



研究生《深度学习》课程 实验报告

实验名称: 疫情微博情绪分类实验

姓 名: 韩浩杰

学 号: 20125164

上课类型: 专业课

日 期: 2020.9.16

一、实验内容

情感分析技术一直是自然语言处理领域研究的重点内容之一。2020 年，新冠肺炎疫情成为了全国人民关注的焦点，众多用户针对此次疫情在新浪微博等社交媒体平台上发表自己的看法，蕴含了非常丰富的情感信息。

基于自然语言处理技术自动识别社交媒体文本中的情绪信息，可以帮助政府了解网民对各个事件的态度，及时发现人民的情绪波动，从而更有针对性地制定政策方针，具有重要的社会价值。尽管之前的社交媒体情感分析技术已经取得了不错的进展，但是如何将之前的研究成果快速高效地应用到疫情相关的数据当中，仍然是一个值得研究的问题。

在本任务当中，要求专门针对疫情相关微博进行情绪分析，来判断微博所表达的情感。微博情绪分类任务旨在识别微博中蕴含的情绪，输入是一条微博，输出是该微博所蕴含的情绪类别。在本次任务中，我们将微博按照其蕴含的情绪分为以下六个类别之一：积极、愤怒、悲伤、恐惧、惊奇和无情绪。要求同学们以提高在测试集上的效果为目标，自己根据数据特点及需要进行数据预处理以及模型设计。

本任务不对模型的选择和设计进行限制，同时要求训练所用数据不脱离给定的数据集范围，不可引入外部语料（例如通过外部语料得到的预训练词向量等），可以以 CNN、RNN 等模型为基础进行设计。

实验报告中需要包含但不限于数据的处理过程介绍（例如分词等）、词向量、模型图、模型中各部分的作用介绍或者使用理由、超参数设置以及训练过程中各指标变化和实验结果分析等。

二、实验环境及实验数据集

2.1 实验环境

1. 操作系统：Windows 10 家庭中文版
2. CPU：AMD Ryzen9 4900H
3. GPU：Nvidia Geforce RTX2060 6GB
4. 内存：16GB
5. 软件：VS Code + Jupyter Notebook + Python 3.6.010

2.2 数据集

本数据集（疫情微博数据集）内的微博内容是在疫情期间使用相关关键字筛选获得的疫情微博，其内容与新冠疫情相关。每条微博被标注为以下六个类别之一：neural（无情绪）、happy（积极）、angry（愤怒）、sad（悲伤）、fear（恐惧）、surprise（惊奇）。疫情微博训练数据集包括 6,606 条微博，测试数据集包含 5,000 条微博。数据集为 json 格式，包含三个字段：数据编号（id），文本（content），情绪标签（label）。示例：{"id": 11, "content": "武汉加油！中国加油！安徽加油！", "label": "happy"}。

三、实验设计

3.1 词向量训练

由于实验要求不引入外部预料，因此需要从给定数据集自己训练词向量，本实验使用 gensim 库提供的 word2vec 对处理过的语句进行训练得到词向量。

3.2 模型设计

本实验采用 Kim 2014 经典的 CNN 文本分类模型 TextCNN，其结构包括嵌入层 Embedding layer、卷积层 CONV layer、激励层 ReLU layer、池化层 Pooling layer、全连接层 FC layer、丢弃层 Dropout layer。

其中嵌入层与 CNN 的输入层作用相同。卷积层中由于每个词向量只对应一个词，滑动长度切割后没有意义，因此卷积核维度只有变长，分别为 2、3、4，而宽度固定为词向量的维度。激励层使用 ReLU 激活，池化层使用最大池化，再使用丢弃层来缓解过拟合，最后由全连接层得到分类特征。模型图如下：

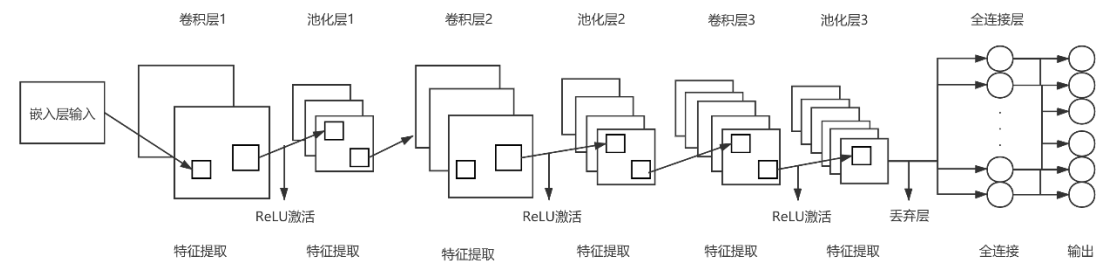
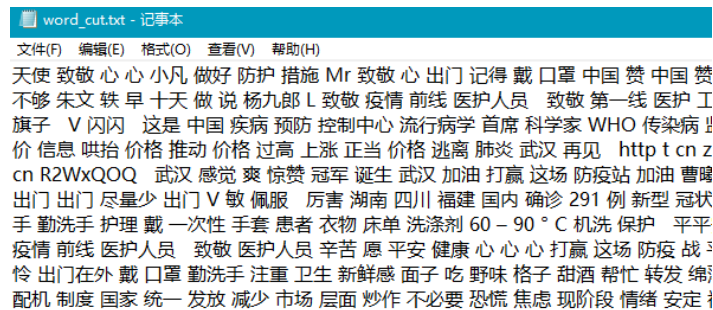


图 1 TextCNN 模型图

四、实验过程

4.1 词向量训练

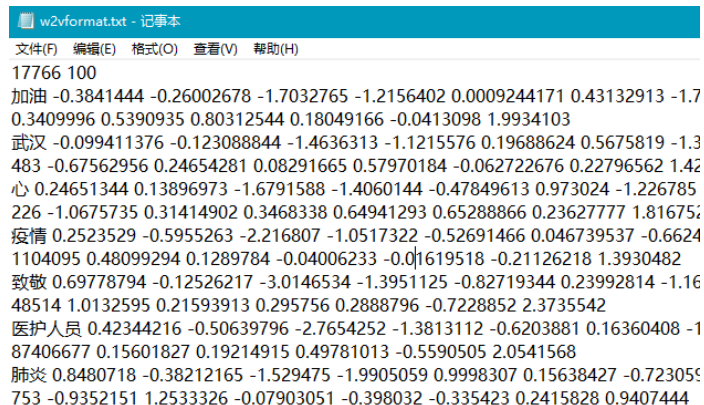
1. 使用 conda 安装 gensim 包，引入 Word2Vec。
2. 使用 json.load 读取给定数据集，存入 dict，分为 train，valid，test 三份，其中数据数量分别为 6606，2000，2000。使用 valid 中的数据对模型效果进行验证并调整参数，test 仅作最终评估使用。
3. 使用 jieba 库提供的分词功能对 train 中句子进行分词、去除停用词并存入 txt 文件以方便后续使用。得到分词文件部分如下：



```
word_cut.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
天使 致敬 心 心 小凡 做好 防护 措施 Mr 致敬 心 出门 记得 戴 口罩 中国 赞 中国 赞
不够 朱文 铁 早 十天 做 说 杨九郎 L 致敬 疫情 前线 医护人员 致敬 第一线 医护 工
旗子 V 闪闪 这是 中国 疾病 预防 控制中心 流行病学 首席 科学家 WHO 传染病 !
价 信息 哄抬 价格 推动 价格 过高 上涨 正当 价格 逃离 肺炎 武汉 再见 http t cn z
cn R2WxQOQ 武汉 感觉 爽 惊喜 冠军 诞生 武汉 加油 打赢 这场 防疫站 加油 曹
出门 出门 尽量少 出门 V 敏 佩服 厉害 湖南 四川 福建 国内 确诊 291 例 新型 冠
状 手 勤洗手 护理 戴 一次性 手套 患者 衣物 床单 洗涤剂 60 - 90 ° C 机洗 保护 平
平 疫情 前线 医护人员 致敬 医护人员 辛苦 愿 平安 健康 心 心 心 打赢 这场 防疫 战
! 怜 出门 在外 戴 口罩 勤洗手 注重 卫生 新鲜感 面子 吃 野味 格子 甜酒 帮忙 转发 绑
配 机 制度 国家 统一 发放 减少 市场 层面 炒作 不必要 恐慌 焦虑 现阶段 情绪 安定 !
```

图 2 分词结果

4. 使用 LineSentence 读取分词文件并调用 Word2Vec 模型对词语进行训练得到词向量。由于数据集中词汇量较小，词向量维度 size 取默认值 100，词向量上下文最大距离 window 取 3，最小词频 min count 取 1。使用 CBOW 连续词袋模型，并采取层序 softmax 采样法进行训练。得到的词向量部分如下：



```
w2vformat.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
17766 100
加油 -0.3841444 -0.26002678 -1.7032765 -1.2156402 0.0009244171 0.43132913 -1.703409996 0.5390935 0.80312544 0.18049166 -0.0413098 1.9934103
武汉 -0.099411376 -0.123088844 -1.4636313 -1.1215576 0.19688624 0.5675819 -1.3483 -0.67562956 0.24654281 0.08291665 0.57970184 -0.062722676 0.22796562 1.42
心 0.24651344 0.13896973 -1.6791588 -1.4060144 -0.47849613 0.973024 -1.226785
226 -1.0675735 0.31414902 0.3468338 0.64941293 0.65288866 0.23627777 1.81675
疫情 0.2523529 -0.5955263 -2.216807 -1.0517322 -0.52691466 0.046739537 -0.6624
1104095 0.48099294 0.1289784 -0.04006233 -0.01619518 -0.21126218 1.3930482
致敬 0.69778794 -0.12526217 -3.0146534 -1.3951125 -0.82719344 0.23992814 -1.1648514 1.0132595 0.21593913 0.295756 0.2888796 -0.7228852 2.3735542
医护人员 0.42344216 -0.50639796 -2.7654252 -1.3813112 -0.6203881 0.16360408 -1.87406677 0.15601827 0.19214915 0.49781013 -0.5590505 2.0541568
肺炎 0.8480718 -0.38212165 -1.529475 -1.9905059 0.9998307 0.15638427 -0.72305
753 -0.9352151 1.2533326 -0.07903051 -0.398032 -0.335423 0.2415828 0.9407444
```

图 3 词向量

4.2 数据预处理

1. 首先观察数据分布与构成，将词语长度反映在直方图中可以看出大部分微博文本长度集中在 50 以内。

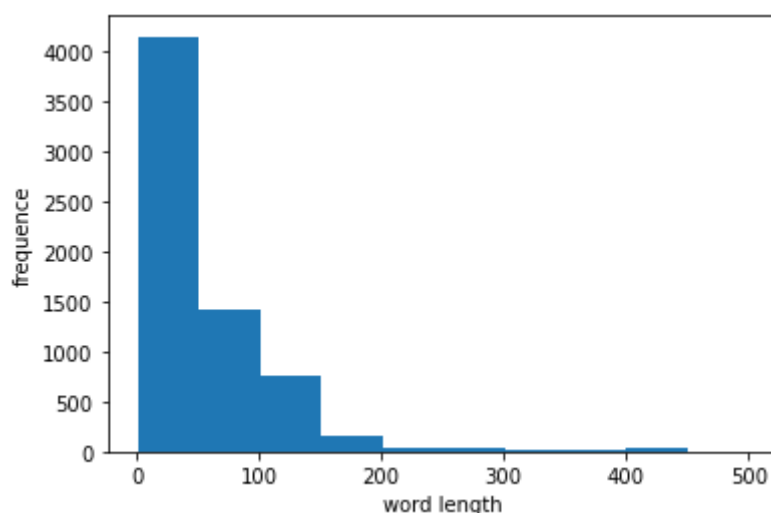


图 4 文本长度分布

2. 使用 `torchtext` 构造文本与标签的 `Field`，分别为 `TEXT` 与 `LABEL`，其中 `TEXT` 域需要对语句长度进行统一修正，根据图 2 将文本长度固定为 50。
3. 对三组数据进行分词处理，并将标签转换为对应的 0-5 整数，根据 `torchtext` 的要求构造文本 `example` 与 `field`，拼接后形成数据集。

```
1. def get_dataset(load_data, text_field, label_field):
2.     fields = [('text', text_field), ('label', label_field)]
3.     examples = []
4.
5.     for data in load_data:
6.         content = words_after_jieba(data['content'])
7.         label = trans_labels(data['label'])
8.         examples.append(Data.Example.fromlist([content, label], fields))
9.
10.    return examples, fields
```

4. 使用 `torchtext` 提供的 `build_vocab` 函数，将 `word2vec` 训练得到的词向量应用于训练数据集映射，从而建立词表，最后将数据装载入数据迭代器。

```
1. TEXT.build_vocab(train_set, vectors=Vectors('w2vformat.txt'))
2. LABEL.build_vocab(train_set)
```

4.3 实现模型

根据模型设计进行实现，其中卷积层使用了 `nn.ModuleList` 进行迭代产生，更为方便，三层卷积的卷积核长分别为 2、3、4，宽为词向量大小 100，最大池化将特征压缩至 1，经过三层卷积后进行拼接处理，然后使用丢弃率为 0.5 的丢弃层缓解过拟合，最终通过全连接层将预测结果分为 6 类。

```
1. class TextCNN(nn.Module):
2.     def __init__(self, vocab_size, embedding_dim, output_size,
3.                   filter_num=100, kernel_list=(2,3,4), dropout=0.5):
4.         super(TextCNN, self).__init__()
5.         self.embedding = nn.Embedding(vocab_size, embedding_dim)
6.         self.convs = nn.ModuleList([nn.Sequential(
7.             nn.Conv2d(1, filter_num, (kernel, embedding_dim)),
8.             nn.ReLU(), nn.MaxPool2d((sentence_len-kernel+1, 1))
9.         ) for kernel in kernel_list])
10.        self.fc = nn.Linear(filter_num * len(kernel_list), output_size)
11.        self.dropout = nn.Dropout(dropout)
12.    def forward(self, x):
13.        x = self.embedding(x) # (batch_size,word_num,embedding_dim)
14.        x = x.unsqueeze(1) # (batch_size,channel_num,word_num,embedding_dim)
15.        out = [conv(x) for conv in self.convs]
16.        out = torch.cat(out, dim=1)
17.        out = out.view(x.size(0), -1)
18.        out = self.dropout(out)
19.        out = self.fc(out)
20.        return out
```

模型具体参数如下图所示：

```
TextCNN(
  (embedding): Embedding(17770, 100)
  (convs): ModuleList(
    (0): Sequential(
      (0): Conv2d(1, 100, kernel_size=(2, 100), stride=(1, 1))
      (1): ReLU()
      (2): MaxPool2d(kernel_size=(49, 1), stride=(49, 1), padding=0, dilation=1, ceil_mode=False)
    )
    (1): Sequential(
      (0): Conv2d(1, 100, kernel_size=(3, 100), stride=(1, 1))
      (1): ReLU()
      (2): MaxPool2d(kernel_size=(48, 1), stride=(48, 1), padding=0, dilation=1, ceil_mode=False)
    )
    (2): Sequential(
      (0): Conv2d(1, 100, kernel_size=(4, 100), stride=(1, 1))
      (1): ReLU()
      (2): MaxPool2d(kernel_size=(47, 1), stride=(47, 1), padding=0, dilation=1, ceil_mode=False)
    )
  )
  (fc): Linear(in_features=300, out_features=6, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
```

图 5 模型参数

4.4 训练模型

1. 分别实现 `train` 函数与 `test` 函数，`train` 中只对 `loss` 值进行观察，`test` 中观察 `loss` 值与三项评价指标 `macro p`、`macro r`、`macro f1`，不进行梯度变化。
2. 三项评价指标直接使用 `sklearn.metrics` 提供的函数进行计算。

```
1. macro_p += precision_score(true_label, pred_label, average='macro')
2. macro_r += recall_score(true_label, pred_label, average='macro')
3. macro_f1 += f1_score(true_label, pred_label, average='macro')
```

3. 使用自己训练过的嵌入词向量来替换随机初始化。

```
1. pretrained_embedding = TEXT.vocab.vectors
2. net.embedding.weight.data.copy_(pretrained_embedding)
3. net.embedding.weight.data[PAD_IDX] = torch.zeros(embedding_size)
4. net.embedding.weight.data[UNK_IDX] = torch.zeros(embedding_size)
```

4. `batch size` 设为 64，学习率经过多次调节后在 0.0001 时表现较好，训练轮数为 50 轮，开始训练。

五、实验结果

实验过程中使用 valid 验证集对超参数进行调整，最终调整结果为 valid loss: 0.8398, valid p: 0.5339, valid r: 0.5066, valid f1: 0.4963 这里不做详细展示。

1. 训练集与测试集 loss 值对比

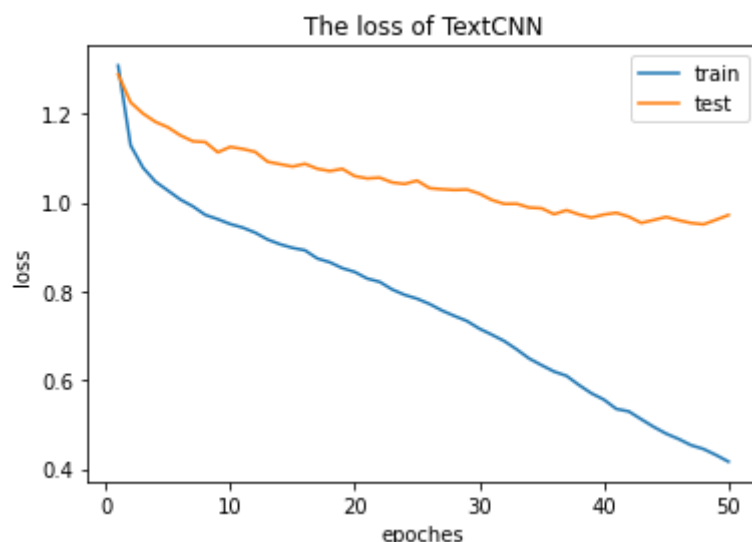


图 6 loss 曲线

最终训练集 loss 值接近，测试集 loss 值接近，与其在验证集上结果接近，虽然从曲线上看出存在一定的过拟合现象，但模型整体拟合能力较好。

2. 评价指标结果

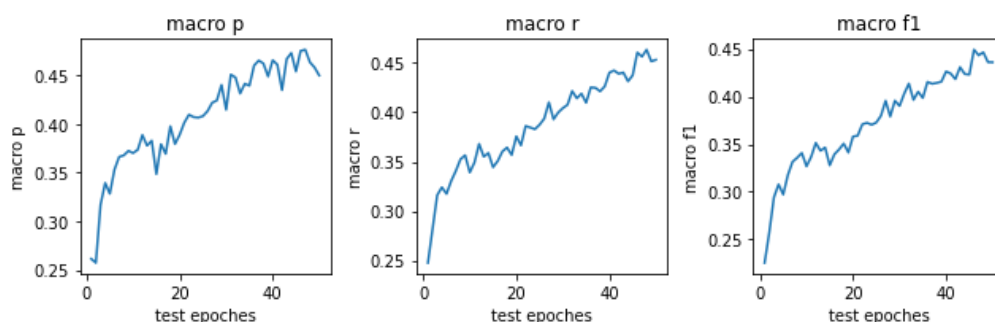


图 7 评价指标曲线

从图 7 可以看出随着训练的进行，文本情绪分类的宏准确率、宏回归值与宏 F1 值都在逐渐上升，最终结果为 test p: 0.4764, test r: 0.4564, test f1: 0.4434，相较验证集有所下降。在实验中尝试仅按照预测结果与真实值的匹配与否作为准确率标准，最终能达到 0.7 以上，这里猜测是计算方法略有不同。限于词典训练所用词汇量较小，能达到这个程度的准确率已经说明模型预测能力较强。

3. 预测结果展示

这里随机取一些样本进行测试：

```
true label: happy, pred label: happy
true label: sad, pred label: neural -> false
true label: neural, pred label: neural
true label: neural, pred label: neural
true label: neural, pred label: neural
true label: neural, pred label: happy -> false
true label: neural, pred label: neural
true label: neural, pred label: neural
true label: happy, pred label: happy
true label: happy, pred label: happy
```

图 8 预测结果 1

```
true label: neural, pred label: angry -> false
true label: angry, pred label: angry
true label: happy, pred label: happy
true label: neural, pred label: happy -> false
true label: happy, pred label: happy
true label: happy, pred label: angry -> false
true label: neural, pred label: neural
true label: angry, pred label: neural -> false
true label: neural, pred label: neural
true label: happy, pred label: happy
```

图 9 预测结果 2

可以看出平均预测准确率接近 0.7 左右，TextCNN 模型效果较好。

六、心得体会

综合实验为四选一，想要挑战一下没有接触过的领域，于是选择了 nlp 的任务。对于数据处理以及词典训练和应用模型这些方面都是从头学习，难度不小，任务量也很大。

首先面临的的就是对于数据的处理，分词在以前自己做词云的时候简单使用过，但划分数据集以及与标签一一对应并装载数据这方面却没有接触过，在同学的帮助下知道了 PyTorch 所提供的 torchtext 包，很大程度上对构造数据集提供了便利。然后是词向量的训练，网络上的教程大多是使用预训练的词向量来进行映射，而本实验要求不使用外部语料，因此需要自己训练。任务要求中提到了 gensim，于是从此下手使用了 word2vec 来进行词向量的训练。词向量的作用是为了将文本转化为二维张量，以便传入模型进行训练。这些前期准备都做好了后就开始模型的构建，这里可以说是按部就班的过程，根据经典模型一步步搭建即可。最终模型训练效果看起来不错，也是我对 nlp 的初次入门。

从一开始对 nlp 一无所知，到逐步开始了解经典模型，这其中不乏课程所学知识的助益。每一次实验模型的构建，参数的调整，结果的分析，都使这次综合实验减轻了不少工作量，这门课程使我受益匪浅。