

研究生《深度学习》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **大作业：疫情微博情绪分类** |
| **姓 名：** | **秦梓鑫** |
| **学 号：** | **21120390** |
| **上课类型：** | **专业课** |
| **日 期：** | **2021年9月22日** |

## 一、实验内容

**1.1 任务背景**

2019年初，新型疫情来势汹汹，对人们的生产生活产生了巨大的影响，引发舆论广泛关注。在以微博为代表的社交媒体上，疫情相关的话题引起了网友们的广泛讨论。

基于自然语言处理技术，深入挖掘微博文本中蕴含的情感态度信息，可以明确公众态度、感知情绪变化、辅助政府决策、引导网络正能量，具有研究意义和社会价值。

本实验的任务是：使用深度学习方法，对给定的疫情微博数据集进行情感分析，输出微博蕴含的情绪类别。任务的优化目标是：提高在测试集上的评估得分。特别地，训练数据不能脱离数据集范围，不可以引入外部语料、预训练模型。

**1.2评估指标**

本任务的评价指标为：宏精准率(macro-Precision)、宏召回率(macro-Recall) 、宏F1值(macro-F1)。

以上指标均适用于多分类问题。其计算方法如下：

（1）首先统计各个类标的TP、FP、FN、TN；

（2）分别计算各自的精准率(Precision)、召回率(Recall)和F1值；

（3）宏精准率由每个类各自的精准率直接取平均值得到；

（4）宏召回率和宏F1值(macro-F1)的计算方法类似。

在类别不均衡的情况下，使用宏系列指标会更关注小类别的分类效果。

## 二、实验设计

本实验基于跨行业数据挖掘标准流程(CRISP-DM)[8]进行实验环节设计。

**2.1 数据理解**

**1. 词频统计**

给定的训练集、测试集通过jieba库分词后，得到词语共计5049个。词频基本符合长尾分布，除了“武汉”“加油”“我们”“疫情”等词语频繁出现外，多数词语出现次数较少。



图2-1：全部微博文本（训练集+测试集）生成的词云；

词的字体大小对应该词出现的频率

**2. 情绪分布**

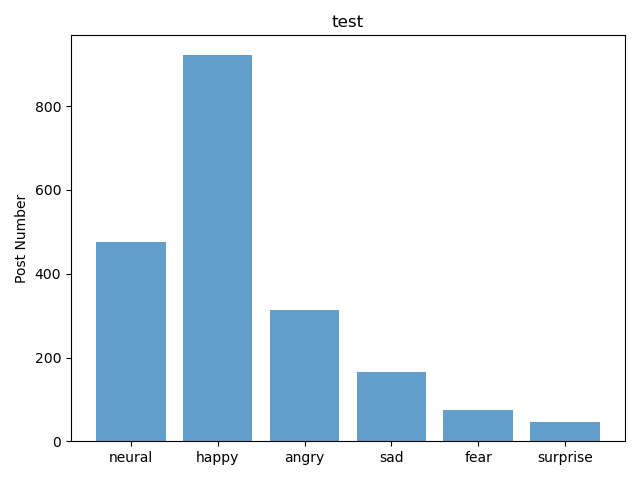
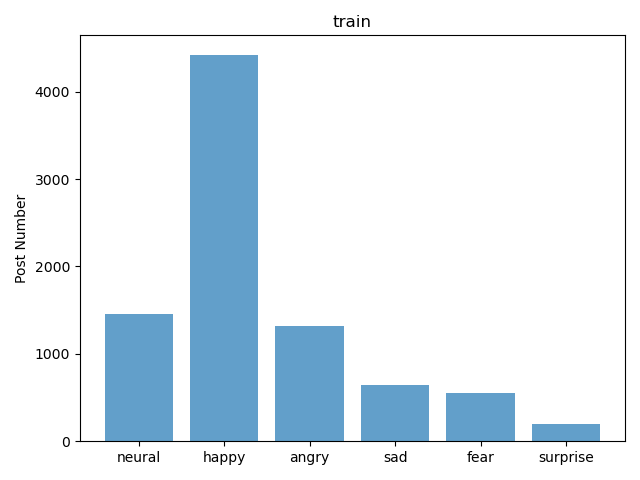


图2-2：文本在训练集和测试集上的情感分布

可以观察到：训练集和测试集的情绪分布基本一致，但也存在样本类不均衡的问题。其中：情绪为积极(happy)的微博文本最多；情绪为悲伤（sad）、恐惧（fear）和惊讶（surprise）的文本较少。

**2.2 数据预处理**

**1. 数据清洗**

在训练过程中，与作者表达情感无关的、过于小众化的文本会造成干扰。此外，文本中还夹杂着一些无语义的特殊符号。

本实验中，我们搜集到以下类型的噪声和干扰，并进行了针对性处理：

|  |  |  |
| --- | --- | --- |
| 类别 | 示例文本 | 处理 |
| @微博用户 | 25//**@皮衣要穿在對的溫度**:宝贝也要注意！ | 删去 |
| 特殊符号、标点符号 | 我参与了@新浪科技 发起的投票【你过年还出去玩么？】，我投给了“不去，在家待着”这个选项，你也快来表态吧~ | 删去 |
| 链接 | 最近很火的鸡蛋面包吃法，[可爱]福建确诊首例新型肺炎病例偶尔来点不一样的早餐厨艺教程 早餐不重样 **t.cn/A6vMKKj4** | 删去 |
| 停用词（没有表达情感信息的虚词） | 哈哈哈哈太可爱**了**！给姥爷**的**防范意识点赞 | 删去 |
| 繁体中文词语 | #**甬**抗肺炎# which disappear after beautiful hover.承诺常常很像蝴蝶，美丽的飞旋然后不见。**固執等著誰 卻驚覺已無法**倒退人生在世 | 翻译为简体中文 |

表2-1：文本中的噪声和干扰

**2. 数据增广**

在前期实验中，我们输出的错误数据(misclassified items)进行了调研，发现集中于小分类样本。因此，我们采用**文本回译**的方式，对悲伤（sad）、恐惧（fear）和惊讶（surprise）三类的样本进行了数据增广，以增加训练数据的表达力。后续实验结果显示，对于小类别数据的小面积增广对指标有一定的提升作用。

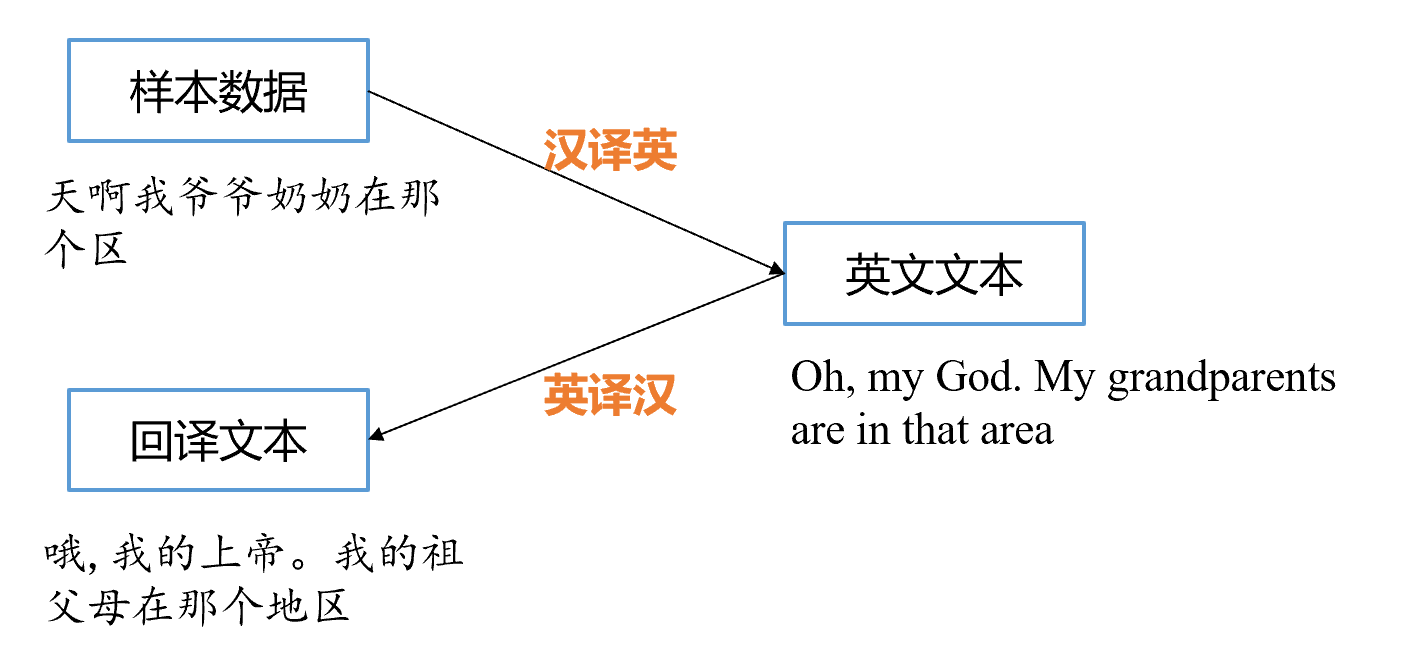


图2-3：文本回译示例

经过数据清洗后的词云分布如图所示。可以感知，筛选后的词集能更明确地反映情绪信息。



图2-4：文本预处理后的词集

**2.3 模型设计**

本次实验中，我们对比了多种模型，最终选择BiLSTM+Attention模型。模型的结构如下图所示。

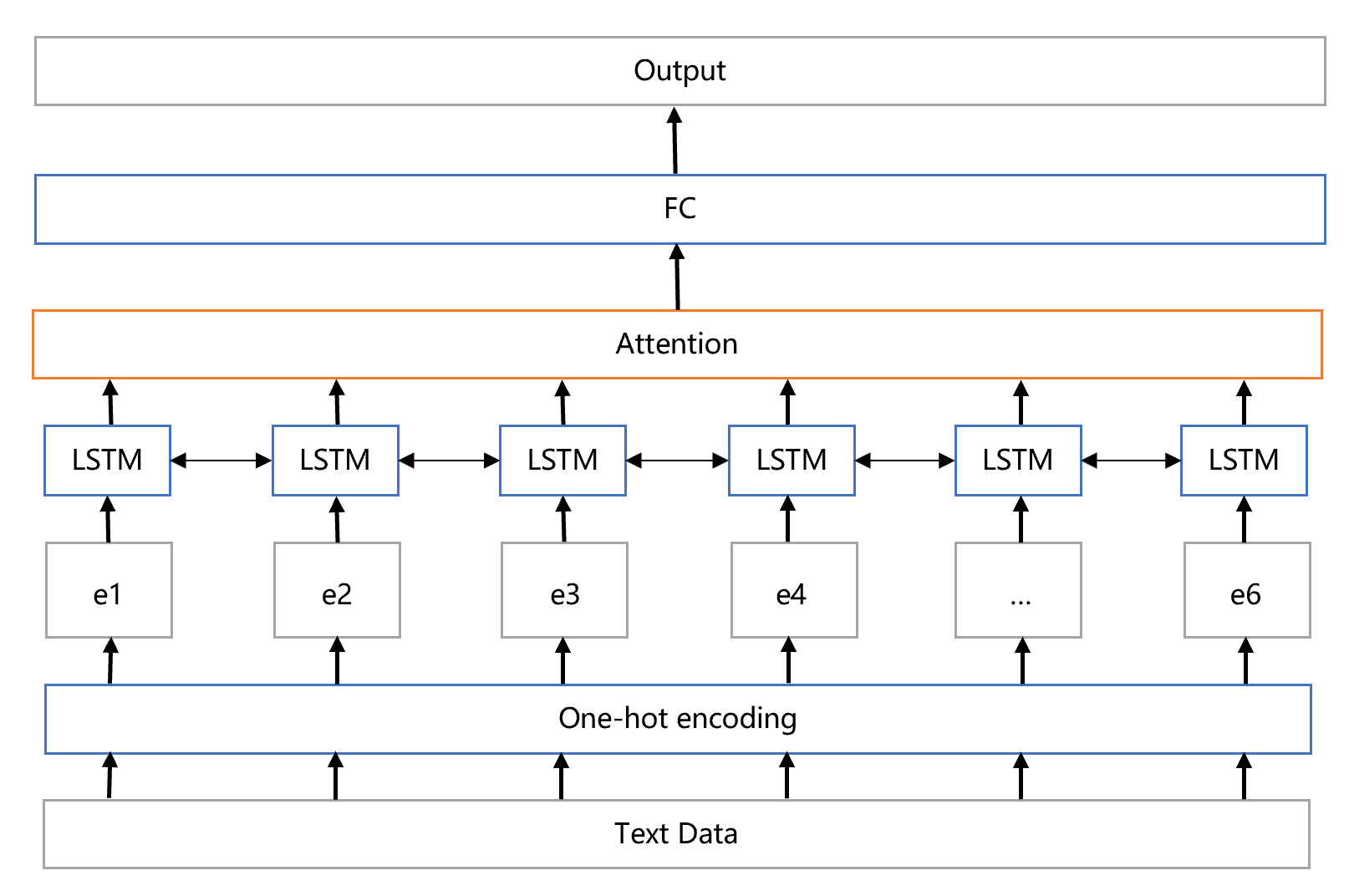


图2-5：模型图示

**（1）文本表示和词嵌入**

首先，文本数据经过清洗、增广和分词之后，以one-hot encoding的方式进行编码。

其中：N是微博文本数量，D是数据集形成的语料库中词的数量。

之后，编码后的张量送去嵌入层(embedding)进行嵌入。

其中：是文本的嵌入长度，是一个超参数；是词嵌入矩阵。本模型中，设置为256。

**（2）双向LSTM**

之后，文本的嵌入表示被依次送入双向LSTM中。

首先介绍LSTM单元；在每一个时刻t，每个LSTM单元完成以下计算：

* 门控单元的计算
* 候选状态、内部状态和输出的更新

其中，是Sigmoid激活函数，是第t个嵌入词向量；都是门控单元；所有的和是模型的参数。在双向LSTM中，我们输入词嵌入，在个LSTM单元中获得输出，最终获得隐藏层表示：

**（3）注意力层**

在注意力层中，每一个隐藏状态被计算和其它状态的注意力分数；随后，注意力分数和隐藏状态最初的值被加权融合，得到文本的最终表示，并送入全连接层中进行六分类的预测。

## 三、实验环境及实验数据集

**1. 实验环境**

实验环境为Linux 3.10.0-1062.el7.x86\_64；运算器为NVIDIA GeForce RTX 2080；框架为：Pytorch 1.6.0；采用Pycharm内置的SSH连接进行交互。

**2. 实验数据集**

数据集以json文件的形式发布。其中，训练集包含8606条中文微博，测试集包含2000条中文微博，每一条微博有唯一的情绪归属。情绪共有六个分类，包括：积极（happy）、愤怒（angry）、悲伤（sad）、恐惧（fear）、惊奇（surprise）和无情绪（neural）。

## 四、实验过程

**1. 数据读取和预处理**

采用python内置的json包进行读取：

1. import json
2. *#读取文件*
3. def resolveJson(path):
4. file = open(path, "rb")
5. file\_json = json.load(file)
6. record = []
7. for i in range(len(file\_json)):
8. fileJson =  file\_json[i]
9. id = fileJson["id"]
10. content = fileJson["content"]
11. label = fileJson["label"]
12. record.append([id,content,label])
13. return record

使用re包进行正则化筛选，实现数据清洗：

1. def clean(json\_text):
2. print("Clean data")
3. *# 加载停用词列表*
4. with open('cn\_stopwords.txt', encoding='utf-8') as f\_stop:
5. stopwords = [line.strip() for line in f\_stop]
6. f\_stop.close()
7. for json\_item in json\_text:
8. text = json\_item[1]
9. import re
10. text = re.sub(r'\/\/\@.\*?(\：|\:)', "",text) *#清除被转发用户用户名*
11. text = re.sub(r'\#.\*?\#',"",text) *# 清除话题*
12. text = re.sub(r'\【.\*?\】', "", text)  *# 清除话题*
13. text = re.sub(r'(https|http)?:\/\/(\w|\.|\/|\?|\=|\&|\%)\*\b', "", text, flags=re.MULTILINE) *# 清除连接*
14. text = re.sub("[\s+\.\!\/\_,$%^\*(+\"\']+|[+——！，。？、~@#￥%……&\*（）]", "", text) *# 去除中文标点符号*
15. *# 去除停用词*
16. outstr=''
17. for word in text.split():
18. if word not in stopwords:
19. if word != '/t':
20. outstr += word
21. json\_item[1] = outstr
22. print("Data is cleaned!")
23. return json\_text

使用有道翻译API进行数据回译，其中：appid和app\_secret需要手动申请。实验过程中也考虑过百度翻译API，但百度翻译API存在请求长度限制，不适用于大规模的文本数据。

1. import jionlp as jio
2. *# 初始化应用*
3. youdao\_free\_api = jio.YoudaoFreeApi()
4. youdao\_api = jio.YoudaoApi(
5. [{'appid': '4f490dca0215d784',
6. 'app\_secret': 'n9Na3ZhZeeub0IOo4BTckpbf9QbFlqie'}])
7. def back\_trans(text):
8. trans\_youdao = youdao\_api(text=text,from\_lang='zh-CHS', to\_lang='en') *#中译英*
9. back\_trans\_yy = youdao\_api(text=trans\_youdao,from\_lang='en', to\_lang='zh-CHS') *#英译中*
10. return back\_trans\_yy

**2. 词嵌入和标签嵌入**

词嵌入的核心步骤如下：

（1）对情感标签进行编码。

1. *#对标签进行编码*
2. def encode\_label(labels,title):
3. encoder = {'neural':0,'happy':1,'angry':2,'sad':3,'fear':4,'surprise':5}
4. labels2 = [encoder[label] for label in labels]

（2）对文本进行分词，整理出词库mywords；根据词库，对文本进行one-hot编码，将文本转化为向量。

1. *# 对文本进行编码*
2. *# 中文分词*
3. import jieba
4. mywords = " ".join(jieba.cut(mytext))
5. *# 将文本token为id列表*
6. def get\_tokenized\_text(text,labels,word\_to\_index):
7. if len(text) == len(labels):
8. vol = len(text)
9. for i in range(vol):
10. temp = []
11. *# 对每一行文本进行分词*
12. import jieba
13. textline = text[i]
14. word\_list = " ".join(jieba.cut(textline))
15. for word in word\_list:
16. if (word in word\_to\_index.keys()):
17. temp.append(int(word\_to\_index[word]))
18. else:
19. temp.append(0)
20. yield [temp, labels[i]]

（4）对长短不一的向量，进行补充(padding)，形成长度相同的向量。补充函数的定义如下：

1. *# 长度不足的文本，用[1]标记进行补足*
2. def padding(x,max\_length):
3. if len(x)>max\_length:
4. text = x[:max\_length]
5. else:
6. text = x + [1] \* (max\_length - len(x))
7. return text

（5）建立文本-情感编码的对应，便于加载到Dataloader

1. *# 文本处理为相同长度的序列*
2. *# 核心模块：文本转向量，向量转固定长度的张量*
3. def process\_text(text,labels,word\_to\_index):
4. data = get\_tokenized\_text(text,labels,word\_to\_index)
5. max\_length = 50
6. labeltensor = torch.IntTensor(labels)
7. samples = []
8. for content in data:
9. text\_to\_sequence = padding(content[0],max\_length)
10. samples.append(text\_to\_sequence)
11. sampletensor = torch.LongTensor(samples) *# Long type will cause error in training,but is essential in embedding*
12. return sampletensor,labeltensor

**3. 模型构建**

本实验所用的包含注意力机制的Bi-LSTM模型，代码实现如下：

1. class BiLSTM\_Attention(nn.Module):
2. def \_\_init\_\_(self, total\_word\_count, hidden\_size, num\_layers):
3. super(BiLSTM\_Attention, self).\_\_init\_\_()
4. self.word\_embeddings = nn.Embedding(total\_word\_count, 1024)
5. self.encoder = nn.LSTM(input\_size=1024,hidden\_size=hidden\_size, num\_layers=num\_layers,batch\_first=True,bidirectional=True)
6. *# 初始时间步和最终时间步的隐藏状态作为全连接层输入*
7. self.w\_omega = nn.Parameter(torch.Tensor(
8. hidden\_size \* 2, hidden\_size \* 2))
9. self.u\_omega = nn.Parameter(torch.Tensor(hidden\_size \* 2, 1)) *#对hiddensize进行降维，以便得到每个h的注意力权重*
10. self.decoder = nn.Linear(2 \* hidden\_size, 6)
11. nn.init.uniform\_(self.w\_omega, -0.1, 0.1)
12. nn.init.uniform\_(self.u\_omega, -0.1, 0.1)
13. def forward(self, inputs):
14. embeddings = self.word\_embeddings(inputs.to(device)).to(device)
15. outputs, \_ = self.encoder(embeddings) *# output, (h, c)*
16. x = outputs.to(device)
17. *# Attention过程*
18. u = torch.tanh(torch.matmul(x, self.w\_omega)).to(device)
19. att = torch.matmul(u, self.u\_omega).to(device)
20. import torch.nn.functional as F
21. att\_score = F.softmax(att, dim=1).to(device)
22. scored\_x = x \* att\_score.to(device)
23. feat = torch.sum(scored\_x, dim=1).to(device)  *# 加权求和*
24. outs = self.decoder(feat)
25. return outs.to(device)

**4. 基准模型(baselines)**

实验中使用了RNN，GRU，LSTM，Transformer四类模型作为对比。受篇幅所限，具体代码不再罗列。

**5. 训练过程**

训练过程如下：

1. def train(net,loss,optimizer,train\_iter,test\_iter,index2sentence):
2. net = net.to(device)
3. num\_epochs = 20
4. score\_log = []
5. for epoch in range(num\_epochs):
6. for x, y in train\_iter:
7. *#print(x.shape,y.shape)*
8. yhat = net(x)
9. yhat = yhat.view(len(yhat), -1)
10. l = loss(yhat, y.long().squeeze().to(device))
11. optimizer.zero\_grad()
12. l.backward()
13. optimizer.step()
14. loss\_test = calculate(net, test\_iter, loss)
15. acc\_train,prec\_tr, recall\_tr, f1\_tr,report\_tr,yhat\_list,label\_list  = evaluate(net, train\_iter)
16. acc\_test,prec\_te,recall\_te,f1\_te,report\_te,yhat\_list\_te,label\_list\_te  = evaluate(net, test\_iter,output=True)
17. print("epoch",epoch,"\*test\*  ","f1:",f1\_te,"loss:", loss\_test, "precision:",prec\_te,"recall:",recall\_te,"acc(hand):",acc\_test)
18. print("train f1:",f1\_tr)

**6. 评估过程**

在评估阶段，我们采用sklearn.metrics中提供的混淆矩阵、宏F1值等计算函数，对比模型输出和标签，评估模型性能。

1. from sklearn.metrics import precision\_recall\_fscore\_support,classification\_report
2. p\_class, r\_class, f\_class, support\_micro = precision\_recall\_fscore\_support(labels\_list,yhat\_list,average='macro')
3. print('Marco Precision:', p\_class)
4. print('Marco Recall:', r\_class)
5. print('Marco F1：', f\_class)
6. print('Confusion Matrix:\n', classification\_report(labels\_list, yhat\_list, labels=[0, 1, 2, 3, 4, 5]))
7. report = classification\_report(labels\_list, yhat\_list, labels=[0, 1, 2, 3, 4, 5])
8. return acc,p\_class,r\_class,f\_class,report,yhat\_list,labels\_list

## 五、实验结果

**1. 模型评估**

本实验的结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 模型 | 宏-F1值 | 宏-精确率 | 宏-召回率 |
| RNN | 0.3358 | 0.3415 | 0.3389 |
| GRU | 0.3949 | 0.4086 | 0.3921 |
| LSTM | 0.4166 | 0.4558 | 0.4160 |
| Transformer | 0.4215 | 0.4470 | 0.4379 |
| **Bi-LSTM-attention** | **0.4904** | **0.5089** | **0.4827** |

表5-1：模型性能对比

其中，Bi-LSTM组的最优结果截图如下：

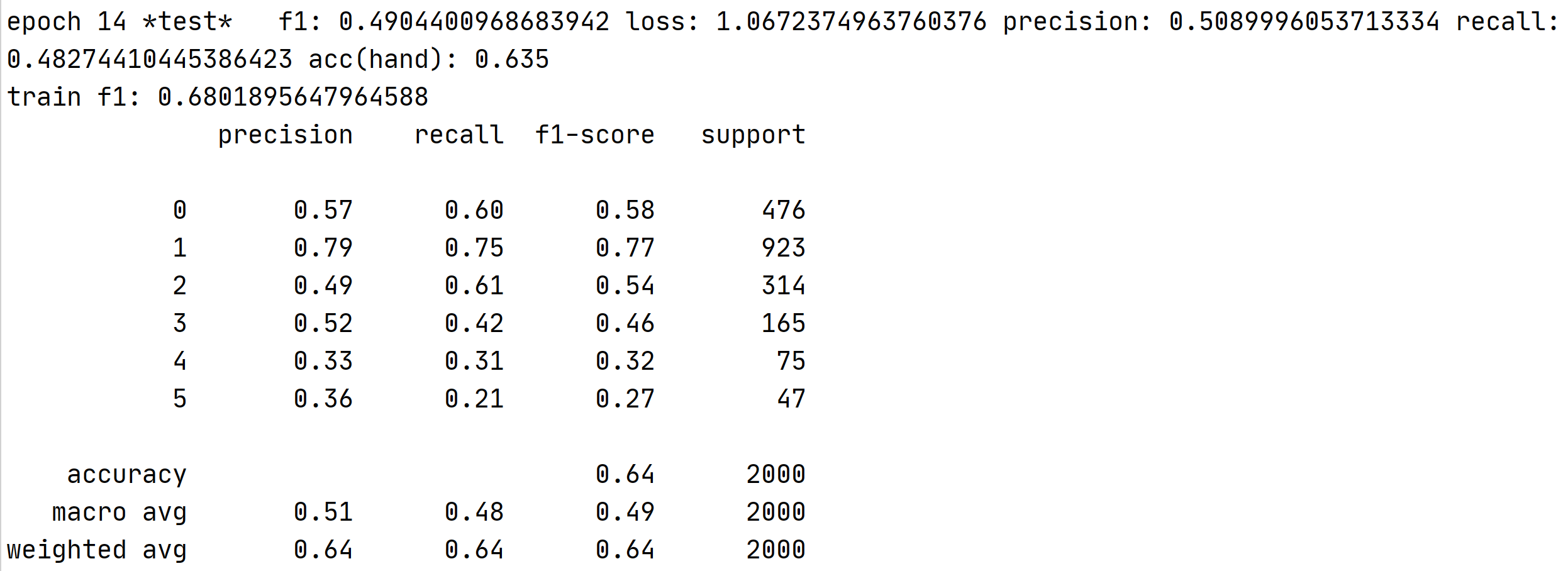


图5-1：Bi-LSTM-attention模型的效果

训练集和测试集的宏F1值变化如下：

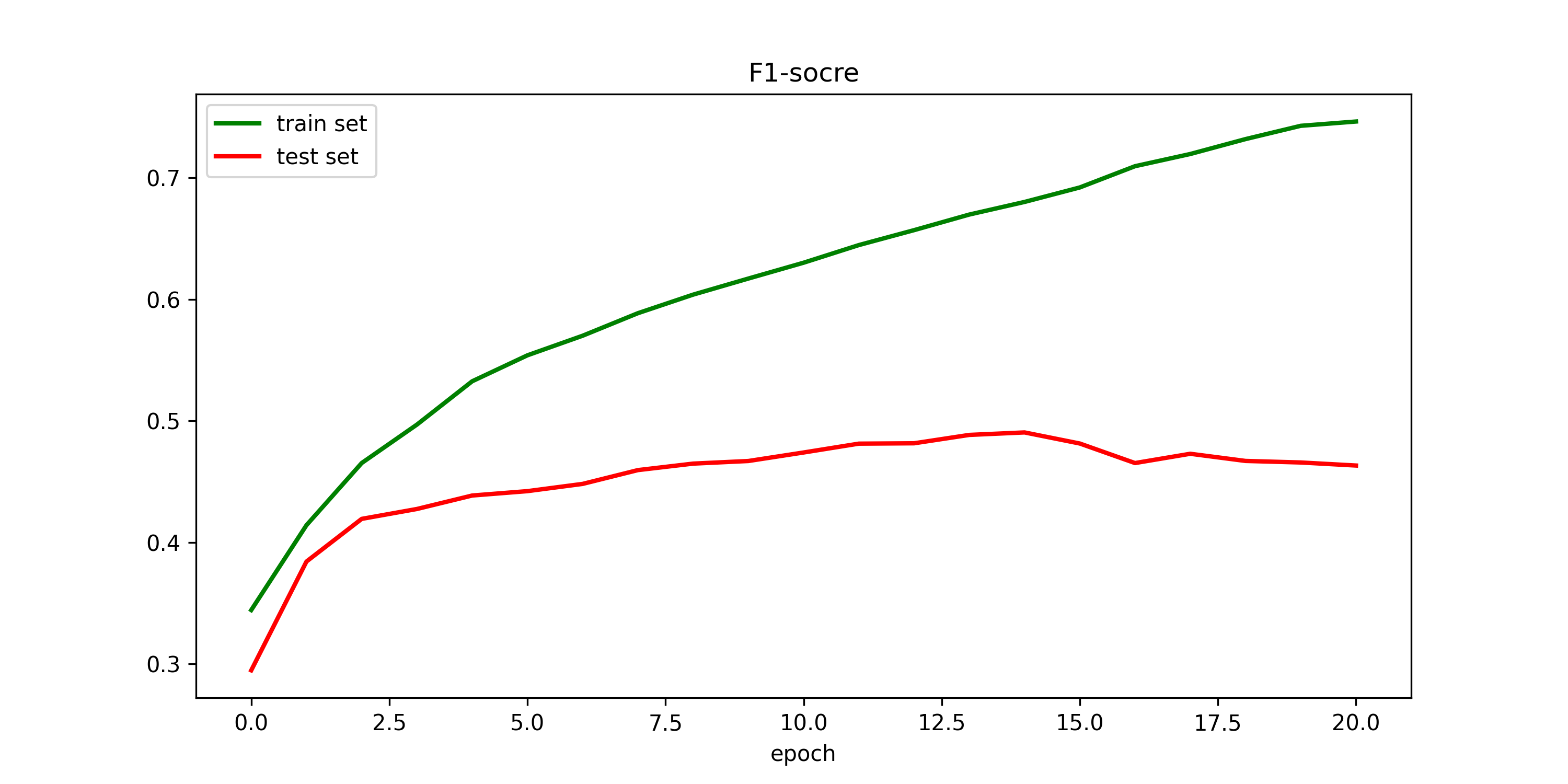


图5-1：Bi-LSTM-attention模型在训练集和测试集上的F1-值变化

训练过程中的超参数如下：

|  |  |
| --- | --- |
| 初始学习率 | 1e-4 |
| 正则化(weight\_decay) | 0.8\*1e-4 |
| 学习率衰减步长 | 10 |
| 学习率衰减率 | 0.5 |
| batch size | 32 |

表5-2：超参数列表

**2. 结论和反思**

实验结果表明：在给定的文本分类任务下，我们构建的Bi-LSTM-attention模型在测试集上达到了0.4904的宏F1值，优于RNN、GRU、LSTM和Transfomer模型。

实验过程中，我们发现了以下问题和值得思考的点：

**（1）过拟合现象严重。**训练集和测试集在情感上的分布大体相似，但是在词汇上具有较大的差异。对此，在不使用任何预训练模型的前提下，我们尽可能地引入了多种策略以增加模型的泛化性能，包括：添加Dropout层、正则化、提前终止训练、学习率衰减等。

**（2）样本不均衡问题。**在调试过程中，我们每次将分类错误(mis-classified)的样本进行输出，发现错误集中于小类别的样本数据。因此，我们尝试了数据增广策略；此外，实验表明，适当减少batchsize有助于对此类数据的学习。

**（3）小数据集的预处理问题。**相比于工业界用于预训练任务的数据集，给定的数据集规模较小。因此，预处理过程一方面会让数据变得更为精细；另一方面，过多的处理会造成信息流失，使得模型可学习的信息量减少，最终“无物可学”。因此，需考虑预处理的适度问题。

## 六、实验心得体会

在本科时，数据挖掘课的文本分类作业是通过WEKA软件“调包”实现的，当时便对文本挖掘产生了巨大的兴趣，但没有机会动手“造轮子”，也不明白梯度下降等算法的原理，不理解为什么代码可以work。

在暑期学期中，通过智能计算数学基础、深度学习两门课程，我初步建立了较为完善的知识体系。在完成了四次实验以及最终的大作业之后，我终于对曾经感到非常困惑的深度学习领域产生了较为清晰的认知。

在大作业的选题时，我决定再次选择文本分类的题目。在完成作业的过程中，我直观地感受到了自身的成长：从对公式的不解到手动推导公式、从调包完成作业到动手实现、从对待不同类型数据的恐慌到能够熟练运用词嵌入、序列采样等方法进行处理……之前道听途说的各种技巧，成为亲手写下的代码时，会觉得心里少了许多浮躁，多了一分踏实。

同时，数据科学精彩、丰富的一面逐渐映入眼帘。疫情数据的生活性、数据处理的艺术性、优化方法的严密性、偏差和误差的偶然性，交融着理性的光芒和感性的色彩，使我对这门学科产生了更深层的理解和敬意。

非常幸运，能在这个暑假修读万老师主讲的《深度学习》课程。希望在以后的学习生活中，也能够一直做到理论结合实践，做到知行合一，止于至善。

## 七、参考文献

[1] 清博大数据-SMP2020微博情绪分类评测汇报

[2] 炬火-SMP2020-微博情绪分类评测汇报

[3] 文本数据增强——回译：<https://github.com/dongrixinyu/JioNLP>

[4] Transformer原理以及文本分类实战：<https://blog.csdn.net/qq_36618444/article/details/106472126>

[5] Attention 扫盲：注意力机制及其 PyTorch 应用实现：

<https://blog.csdn.net/fengdu78/article/details/103849711>

[6] pytorch中RNN参数的详细解释: <https://blog.csdn.net/lwgkzl/article/details/88717678>

[7] NLP:基于jieba和gensim的疫情微博情绪分类：

<https://blog.csdn.net/sunny_1219/article/details/110239752>

[8] Chris Clifto. Cross-Industry Standard Process for Data Mining