新闻文本分类

19351024林威

1. 背景介绍

文本自动分类（text categorization）简称文本分类，是模式识别与自然语言处理密切结合的研究课题。传统的文本分类是基于文本内容的， 研究如何将文本自动划分成政治的、经济的、军事的、体育的、娱乐的等各种类型。这也是人们提到"文本分类"这一术语时通常所指的含义。

本次实践的背景是Datawhale与天池联合发起的0基础入门系列赛事第三场 —— 零基础入门NLP之新闻文本分类挑战赛。赛题以自然语言处理为背景，要求选手根据新闻文本字符对新闻的类别进行分类，这是一个经典文本分类问题。通过这道赛题可以引导大家走入自然语言处理的世界，带大家接触NLP的预处理、模型构建和模型训练等知识点。

> 零基础入门NLP - 新闻文本分类 - 天池大赛 - 阿里云天池

2. 数据集

赛题以新闻数据为赛题数据，数据集报名后可见并可下载。赛题数据为新闻文本，并按照字符级别进行匿名处理。整合划分出14个候选分类类别：财经、彩票、房产、股票、家居、教育、科技、社会、时尚、时政、体育、星座、游戏、娱乐的文本数据。赛题数据由以下几个部分构成：训练集20w条样本，测试集A包括5w条样本，测试集B包括5w条样本。为了预防选手人工标注测试集的情况，比赛数据的文本按照字符级别进行了匿名处理。处理后的赛题训练数据如下：

label	text
6	57 44 66 56 2 3 3 37 5 41 9 57 44 47 45 33 13 63 58 31 17 47 0 1 1 69 26 60 62 15 21 12 49 18 38 20 50 23 57 44 45 33 25 28 47 22 52 35 30 14 24 69 54 7 48 19 11 51 16 43 26 34 53 27 64 8 4 42 36 46 65 69 29 39 15 37 57 44 45 33 69 54 7 25 40 35 30 66 56 47 55 69 61 10 60 42 36 46 65 37 5 41 32 67 6 59 47 0 1 1 68

在数据集中标签的对应的关系如下：

```
{'科技': 0, '股票': 1, '体育': 2, '娱乐': 3, '时政': 4, '社会': 5, '教育': 6, '财经': 7, '家居': 8, '游戏': 9, '房产': 10, '时尚': 11, '彩票': 12, '星座': 13}
```

3. 评价指标

评价标准为类别f1_score的均值，提交结果与实际测试集的类别进行对比，结果越大越好。

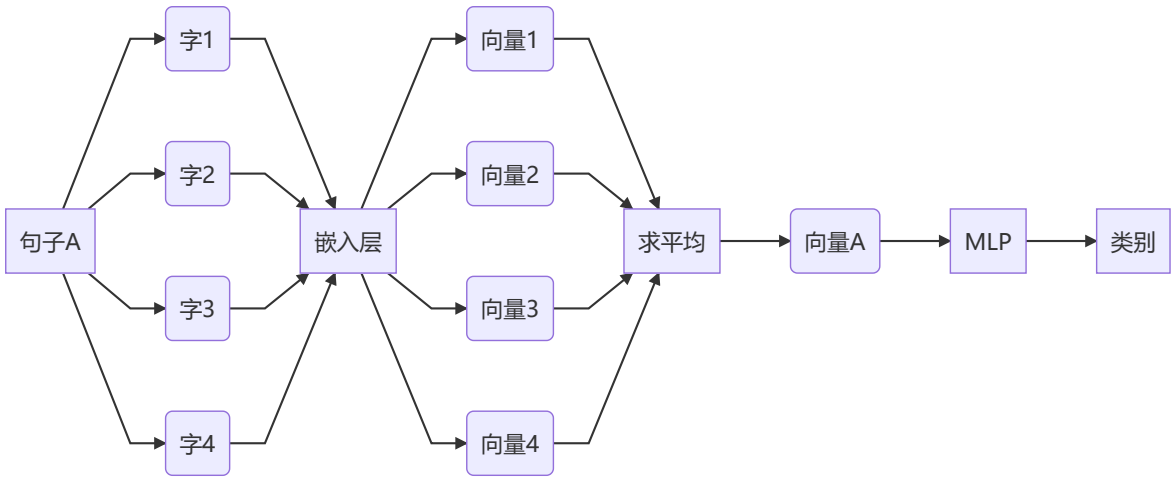
$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

可以通过sklearn来完成f1_score计算：

```python
from sklearn.metrics import f1_score
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
f1_score(y_true, y_pred, average='macro')
```

4. 解题模型

我选择了深度学习的方法来解决这个问题。模型其实非常简单，可以由下图来表示：



模型的输入是一个句子A，它由许多字构成（这里以4个字为例）。实际上，这里的每一个汉字都用了一个整数来表示，这个整数会被作为是嵌入层的输入。嵌入层的作用是把一个整数输入转化为一个指定维数的向量。之后我对这些向量的每个元素分别求平均，得到了向量A。向量A即可被认为是句子A的表示。接着我再用一个多层感知机对向量A进行分类预测，最终得到一个长度为14的向量。该向量中的每个元素即代表对应位置所表示的类别的概率，取概率最大者为模型预测结果。

该模型可以用SGD优化器在训练集上进行训练，损失函数为Cross Entropy。

5. 具体python实现

（一）导入需要用到的Package

我的具体实现是基于Pytorch的，以下是一些可能会用到的库

```python
import torch
from torch.utils.data import Dataset, DataLoader
import pandas as pd
from tqdm import tqdm
from torch import nn
import time
from torch.utils.data.dataset import random_split
from torchtext.data.functional import to_map_style_dataset
from sklearn.metrics import f1_score
```

（二）数据准备

先为训练集和测试集创造两个dataset。

```
class trainDataset(Dataset):
    def __init__(self):
        self.data=pd.read_csv("训练集/train_set.csv/train_set.csv",sep='\t')
    def __len__(self):
        return len(self.data)
    def __getitem__(self,idx):
        item=self.data.iloc[idx,:]
        label=item['label']
        text=item['text'].split(' ')
        text=[int(i) for i in text]
        return label,text

class testDataset(Dataset):
    def __init__(self):
        self.data=pd.read_csv("测试集/test.csv/test_a.csv",sep='\t')
    def __len__(self):
        return len(self.data)
    def __getitem__(self,idx):
        item=self.data.iloc[idx,:]
        text=item['text'].split(' ')
        text=[int(i) for i in text ]
        return text
```

接着将训练集以$0.95:0.05$的比例分为训练集和验证集

```
train_dataset = trainDataset()
test_dataset=testDataset()
num_train = int(len(train_dataset) * 0.95)
split_train_, split_valid_ = \
    random_split(train_dataset, [num_train, len(train_dataset) - num_train])
```

然后把他们都放置入DataLoader当中

```
train_dataloader = DataLoader(split_train_, batch_size=BATCH_SIZE,
                              shuffle=True, collate_fn=collate_batch)
valid_dataloader = DataLoader(split_valid_, batch_size=BATCH_SIZE,
                              shuffle=True, collate_fn=collate_batch)
test_dataloader  = DataLoader(test_dataset,batch_size=BATCH_SIZE,
                              shuffle=False, collate_fn=test_collate_batch)

def collate_batch(batch):
    label_list, text_list, offsets = [], [], [0]
    for (label, text) in batch:
         label_list.append(label)
         processed_text = torch.tensor(text, dtype=torch.int64)
         text_list.append(processed_text)
         offsets.append(processed_text.size(0))
    label_list = torch.tensor(label_list, dtype=torch.int64)
    offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)
    text_list = torch.cat(text_list)
    return label_list.to(device), text_list.to(device), offsets.to(device)

```

```
def test_collate_batch(batch):
    text_list, offsets = [], [0]
    for  text in batch:
         processed_text = torch.tensor(text, dtype=torch.int64)
         text_list.append(processed_text)
         offsets.append(processed_text.size(0))
    offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)
    text_list = torch.cat(text_list)
    return  text_list.to(device), offsets.to(device)
```

（三）定义模型

根据前文描述的思路，定义网络模型。

```
class TextClassificationModel(nn.Module):

    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextClassificationModel, self).__init__()
        self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=True)
        self.fc =  nn.Linear(embed_dim, num_class)
        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        return self.fc(embedded)
```

（四）训练、验证和测试模型

```
def train(dataloader):
    model.train()
    total_acc, total_count = 0, 0
    log_interval = 500
    start_time = time.time()

    for idx, (label, text, offsets) in enumerate(dataloader):
        optimizer.zero_grad()
        predicted_label = model(text, offsets)
        loss = criterion(predicted_label, label)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)
        optimizer.step()
        total_acc += (predicted_label.argmax(1) == label).sum().item()
        total_count += label.size(0)
        if idx % log_interval == 0 and idx > 0:
            elapsed = time.time() - start_time
            print('| epoch {:3d} | {:5d}/{:5d} batches '
                  '| accuracy {:8.3f}'.format(epoch, idx, len(dataloader),
                                              total_acc/total_count))
            total_acc, total_count = 0, 0
            start_time = time.time()
```

```
    torch.save(model,'my_model.pth')
def evaluate(dataloader):
    model.eval()
    total_acc, total_count = 0, 0

    with torch.no_grad():
        for idx, (label, text, offsets) in enumerate(dataloader):
            predicted_label = model(text, offsets)
            loss = criterion(predicted_label, label)
            total_acc += (predicted_label.argmax(1) == label).sum().item()
            total_count += label.size(0)
    return total_acc/total_count
def test(dataloader):
    model.eval()
    res=[]

    with torch.no_grad():
        for idx, (text,offsets) in enumerate(dataloader):
            predicted_label = model(text, offsets)
            result=predicted_label.argmax(1).to('cpu').numpy()
            res.extend(result)
    df=pd.DataFrame()
    df['label']=res
    df.to_csv("test_result.csv")
```

（五）超参设定和主体部分

```
EPOCHS = 5
LR = 5
BATCH_SIZE = 16

criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=LR)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1.0, gamma=0.1)

num_class = 14
vocab_size = 7916
emsize = 64
model = TextClassificationModel(vocab_size, emsize, num_class).to(device)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

for epoch in range(1, EPOCHS + 1):
    epoch_start_time = time.time()
    train(train_dataloader)
    accu_val = evaluate(valid_dataloader)
    if total_accu is not None and total_accu > accu_val:
      scheduler.step()
    else:
        total_accu = accu_val
    print('-' * 59)
    print('| end of epoch {:3d} | time: {:5.2f}s | '
          'valid accuracy {:8.3f} '.format(epoch,time.time() -
epoch_start_time,
                                           accu_val))
    test(test_dataloader)
```

6. 实验结果

经过5个epoch的训练（平均每个epoch需要运行169.85秒），我在验证集上的到的结果为 $Precision = 0.925, f1 = 0.91$

最后我把在测试集上跑的结果上传至比赛官网，结果如下图所示：