## 一、数据预处理

## （一）：用外部数据训练字向量和词向量：

**1 网址：**<https://github.com/SophonPlus/ChineseNlpCorpus/tree/master/datasets/yf_dianping>

**2 文件：utils/**build\_w2v.py

**3 说明：**

440 万条大众点评的评论语料，解压后近2G，用于预训练词向量和字向量，用于对比词向量和字向量的效果。之后再用这次项目的数据集进行增量训练。

## （二）：数据统计分析

**1 文件：utils/statistic\_analysis.ipynb**

**2 说明：**

一是进行简繁体转换，二是对评论内容分词，三是将评论内容转化为字（为字向量模型做准备），四是统计字和词的长度分布，以供确定输入的最大长度时进行参考。最后把处理的结果保存起来备用。

分词后样本的最大长度取400时，可以覆盖94.88%的样本；分字后样本的最大长度取800时，可以覆盖97.62的样本。

对所有样本按长度进行排序，以便于对样本分batch时，每个batch的输入长度不同，从而减少pad。在训练时，发现这种做法可以提升训练速度，但是并没有提升模型的效果。

**3 问题：**

（1）去停用词吗？

涉及到情感分析，最好不去停用词，因为停用词表中包含表示程度、否定、标点符号等影响情感判定的字词，去掉的话可能情感都变了。比如：！！！就表示一种强烈的情感。

（2）表情符号如何处理？

举例：棒棒哒(=^x^=)用点评网买单还优惠了28元，挺划算的^\_^唯一遗憾是逛了一下午饿坏。

原本打算把表情符号找出来，在分词或分字的时候保留。可是发现加入jieba，是不能切分出来的。这个很费时间，这次项目就没做了。

（3）繁体字怎么处理？

繁体字必须转化为简体字，比方说：两条评论比较相似，但是一个繁体一个简体，如果进行转化，那么我们的训练样本相当于增加了。

## （三）：词向量的增量训练

**1 文件：utils/retrain\_w2v.py**

**2 说明：**

首先对输入的内容进行处理：一是把不在字典中词替换为<UNK>,二是如果不足最大输入长度，那就进行PAD，加上<PAD>，三是超过最大长度，就截断。

然后增量训练word2vec，就可以得到<PAD>和<UNK>的字向量。训练好word2vec后，重新建立word2id的字典，同时把输入内容由词转化为id，保存备用，后面我们就可以直接输入模型了。

训练的词向量模型中有249687个词（包含<UNK>和<PAD>），字向量模型中保留了12973个字。

## （四）：输入数据的准备

**1 文件：utils/data\_loader.py**

**2 说明：**

由于分类类别为20类，所以要做20个模型，这一步把20个模型的标签分别拿出来，然后进行one-hot编码，one hot 编码的维度是4.

这一步直接把已经转为id的x和已经进行one hot编码的y，直接准备好，实现数据预处理和训练的解耦。

进行pad的时候，我们的输入最大长度为350，而这里，通过max\_x\_len这个参数，我们可以自由调整真正训练时候的最大输入长度。

由于tensorflow 2.0的输入要求是float32格式，所以进行一下转化。

## （五）：batch 的生成

**1 文件：model\_tf2/batcher.py**

**2 说明：**

这一步用tf.data生成batch。

sample\_num的作用是：一开始要把模型调通，那么只需要取小量数据，比如10000.

## （六）：GPU设置和命令行参数

**1 文件：utils/gpu\_utils.py utils/params\_utils.py**

2 说明：

对gpu和训练参数进行设置。

## 二：模型构建和训练

## （一）：自定义模型

**1 文件：textcnn\_tf2/textcnn\_model.py**

**2 说明：**

用类和对象的方式，搭建了textcnn模型。用keras实现的文本分类模型太多了，不过为了熟悉tensorflow 2.0，还是自己尝试搭了一个。

由于要训练20个模型，RNN的模型太慢了，不好验证参数调整的结果，所以用了速度比较快的CNN模型。可能LSTM的效果更好，但速度太慢，最终也没有尝试。

**3 主要参数：**

|  |  |
| --- | --- |
| 主要参数 | 值 |
| Learning rate | 0.0001 |
| Kernel size | 2,3,4,5 |
| Filter numbers | 128 |
| dropout rate | 0.5 |

## （二）：训练模型和保存模型

**1 文件：**

**textcnn\_tf2/textcnn\_train\_helper.py**

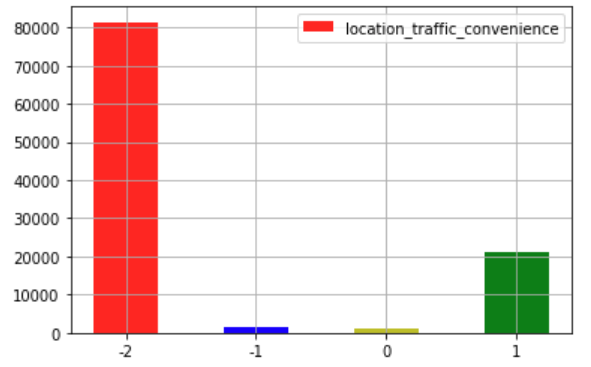
**textcnn\_tf2/textcnn\_train.py**

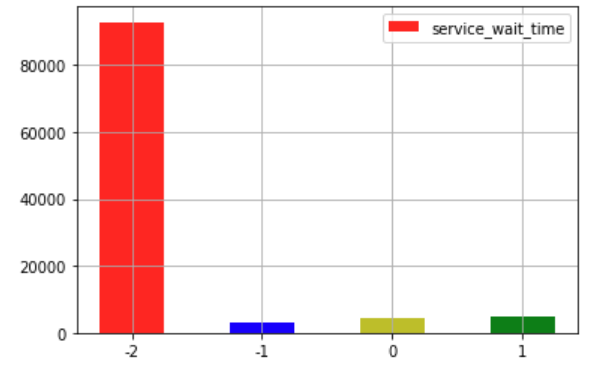
**2 说明：**

在20个小类下统计4类的频率分布，发现类别不平衡是普遍存在的现象，于是计算每个类别的频率，得到类别的权重，用于自定义loss函数，对每个类的损失进行加权，缓解类别不平衡问题（代价敏感学习）。

模型的保存方式，选择保存整个网络结构和参数，而不是仅仅保留参数，便于模型部署。可是由于要做20个模型，每个模型都要保存一次词向量矩阵，导致最后保存的模型文件非常大，每个模型200M，共4G！

**3 类别不平衡情况：**





## 三：参数调整记录

## （一）：第一次调参

F1值平均为 0.64255。主要参数如下，详细的结果看《调参和结果记录表》。

|  |  |
| --- | --- |
| batch\_size | 256 |
| max\_x\_len：输入的最大长度 | 250 |
| Epochs：训练轮次 | 20 |
| embed\_size：词向量维度 | 200 |
| num\_filters：卷积核的通道数 | 128 |
| dense\_units：dropout之后的全连接层的神经元个数 | 64 |
| dropout\_rate：丢弃率 | 0.5 |
| fiters size：卷积核的大小 | 2,3,4,5 |
| word2vec\_type：模型类别（字或词） | word |

## （二）：第二次调参

训练太慢了，于是首先训练集只取10000，验证集取1000，对比字向量和词向量的效果。结果发现词向量的效果好于字向量，于是再取全部的105000个样本进行训练。可能的原因是：这些评论文本比较通顺，非常少有错别字和语法的错误，与微博、贴吧等UGC的文本相比，质量更高，这种情况下，用词向量效果更好。

字模型的最大长度为800，词模型的最大长度为400，覆盖95%以上的样本。

这一次，把early stop的指标设置为loss，结果非常不好，F1值非常低，字向量的模型F1-值为0.5849，词向量的模型F1-值为0.6007。

这说明，用什么指标进行模型评估，就用什么指标做early stop。

|  |  |
| --- | --- |
| batch\_size | 128 |
| max\_x\_len（词/字） | 400/800 |
| epochs | 30 |
| embed\_size | 200 |
| num\_filters | 128 |
| dense\_units | 64 |
| dropout\_rate | 0.5 |
| fiters size | 2,3,4,5, |

## （三）：第三次调参

把早停的指标设置为F1-score，跑全部数据。

词向量的模型尝试在embedding 层加dropout，发现效果有所下降，根据直觉，如果drop 掉的词恰好是表示情感倾向的词，那么模型学习到信息会减少。所以embedding 层不宜进行drop ，给模型以完整的输入；在特征提取完后再进行dropout。把embedding层的dropout用在字向量模型上，效果更差。

增大输入的最大长度，以覆盖95%以上的样本的长度。最好的模型是词向量模型，输入的最大长度为400，embedding层不加dropout，F1值为0.658.

bilstm+attention的模型，在长文本上表现可能更好，不过训练实在太慢了，没有训练了。

## 四：模型的优缺点

## （一）：优点

1 用代价敏感学习来缓解类别不平衡问题，而不是通过下采样或上采样的方式，可以保持数据集的原始分布。

2 用textcnn模型来训练，训练速度快，便于调参时验证想法。

**（二）：缺点或待完善点**

1 可以尝试embedding层对词向量进行微调，或许可以提升效果。

2 可以尝试把 avg pool 和attention 加入，与max pool 的结果进行拼接。

3 可以尝试一些数据增强的方法。

4 没有尝试其他适合长文本分类的模型。