

감정 분석기

with. Chat GPT

11주차 팀과제

201734905 김민수

201939734 김민정

202037620 김홍주

202136606 김도연

목차

- 감정 분석기에 대하여
 - ◆ BERT, KoBERT란?
 - ◆ 감정 분석기란?
- Chat GPT의 수준 평가
 - ◆ original code 설명
 - ◆ original code의 flow graph, pseudo code, test case 요청
 - ◆ code 요청 & 검토
 - ◆ flow graph, pseudo code, test case 요청
- Chat GPT
 - ◆ Chat GPT가 가지고 올 긍정적/부정적 영향
 - ◆ Chat GPT가 여러 분야에 미칠 영향

감정 분석기

- BERT, KoBERT에 대하여

BERT(Bidirectional Encoder Representations from Transformers)는 2018년에 발표된 언어 모델로, 전이 학습(Transfer Learning)을 통해 다양한 자연어 처리(Natural Language Processing) 작업에 사용된다. Transformer 아키텍처를 기반으로 하며, 사전 훈련된 언어 모델로 대량의 텍스트 데이터를 사용하여 학습된다.

"양방향" 모델로서, 기존의 양방향 언어 모델과 달리, BERT는 사전 학습 단계에서 Masked Language Model(MLM)과 Next Sentence Prediction(NSP)이라는 두 가지 목적을 동시에 사용하여 학습한다. MLM은 입력 문장에서 임의의 단어를 마스킹하고, 마스킹된 단어를 예측하는 작업을 학습하는 것이고, NSP는 두 문장의 관계를 예측하는 작업을 학습하는 것이다.

KoBERT는 SKTBrain에서 공개했는데, 한국어 BERT의 변형 버전으로, 한국어 자연어 처리를 위해 사전 훈련된 모델이다. 한국어 위키 5백만 문장과 한국어 뉴스 2천만 문장을 학습한 모델이고, 한국어 텍스트에 대해 BERT와 유사한 아키텍처를 가지고 있으며, 한국어 문장 구조 및 특징을 고려하여 학습된 사전 훈련 모델이다. KoBERT는 한국어 문장의 의미와 문맥을 파악할 수 있으며, 감정 분류, 개체명 인식, 텍스트 분류 등 다양한 자연어 처리 작업에 활용될 수 있다.

- 감정 분석이란?

감정 분석이란, 간단하게 설명하면 감정이 담긴 어떠한 문장을 입력했을 경우에 입력된 문장 안에서 주요 감정을 파악하여 정해진 7개의 감정 중 하나로 분류하는 프로그램이다.

AiHub의 감정 분류를 위한 대화 음성 데이터셋을 이용한다.

happiness(행복), angry(분노), disgust(혐오), fear(공포), neutral(중립), sadness(슬픔), surprise(놀람) 7가지의 감정으로 분류하는데, 예를 들어, "엘리베이터에 갇혔었다." 라는 문장을 입력하면 공포로, "종강 하고싶다." 라는 문장을 입력하면 슬픔으로 분류한다.

Chat GPT의 코딩 수준 평가

- original code 설명

1. 환경 설정(Error발생)

KoBERT가 요구하는 최신 정보를 토대로 필요한 패키지를 설치한다.

```
In [2]: 1 |pip install mxnet
        2 |pip install gluonnlp==0.8.0
        3 |pip install pandas tqdm
        4 |pip install sentencepiece==0.1.91
        5 |pip install transformers==4.8.2
        6 |pip install torch

Requirement already satisfied: mxnet in ./anaconda3/lib/python3.8/site-packages (1.9.1)
Requirement already satisfied: numpy<2.0.0,>1.16.0 in ./anaconda3/lib/python3.8/site-packages (from mxnet) (1.20.3)
Requirement already satisfied: graphviz<0.9.0,>0.8.1 in ./anaconda3/lib/python3.8/site-packages (from mxnet) (0.8.4)
Requirement already satisfied: requests<3,>2.20.0 in ./anaconda3/lib/python3.8/site-packages (from mxnet) (2.26.0)
Requirement already satisfied: idna<4,>=2.5 in ./anaconda3/lib/python3.8/site-packages (from requests<3,>=2.20.0->mxnet) (3.1)
Requirement already satisfied: charset-normalizer<=2.0.0 in ./anaconda3/lib/python3.8/site-packages (from requests<3,>=2.20.0->mxnet) (2.0.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in ./anaconda3/lib/python3.8/site-packages (from requests<3,>=2.20.0->mxnet) (1.26.7)
Requirement already satisfied: certifi<=2017.4.17 in ./anaconda3/lib/python3.8/site-packages (from requests<3,>=2.20.0->mxnet) (2022.12.7)
Requirement already satisfied: gluonnlp==0.8.0 in ./anaconda3/lib/python3.8/site-packages (from gluonnlp==0.8.0) (0.8.0)
Requirement already satisfied: numpy in ./anaconda3/lib/python3.8/site-packages (from gluonnlp==0.8.0) (1.20.3)
Requirement already satisfied: pandas in ./anaconda3/lib/python3.8/site-packages (from gluonnlp==0.8.0) (1.3.3)
Requirement already satisfied: tqdm in ./anaconda3/lib/python3.8/site-packages (from gluonnlp==0.8.0) (4.62.3)
Requirement already satisfied: numpy>=1.17.3 in ./anaconda3/lib/python3.8/site-packages (from pandas) (1.20.3)
Requirement already satisfied: python-dateutil<=2.7.3 in ./anaconda3/lib/python3.8/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz<=2017.3 in ./anaconda3/lib/python3.8/site-packages (from pandas) (2021.1)
Requirement already satisfied: six<=1.5 in ./anaconda3/lib/python3.8/site-packages (from python-dateutil<=2.7.3->pandas) (1.16.0)
```

mxnet이 우선 설치되어야 하며, gluonnlp는 0.8.0으로 설정한다.

2. github에서 KoBERT 파일을 로드하고 KoBERT모델을 불러오기

깃허브 파일의 kobert_tokenizer폴더를 다운받는다.

```
In [3]: 1 |pip install 'git+https://github.com/SKTBrain/KoBERT.git#egg=kobert_tokenizer&subdirectory=kobert_hf'

Collecting kobert_tokenizer
  Cloning https://github.com/SKTBrain/KoBERT.git to /tmp/pip-install-71jm_g80/kobert_tokenizer_4de0280c0eba48a98440efdea3d74e88
  Running command git clone -q https://github.com/SKTBrain/KoBERT.git /tmp/pip-install-71jm_g80/kobert_tokenizer_4de0280c0eba48a98440efdea3d74e88
  Resolved https://github.com/SKTBrain/KoBERT.git to commit 47a69af87928fc24e20f571fe10c3cc9dd9af9a3
```

불러온 후, 우리가 필요한 tokenizer와 model, vocabulary를 불러온다.

tokenizer : tokenizer

model : bertmodel

vocabulary: vocab

```
In [5]: 1 from kobert_tokenizer import KoBERTTokenizer
        2 from transformers import BertModel
        3
        4 tokenizer = KoBERTTokenizer.from_pretrained('skt/kobert-base-v1')
        5 bertmodel = BertModel.from_pretrained('skt/kobert-base-v1')
        6 vocab = nlp.vocab.BERTVocab.from_sentencepiece(tokenizer.vocab_file, padding_token='[PAD]')
        7 #vocab = tokenizer.get_vocab()
        8 tok = tokenizer.tokenize
        9
        10 #model, vocab = BertModel.from_pretrained('skt/kobert-base-v1', tokenizer.vocab_file)
        11
        12 print(vocab)

Vocab(size=8002, unk="[UNK]", reserved=["[CLS]", "[SEP]", "[MASK]", "[PAD]"])
```

이 부분에서 굉장히 많은 오류가 발생한다.

3. 필요한 라이브러리 불러오기 (Error발생)

사전 학습된(pre-trained) 모델인 BERT를 사용할 때는 transformers라는 패키지를 자주 사용하기 때문에 호출하였다.

또한, 학습시간을 줄이기 위해 GPU를 사용한다.

4. 데이터셋 불러오기

이전에 AiHub에 기재되었던 데이터(현재는 없음)를 올려준 블로그를 통해서 다운 받았다.

```
In [6]: 1 data = pd.read_excel('/home/jihwan/MinJeong Archive/감정분류데이터셋.xlsx')
```

```
In [7]: 1 data.sample(n=10)
```

Out[7]:

	Sentence	Emotion	Unnamed: 2	Unnamed: 3	Unnamed: 4	공포	5468
2855	혹시 유산인 가능성도 있나요...?	공포	NaN	NaN	NaN	NaN	NaN
16680	무슨 은행이여 시발	분노	NaN	NaN	NaN	NaN	NaN
35985	사진찍어서 숫자 파악하는거 봐서 저 경찰청장도 나 알아야 겠구만. ㅎㅎㅎ	혐오	NaN	NaN	NaN	NaN	NaN
7340	다들 그만 두셋남...?	놀람	NaN	NaN	NaN	NaN	NaN
36064	노무현 대통령과 당신은 많이 다르다	혐오	NaN	NaN	NaN	NaN	NaN
31329	재미따 재미따 재미따 ㅎㅎㅎ	행복	NaN	NaN	NaN	NaN	NaN
37001	언제 터질지 모르는거 옆에 있으면.	혐오	NaN	NaN	NaN	NaN	NaN
19957	거기에 짐대출까지 끼면 고충이 크겠지.	슬픔	NaN	NaN	NaN	NaN	NaN
19347	난 고령태가 부럽다...	슬픔	NaN	NaN	NaN	NaN	NaN
18338	너무 안가리면 육역을까봐 출가했더니 더육먹었네요ㅜ	슬픔	NaN	NaN	NaN	NaN	NaN

해당 데이터 셋에 대해서 총 7개의 감정 class를 0~6개의 숫자(label)에 대응시켜 data_list에 담아준다.

```
In [8]: 1 data.loc[(data['Emotion'] == "공포"), 'Emotion'] = 0 #공포 => 0
2 data.loc[(data['Emotion'] == "놀람"), 'Emotion'] = 1 #놀람 => 1
3 data.loc[(data['Emotion'] == "분노"), 'Emotion'] = 2 #분노 => 2
4 data.loc[(data['Emotion'] == "슬픔"), 'Emotion'] = 3 #슬픔 => 3
5 data.loc[(data['Emotion'] == "중립"), 'Emotion'] = 4 #중립 => 4
6 data.loc[(data['Emotion'] == "행복"), 'Emotion'] = 5 #행복 => 5
7 data.loc[(data['Emotion'] == "혐오"), 'Emotion'] = 6 #혐오 => 6
8
9 data_list = []
10 for ques, label in zip(data['Sentence'], data['Emotion']) :
11     data = []
12     data.append(ques)
13     data.append(str(label))
14
15     data_list.append(data)
```

5. 입력 데이터셋을 토큰화하기 (Error발생)

각 데이터가 BERT 모델의 입력으로 들어갈 수 있도록 tokenization, int encoding, padding 등을 해주는 코드이다.

에러 발생 부분

- 주석으로 표시해놓은 부분, `nlp.data.BERTSentenceTransform` 은 위에서 에러가 났던, `import gluonnlp as nlp` 에서와 같은 패키지이 다.
- 이 부분에서는 파이썬이 업그레이드되면서 `gluonnlp` 와는 호환이 되지 않으면서 문제가 발생한 것이며, 정확히는 `BERTSentenceTransform` 에서 난 에러이다.
- 이를 해결하기 위해 여러 자료를 찾아보았으며 깃허브도 찾아보았지만, 깃허브에도 업그레이드된 사항이 나와있지 않았다.
- 이후, 다른 사람이 직접 `BERTSentenceTransform` 을 직접 새로 구현하신 분의 자료를 발견하여 코드를 수정하였다. (<https://blog.naver.com/newyeararchive/223097878715>)
 - ◆ py파일에서 class를 복붙하여 코드를 수정한다.
 - ◆ `def __init__`에서 input에 vocab를 받는 부분 추가, `self._vocab = vocab` 을 추가하고,`def __call__`에서 `vocab = self._vocab`로 바꿔주었다.

BERT 모델에 적합한 형태로 데이터를 변환하는 클래스인 `BERTSentenceTransform` 를 재정의

```
In [9]: 1 class BERTSentenceTransform:
2         """BERT style data transformation.
3
4         Parameters
5         -----
6         tokenizer : BERTTokenizer.
7             Tokenizer for the sentences.
8         max_seq_length : int.
9             Maximum sequence length of the sentences.
10        pad : bool, default True
11            Whether to pad the sentences to maximum length.
12        pair : bool, default True
13            Whether to transform sentences or sentence pairs.
14        """
15
16    def __init__(self, tokenizer, max_seq_length, vocab, pad=True, pair=True):
17        self._tokenizer = tokenizer
18        self._max_seq_length = max_seq_length
19        self._pad = pad
20        self._pair = pair
21        self._vocab = vocab
22
23    def __call__(self, line):
24
25        # convert to unicode
26        text_a = line[0]
27        if self._pair:
28            assert len(line) == 2
29            text_b = line[1]
30
31        tokens_a = self._tokenizer.tokenize(text_a)
32        tokens_b = None
33
34        if self._pair:
35            tokens_b = self._tokenizer.tokenize(text_b)
36
37        if tokens_b:
38            # Modifies 'tokens_a' and 'tokens_b' in place so that the total
39            # length is less than the specified length.
40            # Account for [CLS], [SEP], [SEP] with "- 3"
41            self._truncate_seq_pair(tokens_a, tokens_b,
42                                   self._max_seq_length - 3)
43        else:
44            # Account for [CLS] and [SEP] with "- 2"
45            if len(tokens_a) > self._max_seq_length - 2:
46                tokens_a = tokens_a[0:(self._max_seq_length - 2)]
47
48        # The embedding vectors for 'type=0' and 'type=1' were learned during
49        # pre-training and are added to the wordpiece embedding vector
50        # (and position vector). This is not "strictly" necessary since
51        # the [SEP] token unambiguously separates the sequences, but it makes
52        # it easier for the model to learn the concept of sequences.
53
54        # For classification tasks, the first vector (corresponding to [CLS]) is
55        # used as the "sentence vector". Note that this only makes sense because
56        # the entire model is fine-tuned.
57        #vocab = self._tokenizer.vocab
58        vocab = self._vocab
```

```

59     tokens = []
60     tokens.append(vocab.cls_token)
61     tokens.extend(tokens_a)
62     tokens.append(vocab.sep_token)
63     segment_ids = [0] * len(tokens)
64
65     if tokens_b:
66         tokens.extend(tokens_b)
67         tokens.append(vocab.sep_token)
68         segment_ids.extend([1] * (len(tokens) - len(segment_ids)))
69
70     input_ids = self._tokenizer.convert_tokens_to_ids(tokens)
71
72     # The valid length of sentences. Only real tokens are attended to.
73     valid_length = len(input_ids)
74
75     if self._pad:
76         # Zero-pad up to the sequence length.
77         padding_length = self._max_seq_length - valid_length
78         # use padding tokens for the rest
79         input_ids.extend([vocab[vocab.padding_token]] * padding_length)
80         segment_ids.extend([0] * padding_length)
81
82     return np.array(input_ids, dtype='int32'), np.array(valid_length, dtype='int32'),\
83            np.array(segment_ids, dtype='int32')
84
85

```

이후 BERTDataset을 다시 정의한다. 해당 부분은 위와

```

In [10]: 1 from kobert_tokenizer import KoBERTTokenizer
2         from transformers import BertModel
3         from transformers import AdamW
4         from transformers.optimization import get_cosine_schedule_with_warmup
5
6         class BERTDataset(Dataset):
7             def __init__(self, dataset, sent_idx, label_idx, bert_tokenizer, vocab, max_len,
8                          pad, pair):
9                 transform = BERTSentenceTransform(bert_tokenizer, max_seq_length=max_len, vocab=vocab, pad=pad, pair=pair)
10                #transform = nlp.data.BERTSentenceTransform(
11                #    tokenizer, max_seq_length=max_len, pad=pad, pair=pair)
12                self.sentences = [transform[i[sent_idx]] for i in dataset]
13                self.labels = [np.int32(i[label_idx]) for i in dataset]
14
15            def __getitem__(self, i):
16                return (self.sentences[i] + (self.labels[i], ))
17
18            def __len__(self):
19                return (len(self.labels))
20

```

이를 통해 data_train에서 tok 으로 설정했던 부분을 tokenizer 로 설정하여 수정하였다.

```

In [14]: 1 tokenizer = KoBERTTokenizer.from_pretrained('skt/kobert-base-v1')
2         bert_model = BertModel.from_pretrained('skt/kobert-base-v1', return_dict=False)
3         vocab = nlp.vocab.BERTVocab.from_sentencepiece(tokenizer.vocab_file, padding_token='[PAD]')
4
5         #저는 csv 파일을 사용하여 이부분은 dataset_train.values 로 해주었습니다.
6         data_train = BERTDataset(dataset_train, 0, 1, tokenizer, vocab, max_len, True, False)
7         data_test = BERTDataset(dataset_test, 0, 1, tokenizer, vocab, max_len, True, False)
8
9

```

parameter 의 경우, 예시 코드에 있는 값들을 동일하게 설정해주었다.

```

In [12]: 1 # Setting parameters
2         max_len = 64
3         batch_size = 16
4         warmup_ratio = 0.1
5         num_epochs = 5
6         max_grad_norm = 1
7         log_interval = 200
8         learning_rate = 5e-5

```

사이킷런에서 제공해주는 train_test_split 메서드를 활용해 기존 data_list 를 train 데이터 셋과 test 데이터셋으로 나눈다. 5:1 비율로 나누었다.

```

In [13]: 1 #train & test 데이터로 나누기
2         from sklearn.model_selection import train_test_split
3
4         dataset_train, dataset_test = train_test_split(data_list, test_size=0.2, shuffle=True, random_state=34)

```

위에서 구현한 BERTDataset 클래스를 활용해 tokenization, int encoding, padding 을 진행하였다.

```

In [15]: 1 train_data_loader = torch.utils.data.DataLoader(data_train, batch_size=batch_size, num_workers=5)
2         test_data_loader = torch.utils.data.DataLoader(data_test, batch_size=batch_size, num_workers=5)

```

torch 형식의 dataset을 만들어주면서, 입력 데이터셋의 처리가 모두 끝났다.

6. KoBERT 모델 구현하기

```
In [16]: 1 class BERTClassifier(nn.Module):
2     def __init__(self,
3                 bert,
4                 hidden_size = 768,
5                 num_classes=7, ##클래스 수 조정##
6                 dr_rate=None,
7                 params=None):
8         super(BERTClassifier, self).__init__()
9         self.bert = bert
10        self.dr_rate = dr_rate
11
12        self.classifier = nn.Linear(hidden_size , num_classes)
13        if dr_rate:
14            self.dropout = nn.Dropout(p=dr_rate)
15
16        def gen_attention_mask(self, token_ids, valid_length):
17            attention_mask = torch.zeros_like(token_ids)
18            for i, v in enumerate(valid_length):
19                attention_mask[i][:v] = 1
20            return attention_mask.float()
21
22        def forward(self, token_ids, valid_length, segment_ids):
23            attention_mask = self.gen_attention_mask(token_ids, valid_length)
24
25            pooler = self.bert(input_ids = token_ids, token_type_ids = segment_ids.long(), attention_mask = attention_mask.float()).to(torch.device('cpu'))
26            if self.dr_rate:
27                out = self.dropout(pooler)
28            return self.classifier(out)
```

BERT모델을 불러온다.

```
In [17]: 1 #train
2
In [18]: 1 model = BERTClassifier(bertmodel, dr_rate=0.5).to(device)
2
3 #optimizer와 schedule 설정
4 no_decay = ['bias', 'LayerNorm.weight']
5 optimizer_grouped_parameters = [
6     {'params': [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)], 'weight_decay': 0.01},
7     {'params': [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)], 'weight_decay': 0.0}
8 ]
9
10 optimizer = AdamW(optimizer_grouped_parameters, lr=learning_rate)
11 loss_fn = nn.CrossEntropyLoss() # 다중분류를 위한 대표적인 loss func
12
13 t_total = len(train_dataloader) * num_epochs
14 warmup_step = int(t_total * warmup_ratio)
15
16 scheduler = get_cosine_schedule_with_warmup(optimizer, num_warmup_steps=warmup_step, num_training_steps=t_total)
17
18 #정확도 측정을 위한 함수 정의
19 def calc_accuracy(X,Y):
20     max_vals, max_indices = torch.max(X, 1)
21     train_acc = (max_indices == Y).sum().data.cpu().numpy()/max_indices.size()[0]
22     return train_acc
23
24 train_dataloader
```

Out[18]: <torch.utils.data.dataloader.DataLoader at 0x7f6ee7830640>

해당 부분은 예제 코드와 동일하게 사용하였다.

7. 모델 학습시키기

- KoBERT 모델을 학습시키는 코드이다. epoch는 5로 지정하였다.
- 앞서 이 모델에서 학습시킬 수 있도록 입력 데이터셋을 처리하고, 파라미터를 모두 지정하였으므로 예시 코드와 동일하게 진행하였다.
- 이 부분에서 원본 예제 코드와 다른 점은 정확도가 초반에 17%로 나왔다는 것이었다.
- 따라서, 이 부분을 위하여 위에 tokens_a = self_tokenizer(text_a) 로 되어 있던 부분을 tokens_a = self_tokenizer.tokenize(text_a) 로 수정함으로써 정확도를 높였다.

```
In [19]: 1 train_history=[]
2 test_history=[]
3 loss_history=[]
4 for e in range(num_epochs):
5     train_acc = 0.0
6     test_acc = 0.0
7     model.train()
8     for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(train_data_loader)):
9         optimizer.zero_grad()
10        token_ids = token_ids.long().to(device)
11        segment_ids = segment_ids.long().to(device)
12        valid_length = valid_length
13        label = label.long().to(device)
14        out = model(token_ids, valid_length, segment_ids)
15
16        #print(label.shape, out.shape)
17        loss = loss_fn(out, label)
18        loss.backward()
19        torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
20        optimizer.step()
21        scheduler.step() # Update learning rate schedule
22        train_acc += calc_accuracy(out, label)
23        if batch_id % log_interval == 0:
24            print('epoch {} batch id {} loss {} train acc {}'.format(e+1, batch_id+1, loss.data.cpu().numpy(), train_acc / (batch_id+1)))
25            train_history.append(train_acc / (batch_id+1))
26            loss_history.append(loss.data.cpu().numpy())
27        print("epoch {} train acc {}".format(e+1, train_acc / (batch_id+1)))
28        #train_history.append(train_acc / (batch_id+1))
29
30    model.eval()
31    for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(test_data_loader)):
32        token_ids = token_ids.long().to(device)
33        segment_ids = segment_ids.long().to(device)
34        valid_length = valid_length
35        label = label.long().to(device)
36        out = model(token_ids, valid_length, segment_ids)
37        test_acc += calc_accuracy(out, label)
38        print("epoch {} test acc {}".format(e+1, test_acc / (batch_id+1)))
39        test_history.append(test_acc / (batch_id+1))
40
41    #/tmp/ipykernel_5495/1756086221.py:8: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
42    # Please use tqdm_notebook.tqdm instead of 'tqdm.tqdm_notebook'
43    for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(train_data_loader)):
```

```
/tmp/ipykernel_5495/1756086221.py:8: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use tqdm_notebook.tqdm instead of 'tqdm.tqdm_notebook'
for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(train_data_loader)):
```

```
100% |#####| 18201/18200 [02:21<00:00, 13.958s]
```

```
epoch 1 batch id 1 loss 2.98702634447910 train acc 0.225
epoch 1 batch id 200 loss 2.09521502121138 train acc 0.183760867164179
epoch 1 batch id 400 loss 1.71819561552095 train acc 0.2459478392166326
epoch 2 batch id 600 loss 1.3363865271334 train acc 0.302847216257187
epoch 1 batch id 800 loss 1.459418957404186 train acc 0.3327408488138553
epoch 2 batch id 1000 loss 1.361364264916948 train acc 0.3622317626315326
epoch 1 batch id 1200 loss 1.56336867514038 train acc 0.338466777085524
epoch 2 batch id 1400 loss 1.046113066124942 train acc 0.395181223411649
epoch 1 batch id 1600 loss 1.226231337186454 train acc 0.428186422178615
epoch 1 batch id 1800 loss 1.48158169306232 train acc 0.413918988956395
epoch 2 train acc 0.42837151058297
```

```
/tmp/ipykernel_5495/1756086221.py:11: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use tqdm_notebook.tqdm instead of 'tqdm.tqdm_notebook'
for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(test_data_loader)):
```

```
100% |#####| 483/483 [00:10<00:00, 46.71s]
```

```
epoch 1 test acc 0.48845596267575
```

```
100% |#####| 18201/18200 [02:22<00:00, 13.588s]
```

```
epoch 2 batch id 1 loss 1.61045624195167 train acc 0.375
epoch 2 batch id 200 loss 1.338242748971168 train acc 0.4642022080809503
epoch 2 batch id 400 loss 1.22462052071079 train acc 0.4958478392166326
epoch 2 batch id 600 loss 1.2014862220942 train acc 0.506941367294982
epoch 2 batch id 800 loss 1.37261272773062 train acc 0.539513734756617
epoch 2 batch id 1000 loss 1.221986422491118 train acc 0.535238162221632
epoch 2 batch id 1200 loss 1.070387616611816 train acc 0.541378834271489
epoch 2 batch id 1400 loss 0.713881746032144 train acc 0.552128618205495
epoch 2 batch id 1600 loss 0.81381830492307 train acc 0.538258588984965
epoch 2 batch id 1800 loss 0.71154688583097 train acc 0.54243574614369
epoch 2 train acc 0.56823299498217
```

```
100% |#####| 483/483 [00:10<00:00, 46.68s]
```

```
epoch 2 test acc 0.52783538628216
```

```
100% |#####| 18201/18200 [02:23<00:00, 13.476s]
```

```
epoch 3 batch id 1 loss 1.286204892388 train acc 0.4375
epoch 3 batch id 200 loss 0.9363709777015 train acc 0.628213218085773
epoch 3 batch id 400 loss 0.8747506977625 train acc 0.62344140505732
epoch 3 batch id 600 loss 0.99633059251492 train acc 0.624616212070889
epoch 3 batch id 800 loss 0.92614121423878 train acc 0.648377410757575
epoch 3 batch id 1000 loss 0.7603422008490 train acc 0.651189124205143
epoch 3 batch id 1200 loss 0.82134157008368 train acc 0.658133524731237
epoch 3 batch id 1400 loss 0.821186981814895 train acc 0.6884712361702
epoch 3 batch id 1600 loss 0.6752056280187 train acc 0.67507878232371
epoch 3 batch id 1800 loss 0.764487245107156 train acc 0.67918102463681
```

```
epoch 3 batch id 1800 loss 0.2618873285937139 train acc 0.6753321294985461
```

```
epoch 3 train acc 0.642484143791516
```

```
100% |#####| 483/483 [00:10<00:00, 46.68s]
```

```
epoch 3 test acc 0.540391155632585
```

```
100% |#####| 18201/18200 [02:22<00:00, 13.588s]
```

```
epoch 4 batch id 1 loss 1.1537116193771307 train acc 0.5025
```

```
epoch 4 batch id 200 loss 0.85421497192201 train acc 0.762046976222096
```

```
epoch 4 batch id 400 loss 0.803777402355009 train acc 0.7454538633062584
```

```
epoch 4 batch id 600 loss 0.756961903821404 train acc 0.745944931163866
```

```
epoch 4 batch id 800 loss 0.751258555845009 train acc 0.7513849793037315
```

```
epoch 4 batch id 1000 loss 0.494273139581804 train acc 0.758863128861339
```

```
epoch 4 batch id 1200 loss 0.4756567167705086 train acc 0.768384529513185
```

```
epoch 4 batch id 1400 loss 0.537431173245655 train acc 0.775829744612845
```

```
epoch 4 batch id 1600 loss 0.421181478732317 train acc 0.768983786118959
```

```
epoch 4 batch id 1800 loss 0.4681712758991194 train acc 0.78267795659727
```

```
epoch 4 train acc 0.785221338327838
```

```
100% |#####| 483/483 [00:10<00:00, 46.78s]
```

```
epoch 4 test acc 0.544335143634804
```

```
100% |#####| 18201/18200 [02:22<00:00, 13.588s]
```

```
epoch 5 batch id 1 loss 0.81381830492307 train acc 0.6075
```

```
epoch 5 batch id 200 loss 0.8112522429797283 train acc 0.827423713191284
```

```
epoch 5 batch id 400 loss 0.4134868810708494 train acc 0.87115878178155
```

```
epoch 5 batch id 600 loss 0.725997109066252 train acc 0.8181881831847122
```

```
epoch 5 batch id 800 loss 0.81635533832751 train acc 0.834737877115358
```

```
epoch 5 batch id 1000 loss 0.511195958291 train acc 0.84648653546653
```

```
epoch 5 batch id 1200 loss 0.281775315856683 train acc 0.8489797803171399
```

```
epoch 5 batch id 1400 loss 0.40760084264632 train acc 0.848959711744754
```

```
epoch 5 batch id 1600 loss 0.25525159852945 train acc 0.84938138433161
```

```
epoch 5 batch id 1800 loss 0.47521275844463 train acc 0.850117950055124
```

```
epoch 5 train acc 0.8331321385951
```

```
100% |#####| 483/483 [00:10<00:00, 46.10s]
```

```
epoch 5 test acc 0.5438278103815439
```

수정하고 나니, train dataset에 대해서는 0.979, test dataset에 대해서는 0.918의 정확도를 기록했다

8. 직접 입력한 새로운 문장으로 테스트

이제 직접 문장을 만들어 학습된 모델이 다중 분류를 잘 해내는지 알아보기 위해 학습된 모델을 활용하여 다중 분류된 클래스를 출력해주는 predict 함수를 구현한다.

- predict 함수는 예측할 문장을 입력으로 받아 해당 문장에 대한 감정을 예측하고 출력한다.
- 입력 문장을 데이터 형식에 맞게 리스트로 생성하고, 이를 데이터셋 형식으로 변환한다.
- 변환한 데이터셋을 사용하여 데이터 로더를 생성하고, 모델을 eval모드로 바꾼다.
- 데이터 로더에서 배치별로 데이터를 가져와 모델에 입력하여 예측 결과를 얻어 레이블을 생성하고 출력한다.

```
In [20]: def predict(predict_sentence):
1
2
3     data = [predict_sentence, '0']
4     dataset_another = [data]
5
6     another_test = BERTDataset(dataset_another, 0, 1, tokenizer, vocab, max_len, True, False)
7     test_dataloader = torch.utils.data.DataLoader(another_test, batch_size=batch_size, num_workers=5)
8
9     model.eval()
10
11     for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(test_dataloader):
12         token_ids = token_ids.long().to(device)
13         segment_ids = segment_ids.long().to(device)
14
15         valid_length = valid_length
16         label = label.long().to(device)
17
18         out = model(token_ids, valid_length, segment_ids)
19
20
21     test_eval=[]
22     for i in out:
23         logits=i
24         logits = logits.detach().cpu().numpy()
25
26         if np.argmax(logits) == 0:
27             test_eval.append("공포")
28         elif np.argmax(logits) == 1:
29             test_eval.append("놀람")
30         elif np.argmax(logits) == 2:
31             test_eval.append("분노")
32         elif np.argmax(logits) == 3:
33             test_eval.append("슬픔")
34         elif np.argmax(logits) == 4:
35             test_eval.append("중립")
36         elif np.argmax(logits) == 5:
37             test_eval.append("행복")
38         elif np.argmax(logits) == 6:
39             test_eval.append("혐오")
40
41     print(">> 입력하신 내용에서 " + test_eval[0] + " 느껴집니다.")
42
```

질문 및 입력을 무한 반복한다.

입력하면 입력한 내용에 따라 predict() 에서 감정을 분류한 값을 출력한다. 0 을 입력하면 무한 반복이 종료된다.

```
In [21]: 1 #질문 무한반복하기! 0 입력시 종료
2 end = 1
3 while end == 1 :
4     sentence = input("하고싶은 말을 입력해주세요 : ")
5     if sentence == "0" :
6         break
7     predict(sentence)
8     print("\n")
```

- flow graph 요청

[질문]

발화 감정을 분류하는 파이썬 프로그램을 만들어줘. 조건은 1. 분류 감정엔 행복,분노,혐오, 공포,중립, 슬픔,놀람 구분한다 2. 한국어 자연어 처리를 위해 기존 kobert 모델을 fine tuning해서 사용 3. 모델 학습이 끝난 후 사용자가 입력한 문장을 바탕으로 감정 분류하기

이 프로그램에 대한 flow graph 랑 psedu code를 줘

[답변]

프로그램의 구현에 앞서, 다음과 같은 흐름도(flow graph)를 기반으로 설명해 드리겠습니다.

1. 사용자로부터 문장을 입력 받는다.
2. 입력된 문장을 전처리한다.
3. 전처리된 문장을 kobert 모델에 입력으로 전달한다.
4. kobert 모델을 통해 문장의 감정을 예측한다.
5. 예측된 감정을 출력한다.

다음은 위 흐름도에 기반한 의사 코드(pseudocode) 예시입니다.

```
# 필요한 라이브러리 및 모듈 임포트
import torch
from kobert_transformers import get_tokenizer
from kobert_transformers import get_kobert_model

# kobert 모델 로드
tokenizer = get_tokenizer()
model = get_kobert_model()

# 감정 레이블 매핑
emotion_labels = ['행복', '분노', '혐오', '공포', '중립', '슬픔', '놀람']

# 사용자 입력 받기
sentence = input("문장을 입력하세요: ")

# 문장 전처리
tokens = tokenizer.tokenize(sentence)
ids = tokenizer.convert_tokens_to_ids(tokens)
input_ids = torch.tensor([ids])

# 감정 분류
outputs = model(input_ids)
emotion_index = torch.argmax(outputs).item()
emotion = emotion_labels[emotion_index]

# 결과 출력
print("입력된 문장의 감정은:", emotion)
```

위의 코드는 kobert 모델을 사용하여 한국어 문장의 감정을 분류하는 기본적인 흐름을 보여줍니다. 실제로 사용하기 위해서는 kobert 모델을 미리 학습시킨 후, 해당 모델을 로드해야 합니다. 또한, 필요한 라이브러리 및 모듈을 설치해야 정상적으로 실행될 수 있습니다.

해당 프로그램에 대한 간단한 flow graph와 pseudo code 를 제공 받음

**** 구체적인 flow graph 시각화 요청**

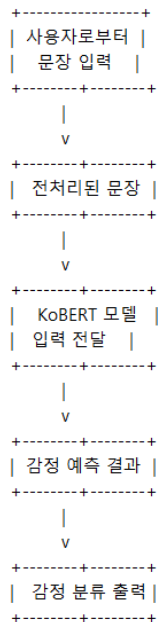
[질문]

해당 프로그램을 구현하기 위해 사용할 프로그램 기법이나 모델 이름을 구체적으로 알려주고 이를 바탕으로 flow graph를 그려줘

[답변]

좋습니다! 한국어 감정 분류를 위해 사용할 수 있는 기법 중 하나는 KoBERT(Korean BERT)입니다. KoBERT는 한국어 자연어 처리를 위해 BERT 모델을 fine-tuning한 모델로, 문장의 의미를 이해하고 감정을 분류하는 데에 사용할 수 있습니다.

따라서 아래와 같이 수정된 flow graph를 제시해 드리겠습니다.



위의 flow graph를 참고하여 코드를 구현하시면 됩니다. KoBERT 모델을 사용하기 위해서는 KoBERT 모델을 미리 다운로드하고 로드해야 합니다. 또한, 전처리와 후처리 단계에서 필요한 함수와 라이브러리도 추가로 사용하게 될 것입니다.

해당 프로그램에 대한 간단한 flow graph 시각화 성공

**** 추가적으로 특정 단계(전처리, 예측 과정) 구체화한 flow graph 요청**

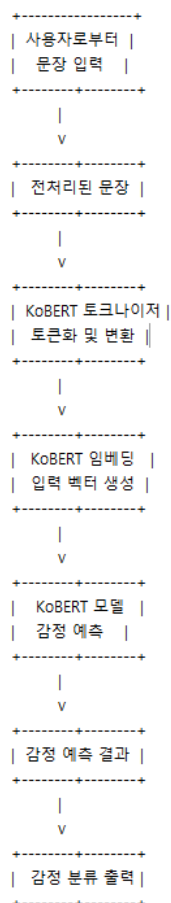
[질문]

전처리 과정이랑 감정 예측 결과에 사용할 구체적인 방법 및 모델을 고려해서 다시 flow graph 를 그려줘

[답변]

감정 분류를 위한 전처리 과정에서는 문장을 토큰화하고 숫자로 변환하여 모델에 입력으로 전달해야 합니다. 이를 위해 KoBERT 모델을 사용하며, KoBERT의 토큰라이저와 임베딩 레이어를 활용합니다. 감정 예측 결과는 KoBERT 모델의 출력을 기반으로 가장 높은 확률을 가진 감정 레이블을 선택하여 분류합니다.

따라서 수정된 flow graph는 다음과 같습니다.



위의 flow graph를 참고하여 코드를 구현하시면 됩니다. 전처리에는 KoBERT의 토큰라이저를 사용하고, 임베딩에는 KoBERT 모델의 임베딩 레이어를 활용합니다. 감정 예측 결과는 KoBERT 모델의 출력을 기반으로 가장 높은 확률을 가진 감정 레이블을 선택하여 분류합니다.

[질문]

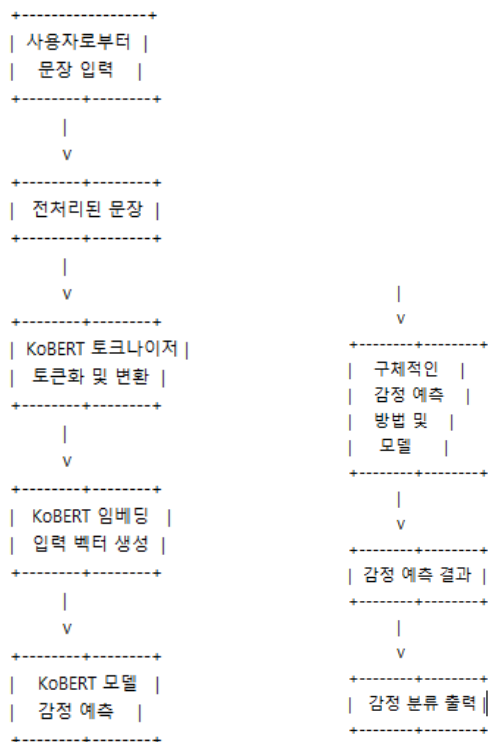
감정 예측 결과를 위해 사용할 모델과 방법은 뭘까? 구체적인 방법을 추가해서 다시 그려줘

[답변]

감정 예측 결과를 위해 일반적으로 사용되는 방법은 다음과 같습니다. 감정 분류 문제는 다중 클래스 분류 문제로 간주되며, 일반적으로 소프트맥스(softmax) 함수를 사용하여 각 클래스에 대한 확률을 계산합니다. 각 클래스의 확률을 계산한 후, 가장 높은 확률을 가진 클래스를 선택하여 감정을 예측합니다.

일반적으로 사용되는 모델 중 하나는 다중 레이어 퍼셉트론(MLP, Multi-Layer Perceptron)입니다. MLP는 하나 이상의 은닉층을 가진 인공 신경망으로, 다중 클래스 분류 문제에 잘 적용됩니다. 다른 모델로는 서포트 벡터 머신(SVM, Support Vector Machine), 랜덤 포레스트(Random Forest), 그래디언트 부스팅(Gradient Boosting) 등도 사용될 수 있습니다.

따라서 수정된 flow graph는 다음과 같습니다.



위의 flow graph를 참고하여 코드를 구현하시면 됩니다. 구체적인 감정 예측 방법과 모델은 사용자의 선택에 따라 다양하게 적용될 수 있습니다. MLP, SVM, Random Forest, Gradient Boosting 등 다양한 모델을 사용하여 감정 예측을 수행할 수 있습니다.

**전처리 과정은 구체적이지만 모델 예측 단계는 사용할 구체적인 알고리즘을 제시 안 하고 후
보군만 제시해줌**

- pseudo code 요청

[질문]

위 프로그램에 대한 pseudo code를 쥐

[답변]

다음은 요청하신 프로그램의 수정된 의사 코드(pseudocode) 예시입니다.

```
# 필요한 라이브러리 및 모듈 импорт
import torch
from kobert_transformers import get_tokenizer
from kobert_transformers import get_kobert_model
from sklearn.metrics import classification_report

# kobert 모델 로드
tokenizer = get_tokenizer()
model = get_kobert_model()

# 감정 레이블 매핑
emotion_labels = ['행복', '분노', '혐오', '공포', '중립', '슬픔', '놀람']

# 사용자 입력 받기
sentence = input("문장을 입력하세요: ")

# 문장 전처리
tokens = tokenizer.tokenize(sentence)
ids = tokenizer.convert_tokens_to_ids(tokens)
input_ids = torch.tensor([ids])

# 감정 분류
outputs = model(input_ids)
emotion_index = torch.argmax(outputs).item()
emotion = emotion_labels[emotion_index]

# 결과 출력
print("입력된 문장의 감정은:", emotion)

# 추가 기능: 감정 분류 결과 평가
# 예시로서 추가한 코드이며, 필요에 따라 사용하실 수 있습니다.
# true_emotion = input("정답 감정을 입력하세요: ")
# true_label = emotion_labels.index(true_emotion)
# predicted_label = emotion_index
# print(classification_report([true_label], [predicted_label], target_names=emotion_labels))
```

위의 의사 코드를 참고하여 실제로 구현하고 실행하면, 사용자로부터 문장을 입력받고 해당 문장의 감정을 분류하는 기능을 갖는 프로그램이 동작합니다. 필요에 따라 추가 기능인 감정 분류 결과 평가를 사용할 수도 있습니다.

해당 프로그램에 대한 pseudo code 구현 성공

**** 코드 형식이 아닌 논리 형식의 pseudo code 요청**

[질문]

다음 규칙을 기반으로 위 프로그램(발화 감정 분류)의 pseudo code를 짜줘

작성 규칙 1 : 구체적 방법보다는 무슨 일을 할 것인가만을 기술한다.

작성 규칙 2 : 입력, 치환, 산술, 관계, 논리, 출력의 기본기능 단위로 한번에 한가지만 기술한다.

작성 규칙 3 : 목적어 + 동사 형식으로 기술한다.

작성 규칙 4 : 처리단계 레벨은 되도록 3단계까지만 표현한다.(들여쓰기 3단계)

작성 순서 1 : 문제 전체에 대한 처리를 순차적으로 표현

작성 순서 2 : 순차적인 표현에서 선택(분기)이나 반복이 필요한 부분에 추가하기

예시 수도코드

문제 : 이름과 국어, 영어 점수를 입력(input)받고, 총점과 평균을 계산하고, 평균을 기준으로 평가하고, 학생 성적을 출력한다. (output)

//작성 순서1 :

//1. 이름, 국어, 영어점수를 입력받는다.

//2. 번호를 매긴다.

//3. 총점을 구한다.

//4. 평균을 구한다.

//5. 평가를 한다.

//6. 평가를 출력한다.

//7. 끝낸다.

//작성순서2 :

//1. 이름, 국어, 영어점수를 입력받는다.

//2. 번호를 매긴다.

//3. 총점을 구한다.

//4. 평균을 구한다.

//5. 평균이 70점 이상인지 판단한다.(선택/분기)

// 5.1 참이면 통과한 것으로 판단

// 5.2 거짓이면 실패한 것으로 판단

//6. 평가를 출력한다.

//7. 끝낸다.

//작성순서3 :

//1. 파일의 끝이 아닌 동안 반복한다.(반복)

// 1.1 이름 국어, 영어점수를 입력받는다.

// 1.2 번호를 매긴다.

// 1.3 총점을 구한다.

// 1.4 평균을 구한다.

// 1.5 평균이 70점 이상인지 판단한다.(선택/분기)

// 1.5.1 참이면 통과한 것으로 판단

// 1.5.2 거짓이면 실패한 것으로 판단

// 1.6 평가를 출력한다.

//2. 끝낸다.

[답변]

좋습니다! 한국어 감정 분류를 위해 사용할 수 있는 기법 중 하나는 KoBERT(Korean BERT)입니다.

KoBERT는 한국어 자연어 처리를 위해 BERT 모델을 fine-tuning한 모델로, 문장의 의미를 이해하고 감정을 분류하는 데에 사용할 수 있습니다.

따라서 아래와 같이 수정된 flow graph를 제시해 드리겠습니다.

```
// 작성 순서 1:
// 1. 사용자로부터 입력 문장을 받는다.
// 2. 문장을 전처리한다.
// 3. 전처리된 문장을 감정 예측 모델에 입력으로 넣는다.
// 4. 모델이 예측한 감정을 가져온다.
// 5. 예측된 감정을 출력한다.
// 6. 끝낸다.

// 작성 순서 2:
// 1. 사용자로부터 입력 문장을 받는다.
sentence = input("문장을 입력하세요: ")

// 2. 문장을 전처리한다.
preprocessed_sentence = preprocess(sentence)

// 3. 전처리된 문장을 감정 예측 모델에 입력으로 넣는다.
predicted_emotion = predict_emotion(preprocessed_sentence)

// 4. 모델이 예측한 감정을 가져온다.
emotion = get_emotion(predicted_emotion)

// 5. 예측된 감정을 출력한다.
print("예측된 감정:", emotion)

// 6. 끝낸다.
```

위의 의사 코드를 참고하여 구현하시면 됩니다. 순차적인 명령문을 사용하여 사용자로부터 문장을 입력받고, 전처리한 후 감정 예측 모델에 입력으로 넣어 감정을 예측하고, 예측된 감정을 출력합니다.

해당 프로그램에 대한 논리형 pseudo code 구현 성공

- test case 요청

[질문]

이번에는 위 발화 분류 프로그램에 대한 test case를 만들어줘

[답변]

다음은 발화 분류 프로그램에 대한 테스트 케이스 예시입니다.

행복한 감정:

입력 문장: "오늘은 정말 기분이 좋아요!"

예상 감정: 행복

분노한 감정:

입력 문장: "이런 일이 또 생기면 화가 나요!"

예상 감정: 분노

혐오하는 감정:

입력 문장: "이런 일을 절대로 다시 경험하고 싶지 않아요."

예상 감정: 혐오

공포한 감정:

입력 문장: "어둠 속에서 무서운 소리가 들려와서 겁이 났어요."

예상 감정: 공포

중립한 감정:

입력 문장: "오늘 날씨가 좋네요."

예상 감정: 중립

슬픈 감정:

입력 문장: "이별이 너무 슬퍼서 마음이 아파요."

예상 감정: 슬픔

놀란 감정:

입력 문장: "정말 놀랐어요! 이런 일이 일어날 줄은 몰랐어요."

예상 감정: 놀람

위의 테스트 케이스 예시를 사용하여 프로그램을 테스트해보세요. 각 입력 문장에 대해 예상되는 감정과 프로그램이 분류한 감정을 비교하여 정확성을 확인할 수 있습니다.

[결과]

하고싶은 말을 입력해주세요 : 오늘은 정말 기분이 좋아요!
>> 입력하신 내용에서 행복이 느껴집니다.

하고싶은 말을 입력해주세요 : 이런 일이 또 생기면 화가 나요!
>> 입력하신 내용에서 분노가 느껴집니다.

하고싶은 말을 입력해주세요 : 이런 일을 절대로 다시 경험하고 싶지 않아요!
>> 입력하신 내용에서 분노가 느껴집니다.

하고싶은 말을 입력해주세요 : 어둠 속에서 무서운 소리가 들려와서 겁이 났어요.
>> 입력하신 내용에서 공포가 느껴집니다.

하고싶은 말을 입력해주세요 : 오늘 날씨가 좋네요
>> 입력하신 내용에서 행복이 느껴집니다.

하고싶은 말을 입력해주세요 : 이별이 너무 슬퍼서 마음이 아파요
>> 입력하신 내용에서 슬픔이 느껴집니다.

하고싶은 말을 입력해주세요 : 정말 놀랐어요! 이런 일이 일어날 줄은 몰랐어요.
>> 입력하신 내용에서 놀람이 느껴집니다.

2,3,번 질문을 제외하곤 chat gpt의 예측 결과가 맞다.

- Chat GPT에게 code 요청 & 검토

[질문]

Please create a Python program based on an AI algorithm that uses kobert to determine what emotions the text you enter has. And tell me the source code.

[답변]

Sure! To determine the emotions present in a text using the KoBERT model, we can utilize the transformers library in Python. This library provides an interface to various pre-trained language models, including KoBERT. You'll need to install the transformers library and the required dependencies. You can do this by running the following command:

```
pip install torch transformers
```

Once you have installed the necessary libraries, you can use the following Python code to analyze the emotions in a given text using KoBERT:

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification

# Load the KoBERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('monologg/kobert')
model = BertForSequenceClassification.from_pretrained('monologg/kobert', num_labels=7)

# Define the emotion labels
emotion_labels = ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surprise']

# Function to analyze the emotions in a given text
def analyze_emotions(text):
    # Tokenize the text
    tokens = tokenizer.encode(text, padding='max_length', truncation=True, max_length=128, return_tensors='pt')

    # Make predictions
    with torch.no_grad():
        outputs = model(**tokens)
        logits = outputs.logits

    # Convert logits to probabilities
    probabilities = torch.softmax(logits, dim=1).squeeze()

    # Map probabilities to emotions
    emotions = {label: prob.item() for label, prob in zip(emotion_labels, probabilities)}

    return emotions

# Example usage
text = "I feel so happy and excited!"
emotions = analyze_emotions(text)
print(emotions)
```

In this code, we first import the necessary libraries, including torch for tensor operations and

BertTokenizer and BertForSequenceClassification from transformers to load the KoBERT model. We then define the emotion labels corresponding to the KoBERT model's output.

The analyze_emotions function takes a text input, tokenizes it using the KoBERT tokenizer, and passes it through the model for emotion analysis. The model's output logits are converted to probabilities using softmax, and the probabilities are then mapped to the emotion labels. The function returns a dictionary containing the emotions and their corresponding probabilities.

Finally, we provide an example usage where we call the analyze_emotions function with a sample text and print the resulting emotions and their probabilities.

Please note that the accuracy and performance of the emotion analysis can vary depending on the quality of the KoBERT model and the training data it was trained on. Also, keep in mind that the KoBERT model is trained on Korean text, so it may not perform optimally on non-Korean text.

[결과값]

Downloading (...)okenizer_config.json: 100%  51.0/51.0 [00:00<00:00, 2.33kB/s]

TypeError Traceback (most recent call last)
<ipython-input-2-974c21815736> in <cell line: 5>()

```
3  
4 # Load the KoBERT tokenizer and model  
----> 5 tokenizer = BertTokenizer.from_pretrained('monologg/kobert')  
6 model = BertForSequenceClassification.from_pretrained('monologg/kobert', num_labels=7)  
7
```

----- 3 frames -----
/usr/lib/python3.10/genericpath.py in isfile(path)
28 """Test whether a path is a regular file"""
29 try:
----> 30 st = os.stat(path)
31 except (OSError, ValueError):
32 return False

TypeError: stat: path should be string, bytes, os.PathLike or integer, not NoneType

SEARCH STACK OVERFLOW

Downloading (...)lve/main/config.json: 100%  426/426 [00:00<00:00, 21.6kB/s]

**** 에러가 발생하여 해당 에러 해결을 요청**

[질문]

The following error occurred on line 5 of the code you made.

"Traceback (most recent call last)"

And the following error occurred on line 30.

"stat: path should be string, bytes, os.PathLike or integer, not NoneType"

Please fix these errors.

[답변]

Apologies for the errors. It seems that the BertForSequenceClassification model in the transformers library has been updated, and there have been some changes to the usage. Please try the modified code below:

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification

# Load the KoBERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('monologg/kobert')
model = BertForSequenceClassification.from_pretrained('monologg/kobert', num_labels=7)

# Define the emotion labels
emotion_labels = ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surprise']

# Function to analyze the emotions in a given text
def analyze_emotions(text):
    # Tokenize the text
    tokens = tokenizer.encode_plus(text, padding='max_length', truncation=True, max_length=128, return_tensors='pt')

    # Make predictions
    with torch.no_grad():
        outputs = model(**tokens)
        logits = outputs.logits

    # Convert logits to probabilities
    probabilities = torch.softmax(logits, dim=1).squeeze()

    # Map probabilities to emotions
    emotions = {label: prob.item() for label, prob in zip(emotion_labels, probabilities)}

    return emotions

# Example usage
text = "I feel so happy and excited!"
emotions = analyze_emotions(text)
print(emotions)
```

In this modified code, the key change is in line 16, where the encode method has been replaced with encode_plus. This change is due to the updated transformers library.

Please give it a try, and let me know if you encounter any further issues.

[결과값]

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-3-134280c7b344> in <cell line: 5>()  
    3  
    4 # Load the KoBERT tokenizer and model  
----> 5 tokenizer = BertTokenizer.from_pretrained('monologg/kobert')  
    6 model = BertForSequenceClassification.from_pretrained('monologg/kobert', num_labels=7)  
    7
```

```
----- 3 frames -----  
/usr/lib/python3.10/genericpath.py in isfile(path)  
    28     """Test whether a path is a regular file"""  
    29     try:  
----> 30         st = os.stat(path)  
    31     except (OSError, ValueError):  
    32         return False
```

TypeError: stat: path should be string, bytes, os.PathLike or integer, not NoneType

SEARCH STACK OVERFLOW

**** 동일한 에러가 발생하여 다시 에러 해결을 요청**

[질문]

The same error has occurred. Please fix the error.

[답변]

I apologize for the inconvenience. To address the error, we can modify the code further. Please try the updated code below:

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification

# Load the KoBERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('monologg/kobert')
model = BertForSequenceClassification.from_pretrained('monologg/kobert', num_labels=7)

# Define the emotion labels
emotion_labels = ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surprise']

# Function to analyze the emotions in a given text
def analyze_emotions(text):
    # Tokenize the text
    tokens = tokenizer.encode_plus(text, padding='max_length', truncation=True, max_length=128, return_tensors='pt')

    # Extract input IDs and attention mask
    input_ids = tokens["input_ids"]
    attention_mask = tokens["attention_mask"]

    # Make predictions
    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits

    # Convert logits to probabilities
    probabilities = torch.softmax(logits, dim=1).squeeze()

    # Map probabilities to emotions
    emotions = {label: prob.item() for label, prob in zip(emotion_labels, probabilities)}

    return emotions

# Example usage
text = "I feel so happy and excited!"
emotions = analyze_emotions(text)
print(emotions)
```

In this updated code, I made a few modifications. First, I extracted the input IDs and attention mask from the encode_plus output using the keys "input_ids" and "attention_mask". Then, I passed these inputs explicitly to the model using model(input_ids, attention_mask=attention_mask).

Please give it a try, and let me know if you encounter any further issues.

[결과]

```
Downloading (...)solve/main/vocab.txt: 100% 77.8k/77.8k [00:00<00:00, 369kB/s]
Downloading pytorch_model.bin: 100% 369M/369M [02:29<00:00, 2.25MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at monologg/kobert and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
{'anger': 0.14082016050815582, 'disgust': 0.13658282160759972, 'fear': 0.186202356822620392, 'joy': 0.05688991850614548, 'neutral': 0.14643071591854095, 'sadness': 0.18198562530021667, 'surprise': 0.11208753287792206}
```

정상적으로 출력

- flow graph 요청

**** 결과가 정상적으로 출력되어 flow graph를 출력할 수 있는 소스코드를 요청**

[질문]

Thanks to you, the error was solved.

In addition, please let me know the code that outputs the flow graph for the result.

[답변]

To visualize the flow graph for the result, you can use the pygraphviz library in Python. This library provides functionality to create and manipulate graphs. You'll need to install the pygraphviz library and its dependencies. You can do this by running the following command:

```
pip install pygraphviz
```

Once you have installed the necessary library, you can use the following modified code to analyze the emotions in a given text using KoBERT and visualize the flow graph:

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification
import pygraphviz as pgv

# Load the KoBERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('monologg/kobert')
model = BertForSequenceClassification.from_pretrained('monologg/kobert', num_labels=7)

# Define the emotion labels
emotion_labels = ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surprise']

# Function to analyze the emotions in a given text
def analyze_emotions(text):
    # Tokenize the text
    tokens = tokenizer.encode_plus(text, padding='max_length', truncation=True, max_length=128, return_tensors='pt')

    # Extract input IDs and attention mask
    input_ids = tokens["input_ids"]
    attention_mask = tokens["attention_mask"]

    # Make predictions
    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits

    # Convert logits to probabilities
    probabilities = torch.softmax(logits, dim=1).squeeze()

    # Map probabilities to emotions
    emotions = {label: prob.item() for label, prob in zip(emotion_labels, probabilities)}
```

```

    return emotions

# Example usage
text = "I feel so happy and excited!"
emotions = analyze_emotions(text)
print(emotions)

# Create and visualize the flow graph
graph = pgv.AGraph(directed=True)

# Add nodes to the graph
graph.add_node("Input", shape="box")
graph.add_node("Tokenizer", shape="box")
graph.add_node("Model", shape="box")
graph.add_node("Softmax", shape="box")
graph.add_node("Output", shape="box")

# Add edges to the graph
graph.add_edge("Input", "Tokenizer")
graph.add_edge("Tokenizer", "Model")
graph.add_edge("Model", "Softmax")
graph.add_edge("Softmax", "Output")

# Set node labels
graph.get_node("Input").attr["label"] = "Input Text"
graph.get_node("Tokenizer").attr["label"] = "Tokenizer"
graph.get_node("Model").attr["label"] = "KoBERT Model"
graph.get_node("Softmax").attr["label"] = "Softmax"
graph.get_node("Output").attr["label"] = "Emotion Predictions"

```

In this modified code, I've added the necessary code to create and visualize the flow graph using the pygraphviz library. After performing the emotion analysis and obtaining the emotions dictionary, the flow graph is created using pygraphviz. Nodes representing different steps, such as "Input," "Tokenizer," "Model," "Softmax," and "Output," are added to the graph, and edges connecting the nodes are added as well. The labels for each node are set accordingly.

The graph layout is determined using the "dot" program, and the resulting flow graph is saved as both a PNG image (flow_graph.png) and an SVG image (flow_graph.svg).

Please make sure you have the pygraphviz library installed and try running the code to visualize the flow graph.

[결과]

```
[11] !pip install pygraphviz
```

```

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pygraphviz
  Downloading pygraphviz-1.10.zip (120 kB)
  120.6/120.6 kB 11.5 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
Building wheels for collected packages: pygraphviz
  error: subprocess-exited-with-error

  × python setup.py bdist_wheel did not run successfully.
  | exit code: 1
  | See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
Building wheel for pygraphviz (setup.py) ... error
ERROR: Failed building wheel for pygraphviz
Running setup.py clean for pygraphviz
Failed to build pygraphviz
ERROR: Could not build wheels for pygraphviz, which is required to install pyproject.toml-based projects

```

**** pygraphviz 라이브러리를 설치하는 과정에서 에러 발생되어 해당 에러에 대한 해결 요청**

[질문]

pip install pygraphviz

There was an error in the code above.

The contents of the error are as follows.

error: subprocess-exited-with-error

× python setup.py bdist_wheel did not run successfully.

| exit code: 1

└> See above for output.

note: This error originates from a subprocess, and is likely not a problem with pip.

Building wheel for pygraphviz (setup.py) ... error

ERROR: Failed building wheel for pygraphviz

Running setup.py clean for pygraphviz

Failed to build pygraphviz

ERROR: Could not build wheels for pygraphviz, which is required to install pyproject.toml-based projects

Please fix the error.

[답변]

I apologize for the inconvenience. It seems that installing pygraphviz using pip might cause issues due to missing dependencies or compatibility problems with certain systems. As an alternative, we can use the graphviz library, which provides a Python interface to Graphviz for creating and rendering graphs. Please follow the updated code below:

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification
import graphviz

# Load the KoBERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('monologg/kobert')
model = BertForSequenceClassification.from_pretrained('monologg/kobert', num_labels=7)

# Define the emotion labels
emotion_labels = ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surprise']

# Function to analyze the emotions in a given text
def analyze_emotions(text):
```

```

# Tokenize the text
tokens = tokenizer.encode_plus(text, padding='max_length', truncation=True, max_length=128, return_tensors='pt')

# Extract input IDs and attention mask
input_ids = tokens['input_ids']
attention_mask = tokens['attention_mask']

# Make predictions
with torch.no_grad():
    outputs = model(input_ids, attention_mask=attention_mask)
    logits = outputs.logits

# Convert logits to probabilities
probabilities = torch.softmax(logits, dim=1).squeeze()

# Map probabilities to emotions
emotions = {label: prob.item() for label, prob in zip(emotion_labels, probabilities)}
|
return emotions

# Example usage
text = "I feel so happy and excited!"
emotions = analyze_emotions(text)
print(emotions)

# Create and visualize the flow graph
graph = graphviz.Digraph(format='png')

# Add nodes to the graph
graphnode("Input", shape="box")
graphnode("Tokenizer", shape="box")
graphnode("Model", shape="box")
graphnode("Softmax", shape="box")
graphnode("Output", shape="box")

# Add edges to the graph
graphedge("Input", "Tokenizer")
graphedge("Tokenizer", "Model")
graphedge("Model", "Softmax")
graphedge("Softmax", "Output")

# Set node labels
graphnode("Input", label="Input Text")
graphnode("Tokenizer", label="Tokenizer")
graphnode("Model", label="KoBERT Model")
graphnode("Softmax", label="Softmax")
graphnode("Output", label="Emotion Predictions")

# Save and display the flow graph
graph.render("flow_graph", view=True)

```

In this updated code, I replaced the pygraphviz library with graphviz to create and render the flow graph. The graphviz library should work without any additional installations or compatibility issues.

Please try running the modified code and let me know if you encounter any further errors.

[결과]

```

✓ [12] import torch
0초 from transformers import BertTokenizer, BertForSequenceClassification
import graphviz

```

graphviz 라이브러리로 변경 후 import 시에 에러가 해결되었고 수정된 소스코드로 run 동작시킴

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at monologg/kobert and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
{'anger': 0.11792969120936258, 'disgust': 0.10929778218269348, 'fear': 0.10193734616041183, 'joy': 0.1825210005044937, 'neutral': 0.140238955616951, 'sadness': 0.21459250152111053, 'surprise': 0.13348370790481567}
'flow_graph.png'

요청한 flow graph가 png 파일로 저장되었음

**** flow graph를 시각화하여 출력해주는 코드를 요청**

[질문]

Please visualize the flow graph.

[답변]

Apologies for the confusion. It appears there was an oversight in the previous response. To visualize the flow graph using the graphviz library, you can modify the code as follows:

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification
import graphviz

# Load the KoBERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('monologg/kobert')
model = BertForSequenceClassification.from_pretrained('monologg/kobert', num_labels=7)

# Define the emotion labels
emotion_labels = ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surprise']

# Function to analyze the emotions in a given text
def analyze_emotions(text):
    # Tokenize the text
    tokens = tokenizer.encode_plus(text, padding='max_length', truncation=True, max_length=128, return_tensors='pt')

    # Extract input IDs and attention mask
    input_ids = tokens["input_ids"]
    attention_mask = tokens["attention_mask"]

    # Make predictions
    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits

    # Convert logits to probabilities
    probabilities = torch.softmax(logits, dim=1).squeeze()

    # Map probabilities to emotions
    emotions = {label: prob.item() for label, prob in zip(emotion_labels, probabilities)}

    return emotions

# Example usage
text = "I feel so happy and excited!"
emotions = analyze_emotions(text)
print(emotions)

# Create and visualize the flow graph
graph = graphviz.Digraph(format='png')

# Add nodes to the graph
graph.node("input", shape="box")
graph.node("Tokenizer", shape="box")
graph.node("Model", shape="box")
graph.node("Softmax", shape="box")
graph.node("Output", shape="box")

# Add edges to the graph
graph.edge("input", "Tokenizer")
graph.edge("Tokenizer", "Model")
graph.edge("Model", "Softmax")
graph.edge("Softmax", "Output")

# Set node labels
graph.node("input", label="Input Text")
graph.node("Tokenizer", label="Tokenizer")
graph.node("Model", label="KoBERT Model")
graph.node("Softmax", label="Softmax")
graph.node("Output", label="Emotion Predictions")

# Save and display the flow graph
graph.render(filename='flow_graph', directory='./', cleanup=True, format='png', view=True)
```

In this updated code, I added the render method to save the flow graph as a PNG file and open it for visualization. The graph will be saved in the current directory with the name flow_graph.png. The cleanup=True argument removes the intermediate DOT file after rendering.

Please give it a try, and the flow graph should be visualized in a separate window. Let me know if you encounter any further issues.

[결과]

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at monologg/kobert and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
{'anger': 0.1805621087551117, 'disgust': 0.18284258246421814, 'fear': 0.14201006293296814, 'joy': 0.065721794962883, 'neutral': 0.09192465990781784, 'sadness': 0.12592074275016785, 'surprise': 0.21101804077625275}
'flow_graph.png'
```

그래프의 시각화를 요청했지만 여전히 이미지 파일로 저장만 할 뿐, 결과값에 그래프를 나타내지 않음

**** 이전에 요청한 코드의 결과값을 그대로 알려준 후 그래프가 시각화되어 보이지 않음을 추가적으로 알리고 그래프의 시각화를 재요청**

[질문]

The result of the code you made is as follows.

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at monologg/kobert and are newly initialized: ['classifier.weight', 'classifier.bias']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

```
{'anger': 0.1805621087551117, 'disgust': 0.18284258246421814, 'fear': 0.14201006293296814, 'joy': 0.065721794962883, 'neutral': 0.09192465990781784, 'sadness': 0.12592074275016785, 'surprise': 0.21101804077625275}
```

flow_graph.png

The flow graph does not appear. Please solve this problem.

[답변]

I apologize for the inconvenience. The warning you received is related to the weights of the model being newly initialized, but it should not affect the flow graph visualization.

As for the flow graph not appearing, it could be due to different factors such as the configuration of your system or any conflicting dependencies. Instead of trying to open the flow graph in a separate window, let's modify the code to directly save the graph to a file without attempting to open it.

Please use the updated code below:

```

import torch
from transformers import BertTokenizer, BertForSequenceClassification
import graphviz

# Load the KoBERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('monologg/kobert')
model = BertForSequenceClassification.from_pretrained('monologg/kobert', num_labels=7)

# Define the emotion labels
emotion_labels = ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surprise']

# Function to analyze the emotions in a given text
def analyze_emotions(text):
    # Tokenize the text
    tokens = tokenizer.encode_plus(text, padding='max_length', truncation=True, max_length=128,
return_tensors='pt')

    # Extract input IDs and attention mask
    input_ids = tokens["input_ids"]
    attention_mask = tokens["attention_mask"]

    # Make predictions
    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits

    # Convert logits to probabilities
    probabilities = torch.softmax(logits, dim=1).squeeze()

    # Map probabilities to emotions
    emotions = {label: prob.item() for label, prob in zip(emotion_labels, probabilities)}

    return emotions

# Example usage
text = "I feel so happy and excited!"
emotions = analyze_emotions(text)
print(emotions)

# Create and visualize the flow graph
graph = graphviz.Digraph(format='png')

# Add nodes to the graph
graph.node("Input", shape="box")
graph.node("Tokenizer", shape="box")
graph.node("Model", shape="box")
graph.node("Softmax", shape="box")
graph.node("Output", shape="box")

# Add edges to the graph
graph.edge("Input", "Tokenizer")
graph.edge("Tokenizer", "Model")
graph.edge("Model", "Softmax")
graph.edge("Softmax", "Output")

# Set node labels
graph.node("Input", label="Input Text")
graph.node("Tokenizer", label="Tokenizer")
graph.node("Model", label="KoBERT Model")
graph.node("Softmax", label="Softmax")
graph.node("Output", label="Emotion Predictions")

# Save the flow graph to a file
graph.render(filename='flow_graph', directory='.', cleanup=True, format='png')

print("Flow graph saved as flow_graph.png")

```

In this modified code, the flow graph is saved to a file directly using `graph.render()`. The file will be saved in the current directory with the name `flow_graph.png`.

Please give it a try, and let me know if the flow graph is saved successfully as `flow_graph.png`.

[결과]

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at monologg/kobert and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
{'anger': 0.11512771248817444, 'disgust': 0.12842847406664166, 'fear': 0.1031372919676688, 'joy': 0.13664271275997162, 'neutral': 0.18580715368257294, 'sadness': 0.13045023381710052, 'surprise': 0.200206456566573}
Flow graph saved as flow_graph.png

결과값까지 알려주며 시각화를 요청했지만 해결이 되지 않았음

**** 요구내용을 그래프의 시각화에서 저장된 이미지 파일의 시각화로 수정하여 요청**

[질문]

I want to print out the saved png file as a result. Let me visualize the image and print it out.

[답변]

Certainly! To visualize and print the saved PNG file, you can use the PIL (Python Imaging Library) library. Make sure you have it installed by running `pip install Pillow`. Then, you can modify the code as follows:

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification
import pydot
from PIL import Image

# Load the KoBERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('monologg/kobert')
model = BertForSequenceClassification.from_pretrained('monologg/kobert', num_labels=7)

# Define the emotion labels
emotion_labels = ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surprise']

# Function to analyze the emotions in a given text
def analyze_emotions(text):
    # Tokenize the text
    tokens = tokenizer.encode_plus(text, padding='max_length', truncation=True, max_length=128, return_tensors='pt')

    # Extract input IDs and attention mask
    input_ids = tokens['input_ids']
    attention_mask = tokens['attention_mask']

    # Make predictions
    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits

    # Convert logits to probabilities
    probabilities = torch.softmax(logits, dim=-1).squeeze()

    # Map probabilities to emotions
    emotions = {label: prob.item() for label, prob in zip(emotion_labels, probabilities)}
```



```

    return emotions

# Example usage
text = "I feel so happy and excited!"
emotions = analyze_emotions(text)
print(emotions)

# Create and visualize the flow graph
graph = pydot.Dot(graph_type='digraph')

# Add nodes to the graph
graph.add_node(pydot.Node("Input", shape="box"))
graph.add_node(pydot.Node("Tokenizer", shape="box"))
graph.add_node(pydot.Node("Model", shape="box"))
graph.add_node(pydot.Node("Softmax", shape="box"))
graph.add_node(pydot.Node("Output", shape="box"))

# Add edges to the graph
graph.add_edge(pydot.Edge("Input", "Tokenizer"))
graph.add_edge(pydot.Edge("Tokenizer", "Model"))
graph.add_edge(pydot.Edge("Model", "Softmax"))
graph.add_edge(pydot.Edge("Softmax", "Output"))

# Set node labels
graph.get_node("Input")[0].set_label("Input Text")
graph.get_node("Tokenizer")[0].set_label("Tokenizer")
graph.get_node("Model")[0].set_label("KoBERT Model")
graph.get_node("Softmax")[0].set_label("Softmax")
graph.get_node("Output")[0].set_label("Emotion Predictions")

# Save the flow graph to a file
graph.write_png("flow_graph.png")
print("Flow graph saved as flow_graph.png")

# Open and display the saved image
image = Image.open("flow_graph.png")
image.show()

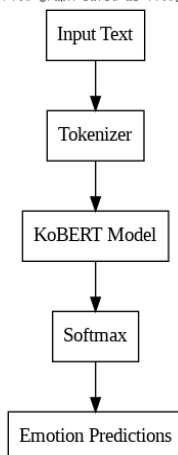
```

In this updated code, I added the PIL library to open and display the saved PNG file using the `Image.open()` and `image.show()` functions, respectively.

Please run the modified code and verify if the flow graph is saved as `flow_graph.png` and displayed successfully. Let me know if you have any further questions or issues!

[결과]

Some weights of BertForSequenceC
 You should probably TRAIN this m
 {'anger': 0.12455475330352783, '
 Flow graph saved as flow_graph.p



저장된 flow graph의 이미지파일의 시각화에 성공함

- pseudo code 요청

**** 해당 알고리즘에 대한 추가적인 답변을 받았으나, 프로그래밍 코드 형식으로 출력되어 프로그래밍 코드가 아닌 논리형식으로 작성된 pseudo code를 재 요청**

[질문]

Tell me the pseudo code for the main steps you told me, not the programming code

[답변]

Certainly! Here's the pseudo code for the main steps of the algorithm:

```
# Pseudo code for Emotion Analysis using KoBERT

# Load the KoBERT model and tokenizer

# Define the emotion labels

# Function to analyze emotions in a given text
function analyze_emotions(text):
    # Tokenize the text

    # Extract input IDs and attention mask

    # Make predictions

    # Convert logits to probabilities

    # Map probabilities to emotions

    # Return the emotions

# Example usage
text = "I feel so happy and excited!"
emotions = analyze_emotions(text)
print(emotions)
```

The above pseudo code outlines the main steps for emotion analysis using KoBERT. Here's a breakdown of each step:

Load the KoBERT model and tokenizer: Load the pre-trained KoBERT model and tokenizer.

Define the emotion labels: Create a list of emotion labels that the model can predict. These labels can include anger, disgust, fear, joy, neutral, sadness, surprise, or any other relevant emotions.

Function to analyze emotions in a given text: Define a function, analyze_emotions, that takes a text

as input and returns the predicted emotions.

Inside the `analyze_emotions` function:

- a. Tokenize the text: Use the tokenizer to convert the input text into tokens that can be understood by the model.
- b. Extract input IDs and attention mask: Obtain the input IDs and attention mask from the tokenized text. These inputs will be used by the model for prediction.
- c. Make predictions: Pass the input IDs and attention mask to the pre-trained KoBERT model to obtain the logits, which represent the model's predictions.
- d. Convert logits to probabilities: Apply a softmax function to the logits to convert them into probabilities, representing the confidence scores for each emotion.
- e. Map probabilities to emotions: Associate each probability with its corresponding emotion label, creating a dictionary or another suitable data structure.
- f. Return the emotions: Return the predicted emotions as a result.

Example usage: Demonstrate the usage of the `analyze_emotions` function by providing a sample text. Store the predicted emotions in a variable (`emotions`) and print the result.

Feel free to translate this pseudo code into any programming language of your choice. Let me know if you have any further questions!

알고리즘에 대한 올바른 pseudo code 및 추가적인 답변을 받음

- test case 요청

**** 추가적으로 해당 프로그램에 대한 여러가지 Test case를 요청**

[질문]

Thanks to you, the problem was solved. Please let me know the test case for additional programming code.

[답변]

Sure! Here's a test case that you can use to evaluate the emotion analysis algorithm:

```
# Test case for emotion analysis using KoBERT

# Define the emotion labels
emotion_labels = ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surprise']

# Test texts
texts = [
    "I am extremely angry right now!",
    "The disgusting smell made me feel sick.",
    "I'm scared of spiders.",
    "This news fills me with joy and excitement!",
    "The weather outside is neither good nor bad.",
    "Feeling really sad about the loss.",
    "Wow, I can't believe it! What a surprise!"
]

# Perform emotion analysis for each text
for text in texts:
    emotions = analyze_emotions(text)

    # Print the text and corresponding emotions
    print("Text:", text)
    for label in emotion_labels:
        print(label + ":", emotions[label])
    print()
```

In this test case, we have a list of texts representing different emotional states. The algorithm will analyze the emotions in each text using the `analyze_emotions` function. The emotions will be printed for each text, showing the probabilities for each emotion label.

Feel free to use this test case or modify it according to your specific needs. Let me know if you have any further questions!

해당 프로그램에 사용 가능한 여러가지 Test case를 받음

Chat GPT

- Chat GPT가 가지고 올 영향

앞으로 Chat GPT와 같은 대화형 인공지능 모델은 효율적인 커뮤니케이션을 제공하여 사용자가 실시간으로 정보를 얻거나 도움을 요청하여 업무, 혹은 일상 생활에서의 효율성을 높일 수 있을 것이다.

또한, 사용자의 선호와 관심사를 학습하여 개인 맞춤형 서비스를 제공할 수도 있다. 이것은 온라인 쇼핑, 엔터테인먼트, 여행 등 다양한 분야에서 사용자들에게 더 나은 경험을 제공할 수 있게 해줄 것이다.

작가, 예술가, 디자이너 등의 창의적인 분야에서도 창작 활동을 지원할 수 있다. 아이디어를 제안하거나 스토리를 개발하는 등의 도움을 주어 창의적인 작업을 더욱 발전시킬 수 있다.

그러나 위와 같은 긍정적인 영향 뿐 아니라 부정적인 영향도 무시할 수는 없다.

발전하는 Chat GPT에 지나치게 의존하게 되면 인간과의 실제 대화나 협력적인 상호작용이 감소하게 될 수 있다. 이로 인해 사회적 고립과 소통의 부족 문제가 발생할 것이다.

또, Chat GPT는 대화 내용을 기반으로 응답을 생성하기 때문에, 민감한 개인정보가 노출될 수도 있을 것이다. 악용될 가능성도 있다.

잘못된 데이터로 교육하게 되면, 인간의 편향이 반영되거나 허위 정보를 전달할 수 있으며, 이것은 사회적 혼란을 야기할 수 있다.

- Chat GPT가 여러 분야에 미치는 영향

첫 번째 Chat GPT는 번역과 자연어 처리 분야에서 놀라운 발전을 이루어 내고 있다. 문장 생성, 시나리오 작성, 기사 작성 등 다양한 창작 콘텐츠 분야에 활용되고 특히 번역 분야에서 놀라운 발전을 이루어 내고 있다. 이러한 발전 덕분에 언어 장벽이 점차 사라지고 국제적인 상호 교류가 원활해지고 있으며 창작자들은 더욱 효율적이고 창의적인 작업을 할 수 있게 되었다.

둘째 Chat GPT는 교육 분야에서도 큰 영향력을 발휘하고 있다. 학생들의 개별적인 학습 스타일, 속도, 선호도에 맞추어 개인화된 학습 자료와 피드백을 제공할 수 있으며 복잡한 개념을 쉽게 설명하거나, 추가적인 학습 자료를 추천해 줄 수도 있다. 특히 Chat GPT는 외국어 학습에 큰 도움을 주고 있다. 학습자와 대화를 통해 발음, 문법, 어휘를 향상시키는데 도움을 주며, 다양한 주제에 대한 대화 연습을 제공할 수 있다. 무엇보다 교육자들은 Chat GPT를 활용하여 학습 자료를 더욱 풍부하고 흥미로운 방식으로 제작할 수 있다. 또한, 교육자들은 학생들의 질문에 대한 답변을 찾거나, 새로운 교육 전략을 고안하는 데 도움을 받을 수 있을 것이다.

셋째 Chat GPT는 데이터 분석 능력을 활용하여 기업이나 정부 조직은 더욱 합리적인 의사 결정을 내릴 수 있고 특히, 빅 데이터 분석을 통해 미래 예측에 도움을 줄 수 있으며 게임 및 엔터테인먼트 산업에서 다양한 캐릭터 개발, 게임 스토리 작성, 사용자와의 인터랙션 등에 활용되며 새로운 경험을 제공할 수 있다. 이로 인해 더욱 풍부하고 혁신적인 엔터테인먼트 콘텐츠가 개발되고 있다.

그밖에 GPT4는 법률 및 금융 분야에서 계약 작성, 법률 자문, 투자 분석 등에 활용되고 있다. 이로 인해 전문가들은 더 빠르고 정확한 서비스를 제공할 수 있게 되었으며 또한 의료 자료 분석, 환자와의 의사소통 등에 활용될 수도 있고 GPT의 도움으로 더욱 정확한 진단을 내려 환자들은 개인화된 치료를 받을 수 있을 것으로 기대된다.

하지만 지금 현재 GPT는 여전히 프로토타입이며 해결해야 할 몇 가지 사항이 있다. GPT는 때때로 질문의 논리와 일치하고 부정확하거나 말도 안 되는 답변을 생성한다. GPT는 외부 정보에 따른 추론 기능을 액세스할 수 없기 때문에 민감한 질문이나 애매한 질문이 주어졌을 때, 같은 의미라도 입력에 따라 전혀 엉뚱한 답변이 나올 수 있다. 하지만 Chat GPT는 현재 다양한 분야에서 큰 영향력을 발휘하고 있으며, 미래에는 더욱 확장될 것으로 예상된다. 이를 통해 인간 사회는 더욱 발전하고, 삶의 질이 향상될 것이다. 그러나 인공지능의 발전에 따른 윤리적, 법적 문제들에 대한 고려와 대응이 필요하며, 인간과 인공지능의 협력을 바탕으로 지속 가능한 발전을 이루어야 할 것으로 판단된다.

참고문헌

<https://hoit1302.tistory.com/159>

https://github.com/SKTBrain/KoBERT/blob/master/scripts/NSMC/naver_review_classifications_pytorch_kobert.ipynb

<https://github.com/SKTBrain/KoBERT/issues>

<https://blog.naver.com/newyeararchive/223097878715>

<http://www.gndomin.com/news/articleView.html?idxno=353288>