华东师范大学计算机科学与技术实验报告

实验课程：数据挖掘	年级：2018	实验成绩：
指导教师：兰曼	姓名：董辰尧	提交作业日期：2021/4/30
实践编号：1	学号：10185102144	实践作业编号：1

一、实验名称：新闻标题采集和分类

新闻网数据的采集，多种不同类型的新闻标题的分类

二、实验目的

掌握本地数据的读写，网络数据的采集，并能自己训练数据预测标题类型

三、实验内容

3.1参考课件L05-WebData.pdf中实例2：新闻网页数据采集，从中国新闻网下载新闻标题和相应类别。

这是一个爬虫，并处理数据的任务。爬虫的主要过程在pdf里面有所展现，我这里需要把爬来的新闻根据原本类型进行编号，并且最后去重。其中爬虫的主要代码如下：

```
url = "http://www.chinanews.com/"
html = requests.get(url) # 获取网页
soup = BeautifulSoup(html.content,"html.parser") # 将该网页转化为 BeautifulSoup 对象
all_news = []
navbar = soup.find(name='ul',attrs={"class":"nav_navcon"}).findAll("a")
for nav in navbar[1:14]: # 取第一栏
    print(nav.string)
    if nav['href'][:5]!="http:":
        url_nav = "http:" + nav['href'] # 获取改板块的url
    print(url_nav)
    html_nav = requests.get(url_nav)
    soup_nav = BeautifulSoup(html_nav.content,"html.parser")

    count = 0 # 控制打印新闻条数
    for n in soup_nav.findAll("li") + soup_nav.findAll("em") + soup_nav.findAll("h1"): # 寻找新闻
        if n.a and n.a.string and len(n.a.string) > 7:
            if nav.string == '时政':
                nav.string = '0'
            if nav.string == '国际':
                nav.string = '1'
            if nav.string == '华人':
                nav.string = '1'
            if nav.string == '社会':
                nav.string = '2'
            if nav.string == '财经':
                nav.string = '3'
            if nav.string == '金融':
                nav.string = '3'
            if nav.string == '产经':
                nav.string = '3'
            if nav.string == '汽车':
                nav.string = '5'
            if nav.string == '港澳':
                nav.string = '4'
            if nav.string == '台湾':
                nav.string = '4'
            if nav.string == '娱乐':
                nav.string = '6'
            if nav.string == '体育':
                nav.string = '7'
            if nav.string == '文化':
                nav.string = '8'
            all_news.append(nav.string+','+n.a.string)
            if count < 3:
                print(n.a.string)
                count += 1
```

去重的主要代码如下：

```
frame=pd.read_csv('all_news.csv',error_bad_lines=False,encoding='utf-8')
data = frame.drop_duplicates(subset=['content'], keep='first', inplace=False)
print(data)
data.to_csv('all_news1.csv', index=0, encoding='utf8')#index = 0 会在写入csv文件的时候去除行索引
```

最后经过大概20次左右（在不同的20天）的爬取，我获得我的最初的训练集（大概有4000行）：

	type	content
0	0	新冠病毒疫苗接种问答来了！涉及34个关键问题
1	0	国家移民管理机构将启用队旗和标志
2	0	美中政策基金会主席：中美知识鸿沟恐将伤害美国外交
3	0	解析新冠重组亚单位蛋白疫苗：精准识别病毒最核心部分
4	0	上海："在沪停留超24小时所有人员需强制登记"系误读
...	...	...
4282	8	《如梦之梦》武汉开启九城巡演大幕 公益首场致敬英雄
4283	8	广西民间艺术家亲身传唱古老山歌 盼传统文化焕新生
4284	8	西藏当代陶瓷文化研究院在拉萨成立
4285	8	湖南南岳祝融火文化园开园 祝融剧场首演
4286	8	书店是一个城市的审美底色

4287 rows × 2 columns

3.2 划分训练集和验证集

这里我采用了9：1的比例进行划分，过程很简单，对于每一条还未划分的新闻标题，产生一个0到1之间的随机数，该数大于0.1就分到训练集，否则分到验证集。主要代码如下：

```
import random

with open('news.csv','r',encoding='utf-8') as f:
    news_list = [e.strip() for e in f.readlines()[1:]]
train = []
test = []

for e in news_list:
    a = random.random()
    if a > 0.1:
        train.append(e)
    else:
        test.append(e)

def data_write(filename,filelist):
    with open(filename,'w',encoding='utf-8') as f:
        for file in filelist:
            f.write(file+'\n')
```

```
data_write('train.txt',train)
data_write('test.txt',test)
```

3.3 使用TextCNN进行训练

3.3.1 TextCNN简介

在写这项作业之前我还没有学到过任何有关NLP方面的知识，在进行网上询问，同学询问之后我决定使用TextCNN进行新闻标题的分类。以下是我对TextCNN的介绍：我们首先把一句话（已经固定好长度，多了删掉，少了补充0）分成一个个的词汇或者字，然后他们映射成数字，作为初始输入。后面经过卷积层（有不同的卷积核，卷积核的高度一般设置成2、3、4、5……），池化层（TextCNN里面一般选择最大池化），池化后我们把得到的结果拼起来最后在全连接层进行类的映射，计算出该句子属于每一类的概率，我们取最大的概率所对应的类就是我们的预测结果。

3.3.2 参数设置

集中设置好参数，这样方便在最后训练的时候调参。主要代码如下：

```
vocab = {}
vocab['[PAD]'] = 0
# 参数设置
device = 'cuda' if torch.cuda.is_available() else 'cpu'
seq_len = 25  # 每句话的长度
embedding_size = 150  # word2vec的维度
kernel_num = 75  # 每一种卷积核的个数
Kernel_list = [2, 3, 4, 5,6]  # N-gram
class_num = 9  # 分类的个数
epoches = 1500  #训练次数
lr = 0.001  # 学习率
```

3.3.3 词向量分配

这一步的意思是我希望新闻里面的每一个出现的字词我都可以有一个数字与之对应，这样方便后面输入的转化，要注意的是由于存在有的新闻字数不足我们设置的每句话的长度的情况，这时需要补0，这就是为什么前面写了vocab['[PAD]'] = 0。主要代码如下：

```
#词向量分配（每一个词都对应一个数字）
def pro_vocab(path):
    with open(path, 'r', encoding='utf-8') as f:
        L = [e.strip() for e in f.readlines()]
        for news in L:
            news = news[2:]
            for word in news:
                if word not in vocab:
                    vocab[word] = len(vocab)

pro_vocab('train.txt')
pro_vocab('test.txt')

vocab_size = len(vocab)
```

3.3.4 将每一句话转化成数字id

```
def word2ids(text):  # 将数据转化成数字id 并多退少补
    ids = [vocab[word] for word in text]
    if len(ids) < seq_len:
        ids += [0] * (seq_len - len(ids))
        return ids
    else:
        return ids[:seq_len]
```

3.3.5 读取训练集测试集

这一步结束后就能返回训练集或测试集所有type以及精活上一步转化成数字id后的列表，主要代码如下：

```
def load_data(path):  # 读取训练集和测试集
    with open(path, 'r', encoding='utf-8') as f:
        turples = [(int(sentence[0]), word2ids(sentence[2:].strip()))
                   for sentence in f.readlines()]
    labels = []
    texts = []
    for turple in turples:
        labels.append(turple[0])
        texts.append(turple[1])
    # 返回标签和经过预处理的文本
    return texts, labels
```

3.3.6 设置本次训练的TextCNN类

__init__对应初始化,forward对应训练过程，主要代码如下：

```
class TextCNN(nn.Module):
    def __init__(self):
        super(TextCNN, self).__init__()
        V = vocab_size
        E = embedding_size
        Ci = 1  # 输入数据的通道数
        Co = kernel_num  # 每一种卷积核的数目
        Kl = Kernel_list  # N-gram
        C = class_num  #输出的维度

        self.embed = nn.Embedding(V, E)
        self.convs = nn.ModuleList([nn.Conv2d(Ci, Co, (K, E)) for K in Kl])
        self.fc = nn.Linear(len(Kl) * Co, C)

    def forward(self, x):
        x = self.embed(x)  # (N, seq_len, E)
        x = x.unsqueeze(1)  # (N, Ci = 1, seq_len, E)
        x = [F.relu(conv(x)).squeeze(3)
             for conv in self.convs]  # [(N, Co, seq_len-ki+1), ...]*len(Ks)
        x = [F.max_pool1d(i, i.size(2)).squeeze(2)
             for i in x]  # [(N, Co), ...]*len(Ks)
        x = torch.cat(x, 1)
        out = F.softmax(self.fc(x), dim=1)
        return out
```

3.3.7 实例化TextCNN类并进行训练+验证

终于到最后一步了，我们只需要根据训练集训练模型，在验证集上跑一遍然后把估计出来的类和真实的类进行比对即可。主要代码如下：

```
model = TextCNN().to(device)  #实例化模型
criterion = nn.CrossEntropyLoss().to(device)  #定义损失函数，为交叉熵损失
optimizer = torch.optim.Adam(model.parameters(), lr=lr)  #使用随机梯度下降算法，学习率为lr

for epoch in trange(epoches):
    pred = model(train_x)
    loss = criterion(pred, train_y)
    if (epoch + 1) % 100 == 0:
        print(loss.item())
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

pred_y = model(test_x)#模型训练好了，这里直接用，计算正确率
L = pred_y.tolist()
total = len(L)
acc = 0
for i, pred in enumerate(L):
    max_index = pred.index(max(pred))
    if max_index == test_y[i]:
        acc += 1

print(acc / total * 100, '%')
```

3.4 对发布的数据进行预测

3.4.1数据预处理

由于我自己整理的数据和发布的数据的样式还是有些不同，所以在这里我不能直接调用上面数据预处理的函数，但是处理数据的基本操作大致相同，主要代码如下：

```
with open('test_4.txt', 'r', encoding='utf-8') as f:
    all_data = f.readlines()
    id_ = []
    texts = []
    for i, line in enumerate(all_data):
        r = line.split()
        id_.append(int(r[0]))
        texts.append(word2ids(r[1]))
#转化为tensor的类型
texts = torch.LongTensor(texts).to(device)
```

3.4.2数据写入

数据预处理完了之后直接用模型跑一下，并且把结果按照要求写入就好了，主要代码如下：

```python
# 我们生成一个新的数据文本，并将所有新闻标题写入新文件
pred = model(texts)
L = pred.tolist()
with open('10185102144-predictions.txt', 'w',encoding='utf-8') as f:
    for i, pred in enumerate(L):
        max_index = pred.index(max(pred))
        f.write(str(id_[i])+'\t'+str(max_index)+'\n')
```

四、实验结果及其分析

4.1训练过程

我导入了tqdm包，方便展示训练结果，训练结果显示是按照进度条来的，没训练100次就打印一次损失，观察损失是否在下降，训练完成后使用模型对验证集进行验证，打印出正确率。下面是截图：

```
 7%|▉      | 100/1500 [00:23<05:24,  4.32it/s]
1.6990526914596558
13%|█▊     | 200/1500 [00:46<05:01,  4.31it/s]
1.6950310468673706
20%|██▋    | 300/1500 [01:09<04:40,  4.27it/s]
1.6937613487243652
27%|███▋   | 400/1500 [01:32<04:17,  4.27it/s]
1.6104191541671753
33%|████▌  | 500/1500 [01:56<03:54,  4.27it/s]
1.609690546989441
40%|█████▍ | 600/1500 [02:19<03:32,  4.24it/s]
1.6094458103179932
47%|██████▎| 700/1500 [02:43<03:08,  4.24it/s]
1.6090658903121948
53%|███████| 800/1500 [03:07<02:45,  4.24it/s]
1.608972191810608
60%|████████| 900
```

看上面结果发现损失是在逐渐下降的

```
]:
```

```python
pred_y = model(test_x)
L = pred_y.tolist()
total = len(L)
acc = 0
for i, pred in enumerate(L):
    max_index = pred.index(max(pred))
    if max_index == test_y[i]:
        acc += 1

print(acc / total * 100, '%')
```

```
63.5 %
```

上图是验证集的正确率

4.2预测结果

预测结果我已经发到了邮箱里

五、问题讨论（实验过程中值得交待的事情）

5.1 调参

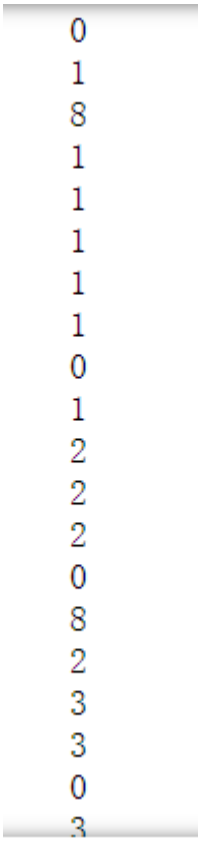
训练模型一般不会一次就能做到完美，所以需要不断地调参。之前没做过TextCNN，调参的过程还是比较繁琐，主要是电脑跑的太慢（配置不太行，后来不得已借助了朋友的台式机的帮助），每一次调参都需要耗费大量的时间，最后正确率在65%左右波动了。

5.2 数据处理

数据处理也是比较头疼的事，自己爬出来的东西，到有分类标签的新闻标题文件，到能进行TextCNN输入的完全转化的矩阵，最后再到预测结果输出，写在另一个txt中。每一步都需要进行数据处理。这个过程还是比较难的，本人经常输出和想要的输出出现差别，于是只能求助于百度等等。最终还是可以圆满解决。

5.3初始数据集

可能是我的初始数据集太小了，而且前5类新闻的占比本身就大，我尝试着打印一下我预测出来的结果，发现竟然几乎都是前5类，后面的4类只有零星几个。估计多爬几天数据，数据量大了可能结果会更好一点



六、结论

这次作业学习到了很多知识，从数据的挖掘，处理，利用，每一步都有更深刻的理解。