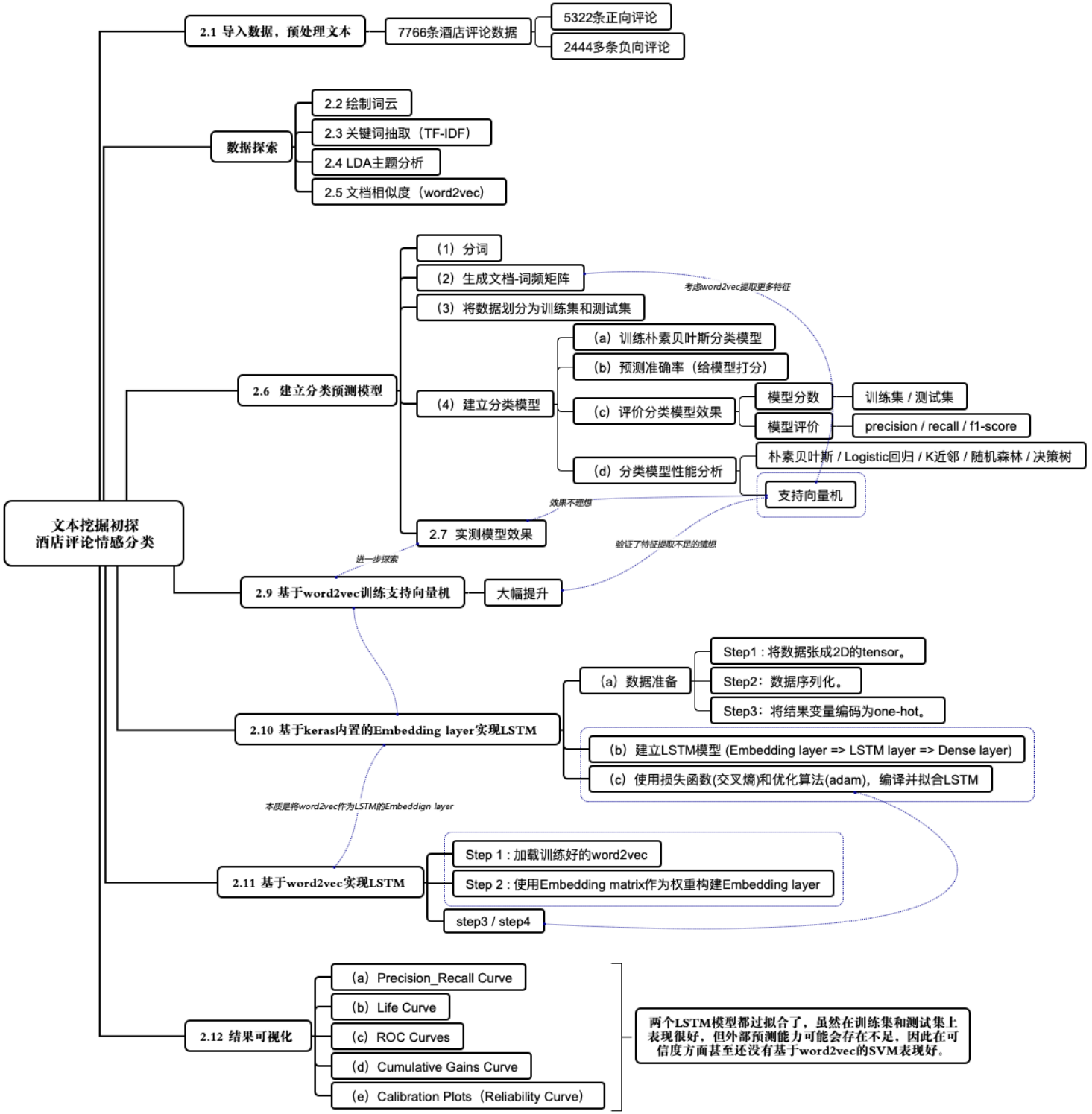
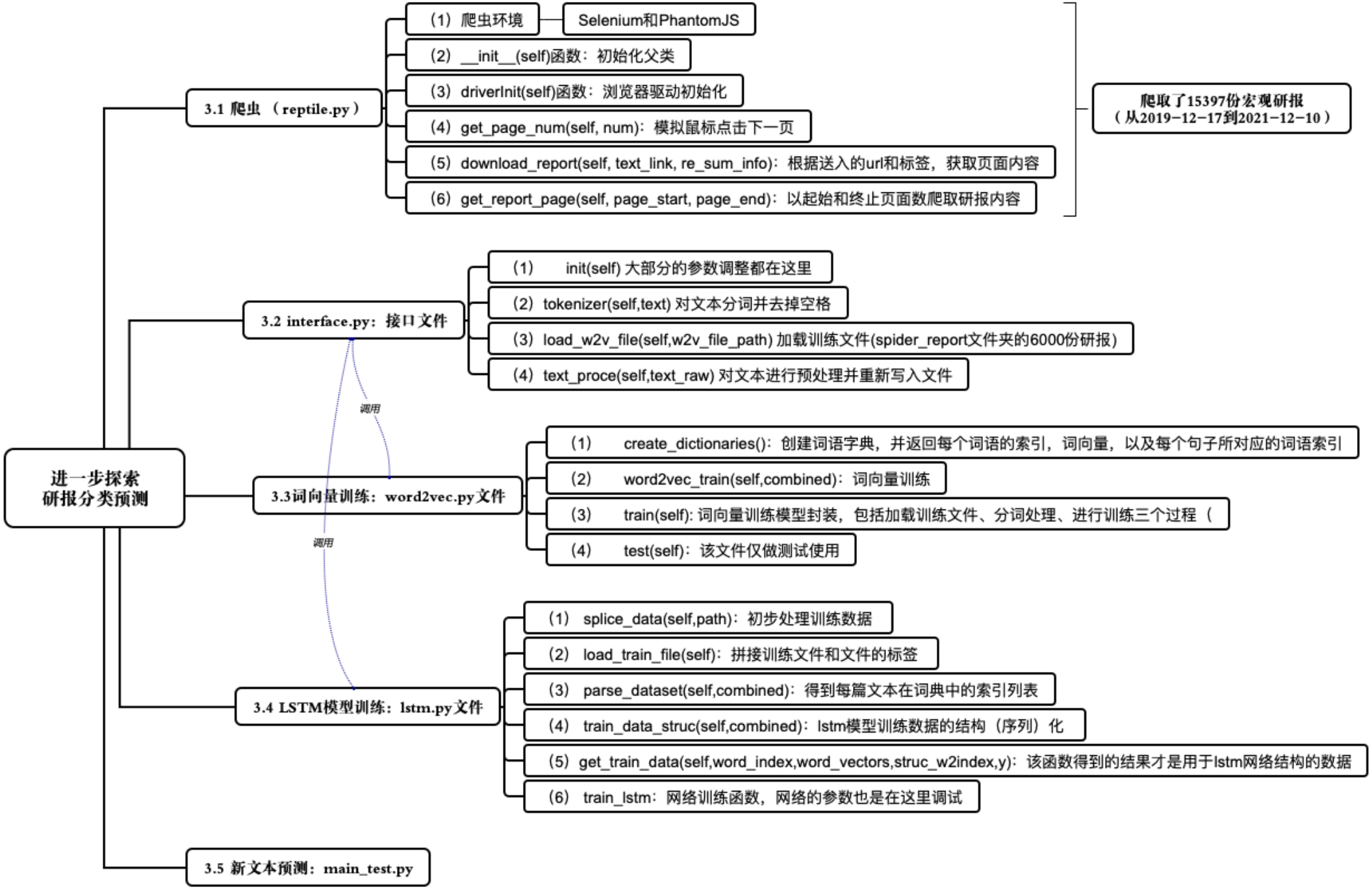

文本挖掘初探
酒店评论情感分类
2.1 导入数据，预处理文本
7766条酒店评论数据
5322条正向评论
2444多条负向评论
数据探索
2.2 绘制词云
2.3 关键词抽取（TF-IDF）
2.4 LDA主题分析
2.5 文档相似度（word2vec）
2.6 建立分类预测模型
（1）分词
（2）生成文档-词频矩阵
（3）将数据划分为训练集和测试集
（4）建立分类模型
（a）训练朴素贝叶斯分类模型
（b）预测准确率（给模型打分）
（c）评价分类模型效果
模型分数
训练集 / 测试集
模型评价
precision / recall / f1-score
（d）分类模型性能分析
朴素贝叶斯 / Logistic回归 / K近邻 / 随机森林 / 决策树
支持向量机
考虑word2vec提取更多特征
2.7 实测模型效果
效果不理想
验证了特征提取不足的猜想
进一步探索
2.9 基于word2vec训练支持向量机
大幅提升
2.10 基于keras内置的Embedding layer实现LSTM
（a）数据准备
Step1：将数据张成2D的tensor。
Step2：数据序列化。
Step3：将结果变量编码为one-hot。
（b）建立LSTM模型 (Embedding layer => LSTM layer => Dense layer)
（c）使用损失函数(交叉熵)和优化算法(adam)，编译并拟合LSTM
本质是将word2vec作为LSTM的Embeddign layer
2.11 基于word2vec实现LSTM
Step 1 : 加载训练好的word2vec
Step 2 : 使用Embedding matrix作为权重构建Embedding layer
step3 / step4
2.12 结果可视化
（a）Precision_Recall Curve
（b）Life Curve
（c）ROC Curves
（d）Cumulative Gains Curve
（e）Calibration Plots（Reliability Curve）
两个LSTM模型都过拟合了，虽然在训练集和测试集上表现很好，但外部预测能力可能会存在不足，因此在可信度方面甚至还没有基于word2vec的SVM表现好。


进一步探索
研报分类预测
3.1 爬虫 （reptile.py）
（1）爬虫环境
Selenium和PhantomJS
（2）__init__(self)函数：初始化父类
（3）driverInit(self)函数：浏览器驱动初始化
（4）get_page_num(self, num)：模拟鼠标点击下一页
（5）download_report(self, text_link, re_sum_info)：根据送入的url和标签，获取页面内容
（6）get_report_page(self, page_start, page_end)：以起始和终止页面数爬取研报内容
爬取了15397份宏观研报
（从2019-12-17到2021-12-10）
3.2 interface.py：接口文件
（1） init(self) 大部分的参数调整都在这里
（2）tokenizer(self,text) 对文本分词并去掉空格
（3）load_w2v_file(self,w2v_file_path) 加载训练文件(spider_report文件夹的6000份研报)
（4）text_proce(self,text_raw) 对文本进行预处理并重新写入文件
调用
3.3词向量训练：word2vec.py文件
（1） create_dictionaries()：创建词语字典，并返回每个词语的索引，词向量，以及每个句子所对应的词语索引
（2） word2vec_train(self,combined)：词向量训练
（3） train(self): 词向量训练模型封装，包括加载训练文件、分词处理、进行训练三个过程（
（4） test(self)：该文件仅做测试使用
调用
3.4 LSTM模型训练：lstm.py文件
（1） splice_data(self,path)：初步处理训练数据
（2） load_train_file(self)：拼接训练文件和文件的标签
（3） parse_dataset(self,combined)：得到每篇文本在词典中的索引列表
（4） train_data_struc(self,combined)：lstm模型训练数据的结构（序列）化
（5）get_train_data(self,word_index,word_vectors,struc_w2index,y)：该函数得到的结果才是用于lstm网络结构的数据
（6） train_lstm：网络训练函数，网络的参数也是在这里调试
3.5 新文本预测：main_test.py

一、实验目的与要求

实验目的：

1．熟悉人工智能与机器学习基本知识与方法；

2．掌握人工智能与机器学习相关技术及原理；

3．运用所学人工智能与机器学习方法和技术，设计解决实际问题的人工智能系统；

实验要求：

1. 实验提交文件为实验报告和相关程序代码，以压缩包的形式提交，命名规则为“学号数字+姓名+Task4”，如 20xx154099 张三 Task4；

2. 所有素材和参考材料需列明出处，实验报告中的图片和程序代码建议标注个人水印或标识信息：姓名，班级，学号信息；

二、实验内容与方法

实验内容：自选（实验、课程讲授内容程序实现或实际应用，建议采用华为云平台）

三、实验步骤与过程

一、文本挖掘

数据挖掘（Data Mining）是指从大量数据中挖掘有趣模式和知识的过程1，近年来备受关注，在快速发展的大数据时代展现了极其重要和广泛的应用前景。其中，从自然语言文本中挖掘用户所感兴趣的模式和知识的方法和技术称为文本数据挖掘（Text Data Mining），也简称为文本挖掘（Text Miining）。这里所说的文本包括普通 TXT 文件、DOC/ DOCX 文件、PDF 文件和 HTML 文件等各类以语言文字为主要内容的数据文件。

典型的文本挖掘包括文本分类、文本聚类、主题模型、情感分析与观点挖掘、话题检测与跟踪、信息抽取和文本自动摘要等技术。在本实验，笔者基于个人兴趣和能力以及本学期人工智能课程所学习的算法（机器学习+深度学习），对文本挖掘进行了一些探索。实验思路如图 1 思维导图所示：

图 1: AIML TASK4 实验思维导图（见前 2 页）

二、文本挖掘初探：酒店评论情感分类

2.1 导入数据，预处理文本

笔者使用 ChnSentiCorp_htl_all 数据集[2]进行分析，该数据集是文本分类（情感分析）的知名中文数据集，包括 7766 条酒店评论数据，其中，5322 条正向评论，2444 多条负向评论。首先利用 pd.read_csv 导入数据，并将积极评论编码为 1，消极评论编码为 0，再合并积极和消极评论，得到本次实验的数据 all_data ,如图 2 所示：

---

[1] [Han et al., 2012] Han JW, Kamber M and Pei JN. 2012. Data mining-concepts and techniques 3rd ed. (范明,孟小锋,译. 数据挖掘:概念与技术.北京:机械工业出版社,2012)

[2] https://raw.githubusercontent.com/SophonPlus/ChineseNlpCorpus/master/datasets/ChnSentiCorp_htl_all/ChnSentiCorp_htl_all.csv

	text	cls
0	"距离川沙公路较近,但是公交指示不对,如果是""蔡陆线""的话,会非常麻烦.建议用别的路线....	1
1	商务大床房，房间很大，床有2M宽，整体感觉经济实惠不错!	1
2	早餐太差，无论去多少人，那边也不加食品的。酒店应该重视一下这个问题了。房间本身很好。	1
3	宾馆在小街道上，不大好找，但还好北京热心同胞很多~宾馆设施跟介绍的差不多，房间很小，确实挺小...	1
4	"CBD中心,周围没什么店铺,说5星有点勉强.不知道为什么卫生间没有电吹风"	1
...	...	...
7761	尼斯酒店的几大特点：噪音大、环境差、配置低、服务效率低。如：1、隔壁歌厅的声音闹至午夜3点许...	0
7762	盐城来了很多次，第一次住盐阜宾馆，我的确很失望整个墙壁黑咕隆咚的，好像被烟熏过一样家具非常的...	0
7763	看照片觉得还挺不错的，又是4星级的，但入住以后除了后悔没有别的，房间挺大但空空的，早餐是有但...	0
7764	我们去盐城的时候那里的最低气温只有4度，晚上冷得要死，居然还不开空调，投诉到酒店客房部，得到...	0
7765	说实在的我很失望，之前看了其他人的点评后觉得还可以才去的，结果让我们大跌眼镜。我想这家酒店以...	0

7766 rows × 2 columns

图 2 酒店评论数据 all_data

2.2 绘制词云

使用 jieba 库对文档进行预处理，首先用 replace 将换行符替换为空格。因为本数据集的研究主题是酒店，为了避免后续“酒店”此词权重过高，也将“酒店”此词替换为空。jieba 分词有三种模式：全模式、精确模式、搜索引擎模式，笔者这里使用全模式和精确模式，通过 jieba.cut 实现中文文本分词。在处理文本之前或之后需要自动过滤掉某些字或词，所以需要使用停用词表，笔者在本次实验过程中使用的是哈工大的中文停用词表[3]。

接着笔者使用 python 非常优秀的词云展示第三方库 wordcloud 库进行词云绘制，主要分为两个步骤，一是使用 WordCloud 创建评论文本的词云对象，包括设置字体、词云图的背景颜色、去掉停用词、最多能显示的词数大小以及最大字体的大小等参数。二是使用 matplotlib.pyplot 库绘制词云，用 plt.imshow(word_cloud)语句进行词云展示（如图 3 所示）。

从图中可见，积极性评论确实更多地出现“不错”、“很好”等表述，消极性评论出现了“前台”“打电话”“差”等表述。比较符合我们的常识，但是词云图显示的主要是关键词（高频词），是一种总览，比较难从中发现一些主题模式。

图 3 词云图展示（左：积极评论，右：消极评论）

2.3 关键词抽取（TF-IDF）

TF-IDF（term frequency–inverse document frequency，词频-逆向文件频率）是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。其主要思想是：如果某个单词在一篇文章中出现的频率 TF 高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。其中，词频（TF）表示词条（关键字）在文本中出现的频率，逆向文件

[3] https://github.com/goto456/stopwords

频率 (IDF) 表示某一特定词语的 IDF，可以由总文件数目除以包含该词语的文件的数目，再将得到的商取对数得到。

笔者使用了 jieba.analyse 库中的 extract_tags 方法提取文档关键词，该方法本质上是基于 TF-IDF 方法，也是实现 TF-IDF 最简单易行的方法。得到积极评论和消极评论的关键词如图 4 所示，可见，提取出积极评论和消极评论都包含酒店、房间、入住、服务、早餐等评价对象词，比较重合。但积极评论出现了不错、方便、干净等好评关键词，消极评论出现了卫生间、晚上、没有等表示酒店可能做得不好的关键词。

```
[('酒店', 0.2355155552225905),
 ('房间', 0.14884893740225225),
 ('不错', 0.117333835955301),
 ('入住', 0.07550899902047907),
 ('服务', 0.06747167002734858),
 ('早餐', 0.06478203399695705),
 ('比较', 0.04772438916662042),
 ('感觉', 0.04519728853107141),
 ('方便', 0.04348479818126573),
 ('前台', 0.039933744763990975),
 ('宾馆', 0.03784634784619873),
 ('非常', 0.03391702937784472),
 ('设施', 0.033731448046670054),
 ('可以', 0.03349605774215204),
 ('服务员', 0.03298656491282223),
 ('携程', 0.03288631403594539),
 ('2008', 0.031174428932069974),
 ('下次', 0.029658054926841853),
 ('干净', 0.027878082766973757),
 ('还是', 0.027665395208601007)]
```

```
[('酒店', 0.2171582545971048),
 ('房间', 0.1499241052188293),
 ('入住', 0.07568336452583642),
 ('携程', 0.07378034900824917),
 ('前台', 0.06920899764906877),
 ('服务员', 0.04958533790822064),
 ('没有', 0.04296153526014419),
 ('服务', 0.04184076221128819),
 ('早餐', 0.037961858627015045),
 ('宾馆', 0.03547194173922455),
 ('我们', 0.033793276715127156),
 ('客人', 0.02945696436252977),
 ('2008', 0.025966786864384125),
 ('设施', 0.025693345443947645),
 ('退房', 0.024945169942903456),
 ('晚上', 0.024010384056660377),
 ('感觉', 0.02298701378327763),
 ('10', 0.022176015059364545),
 ('非常', 0.020989748572476336),
 ('卫生间', 0.01945158625979008)]
```

图 4 TF-IDF 关键词抽取（左：积极评论，右：消极评论）

2.4 LDA 主题分析[4]

主题模型（Topic Model）能够识别在文档里的主题，挖掘语料里隐藏信息，并且在主题聚合、从非结构化文本中提取信息、特征选择等场景有广泛的用途，Latent Dirichlet Allocation（LDA）是其中最具代表性的模型，是一种非监督机器学习技术。由于篇幅所限，笔者在这里将不再赘述算法原理，简单的说，即把文本语料库输入，模型会输出这些文本大概在讲哪些主题。

笔者在使用 gensim 库实现 LDA。首先，同样对文本进行预处理和分词，生成语料库；接着，由于 gensim 所需要的输入格式为每条评论是一个列表，元素为词语，所以需要将文档转换为 list；然后，用 gensim 包提供的函数 corpora.Dictionary，生成文本对应的字典，接着还要把每个文档都变成一个数值向量，笔者在这里使用词袋模型（Bag of words）（调用 doc2bow 函数）将语料库进行数字化，生成 BOW 稀疏向量。最后，使用 gensim.models.ldamodel 的 LdaModel 训练 LDA 模型，LDA 是一种无监督学习方法，可以自由选择主题的个数，笔者在这里设置 num_topics = 1,同时输出训练时间。

结果如图 5 所示，但结果似乎没有太大的差异，笔者考虑可能是因为此数据集过于口语化，且由于停用词表直接使用的是哈工大的停用词表，可能根据文本结构进一步完善停用词表，可以获得更好的训练结果。

```
CPU times: user 3.52 s, sys: 393 ms, total: 3.91 s
Wall time: 1.76 s
```

```
1 pos_ldamodel.print_topics()
```

```
[(0,
  '0.025*"很" + 0.016*"不错" + 0.012*"好" + 0.011*"还" + 0.010*"服务" + 0.010*"都" + 0.008*"住"
+ 0.008*"比较" + 0.007*"不" + 0.007*"入住"')]
```

[4] https://blog.csdn.net/Lyric1/article/details/105126437

```
CPU times: user 3.32 s, sys: 319 ms, total: 3.64 s
Wall time: 1.57 s
```

```
1  neg_ldamodel.print_topics()
```

```
[(0,
  '0.012*"不" + 0.010*"都" + 0.009*"很" + 0.009*"说" + 0.008*"住" + 0.007*"还" + 0.007*"入住" +
0.006*"服务" + 0.006*"前台" + 0.005*"携程"')]
```

图 5 LDA 主题分析（上：积极评论，下：消极评论）

2.5 文档相似度（word2vec）[5]

词向量（word2vec）是一个将单词转换成向量形式的工具。可以把对文本内容的处理简化为向量空间中的向量运算，计算出向量空间上的相似度，来表示文本语义上的相似度。笔者同样使用 Gensim 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。

首先，进行语料的预处理，将文本转换成模型所能理解的稀疏向量，Gensim 的 word2vec 输入的是句子的序列，每个句子是一个单词列表，通过分词以及去除停用词，得到的数据结果如图 6 所示。

```
1  all_data['text_list']
```

```
0       [距离, 川沙, 公路, 较近, 公交, 指示, 不, 蔡陆线, 会, 非常, 麻烦, 建议...
1       [商务, 大床, 房, 很大, 床有, 2M, 宽, 整体, 感觉, 经济, 实惠, 不错, !]
2       [早餐, 太, 差, 去, 人, 不加, 食品, 应该, 重视, 一下, 问题, 本身, 很...
3       [宾馆, 小, 街道, 上, 不大好, 找, 还好, 北京, 热心, 同胞, 很多, ~, ...
4       [CBD, 中心, 周围, 没什么, 店铺, 说, 5, 星, 有点, 勉强, 不, 知道,...
                              ...
7761    [尼斯, 几大, 特点, 噪音, 大, 环境, 差, 配置, 低, 服务, 效率, 低, 1...
7762    [盐城, 很, 多次, 第一次, 住, 盐阜, 宾馆, 的确, 很, 失望, 整个, 墙壁,...
7763    [看, 照片, 觉得, 还, 挺不错, 4, 星级, 入住, 以后, 后悔, 挺大, 空空,...
7764    [去, 盐城, 最低气温, 4, 度, 晚上, 冷得, 要死, 居然, 还, 不开, 空调,...
7765    [说, 实在, 很, 失望, 之前, 看, 其他人, 点评, 后, 觉得, 还, 才, 去,...
Name: text_list, Length: 7766, dtype: object
```

图 6 word2vec 的输入

Word2vec 有很多可以影响训练速度和质量的参数，参数 min_count 是对词进行过滤，频率小于 min-count 的单词则会被忽视，为了更好地保留数据，笔者在这里设置为 1；参数 size 是输出词向量的维数，即神经网络的隐藏层的单元数。值太小会导致词映射因为冲突而影响结果，值太大则会耗内存并使算法计算变慢，大的 size 需要更多的训练数据，但是效果会更好，笔者在这里设置为 300；其他使用默认参数。接着训练模型，训练完毕的模型实质是一个词代表一个向量，如图 7 所示，卫生这个词是一个（300，1）的向量。

```
1  # 训练完毕的模型实质
2  print(w2vmodel.wv["卫生"].shape)
3  w2vmodel.wv["卫生"]
```

```
(300,)

array([-0.21681097, -0.34666854, -0.00703033,  0.14482367,  0.6827701 ,
       -0.2206141 , -0.09363696, -0.14329071,  0.67450756,  0.3988966 ,
       -0.00695775,  0.2362109 , -0.03555465, -0.4070188 , -0.01673803,
        0.18904129,  0.07421786,  0.01655366, -0.27252838, -0.13345917,
       -0.43823028, -0.5161893 , -0.26396695,  0.9807082 ,  0.13538274,
       -0.35403758,  0.15029688,  0.17646515,  0.04811452, -0.08791401,
        0.51134914,  0.65979975, -0.13960259, -0.63649416,  0.5026108 ,
       -0.34404635,  0.5232417 ,  0.36957428, -0.08088782,  0.47919166,
```

图 7 训练完毕的模型实质

同时，可以根据向量度量词语之间的相似度，得到一些有关系的词组，如图 8 所示，可以看到文本中，跟卫生这个词相似度比较高的一些词。

[5] https://blog.csdn.net/qq_41814556/article/details/80990976

```
#词语相似度
w2vmodel.wv.most_similar('卫生')
```

```
[('尤其', 0.9500515460968018),
 ('很香', 0.9347293972969055),
 ('管理水平', 0.9342608451843262),
 ('酒店设备', 0.9311650991439819),
 ('整洁', 0.9282059073448181),
 ('卫生死角', 0.9280813932418823),
 ('高雅', 0.92791348695755),
 ('酒店设施', 0.9239890575408936),
 ('Thepositionofthishotelisverybad', 0.9232931733131409),
 ('令人满意', 0.9230090379714966)]
```

```
w2vmodel.wv.most_similar('早餐')
```

```
[('自助餐', 0.885247528553009),
 ('品种', 0.8770273923873901),
 ('自助', 0.8641621470451355),
 ('丰富', 0.8254192471504211),
 ('五谷杂粮', 0.8252454996109009),
 ('晚餐', 0.8240232467651367),
 ('大杂烩', 0.8142313361167908),
 ('五十多个', 0.8077825903892517),
 ('丁山', 0.8030723333358765),
 ('种类', 0.8013982772827148)]
```

图 8 词语相似度

2.6 建立分类预测模型

（1）分词

同样使用 jieba 进行分词，注意这里需要将分词后的结果转换为字符串形式，不然后面使用 fit_transform 方法的时候会出错，笔者在这里 debug 了很久才发现是这个问题。

（2）生成文档-词频矩阵

本模型使用的文档-词频矩阵作为纳入模型训练的文本特征，结果 wordmtx 为 7766*27530 的稀疏矩阵。笔者在这里使用的是 CountVectorizer 这个常见的特征数值计算类，这是一个文本特征提取方法，对于每一个训练文本，它只考虑每种词汇在该训练文本中出现的频率，将文本中的词语转换为词频矩阵，并通过 fit_transform 函数计算各个词语出现的次数，从而生成文档-词频矩阵。

```
#生成文档词频矩阵
from sklearn.feature_extraction.text import CountVectorizer
counvec = CountVectorizer() # 数据需要转换为可迭代的list格式
wordmtx = counvec.fit_transform(all_data['text_cut'])
```

```
wordmtx
```

```
<7766x27530 sparse matrix of type '<class 'numpy.int64'>'
        with 271948 stored elements in Compressed Sparse Row format>
```

图 9：文档-词频矩阵

（3）将数据划分为训练集和测试集

各个参数为：x_train, x_test, y_train, y_test = train_test_split(wordmtx, all_data.cls, test_size = 0.3, random_state = 111)，需要注意的是参数 random_state 是用来控制随机状态的，固定 random_state 后，每次构建的模型是相同的、生成的数据集是相同的、每次的拆分结果也是相同的，这样后面构建的分类模型才有可比性。

（4）建立分类模型

笔者基于 **sklearn** 库建立了朴素贝叶斯、Logistic 回归、支持向量机、K 近邻、随机森林和决策树六种分类模型。由于各模型实现原理类似，此处以**朴素贝叶斯模型**实现过程为例进行简单展示。

（a）训练朴素贝叶斯分类模型

```
from sklearn import naive_bayes
NBmodel = naive_bayes.MultinomialNB()
NBmodel.fit(x_train, y_train) # 拟合模型
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

图 10 训练朴素贝叶斯分类模型

（b）预测准确率（给模型打分）

sklearn 的 score 方法提供了一个缺省的评估法则对模型进行打分，即用训练好的模型在验证集上进行评分（0~1），1 分代表最好。可以看到，朴素贝叶斯模型在验证集上的得分为 0.86，说明模型的效果较不错。

```
# 预测准确率（给模型打分）
print('训练集：', NBmodel.score(x_train, y_train),
      '，验证集：', NBmodel.score(x_test, y_test))
```

```
训练集： 0.9240250183958794 ， 验证集： 0.8665236051502145
```

图 11 给模型打分

（c）评价分类模型效果

sklearn 中的 classification_report 函数用于显示主要分类指标的文本报告．在报告中显示每个类的精确度，召回率，F1 值等信息。输入的参数分别为 y_true 和 y_pred，此处使用 NBmodel.predict(x_test)得到模型预测的值。输出 support（真实值每个类别出现的次数）、precision（精度：预测对个数/预测值这个类别的总个数）、recall（召回率：预测对的个数/真实值的这个类别的总个数）以及 F1 值（精确度和召回率的调和平均值）。其中各分类的 micro average（微平均值，所有数据结果的平均值）、macro average（宏平均值，所有标签结果的平均值）以及 weighted average（加权平均值，所有标签结果的加权平均值）

```
from sklearn.metrics import classification_report
print(classification_report(y_test, NBmodel.predict(x_test)))
```

```
              precision    recall  f1-score   support

           0       0.84      0.73      0.78       772
           1       0.88      0.93      0.90      1558

   micro avg       0.87      0.87      0.87      2330
   macro avg       0.86      0.83      0.84      2330
weighted avg       0.87      0.87      0.86      2330
```

图 11 评价分类模型效果

（d）分类模型性能分析

精确度和召回率都高时，F1 值也会高，F1 值在 1 时达到最佳值（表示完美的精确度和召回率），最差为 0。在二元分类中，F1 值是测试准确度的量度。同时笔者使用 weighted average 作为模型评价。对各个模型分数和模型评价进行总结，如表 1 所示。可以看到朴素贝叶斯模型和 Logistic 回归模型的分类效果是较好的。

表 1 各分类模型评价

	模型分数		模型评价		
	训练集	验证集	精准率 precision	召回率 recall	f1-score
朴素贝叶斯	0.92	0.87	0.87	0.87	0.86
Logistic 回归	0.99	0.87	0.86	0.87	0.86
支持向量机	0.69	0.67	0.45	0.67	0.54
K 近邻	0.79	0.72	0.73	0.72	0.66
随机森林	0.99	0.81	0.80	0.81	0.80
决策树	1.00	0.76	0.76	0.76	0.76

2.7 实测模型效果

下面是两条从大众点评上复制下来的深大旁边汉普斯酒店的评论，分别使用各个分类模型进行预测。同样在预测前需要将文本向量化。如果预测正确，代表积极评论的 real_string 应该输出为 1，代表消极评论的 fake_string 应该输出为 0。

```
# pos
real_string = "深圳出差，之前经常住的酒店没有房间了，同事推荐了这一家，离公司也很近，酒店不错，服务态度也很好，房间楼层高，视野也很开阔，房间大小适中，床和枕头也很舒适，设施比较新，早餐总体也不错，来深圳出差酒店多了选择。"
# neg
fake_string = "现在已经下午 16:37 了，还没有人收拎房间，房间也明示需要清洁打扫。一个团的人，销售不知道统一安排到一个楼层，前台连授权电梯楼层都不会做，房间床很小。"
```

如图 12 所示，可以看到效果较不错，除了支持向量机将消极评论预测为积极评论外，其他分类模型都预测正确。

图 12 实测模型效果

进一步对比之前的评价结果，可以看到支持向量机在验证集的得分只有 0.67，f1 也只有 0.54，笔者进一步思考为什么其他算法的分类结果的都不错，维度支持向量机表现不好呢？可能有两个原因，一是数据样本量太小，特别是消极评论太少影响了特征提取；二是特征提取笔者在这里仅使用了最常用的文档-词频矩阵，存在较多信息损失，也许尝试 TF-IDF、LDA、词向量等方式提取出文档更多特征，可以使模型效果更好。由于本次实验重点在于算法上，笔者通过特征提取进行进一步探索。

2.8 训练词向量 word2vec

此处训练词向量的方法再之前 2.5 节：文档相似度（word2vec）已经进行了详细介绍，同样的步骤（1）清理文本（2）分词（3）使用 gensim 训练词向量模型，此时得到的时每个词的向量表示。但由于本节是希望将每一段文本都作为向量输入到分类模型，所以需要将这些词语的向量合并起来，笔者在这里用各个词向量直接平均的方式生成整句对应的向量，最后生成建模用的向量矩阵，大小为 7766 rows × 300 columns。（因为定义了向量参数 size=300）

```
# 用各个词向量直接平均的方式生成整句对应的向量
def m_avgvec(words, w2vmodel):
    return pd.DataFrame([w2vmodel.wv[w]
                         for w in words if w in w2vmodel.wv]).agg("mean")
```

```
# 生成建模用矩阵
%time train_vec = pd.DataFrame([m_avgvec(s, w2vmodel) for s in text_list])
train_vec.head()
```

```
CPU times: user 2min 18s, sys: 1.37 s, total: 2min 19s
Wall time: 2min 21s
```

	0	1	2	3	4	5	6	7	8	9	...
0	-0.044079	0.217596	-0.079045	0.190549	-0.055233	-0.071799	0.131174	-0.129751	-0.010107	-0.143796	...
1	0.026745	0.196646	-0.622026	0.168825	0.443178	0.033578	0.138856	-0.668566	-0.210110	-0.229128	...
2	-0.061685	0.189842	-0.179897	0.242424	0.111129	0.034167	0.176460	-0.136949	-0.139791	-0.032267	...
3	-0.008537	0.110301	-0.289733	0.069985	0.118345	-0.090926	0.040970	-0.272730	-0.046034	-0.028124	...
4	0.024302	0.196139	-0.410602	0.057936	-0.070377	-0.241467	0.255243	-0.285963	0.089983	0.075860	...

5 rows × 300 columns 使用了 head,只输出了前 5 行

图 13 训练词向量 word2vec

2.9 基于 word2vec 训练支持向量机

同样基于 sklearn 实现 SVM，步骤如下（1）将词向量需要转换为数组，不然后面训练的时候要报错（2）划分训练集和测试集 （3）初始化并训练模型，得到模型的评价如图 14 所示，可以看到基于 word2vec 的 SVM 在验证集的得分达到 0.81，f1 得分达到 0.81，有了大幅的提升，也验证了笔者对于特征提取不足的猜想。（基于文档-词频矩阵的 SVM：验证集得分只有 0.67，f1 也只有 0.54）

```
print('训练集准确率：', w2v_svm.score(x_train, y_train))
print('验证集准确率：', w2v_svm.score(x_test, y_test))
```

```
训练集准确率： 0.8362766740250184
验证集准确率： 0.8137339055793992
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, w2v_svm.predict(x_test)))
```

```
              precision    recall  f1-score   support

           0       0.76      0.64      0.69       773
           1       0.83      0.90      0.87      1557

   micro avg       0.81      0.81      0.81      2330
   macro avg       0.80      0.77      0.78      2330
weighted avg       0.81      0.81      0.81      2330
```

图 14 基于 word2vec 的 SVM 模型效果

2.10 基于 keras 内置的 Embedding layer 实现 LSTM[6][7]

长短期记忆（Long short-term memory, LSTM）是一种特殊的 RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。LSTM 与 RNN（Recurrent Neural Network，循环神经网络）的主要不同在于其中的神经元（感知机）的构造，RNN 的神经元与一般神经网络的感知机一样。而 LSTM 每个神经元则是一个“记忆细胞”，细胞里有“输入门”（input gate）、遗忘门”（forget gate）、“输出门”（output gate），LSTM 的结构图 15 如下。

[6] https://cloud.tencent.com/developer/article/1661253

[7] https://www.cnblogs.com/liuzhen1995/p/11625684.html

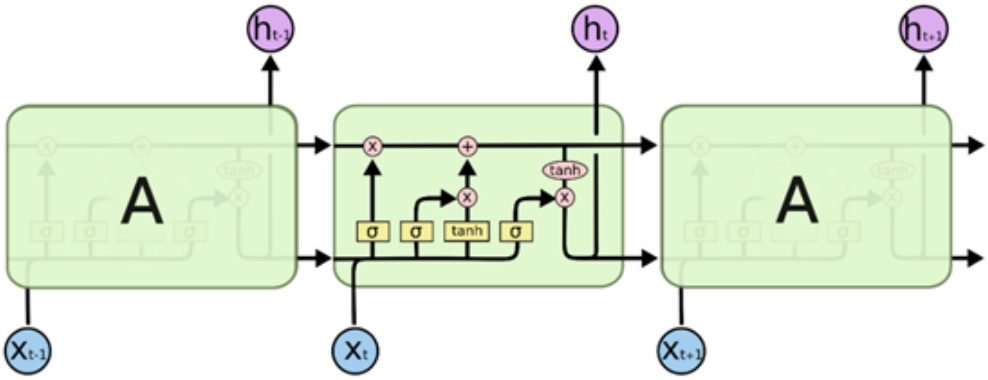


图 15 LSTM 的结构图

LSTM 的具体推导原理较为复杂，笔者不甚熟悉，所以在这里以应用为主，使用 keras 实现 LSTM 方法，keras 实际是以 Tensorflow 搭建的，可以较为方便地实现 LSTM。为了更好地理解 LSTM 的应用，我们先使用 keras 内置的 Embedding layer，该方法本质上是采用编码的方式将文本向量化。

（a）数据准备

首先将分词后的文本数据 text_list 划分为训练集和测试集，然后主要使用 Keras 的 Tokenizer 类进行数据处理，Tokenizer 是一个分词器，用于文本预处理，序列化，向量化等。

Step1 ：将数据张成 2D 的 tensor。调用 Tokenizer 函数对文本进行分词，然后调用 fit_on_texts，实现基于文本集 x_train_lstm 构建词汇表（在调用 texts_to_sequences 之前，必须先调用该方法构建词汇表）

Step2：数据序列化。因为 LSTM 是预测时间序列，即比如通过前 19 个数据去预测第 20 个数据，所有每次输入的数据也必须是一个滑动窗口，这里调用 texts_to_sequences 方法将 tokenizer 的结果生成序列，并且将序列统一为同一长度，为不丢失信息，设置最大 maxlen =100，如果不到 100，则填为 0。

Step3：将结果变量编码为 one-hot。这一步是为了满足 LSTM 的输出格式，因为在利用 categorical_crossentropy 作为损失函数时,需要将标签设定为 one-hot 格式, 即每个标签的长度应转换为一个长度为类别数的向量,该向量除了所属的类别位置为 1 之外,其他位置值为 0。

最后得到各数据的 shape 如图 16 所示：

```
#查看各数据的shape
print('训练集x_train:', x_train_seq.shape)
print('测试集x_test:', x_test_seq.shape)
print('训练集y_train:', y_train_seq.shape)
print('测试集y_test:', y_test_seq.shape)
```

```
训练集x_train: (5436, 100)
测试集x_test: (2330, 100)
训练集y_train: (5436, 2)
测试集y_test: (2330, 2)
```

图 16 各数据的 shape

（b）建立 LSTM 模型 (Embedding layer => LSTM layer => Dense layer)

models 是 Keras 神经网络的核心，这个对象代表所定义的神经网络：有层、激活函数等等属性和功能，这里使用 Sequetial 构建模型，这是 Keras 中的基本网络结构，表示将使用层堆叠起来的网络。

实例化模型后，需要添加三个层：输入层（Embeddign layer）、隐藏层（LSTM layer）和输出层（dense layer）。

- 输入层：使用 keras 内置的 Embedding，该方法本质上是采用编码的方式将文本向量化。

- 隐藏层：指定为 128 个神经元，但当神经元过多的时候，可能效果并不好，因为容易导致过拟合的现象，Dropout 是将上一层神经元进行随机丢弃，有助于解决过拟合的问题，这里笔者设置抛弃 20%的结果
- 输出层：实际上就是 Fully-connected layer，输出为分类结果。

还需要设置 Activation，因为是分类任务，笔者这里定义 softmax 为激活函数（对上一层产生的结果进行修改）

```
# 建立LSTM
lstm_model = Sequential()
lstm_model.add(Embedding(top_words, 128, dropout=0.2)) #设置Embedding layers
lstm_model.add(LSTM(128, dropout_W=0.2, dropout_U=0.2)) # LSTM层指定为128个神经元，# 抛弃20%的结果，防止过拟合

lstm_model.add(Dense(nb_classes))#定义输出层
lstm_model.add(Activation('softmax'))#定义softmax为激活函数，适用于分类任务
lstm_model.summary()
```

```
/Users/fubo/opt/anaconda3/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:3: UserWarning: The `dropout` a
rgument is no longer support in `Embedding`. You can apply a `keras.layers.SpatialDropout1D` layer right after the `E
mbedding` layer to get the same behavior.
  This is separate from the ipykernel package so we can avoid doing imports until
/Users/fubo/opt/anaconda3/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:4: UserWarning: Update your `LS
TM` call to the Keras 2 API: `LSTM(128, dropout=0.2, recurrent_dropout=0.2)`
  after removing the cwd from sys.path.
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, None, 128)	2560000
lstm_3 (LSTM)	(None, 128)	131584
dense_3 (Dense)	(None, 2)	258
activation_3 (Activation)	(None, 2)	0

```
Total params: 2,691,842
Trainable params: 2,691,842
Non-trainable params: 0
```

图 17 建立 LSTM 模型

（c）使用损失函数(交叉熵)和优化算法(adam)，编译并拟合 LSTM

跟机器学习算法一样，训练模型也是输入训练集 X、Y，epochs 为循环训练次数，batch_size 为单次训练的样本数。可以看到拟合了 10 次，最后损失函数降到了 0.0113，准确率达到了 0.9971，模型效果很好。

```
# 编译LSTM
lstm_model.compile(loss = 'binary_crossentropy',# 指定损失函数，交叉熵
                   optimizer = 'adam',
                   metrics = ['accuracy'])#收集附加性能指标，accuracy表示分类准确性
#拟合模型
model_fit = lstm_model.fit(x_train_seq, y_train_seq, batch_size = batch_size, nb_epoch = nb_epoch, verbose = 1)
```

```
/Users/fubo/opt/anaconda3/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:6: UserWarning: The `nb_epoch`
argument in `fit` has been renamed `epochs`.
```

```
Epoch 1/10
5436/5436 [==============================] - 21s 4ms/step - loss: 0.4534 - acc: 0.7842
Epoch 2/10
5436/5436 [==============================] - 20s 4ms/step - loss: 0.2281 - acc: 0.9137
Epoch 3/10
5436/5436 [==============================] - 22s 4ms/step - loss: 0.1359 - acc: 0.9544
Epoch 4/10
5436/5436 [==============================] - 22s 4ms/step - loss: 0.0735 - acc: 0.9772
Epoch 5/10
5436/5436 [==============================] - 20s 4ms/step - loss: 0.0554 - acc: 0.9834
Epoch 6/10
5436/5436 [==============================] - 22s 4ms/step - loss: 0.0297 - acc: 0.9910
Epoch 7/10
5436/5436 [==============================] - 21s 4ms/step - loss: 0.0260 - acc: 0.9926
Epoch 8/10
5436/5436 [==============================] - 21s 4ms/step - loss: 0.0180 - acc: 0.9947
Epoch 9/10
5436/5436 [==============================] - 21s 4ms/step - loss: 0.0198 - acc: 0.9936
Epoch 10/10
5436/5436 [==============================] - 20s 4ms/step - loss: 0.0113 - acc: 0.9971
```

图 18 基于 keras 内置的 Embedding layerLSTM 模型训练结果

模型使用的参数如下：

```
#设置参数
nb_epoch = 10  #epochs次数越多，模型训练效果越好，但所需时间也线性增加
batch_size = 32 #batch_size为分批量将数据用于训练，以减小计算资源的需求
top_words = 20000 #Tokenizer的最大长度
maxlen = 100 #统一序列的最大长度
nb_classes = 2 #one-hot编码长度
```

图 19 模型参数

2.11 基于 word2vec 实现 LSTM

接下来笔者将基于 word2vec 实现 LSTM 模型，该方法的本质是将 word2vec 作为 LSTM 的 Embeddign layer，其余过程和参数跟上述基于 keras 内置的 Embedding layer 实现 LSTM 大致一样。步骤如下：

Step 1： 加载训练好的 word2vec

Step 2： 使用 Embedding matrix 作为权重构建 Embedding layer

Step 3： 基于 Word2Vec embedding 建立 LSTM(embeddign layer => LSTM layer => dense layer)

Step 4： 使用损失函数(交叉熵)和优化算法(adam)，编译并拟合 LSTM 模型

其中，跟上一种方法构建 LSTM 不同之处的核心代码如下，主要是实现使用使用 Embedding matrix 作为权重构建 Embedding layer。

```
# 取出上面训练好的词向量
embedding_matrix = w2vmodel.wv.syn0
#使用 embedding_matrix 的长度作为 top_words
top_words_w2v = embedding_matrix.shape[0]
# 设置 Word2Vec 为 embedding layer
embedding_layer = Embedding(embedding_matrix.shape[0],
                            embedding_matrix.shape[1],
                            weights=[embedding_matrix])
```

从图 20 可以看到，模型拟合了 10 次，最后损失函数降到了 0.0184，准确率达到了 0.9941。

```
Epoch 1/10
5436/5436 [==============================] - 28s 5ms/step - loss: 0.5333 - acc: 0.7355
Epoch 2/10
5436/5436 [==============================] - 27s 5ms/step - loss: 0.3418 - acc: 0.8576
Epoch 3/10
5436/5436 [==============================] - 27s 5ms/step - loss: 0.2317 - acc: 0.9088
Epoch 4/10
5436/5436 [==============================] - 27s 5ms/step - loss: 0.1523 - acc: 0.9415
Epoch 5/10
5436/5436 [==============================] - 26s 5ms/step - loss: 0.1052 - acc: 0.9625
Epoch 6/10
5436/5436 [==============================] - 29s 5ms/step - loss: 0.0698 - acc: 0.9761
Epoch 7/10
5436/5436 [==============================] - 25s 5ms/step - loss: 0.0432 - acc: 0.9877
Epoch 8/10
5436/5436 [==============================] - 28s 5ms/step - loss: 0.0342 - acc: 0.9899
Epoch 9/10
5436/5436 [==============================] - 30s 6ms/step - loss: 0.0316 - acc: 0.9904
Epoch 10/10
5436/5436 [==============================] - 27s 5ms/step - loss: 0.0184 - acc: 0.9941

<keras.callbacks.History at 0x7fe8d5487940>
```

图 20 基于 word2vec LSTM 模型训练结果

2.12 结果可视化[8]

为了更直观的比较模型，此处分别比较了（编号 1）基于 word2vec 实现 SVM、（编号 2）基于 keras 内置的 Embedding layer、（编号 3）基于 word2vec 实现的 LSTM 三种模型的预测能力在各种曲线上的表现。

（a）Precision_Recall Curve

[8] https://blog.csdn.net/qq_39241986/article/details/103047151

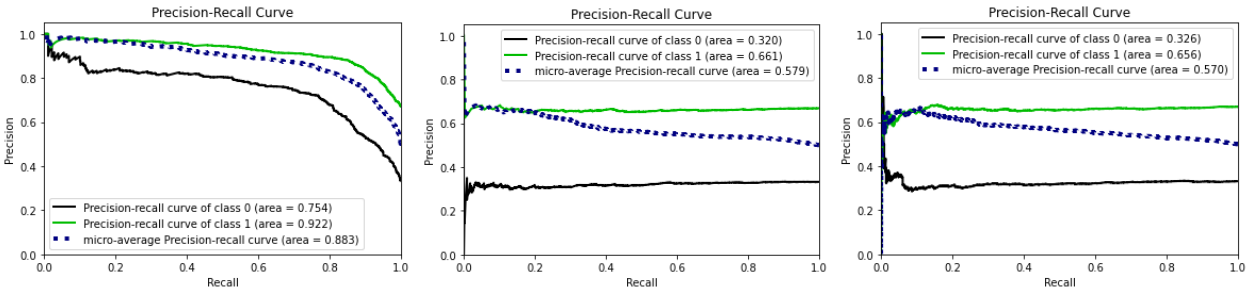


图 21 Precision_Recall Curve（左：svm，中：lstm，右：lstm_w2v）

（b）Life Curve

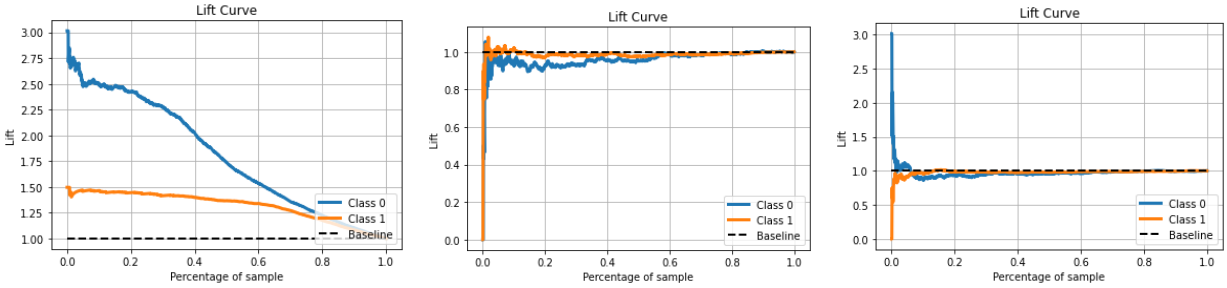


图 23 Life Curve（左：svm，中：lstm，右：lstm_w2v）

（c）ROC Curves

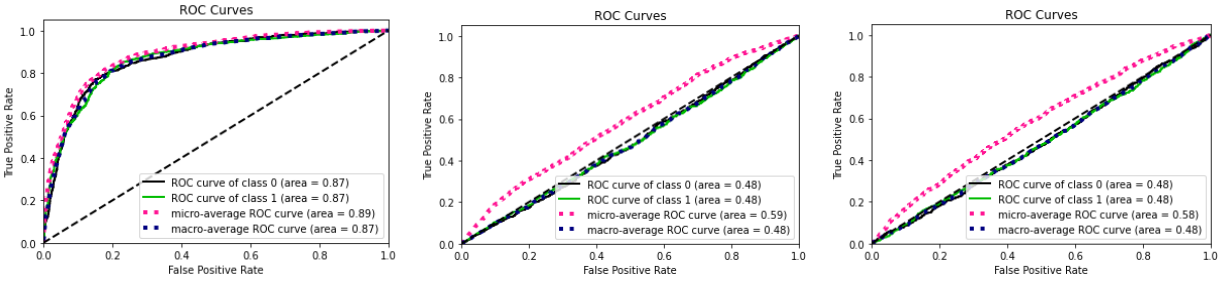


图 24 ROC Curves（左：svm，中：lstm，右：lstm_w2v）

（d）Cumulative Gains Curve

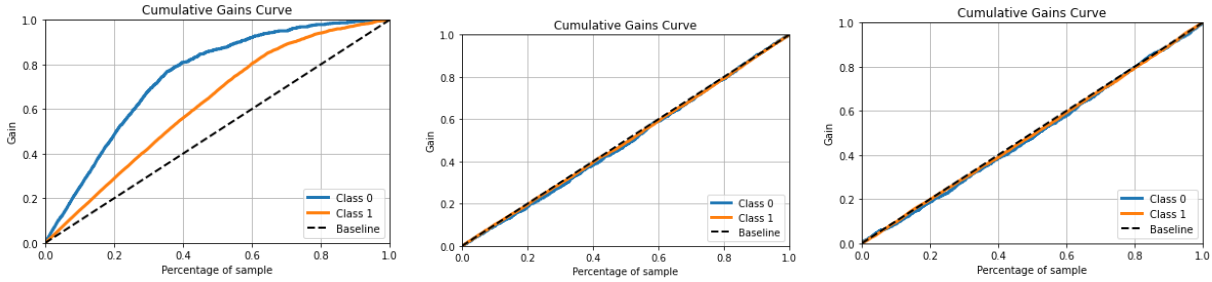


图 25 Cumulative Gains Curve（左：svm，中：lstm，右：lstm_w2v）

（e）Calibration Plots（Reliability Curve）

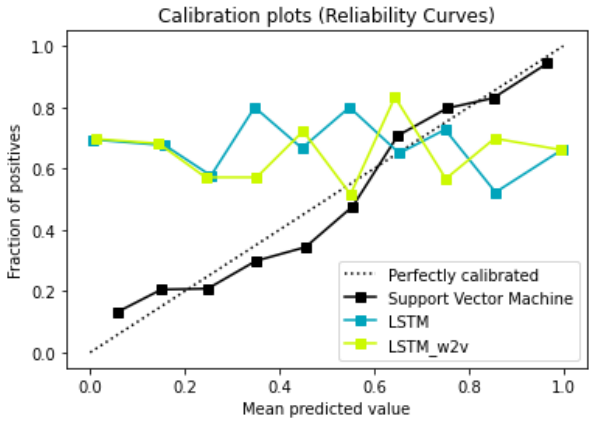


图 26 Calibration Plots

从上述可视化的结果可以看出，一般的 LSTM 模型与基于 wor2vec 实现的 LSTM 模型差异不大，这可能是由于 word2vec 其实也是一种神经网络，因此二者提取到的特征差异不大。但是很遗憾的是，很明显，两个 LSTM 模型都过拟合了，虽然在训练集和测试集上表现很好，但外部预测能力可能会存在不足，因此在可信度方面甚至还没有基于 word2vec 的 SVM 表现好。针对 LSTM 过拟合的问题，可以尝试提高 Dropout 处理的值。

三、进一步探索：研报分类预测

有了前文对 word2vec 和 LSTM 的探索后，笔者将进一步进行应用，笔者首先爬取了东方财富网的宏观研究报告，基于爬取的数据训练 word2vec 和 lstm 模型，再基于模型对研报进行情感分类，这里也从二分类问题扩展为三分类问题。本部分包括（1）爬虫 （2）文本处理 （3）word2vec 词向量训练 （4）Lstm 模型训练 （5）基于模型的新文本预测

3.1 爬虫 （reptile.py）

（1）爬虫环境：

笔者的安装环境是 Mac 下的 Anaconda，使用 Selenium 和 PhantomJS 来构建爬虫环境[9][10]。Selenium 是一个 Web 的自动化测试工具，可以根据指令让浏览器自动加载页面，获取需要的数据，甚至页面截屏，或者判断网站上某些动作是否发生。PhantomJS 是一个基于 Webkit 的“无界面”浏览器，它会把网站加载到内存并执行页面上的 JavaScript，因为不会展示图形界面，所以运行起来比完整的浏览器要高效，把 Selenium 和 PhantomJS 结合在一起，就可以运行一个非常强大的网络爬虫。

（2） __init__(self)函数

- filedir：存储爬取文件的路径
- phantomjs_path：Mac 下使用 PhantomJS 需要将此包放到文件路径下
- url：爬虫要爬取的原始页面（东方财富网宏观研报[11]）

（3）driverInit(self)函数：浏览器驱动初始化

Selenium 库里有个叫 WebDriver 的 API。WebDriver 类似可以加载网站的浏览器，但它可以像 BeautifulSoup 或者其他 Selector 对象一样用来查找页面元素，与页面上的元素进行交互（发送文本、

[9] https://www.cnblogs.com/mayi0312/p/7231242.html
[10] https://blog.csdn.net/lilong117194/article/details/83277075
[11] https://data.eastmoney.com/report/hgyj.html

点击等），以及执行其他动作来运行网络爬虫。

此处通过 webdriver.PhantomJS 调用了驱动，获取初始页面。同时因为页面的网址不变，所以只需第一次获取即可。

（4）get_page_num(self, num)：模拟鼠标点击下一页

- Step1:定位到表格下面页码选择区域，定位元素为："macresearch_table_pager",这里笔者使用的方法是显式等待，即指定查找定位元素，然后设置最长等待时间为 30s，如果在这个时间内，找到了，则继续往下执行；如果在这个时间，还未找到，便会抛出异常。
- Step2: 获取搜索页面的按钮"ipt",通过 find_element_by_class_name 方法实现。
- Step3: 填入页码，模拟点击鼠标。但是注意这里在通过 send_key 送入新页码前必须清空当前的搜索页码，再通过 click()方法模拟点击。
- Step4: 点击后还必须等待固定时长，这里设置为 10s

（5）download_report(self, text_link, re_sum_info)

此函数实现根据送入的 url 和标签，获取页面内容，然后判断报告是否是空页，如果非空，则下载此网页的研报文本，最后保存到本地

（6）get_report_page(self, page_start, page_end)

此函数以起始和终止页面数为爬取标准,两个参数分别是要爬取的页面：起始页面，终止页面。每次获取指定页面，然后获取所有的网页 tr 标签，"tr"，再遍历存储内容的 td 标签，获得：研报序号、报告名称、作者、机构名称、近一月机构宏观研报数量、日期，最后根据送入的 url 和标签，爬取下载研报正文。爬取完毕后，要记得使用 driver.quit()关闭驱动

（7）main():

主函数，创建类 ReportReptile 的类对象，report_obj，并使用 report_obj.get_report_page(1, 2)爬取制定页码的研报数据。

这里以爬取 1-2 页数据为例进行测试，爬取的研报数据（1-2 页）存储在 reptile_files3 这个文件夹，如图 27 所示，可以看到程序正常运行，程序运行结束后，回到文件夹，可以看到爬取保存到本地的研报数据正常（图 28 所示）

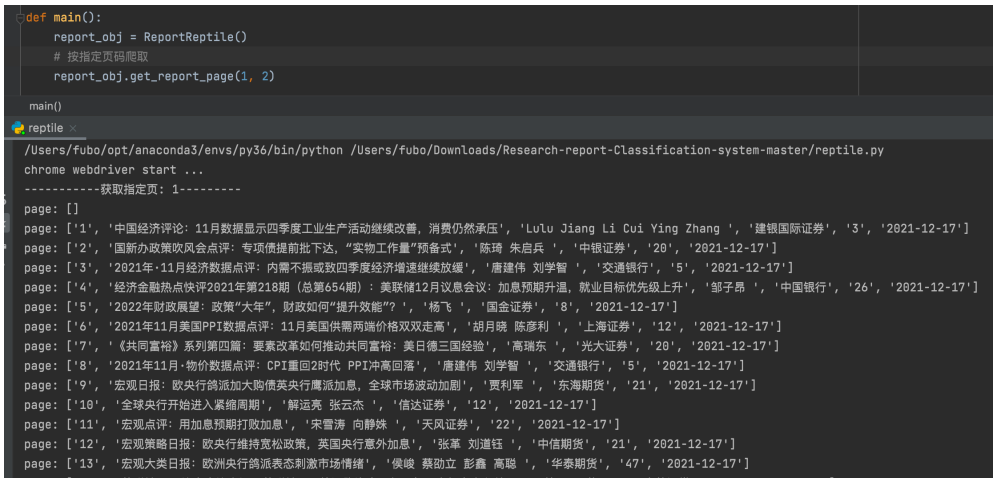


图 27 爬虫程序正常运行

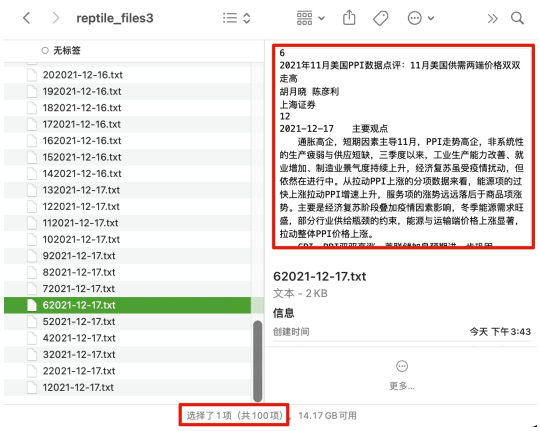


图 28 爬虫保存到本地的文件正常

最终笔者让爬虫爬取了从 2019-12-17 到 2021-12-10 的总共 15397 份宏观研报（如图 29 所示），将用于后续模型的训练和预测，也进一步验证了词爬虫程序的稳健性。

图 29 总共爬取了 15397 份宏观研报

3.2 编写 interface.py：接口文件

由于模型参数等需要不断测试以及对于分词等函数需要被多次调用，所以将参数设置以及一些函数封装成接口文件，即其他文件会调用该文件的函数。

（1） init(self) 大部分的参数调整都在这里

（2）tokenizer(self,text) 对文本分词并去掉空格（注意这里同样需要将文本转化为 list）

（3）load_w2v_file(self,w2v_file_path) 加载训练文件(spider_report 文件夹的 6000 份研报)

（4）text_proce(self,text_raw) 对文本进行预处理并重新写入文件

3.3 词向量训练：word2vec.py 文件

词向量模型上述已经详细介绍，本节的训练过程跟上述思路一致，只是这里进行将训练模型及其词典、向量以 npz、npy、pkl 格式进行了更详细的保存，以供后续进一步读取使用。这些格式文件都是 python 生成的数据文件，一般用于保存训练好的模型文件，使用 np.load()和 np.save()这两个读写磁盘数组数据的函数即可实现读取和保存的过程，这些模型文件最后保存在 word2vec_model 文件夹。这里笔者从 15000 多份爬取的宏观研报中抽取了 6000 份作为词向量模型的训练样本（存储在 spider_report 文件夹）。词向量模型的参数设置已经在接口文件（interface.py）中进行了定义，如图 30 所示：

```
self.filedir_train = "./spider_report/"      # 训练数据集
self.filedir_test = "./test_report/"         # 测试数据集（用于预测）

# word2vec训练时的参数设置
self.vocab_dim = 300                                  # 每个词向量的维度
self.n_exposures = 0                                  # 词频大于0的都会被向量化
self.window_size = 7                                  # 窗口大小
self.cpu_count = multiprocessing.cpu_count()          #多进程模块，统计cpu核数
self.n_iterations = 1
self.w2v_file_path='./spider_report/'                    # 用于训练词向量的文本
self.wordsList='./word2vec_model/wordsList.npy'          # 词典
self.wordVectors='./word2vec_model/wordVectors.npy'   # 词向量
self.Word2vec_model='./word2vec_model/Word2vec_model.pkl'   # word2vec模型保存
self.word_index='./word2vec_model/word_index.npz'     # {词：索引}
self.word_vec='./word2vec_model/word_vec.npz'         # {词：词向量}
```

图 30 word2vec 训练时的参数设置

在用于词向量训练的 word2vec.py 文件中，主要包含以下函数

（1） create_dictionaries(self,model=None,combined=None)：创建词语字典，并返回每个词语的索引，词向量，以及每个句子所对应的词语索引

（2） word2vec_train(self,combined)：词向量训练

（3） train(self)：词向量训练模型封装，包括加载训练文件、分词处理、进行训练三个过程（前两个过程调用了接口文件的 load_w2v_file 和 tokenizer 函数）

（4） test(self)：该文件仅做测试使用

运行 word2vec.py 文件，完成了词向量模型的训练，打开 word2vec_model 文件夹，可以看到模型文件已经生成（如图 31 所示）。将模型保存到本地的另外一个好处是，以后如果还需要使用到关于宏观经济的文本分析，可以直接调用这个已经处理好的模型。

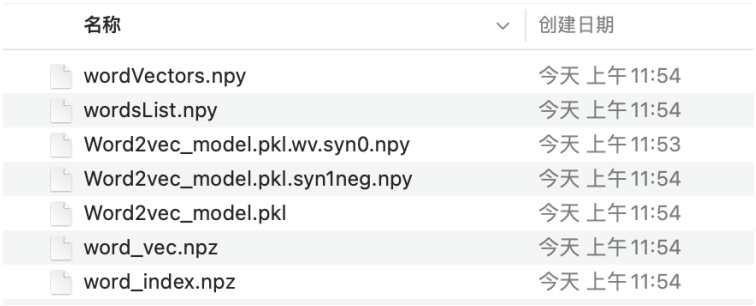


图 31 word2vec_model 文件夹保存的模型数据

接下来笔者对模型进行了一些测试和探索（图 32），以一步加深对词向量模型的理解。

```
def test(self):
    w2indx=np.load(self.word_index)                #加载word_index.npz文件：{词：索引}
    print('w2indx:',w2indx)                        #打印词文件的格式
    w2indx=w2indx['dic'][()]                       #注意必须这种形式读取保存的字典
    print('历史 词索引:',w2indx['历史'])             #打印"历史"这个词在字典的索引
    w2vec=np.load(self.word_vec)                   #加载word_vec.npz文件：{词：词向量}
    print('历史 词向量:',w2vec['dic'][()]['历史']) #读取字典，并打印"历史"这个词的词向量
    wl=np.load(self.wordsList)                     #加载wordsList.npy文件：整个词典
    wl = wl.tolist()                               #也可以以list方式进行读取
    print('词典长度:',len(wl))                      #打印词典list的长度
    #wl = [word.decode('UTF-8')  for word in wl]
    print('期货 词索引:',wl.index('期货'))          #打印期货这个词的索引
    vec=np.load(self.wordVectors)                  #加载wordVectors.npy文件：词向量
    print('词向量长度:',len(vec))                   #打印词向量的长度
    print('第52个词向量:',vec[52])                  #打印第52个词向量
```

图 32 词向量模型测试和探索

测试结果如图 33 所示，笔者已经在注释中进行了详细解释，这里就不再赘述了，这一步看似不重要，但其实如果没有通过这种方式理解好词向量，后续基于词向量构建 LSTM 的 Embedding layer 可能就会出错。

```
word2vec
w2indx: <numpy.lib.npyio.NpzFile object at 0x7f9a21634eb8>
历史 词索引: 28567
历史 词向量: [ 0.76265883 -1.0167947  -0.16652788  0.0785358   0.09174183 -0.25888172
  0.49472123 -0.30813876  0.86362416  0.3464248   0.51972055 -0.19954172
 -0.3510228   0.57495433 -0.0589149   0.39413878  0.04135471  0.31798404
 -0.65226424  0.0464468  -0.645442    0.4220275  -0.08725882 -0.03013898
  0.31150612  0.08778483 -0.13948146  0.07311103 -0.4834107   0.5516148
```

```
词典长度: 52938
期货 词索引: 39380
词向量长度: 52938
第52个词向量: [ 2.79148389e-02 -3.14442217e-02 -9.92166996e-03 -4.56149224e-03
  2.17159819e-02 -1.69127807e-02  3.14552821e-02 -1.62443575e-02
  2.87620872e-02 -3.18282284e-04  1.45910857e-02 -9.33253113e-03
```

图 33 词向量的测试结果

3.4 LSTM 模型训练：lstm.py 文件

LSTM 的原理笔者已经在之前进行了详细介绍，但上一节笔者是使用已经分好类的数据集，再使用 LSTM 算法进行二分类。而本节笔者尝试使用预先分完类的研报样本对 LSTM 模型进行训练，再基于 LSTM 模型进行研报分类预测。这里笔者进一步将分类扩展为三分类问题，包括代表积极情绪的 pos，代表中性情绪的 neu，代表消极情绪的 neg，笔者首先通过人工判别的方式分别从爬取的研报中为每个情感分类找到 20 份研报作为训练集，并全部按照序号命名，存放在 lstm_train_data 文件夹，之下有 pos、neu、neg 文件夹分别存放对应的研报。

同样，这里的难点是进行数据准备，首先对每个文本进行预处理，然后将代表不同情感的三个文件夹及其对应的标签进行拼接，得到的变量 combined 表示文本的拼接，y 表示对应标签的拼接。再构建函数来获取每篇文本在词向量词典中的索引列表，注意这里由于不同的文本长度不同，所以列表长度也不同，基于得到每篇文本所含的词语对应的索引，以后端截断并且填 0 补充的方式，使用 sequence 方法将训练数据结构（序列）化。接着还需要将文本构建为模型用的向量矩阵（变量为 embedding_weights），

方式为：索引为 0 的词语，词向量全为 0，从索引为 1 的词语开始，每个词语对应其词向量形成词向量矩阵；再处理标签数据，分为两步，一是将之前定义的分类标签-1,0,1 转化为 0,1,2，再同样使用 to_categorical 方法将标签转换为 one-hot 编码（-1=0=[1. 0. 0]; 0=1=[0. 1. 0.]; 1=2=[0. 0. 1.]）。这样就得到了用于 lstm 网络结构的数据了。

（1） splice_data(self,path)：初步处理训练数据

（2） load_train_file(self)：拼接训练文件和文件的标签

（3） parse_dataset(self,combined)：得到每篇文本在词典中的索引列表

（4） train_data_struc(self,combined)：lstm 模型训练数据的结构（序列）化

（5）get_train_data(self,word_index,word_vectors,struc_w2index,y)：该函数得到的结果才是用于 lstm 网络结构的数据

a) index_dict:所有的词索引列表(词：索引)

b) word_vectors:所有词的词向量

c) combined:所有文本的索引值

（6） train_lstm：网络训练函数，网络的参数也是在这里调试

LSTM 模型的训练都在 train_lstm 函数(图 35)，同样是实例化模型后，添加三个层：输入层（Embeddign layer）、隐藏层（LSTM layer）和输出层（dense layer），由于这里是三分类问题，使用的激活函数是 relu，且由于前文笔者将 dropout 参数设置为 0.2，结果仍然过拟合，所以笔者在这里设置为 0.5。在输出层外接 softmax，进行最后的 3 分类。然后编译、拟合和评估，方法同前文所示。

```
# 定义网络结构
def train_lstm(self,n_symbols,embedding_weights,x_train,y_train,x_test,y_test):
    nb_classes=3
    print ('Defining a Simple Keras Model...')
    # 实例化模型
    model = Sequential()
    # 输入层 (Embeddign layer)
    model.add(Embedding(output_dim=self.vocab_dim,# 每个词的词向量维度
                        input_dim=n_symbols,# 所有的词的长度加1
                        mask_zero=True,  # 确定是否将输入中的‘0’看作是应该被忽略的‘填充’（padding）值
                        weights=[embedding_weights],# 词向量矩阵
                        input_length=self.input_length))  # 当输入序列的长度固定时，该值为其长度
    # 隐藏层 (LSTM layer) / 三分类
    # 使用单层LSTM 输出的向量维度是50，输入的向量维度是vocab_dim,激活函数relu
    model.add(LSTM(output_dim=50, activation='relu', inner_activation='hard_sigmoid'))
    model.add(Dropout(0.5)) #前文设置为0.2仍然过拟合，此处dropout设置为0.5
    # 输出层 (dense layer)
    #在这里外接softmax，进行最后的3分类
    model.add(Dense(output_dim=nb_classes, input_dim=50, activation='softmax'))
```

图 25 ：模型训练 train_lstm 函数

需要注意的是，笔者在这里对训练的 lstm 模型结果同样进行了保存（在 lstm_model 文件夹），将模型保存为 yml 格式，将向量权重矩阵保存为 h5 文件，这些文件包含模型体系结构和权重，还包括所选损失和优化算法的规范，以便可以读取模型。模型的参数也已经在接口文件进行了设置，如图 36 所示。

```
# lstm的参数设置
self.batch_size = 2       # 每次batch大小
self.n_epoch = 5          # 训练轮数
self.input_length = 550   # 每篇文本的长度固定时，这个值和self.maxlen相等
self.maxlen = 550         # 设定每篇文档的最大长度
self.test_size=0.2        # 测试文本的比例
self.pos_path='./lstm_train_data/pos/'       #积极训练集
self.neg_path='./lstm_train_data/neg/'       #消极训练集
self.neu_path='./lstm_train_data/neu/'       #中性训练集
self.lstm_model='./lstm_model/lstm.yml'      #lstm模型保存
self.lstm_weight='./lstm_model/lstm.h5'
```

图 36 LSTM 模型参数设置

运行文件后，可以看到运行成功，lstm_model 文件夹下也成功保存了 lstm.yml 和 lstm.h5 文件，但是从图 37 可以看到模型在测试集上的准确率只有 0.417，这是因为时间原因，笔者无法花太多时间人工打标签，故总共只有 60 个训练样本进行 lstm 模型训练，但训练数据的质量和数量对于深度学习都是非常重要的，如何获取高质量高数量的训练样本，也就成了本次实验的难题，之后如果有时间，笔者可以使用交叉验证的方式标记更多的训练样本标签，再获取更多的样本后再进行模型训练。但方法笔者已解释清晰，通过花时间标记样本标签进行训练，模型的准确率应该可以提高的。

```
mm=Lstm_nn()
mm.train()
```

```
Lstm_nn › train()
lstm
 44/48 [==========================>...] - ETA: 1s - loss: 9.5243 - acc: 0.4091
 46/48 [==========================>..] - ETA: 0s - loss: 9.8110 - acc: 0.3913
 48/48 [==============================] - 14s 290ms/step - loss: 9.7380 - acc: 0.3958 - val_loss: 9.4022 - val_acc: 0.4167
Evaluate...

  2/12 [====>.........................] - ETA: 0s
  4/12 [=========>....................] - ETA: 0s
  6/12 [==============>...............] - ETA: 0s
  8/12 [===================>..........] - ETA: 0s
 10/12 [========================>.....] - ETA: 0s
 12/12 [==============================] - 0s 32ms/step
Test score: [9.402222335338593, 0.4166666666666667]
```

图 37 lstm 模型训练结果

3.5 新文本预测：main_test.py

此文件是总的调用，提供了词向量训练、lstm 模型训练的接口，同时使用此文件对新文本进行预测。函数 get_cla_rep(self)用来对新的研报进行分类预测，首先加载训练好的 lstm 模型，然后对开始处理要预测的文本，包括遍历测试文件（放在 test_report 文件夹下），调用模型预测，然后输出预测研报的分类结果等。从图 38 设置的提示中可以看到此文件运行成功。

```
Using TensorFlow backend.
loading trained model......
/Users/fubo/Downloads/Research-report-Classification-system-master/main_test.py:37: YAMLLoadWarning: calli
  yaml_string = yaml.load(f)
/Users/fubo/opt/anaconda3/envs/py36/lib/python3.6/site-packages/keras/engine/saving.py:350: YAMLLoadWarnin
  config = yaml.load(yaml_string)
loading weights......
2021-12-18 20:42:24.871235: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instruc
Building prefix dict from the default dictionary ...
Loading model from cache /var/folders/kk/v0bv1b8n67qd9371j5yqxgf00000gn/T/jieba.cache
Loading model cost 0.757 seconds.
Prefix dict has been built succesfully.
```

图 38 新文本预测成功

从爬取的不在训练样本之内的文件中随机抽取 15 个文件进行预测，可以看到预测结果如图 39 所示，即使模型由于训练样本的原因准确率并不高，但是此时分类效果还可以。

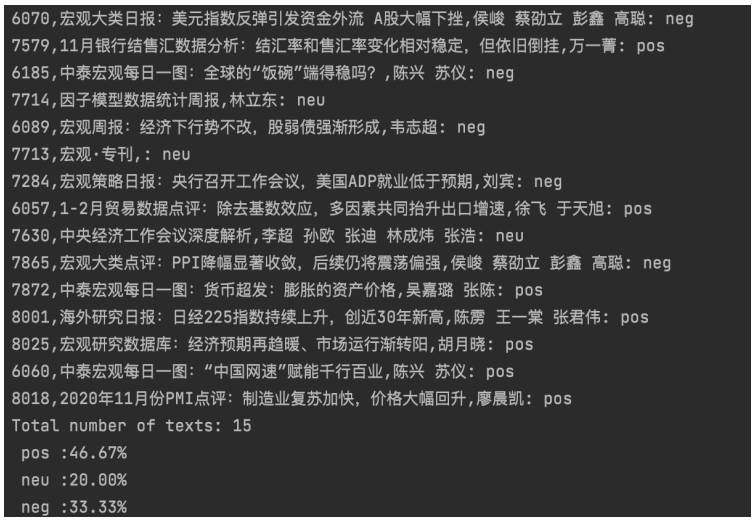


图 39 新文本预测结果

四、实验结论或体会

在本次实验中，笔者基于对文本分析和人工智能的兴趣，自行探索了很多问题，也从中学习到了很多知识，除了加深对机器学习算法的应用，也对深度学习、爬虫、文件操作有了进一步的认识，同时对于函数封装等代码书写规范性问题也有了很大的进步。这个过程中花费了很多时间，最近三周几乎每周这个工作都成了花费时间最多的工作，由于笔者本身的代码基础并不是很好，对于深度学习的应用也是第一次尝试，在所完成的工作中，从运行环境到文本编码到数据类型，中间踩了很多坑，每次都花费了一些时间才找到 bug，所以实验报告的也写的比较详细。但是比较遗憾的是，最后的模型并不能直接使用，因为对于宏观研报的分类可能需要一些专家来帮忙，构建好分类规则，再标签好训练样本，笔者相信通过这一步，所构建的模型可以取得较好的效果，学术界的很多研究已经证实了这一点，期末考试完成后笔者会进行进一步的探索。本学期即将结束，人工智能课程是笔者本学期花费了很多时间的课程，也确实收获很大，在此感谢特别 nice 的高老师的付出，您辛苦了💦。