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

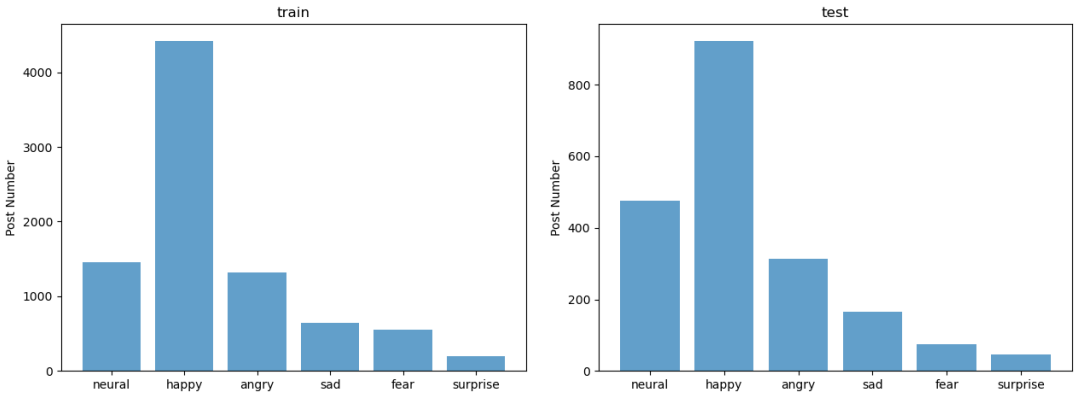

图 2-2：文本在训练集和测试集上的情感分布

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。

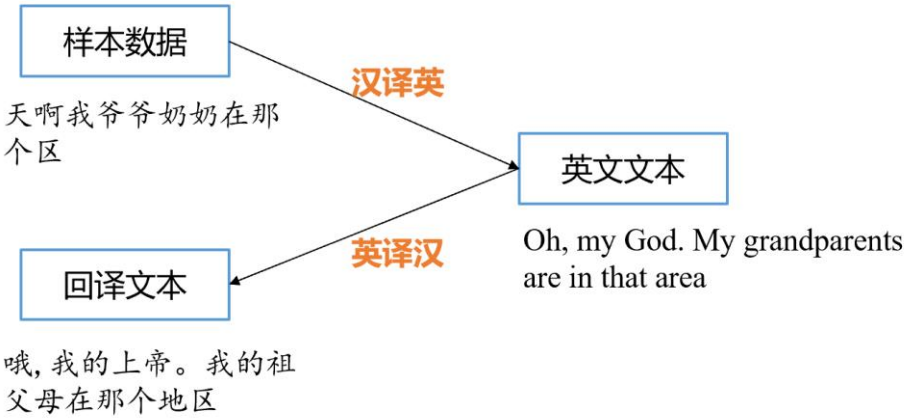

图 2-3：文本回译示例

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。

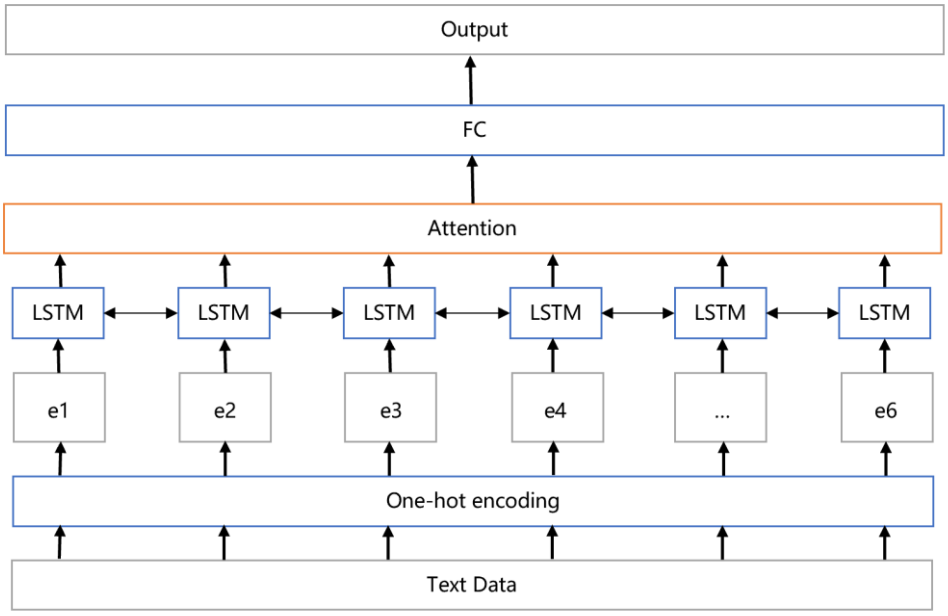

图 2-5：模型图示

（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。

$$X = (x_1, x_2, \dots, x_n), X \in R^{N \times D} \tag{1}$$

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。

$$E = M_{embed}X = (e_1, e_2, \dots, e_N), E \in R^{N \times d} \tag{2}$$

其中：d是文本的嵌入长度，是一个超参数；M_{embed}是词嵌入矩阵。本模型中，d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

$$f_t = \sigma\left(W_f[h_{t-1}, e_t] + b_f\right) \tag{3}$$

$$i_t = \sigma(W_i[h_{t\text{-}1}, e_t] + b_i) \tag{4}$$

$$o_t = \sigma(W_o[h_{t-1}, e_t] + b_o) \tag{5}$$

- 候选状态、内部状态和输出的更新

$$\tilde{c_t} = tanh(W_c e_t + U_c h_{t-1} + b_c) \tag{6}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t} \tag{7}$$

$$h_t = o_t \odot tanh(\mathrm{c_t}) \tag{8}$$

其中，σ是 Sigmoid 激活函数，e_t是第 t 个嵌入词向量；c_t, f_t, i_t, o_t都是门控单元；所有的W和b是模型的参数。在双向 LSTM 中，我们输入词嵌入$E = (e_1, e_2, \dots, e_N)$，在$2L$个 LSTM 单元中获得输出$h = (h_1, h_2, \dots, h_{2L})$，最终获得隐藏层表示：

$$h_i = \overrightarrow{\mathrm{h_i}} \oplus \overleftarrow{h_i} \in R^{2L} \tag{9}$$

（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。

$$a_i = tanh(W_h h_i + b_h) \in [-1,1] \tag{10}$$

$$w_i = \frac{exp(a_i)}{\sum_{t=1}^{N} exp(a_t)}, \sum_{i=1}^{N} w_i = 1 \tag{11}$$

$$r = \sum_{i=1}^{N} w_i h_i, r \in R^{2L} \tag{12}$$

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：

```
import json

#读取文件
def resolveJson(path):
    file = open(path, "rb")
    file_json = json.load(file)
    record = []
    for i in range(len(file_json)):
        fileJson =  file_json[i]
        id = fileJson["id"]
        content = fileJson["content"]
        label = fileJson["label"]
        record.append([id,content,label])
    return record
```

使用 re 包进行正则化筛选，实现数据清洗：

```
def clean(json_text):
    print("Clean data")
    # 加载停用词列表
    with open('cn_stopwords.txt', encoding='utf-8') as f_stop:
        stopwords = [line.strip() for line in f_stop]
        f_stop.close()

    for json_item in json_text:
        text = json_item[1]
        import re
        text = re.sub(r'\/\/\@.*?(\: |\:)', "",text) #清除被转发用户用户名
        text = re.sub(r'\#.*?\#',"",text) # 清除话题
        text = re.sub(r'\【.*?\】', "", text)  # 清除话题
        text = re.sub(r'(https|http)?:\/\/(\w|\.|\/|\?|\=|\&|\%)*\b', "", tex
t, flags=re.MULTILINE) # 清除连接
        text = re.sub("[\s+\.\!\/_,$%^*(+\"\']+|[+——！，。？、~@#￥%……&*
（）]", "", text) # 去除中文标点符号
        # 去除停用词
        outstr=''
        for word in text.split():
```

```
                if word not in stopwords:
                    if word != '/t':
                        outstr += word
            json_item[1] = outstr
        print("Data is cleaned!")
        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。

```
import jionlp as jio

# 初始化应用
youdao_free_api = jio.YoudaoFreeApi()
youdao_api = jio.YoudaoApi(
        [{'appid': '4f490dca0215d784',
          'app_secret': 'n9Na3ZhZeeub0IOo4BTckpbf9QbFlqie'}])

def back_trans(text):
    trans_youdao = youdao_api(text=text,from_lang='zh-CHS', to_lang='en') #中译英
    back_trans_yy = youdao_api(text=trans_youdao,from_lang='en', to_lang='zh-CHS') #英译中
    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。

```
#对标签进行编码
def encode_label(labels,title):
        encoder = {'neural':0,'happy':1,'angry':2,'sad':3,'fear':4,'surprise':5}
        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。

```
# 对文本进行编码
# 中文分词
import jieba
```

```
    mywords = " ".join(jieba.cut(mytext))
    # 将文本token为id列表
    def get_tokenized_text(text,labels,word_to_index):
        if len(text) == len(labels):
            vol = len(text)
            for i in range(vol):
                temp = []
                # 对每一行文本进行分词
                import jieba
                textline = text[i]
                word_list = " ".join(jieba.cut(textline))
                for word in word_list:
                    if (word in word_to_index.keys()):
                        temp.append(int(word_to_index[word]))
                    else:
                        temp.append(0)
                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：

```
    # 长度不足的文本，用[1]标记进行补足
    def padding(x,max_length):
        if len(x)>max_length:
            text = x[:max_length]
        else:
            text = x + [1] * (max_length - len(x))
        return text
```

（5）建立文本-情感编码的对应，便于加载到 Dataloader

```
    # 文本处理为相同长度的序列
    # 核心模块：文本转向量，向量转固定长度的张量
    def process_text(text,labels,word_to_index):
        data = get_tokenized_text(text,labels,word_to_index)
        max_length = 50
        labeltensor = torch.IntTensor(labels)
        samples = []
        for content in data:
            text_to_sequence = padding(content[0],max_length)
            samples.append(text_to_sequence)
        sampletensor = torch.LongTensor(samples) # Long type will cause error in training,
    but is essential in embedding
```

```
        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：

```
class BiLSTM_Attention(nn.Module):
    def __init__(self, total_word_count, hidden_size, num_layers):
        super(BiLSTM_Attention, self).__init__()
        self.word_embeddings = nn.Embedding(total_word_count, 1024)
        self.encoder = nn.LSTM(input_size=1024,hidden_size=hidden_size, num_layers=num_layers,batch_first=True,bidirectional=True)
        # 初始时间步和最终时间步的隐藏状态作为全连接层输入
        self.w_omega = nn.Parameter(torch.Tensor(
            hidden_size * 2, hidden_size * 2))
        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
        self.decoder = nn.Linear(2 * hidden_size, 6)
        nn.init.uniform_(self.w_omega, -0.1, 0.1)
        nn.init.uniform_(self.u_omega, -0.1, 0.1)

    def forward(self, inputs):
        embeddings = self.word_embeddings(inputs.to(device)).to(device)
        outputs, _ = self.encoder(embeddings) # output, (h, c)
        x = outputs.to(device)
        # Attention 过程
        u = torch.tanh(torch.matmul(x, self.w_omega)).to(device)
        att = torch.matmul(u, self.u_omega).to(device)
        import torch.nn.functional as F
        att_score = F.softmax(att, dim=1).to(device)
        scored_x = x * att_score.to(device)
        feat = torch.sum(scored_x, dim=1).to(device)  # 加权求和
        outs = self.decoder(feat)
        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：

```
def train(net,loss,optimizer,train_iter,test_iter,index2sentence):
```

```
        net = net.to(device)
        num_epochs = 20
        score_log = []
        for epoch in range(num_epochs):
            for x, y in train_iter:
                #print(x.shape,y.shape)
                yhat = net(x)
                yhat = yhat.view(len(yhat), -1)
                l = loss(yhat, y.long().squeeze().to(device))
                optimizer.zero_grad()
                l.backward()
                optimizer.step()
            loss_test = calculate(net, test_iter, loss)
            acc_train,prec_tr, recall_tr, f1_tr,report_tr,yhat_list,label_list  = evaluate(net, train_iter)
            acc_test,prec_te,recall_te,f1_te,report_te,yhat_list_te,label_list_te  = evaluate(net, test_iter,output=True)
            print("epoch",epoch,"*test*  ","f1:",f1_te,"loss:", loss_test, "precision:",prec_te,"recall:",recall_te,"acc(hand):",acc_test)
            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。

```
from sklearn.metrics import precision_recall_fscore_support,classification_report
p_class, r_class, f_class, support_micro = precision_recall_fscore_support(labels_list,yhat_list,average='macro')
print('Marco Precision:', p_class)
print('Marco Recall:', r_class)
print('Marco F1: ', f_class)
print('Confusion Matrix:\n', classification_report(labels_list, yhat_list, labels=[0, 1, 2, 3, 4, 5]))
report = classification_report(labels_list, yhat_list, labels=[0, 1, 2, 3, 4, 5])
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：

```
epoch 14 *test*   f1: 0.4904400968683942 loss: 1.0672374963760376 precision: 0.5089996053713334 recall:
0.48274410445386423 acc(hand): 0.635
train f1: 0.6801895647964588
              precision    recall  f1-score   support

           0       0.57      0.60      0.58       476
           1       0.79      0.75      0.77       923
           2       0.49      0.61      0.54       314
           3       0.52      0.42      0.46       165
           4       0.33      0.31      0.32        75
           5       0.36      0.21      0.27        47

    accuracy                           0.64      2000
   macro avg       0.51      0.48      0.49      2000
weighted avg       0.64      0.64      0.64      2000
```

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：

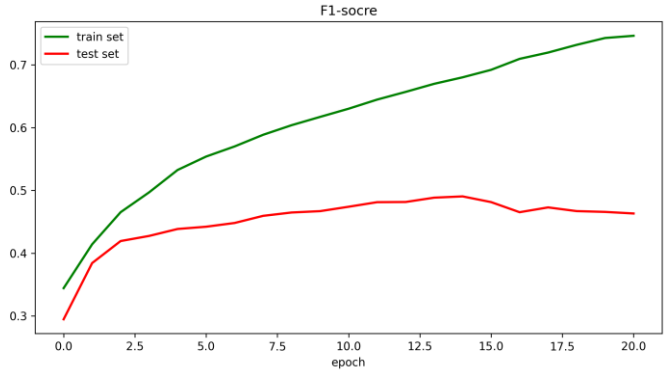

图 5-1：Bi-LSTM-attention 模型在训练集和测试集上的 F1-值变化

训练过程中的超参数如下：

初始学习率	1e-4
L_2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、L_2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