D:\myprojects\nlp\bert\bert_imdb.py

```python
# Huggingface transformers를 이용한 Bert 모델 구현 [문장 분류]
# pip install transformers[torch] datasets torch

import torch,sys
from datasets import load_dataset
from transformers import Trainer, TrainingArguments
from transformers import BertForSequenceClassification
from transformers import BertTokenizer
from transformers import pipeline

def tokenize_function(examples):
    return tokenizer(examples['text'], padding="max_length", truncation=True)

# Check for GPU availability
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print("Using device:", device)

#================================================================
def train_bert(totalEpochs=3) :
    # Load dataset
    dataset = load_dataset('imdb', split=['train', 'test'])
    train_dataset, test_dataset = dataset

    # pre-trained tokenizer 설정
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

    # imdb 데이터 세트의 5%만 사용. 전체 사용시엔 이 부분을 comment할 것
    '''
    train_dataset = train_dataset.shuffle(seed=42).select(range(int(0.05 * len(train_dataset))))
    test_dataset = test_dataset.shuffle(seed=42).select(range(int(0.05 * len(test_dataset))))
    '''
    train_dataset = train_dataset.map(tokenize_function, batched=True)
    test_dataset = test_dataset.map(tokenize_function, batched=True)

    # BERT pre-trained model 설정
    model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
    model.to(device)

    # 학습 파라미터 설정
    training_args = TrainingArguments(
        output_dir='./results',          # output directory
        num_train_epochs=totalEpochs,    # number of training epochs
        per_device_train_batch_size=8,   # batch size for training
        per_device_eval_batch_size=16,   # batch size for evaluation
        warmup_steps=500,                # number of warmup steps for learning rate scheduler
        weight_decay=0.01,               # strength of weight decay
        logging_dir='./logs',            # directory for storing logs
        logging_steps=10,
        evaluation_strategy="epoch",     # evaluation is done at the end of each epoch
    )

    # trainer 설정
    trainer = Trainer(
        model=model,
```

```
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=test_dataset
    )

    # 모델 학습 시키기
    trainer.train()

    # 학습결과 저장
    model_path = "bert-finetuned-imdb"  # 결과를 저장할 폴더명
    model.save_pretrained(model_path)
    tokenizer.save_pretrained(model_path)


#================================================================================
# main, train or predict
#================================================================================
showModelFlag = True
if __name__ == "__main__" :
    print('\n','='*80)
    print('Usage [Training]: python bert_imdb.py   train   totalEpochNo[예:10]')
    print('Usage [Prediction]: python bert_imdb.py   predict   ')

    if len(sys.argv) >= 2 :
            #-------------------------------- 학습
        if 'train' in sys.argv[1] and int(sys.argv[2])> 0:
            totalEpochs = int(sys.argv[2])
            train_bert(totalEpochs)

            # Test prediction using a sentence
        elif 'predict' in sys.argv[1] :
            print('\n*** Model loading......')
            modelPath = 'bert-finetuned-imdb'
            model = BertForSequenceClassification.from_pretrained(modelPath)
            model = model.to(device)
            tokenizer = BertTokenizer.from_pretrained(modelPath)
            classifier = pipeline('text-classification', model=model, tokenizer=tokenizer,
device=0 if device == 'cuda' else -1)
            '''
            state_dict = torch.load(modelPath)
            # Update the current model state with the loaded state dictionary
            model.load_state_dict(state_dict)
            '''
            print("*** Model loaded successfully from ", modelPath, '\n')

            if showModelFlag :
                print(model)

            while True:
                query_sentence = input("\n*** Type a sentence or 'quit' to stop: ")
                if 'quit' in query_sentence :
                    break
                print("===> query sentence:", query_sentence)
                predicted_output = classifier(querySentence)
                print("===> Predicted output:", predicted_output, '\n')

```