kobert_train.py

```python
from random import random
import torch
from torch import nn
import gluonnlp as nlp
import numpy as np
from tqdm import tqdm_notebook
import pandas as pd
#kobert
from kobert.utils import get_tokenizer
from kobert.pytorch_kobert import get_pytorch_kobert_model
#transformers
from transformers import AdamW
from transformers.optimization import get_cosine_schedule_with_warmup
from sklearn.model_selection import train_test_split
from bertDataset import BERTDataset
from bertClassifier import BERTClassifier

class Transformers_kobert():
    def __init__(self):
        self.max_len = 100
        self.batch_size = 16
        self.warmup_ratio = 0.1
        self.num_epochs = 10
        self.max_grad_norm = 1
        self.log_interval = 200
        self.learning_rate = 5e-5

        self.device = torch.device("cuda:0")
        #BERT 모델, Vocabulary 불러오기
        self.bertmodel, self.vocab = get_pytorch_kobert_model()
        #BERT 모델 불러오기
        self.model = BERTClassifier(self.bertmodel,  dr_rate=0.5).to(self.device)
        self.dataset_train, self.dataset_test = self.pretrain_model()
        tokenizer = get_tokenizer()
        self.tok = nlp.data.BERTSPTokenizer(tokenizer, self.vocab, lower=False)
        self.optimizer = self.load_bertmodel()
        self.scheduler = self.scheduler()



    def hook(self):
        self.model_train()
        self.save_model()

    def pretrain_model(self):
        chatbot_data = pd.read_excel('C:/MyProject/chatbot/data/한국어_감정_데이터셋.xlsx')

        chatbot_data.loc[(chatbot_data['Emotion'] == "분노"), 'Emotion'] = 0  #분노 => 2
        chatbot_data.loc[(chatbot_data['Emotion'] == "슬픔"), 'Emotion'] = 1  #슬픔 => 3
        chatbot_data.loc[(chatbot_data['Emotion'] == "행복"), 'Emotion'] = 2  #행복 => 5

        data_list = []
        for q, label in zip(chatbot_data['Sentence'], chatbot_data['Emotion'])  :
            data = []
            data.append(q)
            data.append(str(label))

            data_list.append(data)

        return train_test_split(data_list, test_size=0.25, random_state=0)

    def train_tokenizer(self):
        #토큰화
        data_train = BERTDataset(self.dataset_train, 0, 1, self.tok, self.max_len, True, False) # dataset, sent_idx, label_idx, bert_tokenizer, max_len,
        train_dataloader = torch.utils.data.DataLoader(data_train, batch_size=self.batch_size, num_workers=0)
        return train_dataloader

    def test_tokenizer(self):
        #토큰화
        data_test = BERTDataset(self.dataset_test, 0, 1, self.tok, self.max_len, True, False)
        test_dataloader = torch.utils.data.DataLoader(data_test, batch_size=self.batch_size, num_workers=0)
        return test_dataloader


    def load_bertmodel(self):
        model = self.model
        #optimizer와 schedule 설정
        no_decay = ['bias', 'LayerNorm.weight']
        optimizer_grouped_parameters = [
            {'params': [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)], 'weight_decay': 0.01},
            {'params': [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)], 'weight_decay': 0.0}
        ]

        return AdamW(optimizer_grouped_parameters, lr=self.learning_rate)

    def scheduler(self):
        t_total = len(self.train_tokenizer()) * self.num_epochs
        warmup_step = int(t_total * self.warmup_ratio)

        return get_cosine_schedule_with_warmup(self.optimizer, num_warmup_steps=warmup_step, num_training_steps=t_total)


    #정확도 측정을 위한 함수 정의
    def calc_accuracy(self,X,Y):
        max_vals, max_indices = torch.max(X, 1)
        train_acc = (max_indices == Y).sum().data.cpu().numpy()/max_indices.size()[0]
        return train_acc

    def model_train(self):
        device = self.device
        model = self.model
        loss_fn = nn.CrossEntropyLoss()
        optimizer = self.optimizer
        for e in range(self.num_epochs):
            train_acc = 0.0
            test_acc = 0.0
            model.train()
            for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(self.train_tokenizer())):
                optimizer.zero_grad()
                token_ids = token_ids.long().to(device)
                segment_ids = segment_ids.long().to(device)
                valid_length= valid_length
                label = label.long().to(device)
                out = model(token_ids, valid_length, segment_ids)
                loss = loss_fn(out, label)
                loss.backward()
                torch.nn.utils.clip_grad_norm_(model.parameters(), self.max_grad_norm)
                optimizer.step()
                self.scheduler.step()  # Update learning rate schedule
                train_acc += self.calc_accuracy(out, label)
                if batch_id % self.log_interval == 0:
                    print("epoch {} batch id {} loss {} train acc {}".format(e+1, batch_id+1, loss.data.cpu().numpy(), train_acc / (batch_id+1)))
            print("epoch {} train acc {}".format(e+1, train_acc / (batch_id+1)))

            model.eval()
            for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(self.test_tokenizer())):
                token_ids = token_ids.long().to(device)
                segment_ids = segment_ids.long().to(device)
                valid_length= valid_length
                label = label.long().to(device)
                out = model(token_ids, valid_length, segment_ids)
                test_acc += self.calc_accuracy(out, label)
            print("epoch {} test acc {}".format(e+1, test_acc / (batch_id+1)))

    #모델 저장
    def save_model(self):
        model = self.model
        PATH = 'C:/MyProject/chatbot/save/chatbot_v10.pth'
        torch.save(model, PATH)
```