같다.
‘(정밀도)= $TP/(TP+FP)$ ’. 여기서 **TP**는 “양성”으로 예측하고 실제로 “양성”인 샘플의 수이고, **FP**는 “양성”으로 예측했지만 실제로는 “음성”인 샘플의 수이다.
- (2) **recall**(재현율): 재현율은 실제 “양성”인 샘플 중에서 모델이 “양성”으로 예측한 샘플의 비율을 나타낸다. 즉, 모델이 실제 “양성”인 샘플을 얼마나 잘 찾아내는지 나타내는 지표이다. 재현율 공식은 다음과 같다.
‘(재현율)= $TP/(TP+FN)$ ’. 여기서 **TP**는 위와 동일하고, **FN**은 “음성”으로 예측했지만 실제로는 “양성”인 샘플의 수이다.

- (3) **f1-score(F1 점수)**: F1 점수는 정밀도와 재현율의 조화 평균으로, 정밀도와 재현율을 동시에 고려하여 모델을 평가하는 지표로 사용된다. **F1-score** 공식은 다음과 같다. $(F1-score) = 2 * (precision * recall) / (precision + recall)$. F1-score는 정밀도와 재현율이 균형을 이룰 때, 가장 높은 값을 갖는다.
- (4) **support**: Support는 각 클래스별로 실제로 존재하는 샘플의 수를 의미한다.
- (5) **accuracy(정확도)**: accuracy는 전체 샘플 중에서 올바르게 예측한 샘플의 비율을 나타낸다. 위에서 accuracy가 0.23으로 나타나고 있다. 이는 모델이 전체 샘플 중에서 약 23%만큼의 샘플을 정확하게 예측했다는 것을 의미한다.
- (6) **macro average(매크로 평균)**: macro average는 각 클래스에 대한 지표들의 평균을 계산한 값이다. 위에서 macro average가 0.24, 0.23, 0.23으로 각각 표시되어 있다. 이 값들은 precision, recall, f1-score에 대한 macro average를 나타내며, 각 클래스의 성능을 동일한 가중치로 고려한 결과이다.
- (7) **weighted average(가중 평균)**: weighted average는 각 클래스에 대한 지표들을 해당 클래스의 샘플 수로 가중치를 주어 계산한 평균값이다. 위에서 weighted average가 0.24, 0.23, 0.23으로 각각 표시되어 있다. 이 값들은 precision, recall, f1-score에 대한 weighted average를 나타내며, 각 클래스의 샘플 수에 따라 가중치를 부여한 결과이다.

4.8. 모델 원격 저장소에 올리기

Hugging Face는 자연어 처리 모델을 저장하고 공유할 수 있는 온라인 플랫폼이다. 사용자들은 Hugging Face를 통해 다양한 사전 훈련된 모델, 토큰라이저, 임베딩, 데이터셋 등을 찾고 활용할 수 있으며, 자신이 훈련한 모델을 업로드하여 커뮤니티와 공유할 수도 있다. 또한, Hugging Face는 github처럼 버전 관리를 지원하여 모델 및 자원이 이전 버전과 비교하고 효율적으로 작업할 수 있도록 도와준다. 이를 통해 자연어 처리 커뮤니티와의 협력과 지식 공유를 촉진하여 자연어 처리 작업을 보다 효율적으로 진행할 수 있다. 훈련한 BERT 모델을 웹 서버에 배포하는 것이 목표였지만, 대용량의 딥러닝 모델 파일을 로컬에 보유하고 있는 것은 어려움이

있었다. 또한, Github에 업로드하기 어려운 큰 파일이었다. 이러한 이유로 Hugging Face의 방식을 사용하여 모델을 업로드하기로 결정했다. 모델은 Hugging Face의 모델 저장소에 업로드되었다.

(<https://huggingface.co/burningfalls/my-fine-tuned-bert>) 이를 통해, 로컬에 딥러닝 모델 파일을 보유하지 않고도, 몇 줄의 코드로 모델을 로드하고 사용할 수 있게 되었다.

The screenshot shows the Hugging Face interface for the model 'burningfalls/my-fine-tuned-bert'. The top navigation bar includes the Hugging Face logo, a search bar, and a menu icon. The model page header shows the model name, a 'like' button with a count of 1, and various tags: Text Classification, TensorFlow, Transformers, AI-Hub, English, Korean, and bert. The license is listed as apache-2.0. Below the header are buttons for 'Train', 'Deploy', and 'Use in Transformers'. The main content area shows the 'Files' tab selected, displaying a list of files and their commit history. The files include .gitattributes, README.md, config.json, special_tokens_map.json, tf_model.h5, tokenizer.json, tokenizer_config.json, and vocab.txt. The commit history for each file shows the contributor 'burningfalls' and the commit message 'Update README.md' or '[UPLOAD] my fine-tuned bert'.

File	Size	Commit Message	Time
.gitattributes	1.48 kB	initial commit	12 days ago
README.md	3.91 kB	Update README.md	6 minutes ago
config.json	3.05 kB	[UPLOAD] my fine-tuned bert	12 days ago
special_tokens_map.json	125 Bytes	[UPLOAD] my fine-tuned bert	12 days ago
tf_model.h5	443 MB	[UPLOAD] my fine-tuned bert	12 days ago
tokenizer.json	752 kB	[UPLOAD] my fine-tuned bert	12 days ago
tokenizer_config.json	426 Bytes	[UPLOAD] my fine-tuned bert	12 days ago
vocab.txt	248 kB	[UPLOAD] my fine-tuned bert	12 days ago

BERT 모델을 로드하는 방식도 약간 변경되었다. 이전에는 로컬에 저장된 모델 위치인 'MODEL_SAVE_PATH'에서 가져왔지만, 현재는 서버에서 직접 가져온다. 변경된 방법은 다음과 같다.

```
from transformers import AutoTokenizer, TFAutoModelForSequenceClassification

BERT_PATH = "burningfalls/my-fine-tuned-bert"

loaded_tokenizer = AutoTokenizer.from_pretrained(BERT_PATH)
loaded_model = TFAutoModelForSequenceClassification.from_pretrained(BERT_PATH)
```

위와 같이 코드를 작성하면, Hugging Face에 업로드된 모델을 로드하여 사용할 수 있다.

5. GPT-3.5-turbo

5.1 모델 소개

Openai에서 개발한 강력한 언어 모델인 GPT는 자연어 이해와 생성에 탁월한 성능을 보인다. 이 모델은 사전 훈련된 대규모 데이터셋을 기반으로 학습되어 문장의 맥락을 이해하고 자연스러운 답변을 생성할 수 있다. 이는 대화형 시스템에서 중요한 요소로 작용하며 사용자의 질문이나 입력에 대해 자연스러운 대화를 제공하는 데에 큰 도움이 된다. 또한 GPT는 뛰어난 문맥 파악 능력과 다양한 주제에 대한 지식을 가지고 있다. 이 모델은 사전 훈련된 데이터셋을 통해 다양한 도메인의 정보를 학습하고 이를 바탕으로 다양한 주제에 대한 응답을 생성할 수 있다. 사용자와 대화하는 과정에서 GPT는 유창하고 자연스러운 문장을 생성하여 사람과의 대화가 이루어지는 것처럼 느껴지게 할 수 있다. 본 프로젝트에서는 감정 분석을 위해 파인튜닝된 BERT 모델을 활용하고 있다. 이 모델은 문장에서 감정을 추출하는 데에 탁월한 성능을 보인다. 따라서 우리는 GPT-3.5-turbo 모델을 선택하여 추출한 감정

정보를 챗봇의 응답 문장에 효과적으로 반영하고자 한다. **GPT-3.5-turbo** 모델은 **GPT3-davinci** 모델을 파인튜닝하는 것에 대한 대안으로, 최소한의 비용으로 최대한 원하는 응답 문장을 생성할 수 있다는 장점이 있다. 물론 **GPT3-davinci** 모델을 파인튜닝하여 응답 문장에 감정을 반영하는 방법도 있지만 본 프로젝트에서는 데이터셋의 부족과 장비의 부족, 그리고 예산의 제약으로 인해 이 방법을 채택하기 어려웠다. 이러한 이유로 인해 **GPT-3.5-turbo** 모델을 선택하였고, 해당 모델은 충분히 우리가 원하는 수준의 답변을 반환할 수 있으며, 자연스러운 문장 생성에도 문제가 없다고 판단하였다. 해당 모델을 사용함으로써 사용자와의 대화에서 챗봇이 자연스럽게 효과적인 응답을 제공할 수 있으며 성능을 향상시킬 수 있다.

5.2 모델 성능

GPT 모델은 자연어 처리 분야에서 높은 성능을 보이며, 특히 챗봇 분야에서의 수치적 성과와 다양한 응용 가능성이 주목받고 있다. 먼저, **GPT**는 문장 생성과 이해에서 뛰어난 성능을 보여준다. 사전 훈련된 대규모 데이터셋을 기반으로 학습된 **GPT**는 문장의 맥락을 이해하고 자연스러운 응답을 생성할 수 있다. 이는 챗봇 시스템에서 매우 중요한 요소로 작용하며, 사용자의 질문이나 입력에 대해 유창하고 자연스러운 대화를 제공할 수 있다. 뿐만 아니라, **GPT**는 챗봇 분야에서도 높은 수준의 성능을 보여주고 있다. 이 모델은 다양한 주제에 대한 지식과 문맥 파악 능력을 갖추고 있어 사용자와의 대화를 더욱 자연스럽게 유창하게 만들어 준다. 이를 통해 챗봇은 사용자의 요구에 맞춰 다양한 정보를 제공하거나 문제를 해결하는 데 도움을 줄 수 있다. **GPT**의 쓰임새는 다양한 분야에 걸쳐 확장되고 있다. 본 서비스에서도 만들고자 하고 만든 챗봇은 그 중에서도 가장 인기 있는 응용 분야 중 하나이다. 챗봇은 고객 서비스, 상담, 정보 제공 등 다양한 영역에서 활용되며, 사용자와의 원활한 대화를 통해 효과적인 커뮤니케이션을 이룰 수 있다. **GPT** 모델은 이러한 챗봇 시스템에서 강력한 성능을 발휘하여 사용자들에게 뛰어난 경험을 제공하고 있다. 이러한 성과와 다양한 응용 가능성으로 인해 **GPT**는 현재 다양한 산업 분야에서 활발히 활용되고 있다. 챗봇을 비롯한 자연어 처리 기술은 고객 서비스, 마케팅, 교육, 의료 등 여러 분야에서 혁신적인 솔루션을 제공하고 있으며, **GPT** 모델은 이러한 분야에서 탁월한 성능을 발휘하여 많은 관심과 기대를 받고 있다.

5.3 모델 불러오기

본 프로젝트는 챗봇 서비스의 응답 문장 생성모델로써 Openai의 GPT-3.5-turbo 모델을 활용하였다. GPT-3.5-turbo 모델은 Openai의 ChatGPT에 이용되는 모델로서 뛰어난 문장 생성 능력을 가지고 있다. GPT-3.5-turbo model의 입력 문장 형태는 “User Input Sentence ;; (Fine-Tuned BERT가 분류한 Input Sentence의 감정)”이다. 다른 GPT-3 모델과는 다르게 GPT-3.5-turbo 모델은 문장을 생성하는 데에 있어서 parameter 및 모델 커스텀의 개념으로 'system'이라는 role에 문장을 입력하여 이용한다. 해당 role에 생성 문장이 만족해야 하는 조건인 preliminaries를 입력하면 GPT-3.5-turbo 모델은 그러한 조건을 만족하는 문장을 생성한다. 본 프로젝트에서는 'system'이라는 role에 입력할 문장으로 “친구, 일상대화, 반말, ';;' 뒤에 있는 감정을 답변 문장에 반영”이라는 조건을 활용하였다. 응답 문장의 조건으로 친구와 대화하는 것처럼 응답하고 일상 대화의 형태를 띄며 반말의 친근한 말투를 사용하고 “;;” 뒤에 BERT 모델이 분류한 감정을 Input Sentence와 함께 넣음을 알려주어 답변 문장에 감정을 반영할 수 있게 하였다. 따라서 'system'이라는 role을 통해 GPT-3.5-turbo 모델을 본 프로젝트의 의도와 맞게 커스텀 하였다.

아래는 GPT를 커스텀하기 위해 사용한 'role: system'의 content의 내용이다.

```
# GPT_OPTION: GPT 커스텀
GPT_OPTION = "친구, 일상대화, 반말, ';;' 뒤에 있는 감정을 답변 문장에 반영"
```

GPT-3.5-turbo 모델을 활용하여 문장을 생성하는 과정은 다음과 같다. 먼저 User Input Sentence를 앞에서 Fine-tuning을 진행한 BERT Model을 거쳐 감정을 분석한다. 그 이후 User Input Sentence + 감정을 'role: user'로 GPT Model에게 전달할 메시지 리스트에 추가한다. OpenAI Chat API를 사용하여 해당 내용을 GPT-3.5-turbo 모델의 입력으로 사용하면, 해당 모델은 답변을 result로써 return한다. 그리고 return 받은 내용을 'role: assistant'로 계속하여 메시지 리스트에 저장함으로써 대화의 흐름이 이전과 계속 이어지도록 해당 내용은 generate_answer라는 함수로서 정의되어 server를 구동하는 코드 내에서 input이 들어올 때마다 반복 실행된다. 해당 함수 코드의 내용은 아래와 같다.

```
def generate_answer(question, feel):
    # {질문+감정}을 'role: user'로 메시지 리스트에 추가
    messages.append({"role": "user", "content": question + ' ;; ' + feel})

    # OpenAI Chat API를 사용하여 답변 생성
    result = openai.ChatCompletion.create(
        model=GPT_NAME,
        messages=messages
    )
    answer = result.choices[0].message.content

    # {답변}을 'role: assistant'로 메시지 리스트에 추가
    messages.append({"role": "assistant", "content": answer})

    return answer
```

결과적으로 GPT-3.5-turbo model은 User Input Sentence와 같은 감정을 가진 Output Sentence를 반환한다.

6. 모델 실행

6.1 서버 개요

NLP Model Server는 자연어 처리(NLP) 모델을 활용한 웹 서비스를 제공한다. Flask framework를 사용하여 구축되었으며, fine-tuned BERT와 GPT-3.5 Turbo 모델을 통해 감정 분류 및 자동 응답 기능을 제공한다. 이를 통해 사용자는 자연어 질문을 입력하고, 해당 질문에 대한 감정이 반영된 AI 기반의 응답을 받을 수 있다.

6.2 목표

- (1) 자연어 처리(NLP) 모델을 웹 서비스에 통합하여 사용자가 감정이 반영된 자동 응답 기능을 활용할 수 있도록 제공한다: 서버의 목표는 자연어 처리(NLP) 모델을 웹 서비스에 통합하여 사용자에게 감정이 반영된 자동 응답 기능을

제공하는 것이다. 이를 위해 **BERT**와 **GPT-3.5 Turbo** 모델을 사용한다. **BERT** 모델을 활용하여 사용자가 입력한 질문에 대한 감정을 분류하고, **GPT-3.5 Turbo** 모델을 사용하여 질문과 감정 정보를 기반으로 자연스러운 응답을 생성한다. 이를 통해 사용자는 감정이 반영된 대화형 **AI** 시스템을 경험할 수 있다.

- (2) 입력과 결과 출력이 가능하도록 구현한다: 서버는 **Flask framework**를 사용하여 웹 애플리케이션을 구현한다. **Flask**는 사용자와의 상호작용을 지원하는 간단하고 가벼운 웹 프레임워크이다. 웹 인터페이스를 통해 사용자는 직관적인 방식으로 질문을 입력하고, 결과를 확인할 수 있다. 또한, **CORS(Cross-Origin Resource Sharing)** 설정을 통해 다른 도메인에서의 요청을 허용하여 웹 브라우저에서 애플리케이션에 접근할 수 있도록 한다.
- (3) **BERT**와 **GPT-3.5 Turbo** 모델을 활용하여 질문에 대한 감정 분류와 자동 응답을 제공하며, **AI** 모델의 성능과 품질을 최대한 유지한다: 서버는 **BERT**와 **GPT-3.5 Turbo** 모델을 사용하여 질문에 대한 감정 분류와 자동 응답 기능을 제공한다. **BERT** 모델은 질문에 대한 감정을 정확하게 분류하기 위해 사용된다. **GPT-3.5 Turbo** 모델은 질문과 감정 정보를 기반으로 자연스러운 응답을 생성한다. 미리 학습된 모델을 사용하여 모델의 성능과 품질을 최대한 유지하며, 필요에 따라 추가적인 모델 튜닝을 수행한다.
- (4) 사용자가 손쉽게 프로젝트를 실행하고 활용할 수 있도록, 프로젝트 구성을 간결하게 유지한다: 서버는 사용자가 손쉽게 서버를 실행하고 활용할 수 있도록 구성되어 있다. **OpenAI API key**를 설정하기 위해 ‘**mykey.py**’ 파일을 사용하고, 간단한 수정으로 **API** 키를 입력할 수 있도록 한다. 필요한 **Python** 패키지들을 ‘**requirements.txt**’ 파일에 명시하여 쉽게 설치할 수 있도록 한다. 또한, **Flask** 애플리케이션을 실행하기 위해 간단한 명령어를 제공하여, 사용자가 프로젝트를 쉽게 실행할 수 있도록 한다.

6.3 구현 내용

- (1) **Flask Application**: **Flask** 애플리케이션은 ‘**app.py**’ 파일을 통해 구현되었다. 이 파일은 웹 서비스의 메인 파일로 사용되며, **Flask** 프레임워크를 활용하여 다양한 기능을 제공한다. ‘**app.py**’는 다음과 같은 주요 기능을 포함한다.
 - a. **Flask** 애플리케이션을 생성하고, **CORS** 설정을 통해 다른 도메인에서의

요청을 허용한다.

- b. `/predict` 경로로 **POST** 요청이 들어오면 실행되는 `predict()` 함수를 정의한다.
- c. 요청에서 질문을 추출하고 감정 분류 기능을 호출하여, 질문에 대한 감정을 예측한다.
- d. 자동 응답 기능을 호출하여 질문과 감정 정보를 기반으로 응답을 생성한다.
- e. 생성된 응답을 **JSON** 형식으로 반환한다.

(2) 감정 분류 기능: **BERT** 모델을 활용하여 자연어 질문에 대한 감정 분류를 수행한다. 구체적인 구현 내용은 다음과 같다.

- a. `'sentiments.py'` 파일에는 감정 레이블과 인덱스가 정의되어 있다. **BERT** 모델의 출력을 감정 레이블로 변환하기 위해 이 파일을 활용한다.
- b. **BERT** 모델의 `tokenizer`와 `sequence classification model`을 로드한다.
- c. 텍스트 분류용 파이프라인을 생성하고, 입력 질문에 대한 감정을 예측한다.
- d. 예측된 감정 인덱스를 감정 레이블로 변환하여 반환한다.

(3) 자동 응답 기능: **GPT-3.5 Turbo** 모델을 활용하여 자동 응답을 생성한다. 구체적인 구현 내용은 다음과 같다.

- a. **OpenAI Chat API**를 사용하여 **GPT-3.5 Turbo** 모델과 대화한다.
- b. 사용자의 질문과 감정 정보를 메시지로 전달하여 **GPT** 모델에 입력한다.
- c. 모델은 입력 메시지를 기반으로 자연스러운 응답을 생성하고 반환한다.

(4) **OpenAI API key**: `'mykey.py'` 파일을 통해 **OpenAI API key**를 관리한다. 이 파일에 **API key**를 입력하여 모델과의 연결을 설정할 수 있다. `'app.py'`에서 `'mykey.py'` 파일을 `import`하여 **API key**를 사용하여 **OpenAI**와의 연결을 설정한다.

(5) **Web Interface**: **Flask**를 통해 구현된 웹 인터페이스는 사용자와의 상호작용을 지원한다. 웹 페이지에서 질문을 입력하고 제출하면 서버에 해당 질문이 전송되며, 모델의 응답은 웹 페이지에 출력된다. 사용자는 웹 인터페이스를 통해 질문을 간편하게 입력하고 모델의 결과를 확인할 수 있다.

6.4 사용 방법

- (1) 'mykey.py' 파일에 OpenAI API key를 입력한다.
- (2) 필요한 Python 패키지들을 설치하기 위해 다음 명령을 실행한다.

```
pip install -r requirements.txt
```

- (3) Flask application을 실행한다.

```
python app.app.py
```

- (4) Web browser에서 'http://localhost:5000'으로 접속하여 NLP Model Server를 이용할 수 있다. 질문을 입력하고 결과를 확인할 수 있다.

6.5 비교 및 분석

GPT 모델이 생성한 output이 감정을 담고 있는 답변이라는 증거

- <문장>이 담고 있는 감정이 뭐야? 라고 질문
- 그냥 GPT의 답변 vs 만든 GPT의 답변

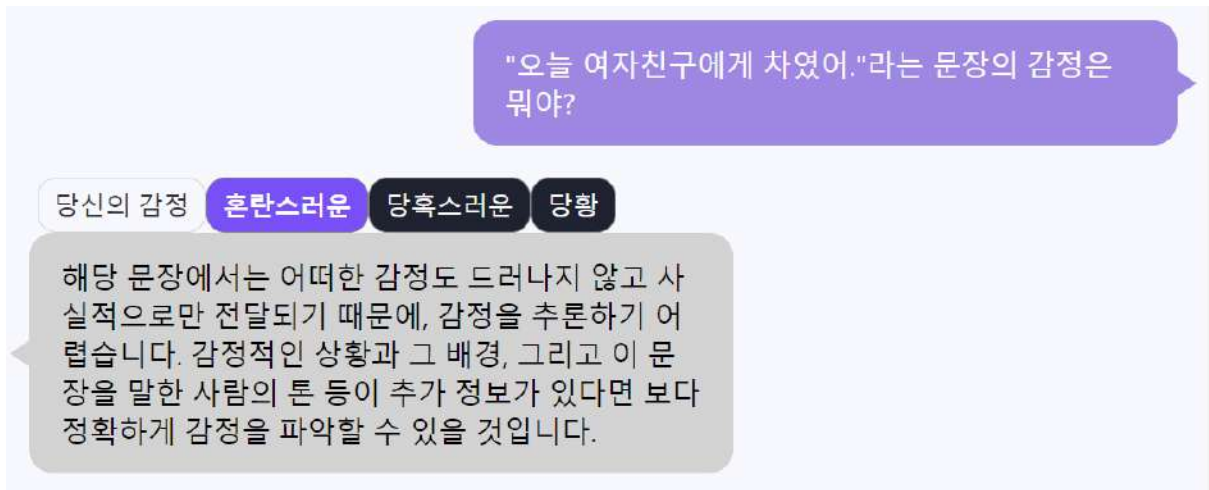
GPT 모델의 생성 문장이 입력으로 준 **feel**을 반영한다는 증명은 다음과 같다.

question으로는 '<문장>'라는 문장의 감정이 뭐야? 라는 질문을 동일하게 부여하였다.

첫 번째는 GPT-3.5-turbo model의 입력으로 감정인 **feel**을 주지 않은 경우이다.

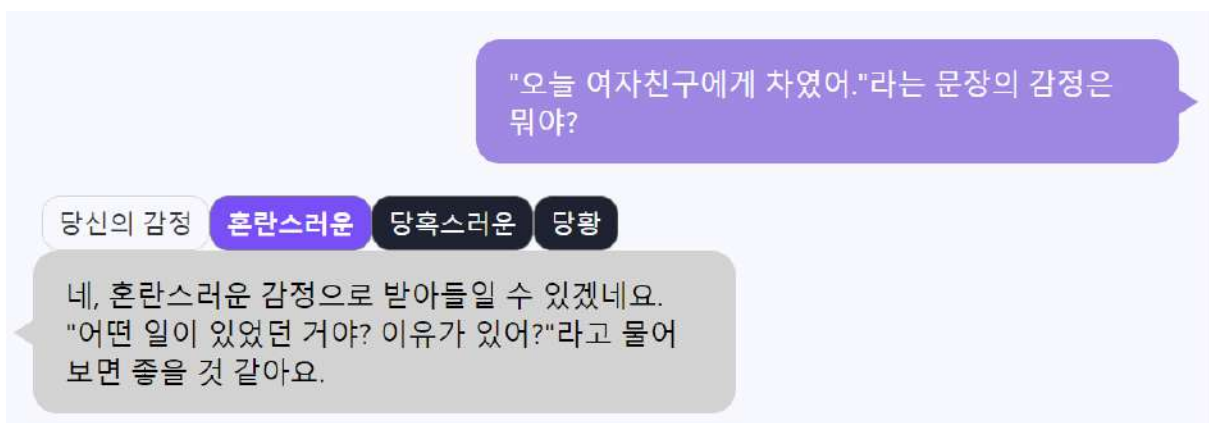
```
# (1) GPT에게 feel을 입력으로 주지 않은 경우messages.append({"role": "system", "content": "친구, 일상대화, 반말"})messages.append({"role": "user", "content": question})
```

(1)의 경우, BERT 모델을 거쳐 Input Sentence로부터 추출한 Feel을 GPT-3.5-turbo model의 입력으로 넣어주지 않아 GPT가 정확하게 문장의 감정을 인식하지 못한다.



(2) GPT에게 feel을 입력으로 준 경우 `messages.append({"role": "system", "content": "친구, 일상대화, 반말, “;” 뒤에 있는 감정을 답변 문장에 반영"})` `messages.append({"role": "user", "content": question + “;” + feel})`

(2)의 경우 내부적으로 Input Sentence를 BERT 모델을 거쳐 추출한 Input과 GPT가 Output으로써 답변한 감정이 정확하게 일치함을 확인할 수 있기 때문에, 본 프로젝트에서 활용한 GPT-3.5-turbo model과 'role: system'의 내용으로 넣은 부분이 출력 문장에서 감정이 반영되도록 유도하였음을 증명하였다.

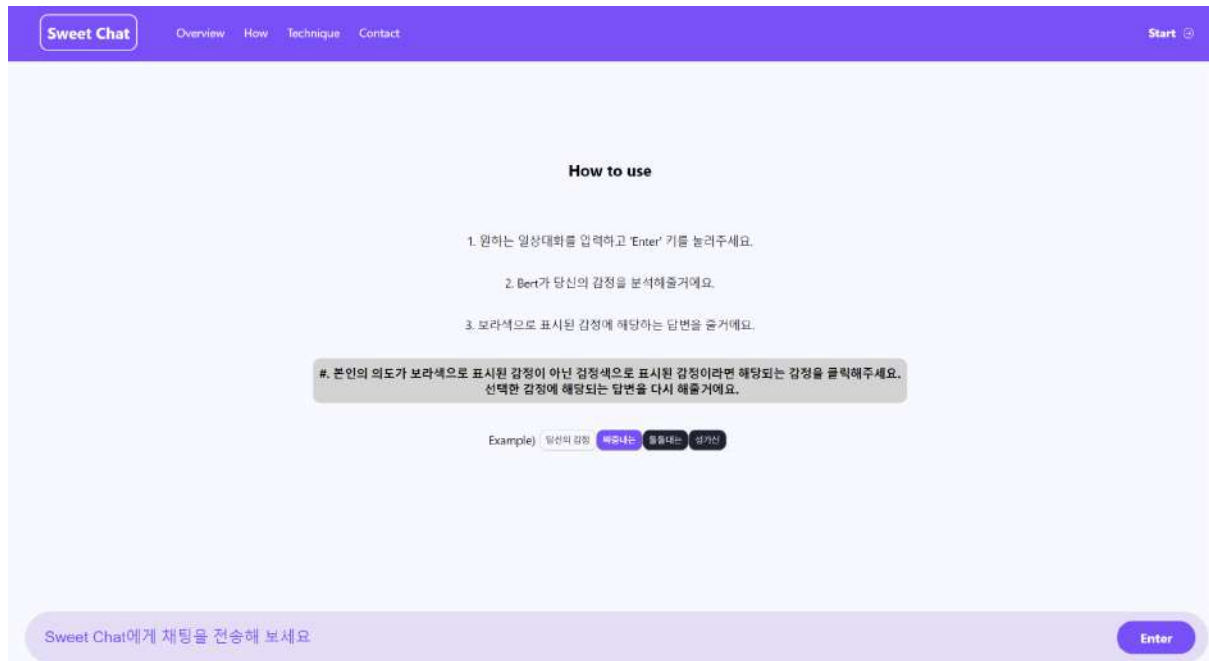


6.6 결과 및 평가

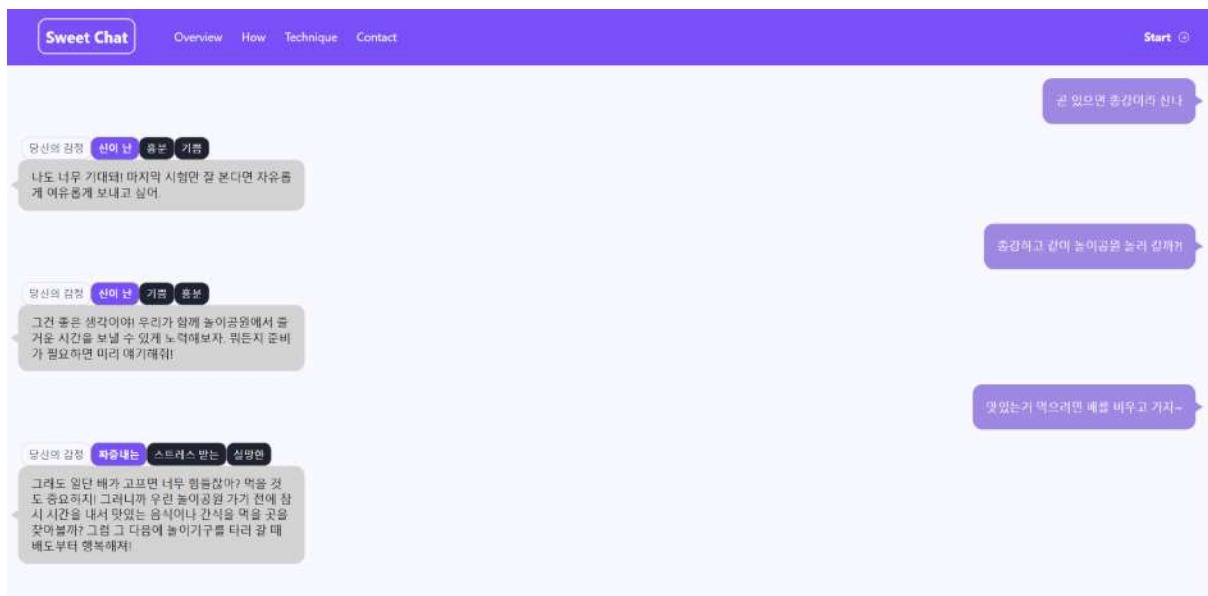
NLP Model Server는 자연어 처리 모델인 BERT와 GPT-3.5 Turbo 모델을 웹 서비스에 통합하여, 사용자의 질문에 대한 정확한 감정 분류와 자연스러운 자동 응답 기능을 제공한다. 해당 기능들에 대한 결과 및 평가는 아래와 같다.

- (1) 감정 분류 결과의 정확성: fine-tuned BERT 모델을 활용한 감정 분류 기능은 사용자의 자연어 질문에 대한 감정을 정확하게 분류한다. 이를 통해, 사용자의 질문이 어떤 감정을 담고 있는지를 신뢰성 있게 예측할 수 있다.
- (2) 자동 응답의 자연스러움: GPT-3.5 Turbo 모델을 활용한 자동 응답 기능은, 입력된 질문과 감정 정보를 바탕으로 자연스러운 응답을 생성한다. 모델은 대화 스타일로 학습되었기 때문에, 사용자와의 상호작용에 맞는 자연어로 응답한다.
- (3) 확장성과 유연성: Flask 프레임워크를 사용하여 구현된 NLP Model Server는 확장성과 유연성을 제공한다. Flask의 가벼운 구조와 모듈화된 설계를 활용하여 새로운 기능의 추가나 기존 기능의 수정이 용이하다. 또한, API endpoint를 통해 다양한 클라이언트와의 통합이 가능하다.
- (4) 보안 측면: OpenAI API 키인 'mykey.py' 파일에 저장되는 OpenAI API 키는 보안에 주의해야 한다. 해당 파일은 권한이 제한된 디렉토리에 저장되어야 하며, 액세스 권한을 최소한으로 설정하여 외부로부터의 접근을 제한해야 한다. OpenAI API 키가 노출되면 악의적인 사용자가 모델에 접근할 수 있으므로, 보안 조치를 철저히 적용해야 한다.

7. 서비스 시연



채팅 전송 화면 사진



채팅 예시 사진 (답변에의 감정 분석, 연속적인 대화 흐름 유지, 응답 문장이 감정을 잘 반영하지 않고 있다고 생각할 시 새로운 응답 문장을 생성하는 기능)

8. 결론

8.1. 최종 연구 성과

연구 초기에는 단순히 **BERT** 모델만을 활용하고 사용자 입력 문장의 감정을 분석해주는 서비스를 만드려 했지만 거기서 더 나아가 심화된 프로젝트를 진행하고자 하여 감정을 반영한 문장 생성 알고리즘에 대해서 알아보고 연구해 **Fine-tuned BERT**와 **GPT**를 파이프라이닝한 감정을 파악하는 챗봇을 구현하고자 하였다. 따라서 주요 목표와 세부 목표를 설정하여 본 프로젝트가 달성하고자 한 “감정을 파악하는 챗봇”이라는 연구 결과물을 만들고자 하였다. 본 프로젝트의 주요 목표와 세부 목표는 ‘자연어처리 관련 자유주제 개발’이라는 주제에 맞추어 자연어처리 기술을 활용하여 **BERT&GPT** 파이프라이닝 감성 대화 챗봇을 개발하는 것이다. 해당 주요 목표를 달성하기 위해 설정했던 세부 목표들은 **BERT** 모델 **fine-tuning**, **GPT-3.5-turbo** 모델을 활용한 응답 문장 생성, 사용자 **Input Sentence**와 동일 감정이 담긴 **Output Sentence**를 생성하기 위한 **GPT** 모델 튜닝이 있었다. 해당 목표를 수행하며 얻은 **BERT** 부분에서의 성과는 다음과 같다. **BERT** 모델을 본 프로젝트가 수행하고자 하는 감정 분류를 정확하게 수행하도록 만들기 위해서 데이터셋 전처리, 학습 파라미터 설정(**batch size**, **learning rate**), 토큰나이징 및 **fine-tuning**을 수행하였다. 해당 **fine-tuned BERT** 모델의 테스트 성능은 랜덤으로 감정을 예측하는 모델보다 약 13.6배의 감정 분류 성능 향상을 보여주었다. (1.72% -> 23.41%). **Fine-Tuned BERT** 모델을 활용한 감정 분류 기능은 사용자로부터 입력으로 들어온 자연어 질문에 대한 감정을 **fine-tuned** 되지 않은 기존의 모델보다 정확하게 분류하였다. 따라서 사용자의 질문이 어떤 감정을 가지고 있는지를 분류할 때 신뢰성 있게 예측할 수 있었다. **GPT-3.5-turbo** 부분에서의 성과는 다음과 같다. 강력한 언어 모델인 **GPT**를 활용하여 문장의 맥락을 이해하고 자연스러운 답변을 생성하는 챗봇을 만들 수 있었으며 ‘**role**’: ‘**system**’ 파트를 통하여 **BERT** 모델에서 추출한 감정 정보를 챗봇의 응답 문장에 효과적으로 반영하였다. 출력 문장에서 **BERT**가 입력 문장으로부터 분류한 감정 반영의 증거는 보고서의 6.5 부분에 **preliminaries**로 ‘**BERT** 모델을 거쳐 추출한

감정을 반영해라'라고 적어준 경우와 적어주지 않은 경우의 챗봇의 응답을 통해 명확하게 알아볼 수 있었다. 데이터셋의 부족과 장비의 부족, 예산의 제약을 극복하기 위해 해당 모델을 선정하여 원하는 수준의 답변을 사용자와의 대화에서 획득할 수 있었으며 모델의 적절한 튜닝을 통해 자연스럽게 효과적인 응답을 제공하여 챗봇 서비스의 성능과 만족도를 향상시킬 수 있었다. 마지막은 챗봇을 만들기 위한 **Client & Server** 구축에서의 성과이다. 자연어 처리 모델을 활용한 웹 서비스를 제공하기 위하여 **Flask framework**를 사용하여 구축하였다. 입력과 결과 출력 및 사용자와의 상호작용을 위하여 해당 방식을 채택하였으며 사용자는 직관적인 방식으로 질문 입력 및 결과를 서비스에서 수행할 수 있다. 또한 **CORS** 설정을 통해 다른 도메인에서의 요청을 허용하여 웹 브라우저에서 애플리케이션에 접근할 수 있도록 하였다. 웹 서비스의 메인 파일인 'app.py'를 통해 **server side**를 구축하였고 사용자로부터 **POST** 요청이 들어오면 실행되는 함수를 정의하여 요청에서 질문을 추출하고 감정 분류 기능을 호출하여 질문에 대한 감정을 예측, 해당 응답을 반환하였다.

주요 목표와 세부 목표를 설정하고 해당 목표를 달성하기 위해 연구를 수행하는 과정에서 **BERT, GPT, Client, Server** 면에서 모두 만족할 만한 성과를 얻을 수 있었으며 사용자와 직관적으로 소통하며 대화하는 **BERT&GPT** 파인튜닝 및 파이프라이닝을 통한 감성 대화 챗봇을 구현할 수 있었다.

8.2. 기대 효과

BERT&GPT 파인튜닝을 활용한 감성 대화 챗봇은 사용자의 입력 문장에서 감정을 정확하게 분석하고, 그에 맞는 감정을 담은 응답을 생성한다. 이를 통해 챗봇은 사용자의 감정을 인식하고 공감하여 대응할 수 있다. 사용자는 챗봇을 통해 자신의 전하고자 하는 감정을 이해받고 공감받으며, 감정적인 도움을 얻을 수 있다. 또한 챗봇과 사용자 간의 대화가 자연스러운 흐름을 갖고, 사용자가 챗봇과의 상호작용을 현실적이고 편안한 경험으로 느낄 수 있다. 감정에 따라서 챗봇의 응답이 조율되므로 사용자는 자신에게 맞는 대화 또는 조언을 받을 수 있다. 이를 통해 사용자는 개인적인 관심사나 문제에 대해 더 나은 해결책이나 지원을 받을 수 있다는 개인

맞춤형 서비스 제공을 할 수 있다. 언제나 어디서든 자신의 감정에 공감해 주며 대화를 나눌 수 있는 서비스를 제공함으로써, 이를 통해 감정 부문에서 소외되는 사람들의 수를 줄이고 다양한 사회적 배경과 상황에서 사용자들이 감정적인 지원을 받을 수 있는 기회를 제공한다. 이는 사회 전체적인 행복 증진으로 이어질 것으로 생각된다.

8.3. 추후 연구 방향

감성 대화 챗봇은 사용자와의 상호작용을 기반으로 개인 정보를 처리하므로, 개인정보 보호와 보안을 강화하는 것이 중요하다. 따라서 추가적인 연구 방향으로써 사용자의 개인 정보를 안전하게 관리하고 보호하기 위한 기술 및 방법에 대한 개선 및 연구를 진행할 수 있을 것으로 생각된다.

9. 참고문헌

Park, S., & Kim, K. J. (2018). A Study on the Development of Relationship Coaching System. Journal of the Korea Society of Computer and Information, 23(12), 23-30.

Lee, J. A., & Jeong, M. (2020). A Study on the Development of AI-based Couple Counseling System. Journal of Digital Convergence, 18(5), 225-233.

Kim, S. Y., & Shin, D. (2019). A Study on the Design of Relationship Coaching App Based on the Analysis of User Needs. Journal of Digital Convergence, 17(10), 99-106.

Lee, S., Lee, J., & Park, Y. (2020). Analysis of Relationship Coaching Apps and Development of Recommendation System. Journal of Digital Convergence, 18(8), 389-397.

Moon, S., Kim, Y., & Lee, M. (2021). Design of Personalized Relationship Coaching Service Based on Machine Learning. Journal of Digital Convergence, 19(1), 115-122.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. (2017). Attention Is All You Need.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding