张芷敏 信息管理系 2000093005

文本数据集分类实验报告

一、 数据源

数据来源：
https://raw.githubusercontent.com/SophonPlus/ChineseNlpCorpus/master/datasets/waimai_10k/waimai_10k.csv

(一) 数据集描述

	label	review
0	1	很快，好吃，味道足，量大
1	1	没有送水没有送水没有送水
2	1	非常快，态度好。
3	1	方便，快捷，味道可口，快递给力
4	1	菜味道很棒！送餐很及时！

该数据是外卖平台用户评论的情感分类数据集，数据中有如下两个字段

字段	特征
label	评论的情感分类，以[0,1]表示，0 为负面，1 为正面
review	评论的文本内容

总数据量有 11987 条，其中正面的文本有 4000 条，而负面的有 7987 条，数据集中不存在重复数据，也没有缺失值。

二、 数据预处理方法

(一) 分词

在开始训练之前，我们需要对文本数据进行分词处理。

函数	作用	输出
get_stopword_list	读取文件生成停用词表	停用词表
cutSentence	以 jieba 全模式分词为猪，引入对于停用词表的调整与判断实现对 review 的切分，	一串以空格分开的 String

利用 cutSentence 函数对 data.review 进行 jieba 全模式分词。

(二) 训练集和测试集的拆分

利用 sklearn.model_selection 的 train_test_split 以 6：4 的比例将数据集拆分成训练集和测试集。以下为训练集和测试集的数据描述。

	训练集 train_set	测试集 test_set
数据量	7192	4795
正面数据	2390	1610
负面数据	4802	3185
正面数据占比	33.23%	33.58%

三、 机器学习——SVM

(一) 特征提取

为了进行文本特征提取，使用 CountVectorizer 的 fit_transform 函数进行特征数值计算，计算每一个词汇在文本中出现了频率，将训练文本中的词汇转换成词频矩阵。

获得词频矩阵后，用 TfidfTransformer 的 fit_transform 函数统计矩阵中每个词语的 TF-IDF 值。

随后，对测试集使用通过计算训练集相同的参数来对测试集进行尺度化。

由此我们得到文本中每个文字对应的 TF-IDF。

(二) SVM 模型训练

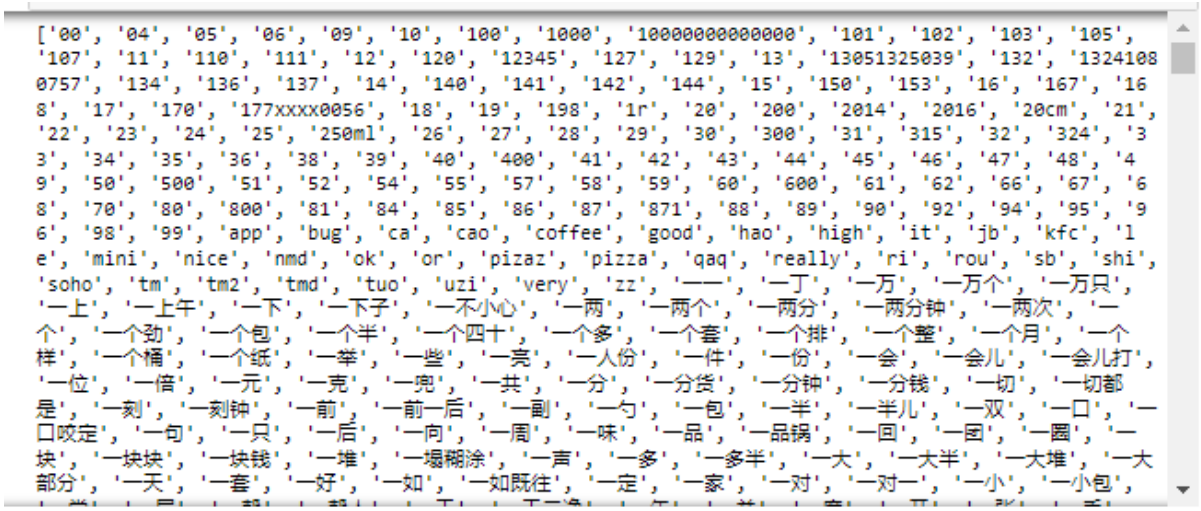
支持向量机 SVM 能够将数据集合压缩到支持向量集合，能够得到现有信息下的最优解，非常适合用于解决文本二分类问题。使用支持向量机构造分类器，将已有的 tf-idf 作为输入训练分类器，然后通过输入测试集测试模型分类效果。

(三) 模型评价与调参

1. 模型 1

	precision	recall	f1-score	support
0	0.88	0.93	0.90	3185
1	0.84	0.74	0.79	1610
accuracy			0.87	4795
macro avg	0.86	0.84	0.85	4795
weighted avg	0.87	0.87	0.86	4795

以上为第一次训练的 classification report.



在过程中可以发现 vectorizer 在生成词频矩阵时过滤掉了单个的字，如上图。

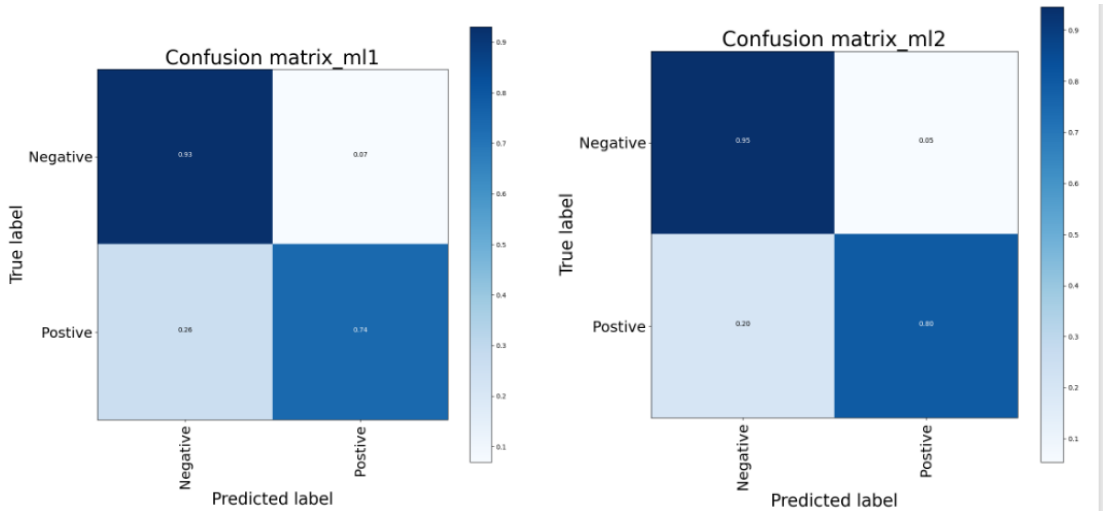
2. 模型 2

这是因为 CountVectorizer 中默认 token_pattern=r"(?u)\b\w\w+\b"，为了解决这个问题我们在实例化 CountVectorizer 的时候指定 token_pattern=r"(?u)\b\w+\b"，再次进行训练。

	precision	recall	f1-score	support
0	0.90	0.95	0.92	3185
1	0.88	0.80	0.84	1610
accuracy			0.90	4795
macro avg	0.89	0.87	0.88	4795
weighted avg	0.89	0.90	0.89	4795

可以看到和上图相比，模型在预测负面评论时，准确率，召回率和 F 值上都有些许的提升，而在预测正面评论时，召回率和 f 值有所下降，但准确率是有所提高的。

纳入长度为 1 的字后，单词数从 6875 增加至 8094。



以上为前后的混淆矩阵对比，模型 2 在召回率上表现得比模型 1 优秀，因此以模型 2 作为基准和深度学习的文本分类作对比。

四、 深度学习——LSTM

(一) 向量化

Gensim 是一款开源的第三方 Python 工具包，用于从原始的非结构化的文本中，无监督地学习到文本隐层的主题向量表达，word2vec 是 gensim 中一个利用神经网络模型获得词的分布式表示的工具，通过 word2vec 可以测量出词与词之间的相似性，挖掘它们之间的联系。

在词间关系上，利用 gensim 中的 word2vec 模型通过对分词后再处理成 list 的 review 进行向量化，训练后得到词间关系，其中单词量为 2999,进行 10 轮训练。以下是‘难吃’的其他词的关联度。

```
[('腻', 0.8233961462974548),
 ('失望', 0.8056866526603699),
 ('最难', 0.7990826368331909),
 ('恶心', 0.7910643219947815),
 ('没吃过', 0.7813606262207031),
 ('老', 0.7387266755104065),
 ('真难', 0.7344042062759399),
 ('简直', 0.732003390789032),
 ('硬', 0.7282747030258179),
 ('腻', 0.7278940081596375)]
```

(二) LSTM 模型构造与训练

1. 预训练 embedding 矩阵

利用 keras 中的 preprocessing 模块中的 tokenizer 学习文本的词典，获取单词和数字的映射关系，输出打印单词数，然后利用 texts_to_sequences，根据这个映射关系的词典将数据集中 text 字段中分词好的列表中的每个词转换成数字。由于 keras 只能接受长度相同的序列输入，因此需要利用 preprocessing 的 pad_sequences 函数对长度不同的 text 进行填充，使他们成为长度相同的序列。

为了配合 keras 需要的数据格式，reshape 训练集和测试集，将数据变换成 n 行 1 列。

2. 建立模型

在模型上，使用 Sequential 顺序模型，是线性不分叉的结构顺序，层和层之间不会跨层产生联系，是多个网络层的线性堆叠。案例总共添加了 4 层，分别是嵌入(embedding)层，随机失活（dropout）层，LSTM 层以及全连接层。

嵌入层主要是将离散变量转变为连续向量，也可以对数据进行降维和升维，虽然 embedding 层会找到合适的向量刻画词间关系，但是通过上文 word2vec 得到的词的分布式表示矩阵初始化嵌入层中的嵌入矩阵，能够有效加速模型的收敛速度。

Dropout 层通过在每次训练中以一定概率忽略一些神经元达到防治数据过拟合的作用，此次训练的 dropout 层采用 50%的概率，此举不但可以减少数据过拟合发生的可能性，还可以一定程度地降低训练模型所需的时间。

LSTM 层是带有长时记忆的循环神经网络，和只与上一节点有关系的 RNN 不同，LSTM 通过在 RNN 上只用来储存上一时刻的信息的 cellstate 上加入输入门、遗忘门和输出门，用于有条件地在 cell state 存储更之前的信息作为下一时刻的输入，从而将 RNN 的短时记忆拓展成长时记忆。

全连接层对 LSTM 提取出的特征作非线性变化，提取这些特征之间的关联，最后映射到输出空间上。这里使用 sigmoid 作为激活函数，输出为 1 维。

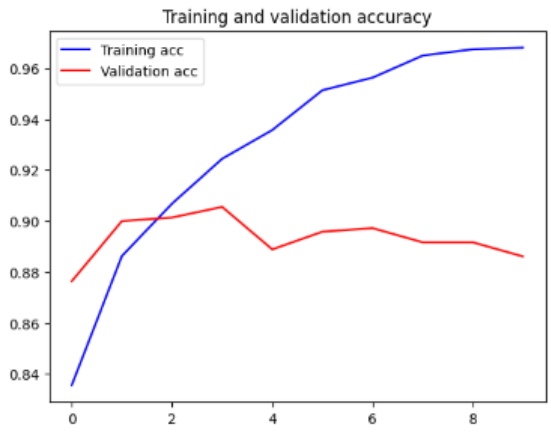
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 300)         3135300

 dropout (Dropout)           (None, None, 300)         0

 lstm (LSTM)                 (None, 100)               160400

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 3,295,801
Trainable params: 3,295,801
Non-trainable params: 0
_________________________________________________________________
```

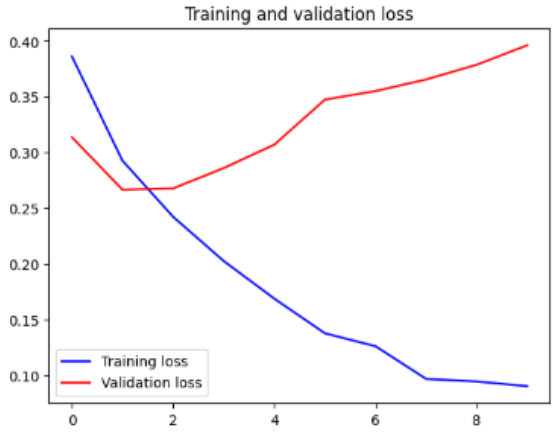
上图为模型的概览，包括各层的输出维度。

用 compile 函数配置模型的学习过程，模型优化器 optimizer 为 adam ，损失函数 loss 为 binary_crossentropy，评估标准 metrics =accuracy。

对于模型的回调函数，为了避免经过一定 epoch 迭代之后，模型效果不再提升，原有的学习率可能已经不再适应该模型，可以利用回调函数中的 ReduceLROnPlateau 参数做到在训练时调整学习率，当标准评估 val_loss 停止提升时，降低学习速率。而利用 EarlyStopping 参数则可以定义何时停止训练，当被监测的数量 val_acc 不再提升时本模型停止训练。

随后用训练集对数据进行训练了，在每次训练时，设置验证集 validation_split=0.1，每次训练时将 10%的训练集划分出来作为验证集，进行 10 轮训练,gonghao





以上为对模型训练过程的可视化，可以发现再第二次训练时模型就过拟合了。于是决定重新调整参数，做过将 batch_size 调整成 16、在全连接层正则化、调整 LSTM 的神经元数变为 64、128 等尝试，结果都不理想，仍然会在模型第 2 轮训练完成前发生过拟合。因此决定沿用第一次的模型与 SVM 模型进行对比分析。

	precision	recall	f1-score	support
0	0.91	0.93	0.92	3185
1	0.85	0.82	0.84	1610
accuracy			0.89	4795
macro avg	0.88	0.88	0.88	4795
weighted avg	0.89	0.89	0.89	4795

以上为 LSTM 模型训练结果。

五、 模型评价与分析

```
print(report_ml2)
```

	precision	recall	f1-score	support
0	0.90	0.95	0.92	3185
1	0.88	0.80	0.84	1610

```
print(report_lstm)
```

	precision	recall	f1-score	support
0	0.91	0.93	0.92	3185
1	0.85	0.82	0.84	1610

左图为 SVM 模型分类结果，右图为 LSTM 模型分类结果，可以发现 SVM 不管是在准确率、召回率、F1 值上都是略优于 LSTM 的，其中在 LSTM 仅在正面分类上召回率比 SVM 的召回率高了 2%。以下对两者在分类效果上做更深入的分析。

```
1/1 [==============================] - 0s 25ms/step
不好吃
机器学习预测结果：正面
深度学习预测结果：负面

1/1 [==============================] - 0s 25ms/step
不太好吃
机器学习预测结果：正面
深度学习预测结果：负面

1/1 [==============================] - 0s 24ms/step
有点咸，但是总体来说还不错
机器学习预测结果：正面
深度学习预测结果：正面

1/1 [==============================] - 0s 28ms/step
如果可以更便宜会好点
机器学习预测结果：正面
深度学习预测结果：正面

1/1 [==============================] - 0s 34ms/step
还以为会很大份，原来只有那么一点
机器学习预测结果：负面
深度学习预测结果：正面

1/1 [==============================] - 0s 33ms/step
还以为会很大份，结果只有那么一点
机器学习预测结果：正面
深度学习预测结果：正面

1/1 [==============================] - 0s 30ms/step
原来那么小份
机器学习预测结果：负面
深度学习预测结果：负面
```

```
1/1 [==============================] - 0s 36ms/step
包装不错
机器学习预测结果：正面
深度学习预测结果：正面

1/1 [==============================] - 0s 31ms/step
蛮好吃的
机器学习预测结果：正面
深度学习预测结果：正面

1/1 [==============================] - 0s 40ms/step
家人都很喜欢
机器学习预测结果：正面
深度学习预测结果：正面

1/1 [==============================] - 0s 33ms/step
吃出了奇怪的东西
机器学习预测结果：负面
深度学习预测结果：负面

1/1 [==============================] - 0s 34ms/step
不够我吃
机器学习预测结果：负面
深度学习预测结果：正面

1/1 [==============================] - 0s 31ms/step
不够吃
机器学习预测结果：负面
深度学习预测结果：正面
```

可以看到面对相同的评论，SVM 和 LSTM 存在分类不一致的现象，SVM 在遇到情况 1 和 2 这种有着“不好吃”的词语的情况的时候会判断错误，检查数据集和 jieba 的分词结果，发现数据集中有 375 个”不好吃”这个词结果中没有“不好吃”这个词，于是通过 jieba.add_word(“不好吃”)把这个词录进 jieba，再次进行预测（这部分代码由于只是加了个词就不另外提交了）。

```
不好吃
机器学习预测结果：负面

不太好吃
机器学习预测结果：正面
```

发现这次模型在分类”不好吃”时正确地将它分成负面，但是模型在面对”不太好吃”这个评论是仍然将他分成正面，检查数据集发现数据集中有 26 个”不太好吃”，由此可见该模型受 jieba 的影响非常大，jieba 是否有录用相关词语会直接地影响模型的分类效果，而 LSTM 在这种情况上并没有受到影响。

对于第 3 和第 4 个情况，很遗憾地 LSTM 将他们全分错了，但 SVM 但面对”结果”和”原来”这两个词时产生了不同的分类结果，把”原来”分成负面，”结果”分成正面。

```
全数据中搜索
搜索词: 结果
positve: 24
total count: 272

搜索词: 原来
positve: 6
total count: 28

训练集中搜索
搜索词: 结果
positve: 17
total count: 158

搜索词: 原来
positve: 2
total count: 13
```

```
全数据中搜索
搜索词: 不够
positve: 22
total count: 76

训练集中搜索
搜索词: 不够
positve: 12
total count: 45

全数据中搜索
搜索词: 不够吃
positve: 0
total count: 0

训练集中搜索
搜索词: 不够吃
positve: 0
total count: 0
```

观察这两个单词在数据中的情况，发现他们在里面通常都属于负面情况，不同的是“原来”占比 15%,“结果”占比 10%，推测是由于逆文档频率的计算上的差异导致这种结果的不同。

对于第 5 和第 6 中情况，机器学习正确地将他们分成负面，但深度学习则错误地分成正面，参考上图可以发现不够这个词在数据集总共出现 76 次，其中超过 75%是负面的，jieba 分词中有”不够”这个词。由于 jieba 没有“不够吃”这个词，因此自行在数据集中搜索发现数据集中其实有 25 个”不够吃”，其中 17 个为负面，8 个为正面，但和情况 1 和 2 的同样不存在此表中的”不好吃”相比，LSTM 并没有将他们辨认出来。

由此可见 SVM 模型在利用 ifidf 进行数据预处理后，面对数据量的稀少仍然能够很好地完成任务，但受到分词结果的限制，而 LSTM 则相反，它对分词要求较低，但对数据量要求大，在面对有较多样本的情况能更好地做出分类。

六、 总结与不足

(一) 总结

机器学习和深度学习模型各有优缺，在特征工程和数据量上的要求不同，机器学习对于特征提取的要求较高，但它能够在小样本上反应出更好的结果，且相较于深度学习，机器学习对于设备的要求较低，训练耗时较短。而深度学习模型由于能够直接从数据中学习与提取特征，对于特征工程的要求较低，可是对运行设备的要求较高，训练耗时长，对数据量要求也较高。

在样本受限时，通过人工进行特征提取后进行机器学习会是一个更好的分类方式，而当具有足够的计算资源和样本量时，深度学习能够更容易取得比机器学习更佳的表现。

(二) 不足与改进之处

在 SVM 部分，没有在分词上进行更多的特征处理如加词、停用词、同义词等。

在 LSTM 部分，没能成功解决模型过拟合的问题。推测过拟合的原因之一还有数据，数据量不足以及数据分布不平均是可能原因。在此数据集中，正面样本仅占比33%左右，数据分布不平均，由于时间关系并没有进一步对数据进行处理，可以尝试使用过采样，让正面样本重复出现达到平均数据分布的效果。