作家风格识别

3210104591 秦雨扬

实验背景

作家风格是作家在作品中表现出来的独特的审美风貌。
通过分析作品的写作风格来识别作者这一研究有很多应用，比如可以帮助人们鉴定某些存在争议的文学作品的作者、判断文章是否剽窃他人作品等。
作者识别其实就是一个文本分类的过程，文本分类就是在给定的分类体系下，根据文本的内容自动地确定文本所关联的类别。
写作风格学就是通过统计的方法来分析作者的写作风格，作者的写作风格是其在语言文字表达活动中的个人言语特征，是人格在语言活动中的某种体现。

实验要求

a）建立深度神经网络模型，对一段文本信息进行检测识别出该文本对应的作者。
b）绘制深度神经网络模型图、绘制并分析学习曲线。
c）用准确率等指标对模型进行评估。

数据集

该数据集包含了 8438 个经典中国文学作品片段，对应文件分别以作家姓名的首字母大写命名。
数据集中的作品片段分别取自 5 位作家的经典作品，分别是：

序号	中文名	英文名	文本片段个数
1	鲁迅	LX	1500 条
2	莫言	MY	2219 条
3	钱钟书	QZS	1419 条
4	王小波	WXB	1300 条
5	张爱玲	ZAL	2000 条

- 其中截取的片段长度在 100~200 个中文字符不等
- 数据集路径为 `dataset/` 以作者名字首字母缩写命名

实验过程

这个实验可能相对来说没有什么波折，因为之前在实验室和学长有做过LLM相关的工作，当时有复现过BertForTokenClassification，使用的是huggingface的一个教程

于是我看到题目自然而然想到了BERT,但是这里是直接的BertForSequenceClassification，然后根据数据集应当选用bert-chinese-base的预训练模型。于是乎就做完了，val精度在98%左右，基本算完成任务了

唯一的小插曲是因为本地2.1版本的权重无法被mo平台支持，只能被迫在线训练，算是熟悉了一下mo平台。

源码

train

```
import os
from sklearn.metrics import accuracy_score
import numpy as np
import torch
import jieba as jb
from torch.utils.data import TensorDataset, DataLoader, random_split
from transformers import BertTokenizer
from transformers import BertForSequenceClassification, AdamW
from transformers import get_linear_schedule_with_warmup
from tqdm import tqdm
model_path = './results/'

tokenizer = BertTokenizer.from_pretrained('tokenizer', do_lower_case=True)


def load_data(path):
    """
    读取数据和标签
    :param path:数据集文件夹路径
    :return:返回读取的片段和对应的标签
    """
    sentences = []  # 片段
    target = []  # 作者

    # 定义lebel到数字的映射关系
    labels = {'LX': 0, 'MY': 1, 'QZS': 2, 'WXB': 3, 'ZAL': 4}

    files = os.listdir(path)
    for file in files:
        if not os.path.isdir(file) and not file[0] == '.':
            with open(os.path.join(path, file), 'r',  encoding='UTF-8') as f:  #
打开文件
                for line in f.readlines():
                    sentences.append(line)
                    target.append(labels[file[:-4]])
    return sentences, target


# Function to get token ids for a list of texts
def encode_fn(text_list):
    all_input_ids = []
    for text in text_list:
        input_ids = tokenizer.encode(
            text,
            add_special_tokens=True,  # 添加special tokens， 也就是CLS和SEP
            max_length=160,  # 设定最大文本长度
            padding='max_length',  # pad到最大的长度
            return_tensors='pt',  # 返回的类型为pytorch tensor
            truncation=True
        )
        all_input_ids.append(input_ids)
    all_input_ids = torch.cat(all_input_ids, dim=0)
```

```
    return all_input_ids


def flat_accuracy(preds, labels):
    """A function for calculating accuracy scores"""

    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return accuracy_score(labels_flat, pred_flat)


if __name__ == '__main__':
    if os.path.exists(model_path):
        print(model_path + " is exist")
    else:
        print(model_path + " is not exist")
        os.mkdir(model_path)
    x_train, y_train = load_data('./dataset')
    # print(x_train[:5])
    all_input_ids = encode_fn(x_train)
    # print(all_input_ids[:5])
    labels = torch.tensor(y_train)
    epochs = 10
    batch_size = 16
    num_labels = 5
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

    # Split data into train and validation
    dataset = TensorDataset(all_input_ids, labels)
    train_size = int(0.90 * len(dataset))
    val_size = len(dataset) - train_size
    train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

    # Create train and validation dataloaders
    train_dataloader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
    val_dataloader = DataLoader(val_dataset, batch_size=batch_size,
shuffle=False)
    # Load the pretrained BERT model
    model = BertForSequenceClassification.from_pretrained('tokenizer',
num_labels=num_labels,

 output_attentions=False,

 output_hidden_states=False)
    model.to(device)
    best_acc = 0
    # create optimizer and learning rate schedule
    # optimizer = AdamW(model.parameters(), lr=2e-5)
    optimizer = AdamW(model.parameters(), lr=2e-5)
    milestones = [1,3,5,7,9]
    # 学习率下降的方式，acc三次不下降就下降学习率继续训练，衰减学习率
    scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones,
gamma=0.5, last_epoch=-1, verbose=False)
    total_steps = len(train_dataloader) * epochs
    # scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,
num_training_steps=total_steps)
```

```
    for epoch in range(epochs):
        # 训练模型
        model.train()
        total_loss, total_val_loss = 0, 0
        total_eval_accuracy = 0
        # bar = tqdm(total=len(train_dataloader))
        for step, batch in tqdm(enumerate(train_dataloader)):
            model.zero_grad()
            outputs = model(batch[0].to(device), token_type_ids=None,
attention_mask=(batch[0] > 0).to(device),
                            labels=batch[1].to(device))
            loss = outputs[0]
            logits = outputs[1]
            total_loss += loss.item()
            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            optimizer.step()
            # bar.update(1)
            # if step % 50 == 0:
            #     print("step: {0} loss: {1}".format(step, loss))
        # 验证模型
        scheduler.step()
        model.eval()
        for i, batch in tqdm(enumerate(val_dataloader)):
            with torch.no_grad():
                outputs = model(batch[0].to(device), token_type_ids=None,
attention_mask=(batch[0] > 0).to(device),
                                labels=batch[1].to(device))
                loss = outputs[0]
                logits = outputs[1]
                total_val_loss += loss.item()
                logits = logits.detach().cpu().numpy()
                label_ids = batch[1].cpu().numpy()
                total_eval_accuracy += flat_accuracy(logits, label_ids)
                # print("eval step: {0} loss: {1}".format(i, loss))
        avg_train_loss = total_loss / len(train_dataloader)
        avg_val_loss = total_val_loss / len(val_dataloader)
        avg_val_accuracy = total_eval_accuracy / len(val_dataloader)
        if(avg_val_accuracy>=best_acc):
            model.save_pretrained(model_path)
            tokenizer.save_pretrained(model_path)
            best_acc=avg_val_accuracy
        print("Saved model")
        print('Train loss     : {0}'.format(avg_train_loss))
        print('Validation loss: {0}'.format(avg_val_loss))
        print('Accuracy: {0}'.format(avg_val_accuracy))
```

test

```
import numpy as np
import torch
from transformers import BertTokenizer
from transformers import BertForSequenceClassification


model_path = './results/'
```

```
def predict(text):
    Tokenizer = BertTokenizer.from_pretrained(model_path)
    model = BertForSequenceClassification.from_pretrained(model_path)
    text_list = []
    labels = []
    text_list.append(text)
    label = 0
    labels.append(label)
    tokenizer = Tokenizer(
        text_list,
        padding=True,
        truncation=True,
        max_length=128,
        return_tensors='pt'  # 返回的类型为 pytorch tensor
    )
    input_ids = tokenizer['input_ids']
    token_type_ids = tokenizer['token_type_ids']
    attention_mask = tokenizer['attention_mask']

    # model = model.cuda()
    model.eval()
    preds = []
    # for i, batch in enumerate(pred_dataloader):
    with torch.no_grad():
        outputs = model(
            input_ids=input_ids,
            token_type_ids=token_type_ids,
            attention_mask=attention_mask
        )

    logits = outputs[0]
    logits = logits.detach().cpu().numpy()
    preds += list(np.argmax(logits, axis=1))
    labels = {0: 'LX', 1: 'MY', 2: 'QZS', 3: 'WXB', 4: 'ZAL'}
    prediction = labels[preds[0]]
    return prediction


if __name__ == '__main__':
    target_text = "中国中流的家庭，教孩子大抵只有两种法。其一是任其跋扈，一点也不管，\
            骂人固可，打人亦无不可，在门内或门前是暴主，是霸王，但到外面便如失了网的蜘蛛一般，\
            立刻毫无能力。其二，是终日给以冷遇或呵斥，甚于打扑，使他畏葸退缩，彷佛一个奴才，\
            一个傀儡，然而父母却美其名曰“听话”，自以为是教育的成功，待到他们外面来，则如暂出樊笼的\
            小禽，他决不会飞鸣，也不会跳跃。"

    print(predict(target_text))
```