|  |  |
| --- | --- |
| 参赛编号： | CDA03288 |
| 赛题题号： | B |

餐饮服务评价情感倾向分析

摘 要

随着科技的发展，以及生活的进步，人们的生活质量正在逐渐升高，餐饮行业也正在中国市场迅速的发展，餐饮外卖市场逐渐成熟。本文对餐饮服务评价情感倾向浸信分析，研究了所给商家的评论信息，找寻商家的优缺点，对于不足之处提出相关策略。最后选取使用深度学习神经网络模型对问题进行建模解决。

针对问题一，首先读取相关数据，进行数据预处理，查看数据的分布特征，判断是否含有缺失值，对comment特征进行分词操作，并去除相关停用词，统计相关词的数量，进行词云展示，以target特征作为基准，分别统计顾客积极情绪和消极情绪评价数量最多的前10个词，最后对词频进行可视化展示。

针对问题二，首先提取所给数据中的时间特征，如月、日、星期、小时等时间特征。接下来对一天24小时进行分析，利用分组聚合操作求得每小时的平均评价数量，并进行可视化展示，得出在1点-4点、9点-16点17点-20点这三个时间段内负面情绪数量略高于正面情绪数量，其他时间基本持平。对一月31天进行分析，统计每天的平均正负面情绪数量，进行可视化展示，得出正负面情绪评论呈现周期性变化，每隔3天或4天正负面情绪数量就会发证一次波动。最后对星期进行分析，统计周一到周日每天的平均正负面情绪数量变化，进行可视化展示，得出周一周二正负面情绪比值较为稳定，周三到周五顾客负面情绪占比逐渐减少，以至于周五的正面情绪数量会大于负面情绪数量，周六周日两天正负面情绪比值较为稳定。

针对问题三，首先求得积极情绪最多的商家id为1041，接下来选取1041商家的数据进行分析，使用TFIDF算法进行关键词提取，并进行权重的计算，选取权重最大的前10个词作为商家的优点进行分析，并将关键词进行可视化便于更好的分析。

针对问题四，首先求得积极情绪最多的商家id为971，然后求得该商家好评率仅为12%，选取该商家数据使用TFIDF提取关键词进行分析，得出该商家应该在出餐速度、餐具分配、份量这三个方面可以进行改进，能够提高顾客的积极情绪。

针对问题五，首先对文本进行预处理，选择comment的最大长度为80，大于80的进行截断，小于80的进行填充，通过预训练模型对文本进行文本向量化，输入到神经网络模型中进行训练，计算其Precision、Recall、F1-score等相关评估函数，最终模型的准确率收敛在97.2%左右。最后进行测试数据test.xlsx的预测并保存结果。

**关键词:** 预处理；可视化；关键词提取；文本向量化；神经网络

目录

目录

[1 数据探索性分析 1](#_Toc123757925)

[1.1读取数据 1](#_Toc123757926)

[1.2 数据的基本信息 1](#_Toc123757927)

[1.3 标签的主要分布 1](#_Toc123757928)

[1.4 文本长度的分布 2](#_Toc123757929)

[2 问题一制作词云和查找评论前10词 4](#_Toc123757930)

[2.1 文本分词和去除停用词 4](#_Toc123757931)

[2.2 展示词云 4](#_Toc123757932)

[2.3 积极情绪评论最多的10个词 6](#_Toc123757933)

[2.4 消极情绪评论最多的10个词 7](#_Toc123757934)

[3 问题二分析用户评论情绪与时间的关系 8](#_Toc123757935)

[3.1 从数据中提取时间信息 8](#_Toc123757936)

[3.2 分析每小时和情绪之间的关系 8](#_Toc123757937)

[3.3 分析每天和情绪之间的关系 10](#_Toc123757938)

[3.4 分析星期和情绪之间的关系 11](#_Toc123757939)

[4 问题三分析积极情绪商家及其优点 13](#_Toc123757940)

[4.1 分析积极情绪最多的商家 13](#_Toc123757941)

[4.2 分析积极情绪最多的商家 13](#_Toc123757942)

[5 问题四分析消极情绪商家以及改进策略 15](#_Toc123757943)

[5.1 分析消极情绪最多的商家 15](#_Toc123757944)

[5.2 分析消极情绪最多的商家 16](#_Toc123757945)

[5.3 改进策略 17](#_Toc123757946)

[6 问题五模型的建立和评估测试 18](#_Toc123757947)

[6.1 建立餐饮评论情感倾向模型 18](#_Toc123757948)

[6.2 情感倾向模型的训练和评估 21](#_Toc123757949)

[6.3 对附件test.xlsx进行预测 24](#_Toc123757950)

[7 附录 25](#_Toc123757951)

# 1 数据探索性分析

## 1.1读取数据

在进行数据分析之前，首先要进行的就是数据探索性分析(Exploratory Data Analysis)，了解本赛题的主要任务，以及数据的分布，从而对数据整体有个基本的理解，首先读取所给数据，如代码1所示。

**代码1 读取文件**

# 导入所需要的包

**import** pandas as pd

# 读取数据

train **=** pd.read\_excel('./data/data.xlsx')

test **=** pd.read\_excel('./data/test.xlsx')

data\_数据说明 **=** pd.read\_excel('./data/字段说明.xlsx')

## 1.2 数据的基本信息

查看数据的形状，可以看出训练集中含有17953条数据和5个特征，而测试集只含有1500条数据和5个特征。如代码2所示。

**代码2 查看数据的形状**

print("train.shape,test.shape",train.shape,test.shape)

接下来需要查看一些所给数据中是否包含缺失值，通过观察，发现训练集没有缺失值，其中测试集中target特征是空值，需要后期进行预测。如代码3所示。

**代码3 查看缺失值**

# 查看数据是否有缺失值

train.isnull().sum(),test.isnull().sum()

## 1.3 标签的主要分布

在进行NLP文本任务前，首先要查看标签的分布，标签分布对结果有着重要的影响。如代码4所示。

**代码4 查看标签分布**

# 设置画图所需要的参数

sns.set(style**=**'whitegrid', palette**=**'muted', font\_scale**=**1.2)

HAPPY\_COLORS\_PALETTE **=** ["#01BEFE", "#FFDD00", "#FF7D00", "#FF006D", "#ADFF02", "#8F00FF"]

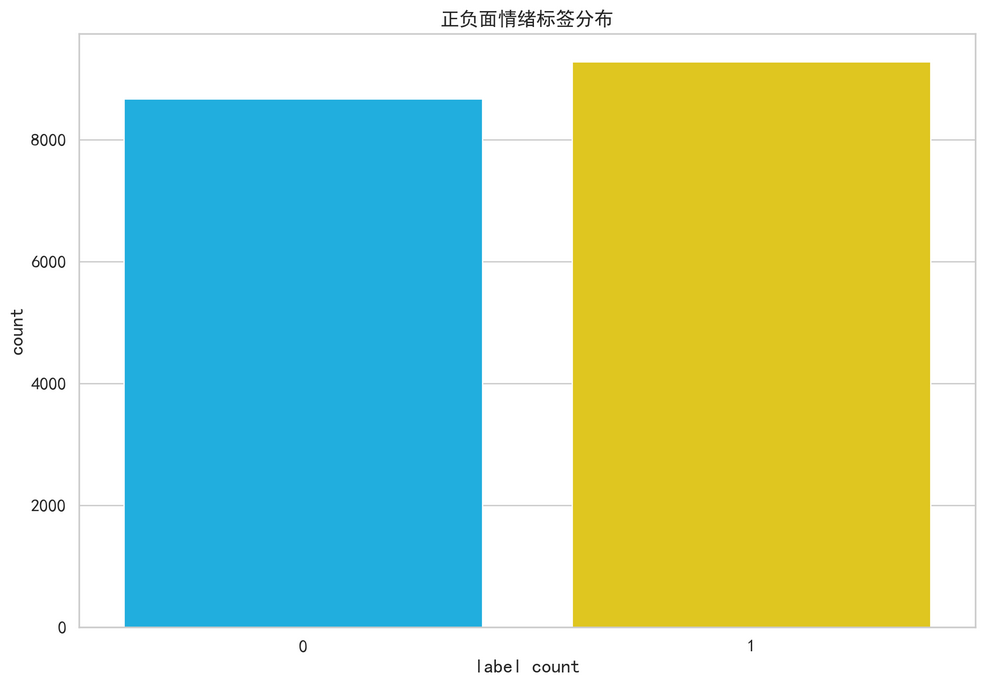
sns.set\_palette(sns.color\_palette(HAPPY\_COLORS\_PALETTE))

rcParams['figure.figsize'] **=** 12, 8

# 开始画图

sns.countplot(x**=**"target", data**=**train)

plt.xlabel('label count')



**图1 标签的分布**

在17953条数据中，可以看出，标签为1(负面评价)数量略多于标签为0(积极评价)，其中负面情绪为9281条，正面情绪为8672条，分布较为均匀。

## 1.4 文本长度的分布

在训练集中可以看出共包含17953条数据，接下来需要分析每条数据的文本长度，分析文本长度是为了理解评论包含的信息，对评论有个大致理解。如代码5所示。

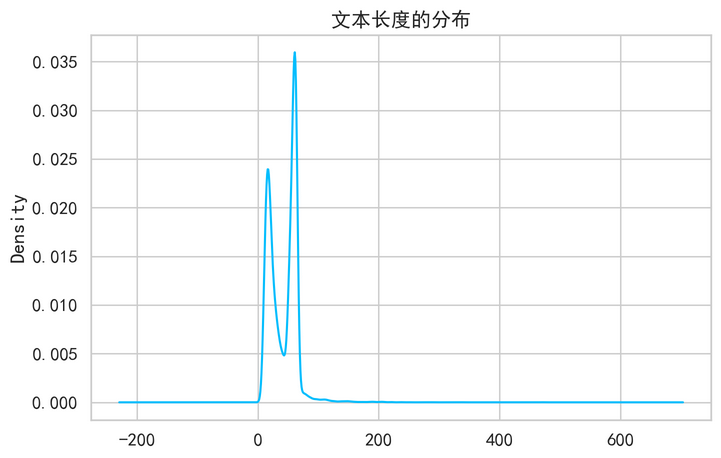
**代码5 查看评论文本长度分布**

plt.figure(figsize**=**(8,5))

plt.title('文本长度的分布')

plt.grid()

train['text\_len'].plot(kind**=**'kde')



**图2 文本长度整体分布**

由上图可以看出，每条数据的评论长度绝大多数分布在[0,100]之间，有个别的评论长度偏大，导致图形过于集中，接下来需找出评论长度偏大的文本。如代码6所示。

**代码6 分析文本过长的数量**

sum(train['text\_len']>100) # comment文本长度大于100的个数

sum(train['text\_len']>200) # comment文本长度大于200的个数

结果如表1所示，发现有个别的comment长度还是偏大的，在后续问题中将会对comment长度进行处理。

**表1 较长文本数量分布**

|  |  |
| --- | --- |
| 评论长度 | 数量 |
| 大于100 | 208 |
| 大于200 | 21 |
| 大于300 | 2 |
| 大于400 | 1 |

# 2 问题一制作词云和查找评论前10词

## 2.1 文本分词和去除停用词

首先将所给的评论内容进行拼接，这样便于分词操作，可以发现在大多数评论中都含有text字符，可以没有情感上的含义，可以将text字符进行删除，再将所有文本进行拼接，便于求词频。如代码7所示。

**代码7 拼接文本**

text **=** ''

**for** i **in** train['comment'].tolist():

    text**+=**i.replace('text','')

接下来进行分词和去除停用词操作，分词使用的是jieba函数进行分词，分词之后仍会含有一些无法表达情感含义的词，可以使用去除停用词操作过滤掉某些无用词。如代码8所示。

**代码8 分词和停用词**

words **=** jieba.lcut(text)

stop **=** pd.read\_csv('./stoplist.txt', header**=**None, encoding**=**'utf-8',engine**=** 'python' , sep**=**'limh')

stop **=** [' ', '　'] **+** list(stop[0])

stop **=** set(stop)

# 去停用词

words **=** [word **for** word **in** words **if** word **not** **in** stop]

word\_num **=** pd.DataFrame(words, columns**=**['word'])

word\_num **=** word\_num[word\_num['word'] !**=** '\n']

word\_num['count'] **=** 1

word\_num **=** word\_num.groupby('word').sum()

word\_many **=** word\_num[word\_num['count'] > 50]

## 2.2 展示词云

上述问题中已经进行了分词和停用词操作，下面可以根据词频进行词云的展示，首先需要找到一张背景图片作为词云的图床，根据所计算的词频画出词频图。如代码9所示。

**代码9 词云的展示**

back\_pic **=** imread("E:\\222.png")  # aixin.jpg # 设置背景图片

wc **=** WordCloud(font\_path**=**'C:\\Windows\\Fonts\\simkai.TTF',  # 设置字体 使用的 windows 自带的字体

               background\_color**=**"white",  # ="white", #背景颜色

               max\_words**=**2000,  # 词云显示的最大数

               mask**=**back\_pic,  # 设置背景图片

               max\_font\_size**=**200, random\_state**=**42, )

# 生成词云

wc.fit\_words(word\_many['count'])

# 绘图

plt.figure(figsize**=**(30, 15))

plt.imshow(wc)

plt.axis('off')



**图3 词云图**

上图为词云图的绘制结果，上图可以看出人们关心的主要是：味道、环境、服务和价格，充分体现了餐饮外卖市场的几个重要因素。

## 2.3 积极情绪评论最多的10个词

本题为寻找积极评价最多的前10词，通过上图的词云分析图可以看出，其中包含了很多无法表示情绪的词，例如：吃、真的、东西等相关词，无法表达情感思想，故可以在停用词操作过滤相关无用词。同理可得出词频，然后通过可视化操作展示词频的数量分布。如代码10所示。

**代码10 积极情绪前10个词**

## 定义词频函数

**def** count\_word(train):

    text **=** ''

**for** i **in** train['comment'].tolist():

        text**+=**i.replace('text','')

    # 分词、去停用词

    words **=** jieba.lcut(text)

    stop **=** pd.read\_csv('./stoplist.txt', header**=**None, encoding**=**'utf-8',engine**=** 'python' , sep**=**'limh')

    b **=** stop.drop\_duplicates()

    stop **=** [' ', '　','不错','好吃','难吃','喜欢','差','吃','东西','太','菜','点','送','饭','慢','店'] **+** list(stop[0])

    stop **=** set(stop)

    # 去停用词

    words **=** [word **for** word **in** words **if** word **not** **in** stop]

    word\_num **=** pd.DataFrame(words, columns**=**['word'])

    word\_num **=** word\_num[word\_num['word'] !**=** '\n']

    word\_num['count'] **=** 1

    word\_num **=** word\_num.groupby('word').sum()

    word\_many **=** word\_num[word\_num['count'] > 50]

**return** word\_num.sort\_values(by**=**'count',ascending**=**False)

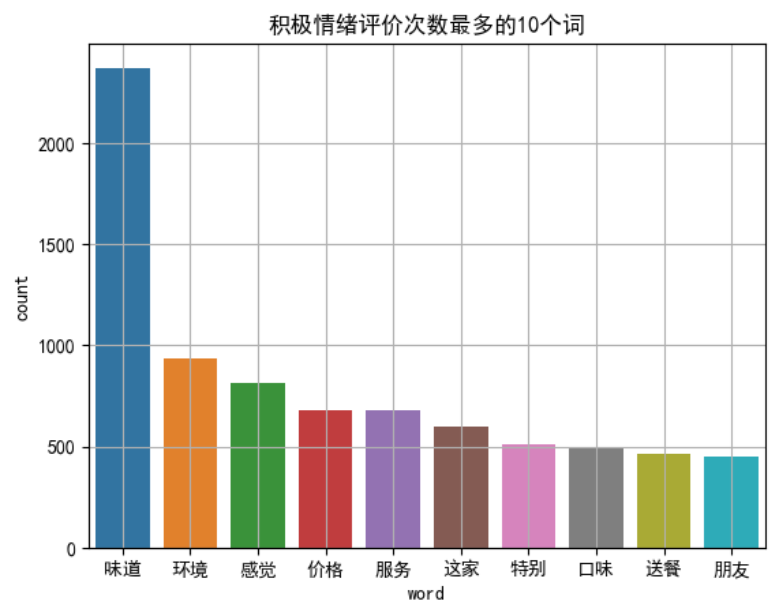
# 积极评价

positive **=** count\_word(train[train['target']**==**1])

plt.title('积极情绪评价次数最多的10个词')

sns.barplot(x**=**positive.iloc[:10].index, y**=**"count", data**=**positive.iloc[:10])

plt.grid()



**图4 积极情绪评价次数最多的10个词**

从上图可以看出，味道一词远远领先于其它词，说明人们把味道放在了第一位，只要味道好，顾客就开心，同时也说明味道与积极情绪相关性较高。

## 2.4 消极情绪评论最多的10个词

消极情绪统计前10词的实现如代码11所示。

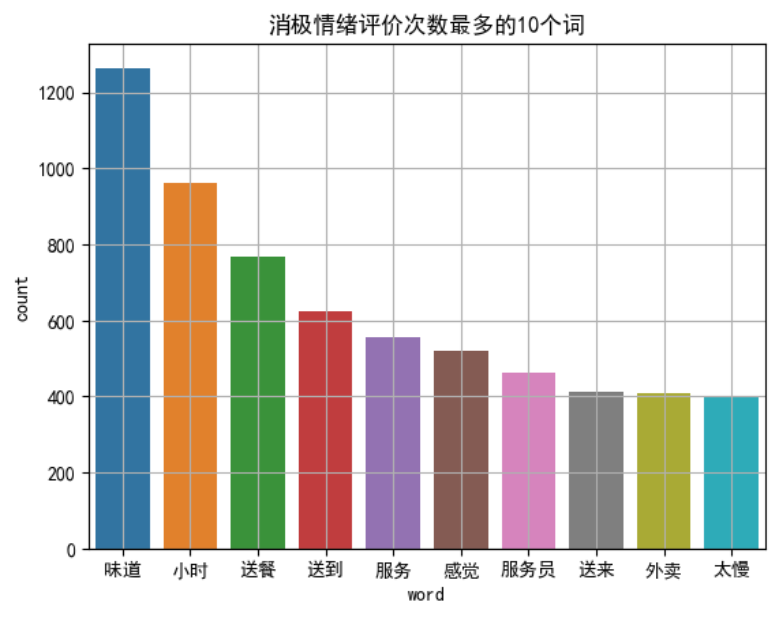
**代码11 消极情绪前10个词**

nag **=** count\_word(train[train['target']**==**1])

plt.title('消极情绪评价次数最多的10个词')

sns.barplot(x**=**nag.iloc[:10].index, y**=**"count", data**=**nag.iloc[:10])

plt.grid()



**图4 消极情绪评价次数最多的10个词**

# 3 问题二分析用户评论情绪与时间的关系

## 3.1 从数据中提取时间信息

首先需要对时间进行处理，提取数据中的时间信息，通过python分割字符串操作进行时间的提取。如代码11所示。

**代码11 消极情绪前10个词**

train['year'] **=** train['timestamp'].apply(**lambda** x:str(x)[:4])

train['month'] **=** train['timestamp'].apply(**lambda** x:str(x)[5:7])

train['day'] **=** train['timestamp'].apply(**lambda** x:str(x)[8:10])

train['hour'] **=** train['timestamp'].apply(**lambda** x:str(x)[11:13])

train['muin'] **=** train['timestamp'].apply(**lambda** x:str(x)[14:16])

train['sec'] **=** train['timestamp'].apply(**lambda** x:str(x)[17:19])

## 3.2 分析每小时和情绪之间的关系

提取了时间信息之后，对target特征进行统计分析，通过可视化操作，可以更加明显的观察出评论和target之间的关系。如代码12所示。

**代码12 情绪与时间小时的平均变化关系**

data **=** pd.DataFrame(columns**=**['hour','positive','negative'])

data['hour'] **=** range(1,25)

data['positive'] **=** train.groupby('target')['hour'].value\_counts().sort\_index().tolist()[:24]

data['negative'] **=** train.groupby('target')['hour'].value\_counts().sort\_index().tolist()[24:]

plt.figure(figsize**=**(10,6))

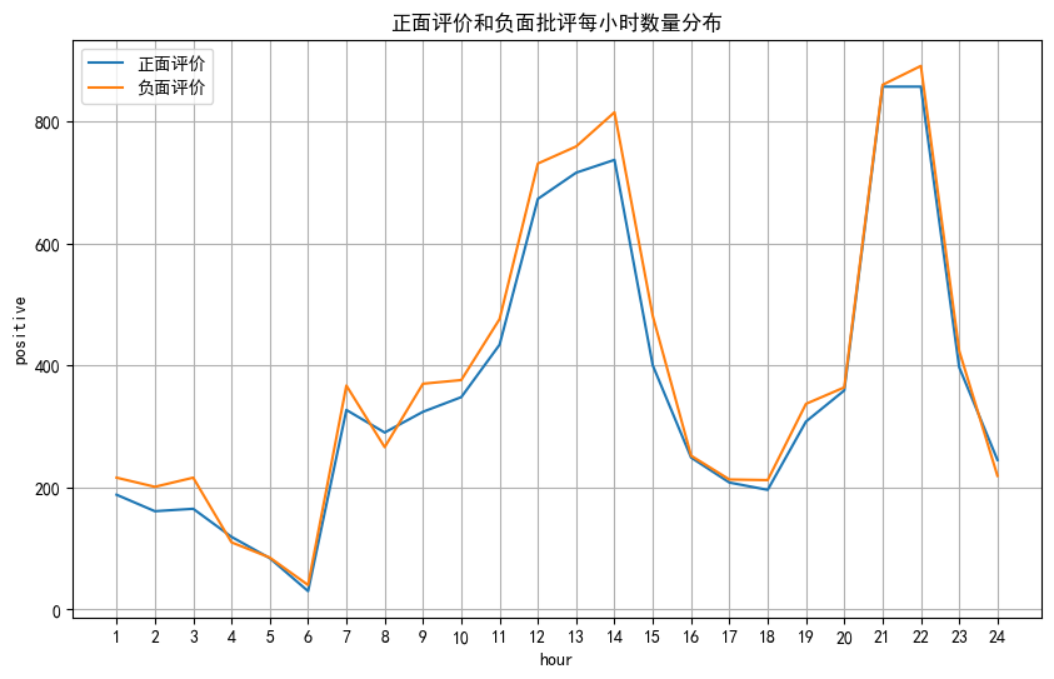
sns.lineplot(x**=**'hour',y**=**'positive',data**=**data,label**=**'正面评价')

sns.lineplot(x**=**'hour',y**=**'negative',data**=**data,label**=**'负面评价')

plt.xticks(data['hour'])

plt.title('正面评价和负面批评每小时数量分布')

plt.grid()



**图5 评价数量随小时的变化**

通过上图可以看出，正面情绪评价和负面情绪评价有着极强的正相关，在一天24小时中，其中1点-4点，8点-16，以及17点-20点，这三个时间段负面情绪评价是高于正面情绪评价数量，可能与外界环境有关。

1点-4点时间段，商家可能处于疲劳状态，导致出餐慢等一系列问题可能会导致顾客负面情绪较多。8点-16点以及17点-20点这两个时间段处于就餐高峰期，可能会导致出餐速度较慢等一系列问题从而使顾客负面情绪较多。

接下来进行将分析正负面评价和星期的关系，首先需要把日期转化为星期，

## 3.3 分析每天和情绪之间的关系

通过上述分析，可以看出在一天24小时之内的顾客情绪的变化，可能受客观因素的影响，接下来将分析顾客在一个月之内每天的平均情绪变化。如代码13所示。

**代码13 情绪与每天的平均变化关系**

data **=** pd.DataFrame(columns**=**['day','positive','negative'])

data['day'] **=** range(1,32)

data['positive'] **=** train.groupby('target')['day'].value\_counts().sort\_index().tolist()[:31]

data['negative'] **=** train.groupby('target')['day'].value\_counts().sort\_index().tolist()[31:]

plt.figure(figsize**=**(20,8))

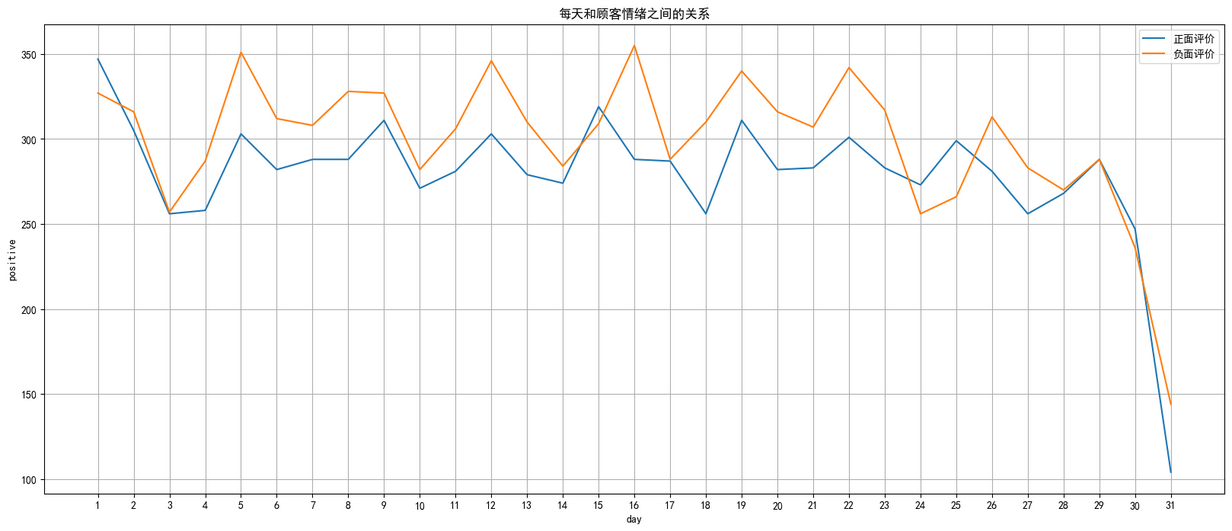
sns.lineplot(x**=**'day',y**=**'positive',data**=**data,label**=**'正面评价')

sns.lineplot(x**=**'day',y**=**'negative',data**=**data,label**=**'负面评价')

plt.xticks(data['day'])

plt.title("每天和顾客情绪之间的关系")

plt.grid()



**图6 评论数量随天的变化**

通过上图得出每天的变化，由于多数月份没有31号，故导致31号总评论数量偏少，不影响结果分析。可以看出顾客的积极情绪和消极情绪都是呈现周期性变化，大约3-4天为一个周期，但无法找出具体的规律，下面可以通过提取星期特征再进一步观察。

## 3.4 分析星期和情绪之间的关系

对所给的数据进行拼接，并将其数据类型转化为datetime数据类型，再进一步提取星期的信息。如代码14所示。

**代码14 提取数据中的星期特征**

train['weekday'] **=** pd.to\_datetime(train['year'].astype(str)**+**'-'**+**

train['month'].astype(str)**+**'-'**+**train['day'].astype(str)).dt.weekday

data **=** pd.DataFrame(columns**=**['weekday','positive','negative'])

data['weekday'] **=** range(1,8)

data['positive'] **=** train.groupby('target')['weekday'].value\_counts().sort\_index().tolist()[:7]

data['negative'] **=** train.groupby('target')['weekday'].value\_counts().sort\_index().tolist()[7:]

提取出星期信息之后，接下来进行数据统计和可视化操作。如代码15所示。

**代码15 情绪与星期的平均变化关系**

total\_width, n **=** 0.8, 2

width **=** total\_width **/** n

name\_list **=** data['weekday'].tolist()

num\_list **=** data['positive'].tolist()

num\_list1 **=** data['negative'].tolist()

x **=**list(range(len(num\_list)))

total\_width, n **=** 0.8, 2

width **=** total\_width **/** n

plt.bar(x, num\_list, width**=**width, label**=**'positive',fc **=** 'r')

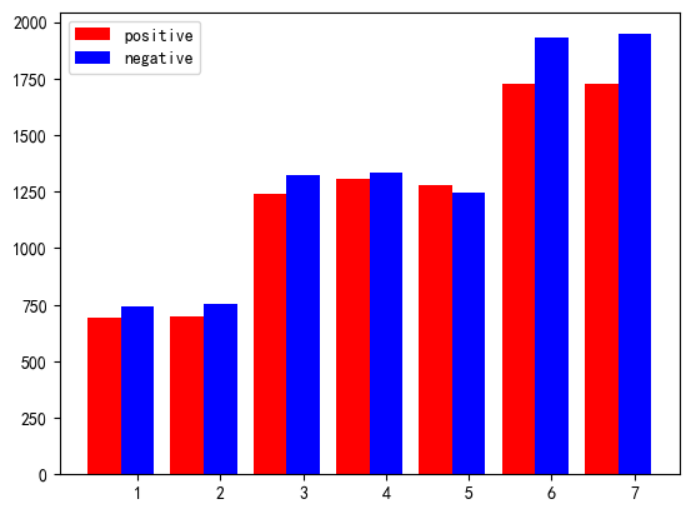
**for** i **in** range(len(x)):

    x[i] **=** x[i] **+** width

plt.bar(x, num\_list1, width**=**width, label**=**'negative',tick\_label **=** name\_list,fc **=** 'b')

plt.legend()

plt.show()



**图7 评论数量随星期的变化**

从以上柱状图可以清楚的看出顾客情绪与时间的关系，在周一周、二两天顾客点餐数量较少，负面情绪略高于正面情绪。在周三、周四、周五三天顾客情绪负面情绪在相对减少。在周六、周日两天顾客订餐数量较多，同时顾客的消极情绪也相对增高。

最后查看一下负面情绪和星期的占比，如代码16所示。

**代码16 计算星期负面情绪占比**

[y**/**(x**+**y) **for** x, y **in** zip(data['positive'].tolist(),data['negative'].tolist())]

**表2 每天负面情绪占比**

|  |  |
| --- | --- |
|  | 负面情绪占比 |
| 星期一 | 0.518 |
| 星期二 | 0.518 |
| 星期三 | 0.516 |
| 星期四 | 0.506 |
| 星期五 | 0.439 |
| 星期六 | 0.528 |
| 星期七 | 0.53 |

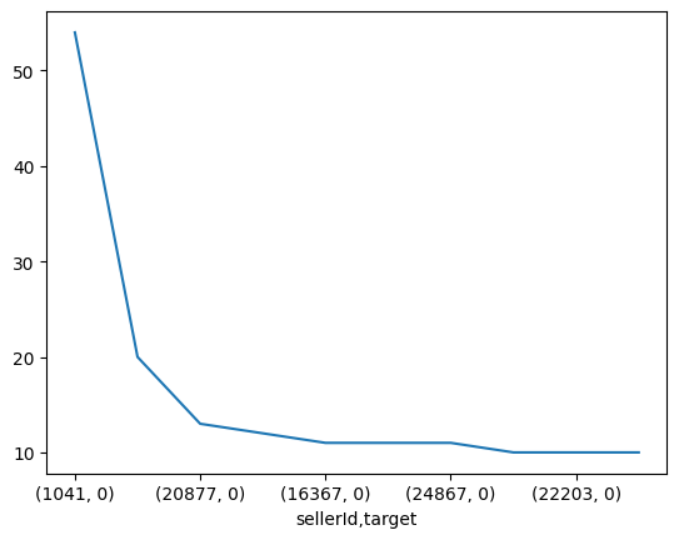
# 4 问题三分析积极情绪商家及其优点

## 4.1 分析积极情绪最多的商家

数据中给出了商家的sellerid参数，作为商家的唯一标识，可以用分组聚合操作进行商家积极情绪的数量。如代码17所示。

**代码17 积极情绪排名前10商家展示**

train[train['target']**==**0].groupby('sellerId')['target'].value\_counts().sort\_values(ascending **=** False)[:10].plot()



**图8 积极情绪商家排名展示**

可以看出id为1041的商家积极评论最多，一共为54条评论，接下来选取id为1041的商家信息，并进行分析，首先查看一下该商家的评论分布，通过观察，得出商家积极评论有54条，消极评论只有4条，好评率高达93%。

## 4.2 分析积极情绪最多的商家

积极情绪最多的商家id为1041，通过TFIDF进行关键词的提取，TFIDF计算公式如下：

其中，表示一个词或词语，表示文档，表示文档集合，表示词在文档中出现的频率，表示词在文档集合中的逆文档频率。

TF(Term Frequency)为词频，即某个给定词语在该文章中出现的频率，TF计算公式如下：

其中，表示词在文档中出现的次数，表示词在文档中出现的次数，表示文档中总的词数。

IDF(Inverse Document Frequency)为逆向文档频率，指的是一个词的普遍重要性程度，TFIDF计算公式如下：

其中表示文档集合中的文档数量，表示在文档集合中包含词的文档数量。

使用TFIDF进行文本词权重提取。如代码18所示：

**代码18 TFIDF提取关键词**

**from** jieba **import** analyse

lis **=** jieba.analyse.extract\_tags(text, withWeight **=** True, topK**=**10)

通过以上代码可以提取相应的关键词，使用画图函数可以更加直观的展示每个词的权重。如代码19所示：

**代码19 商家评论最重要的10个词**

score **=** pd.DataFrame(columns**=**['关键词','权重'])

score['关键词'] **=** [i[0] **for** i **in** lis]

score['权重'] **=** [i[1] **for** i **in** lis]

plt.barh(score['关键词'], score['权重'], height**=**0.7, color**=**'#008792'

, edgecolor**=**'#005344')

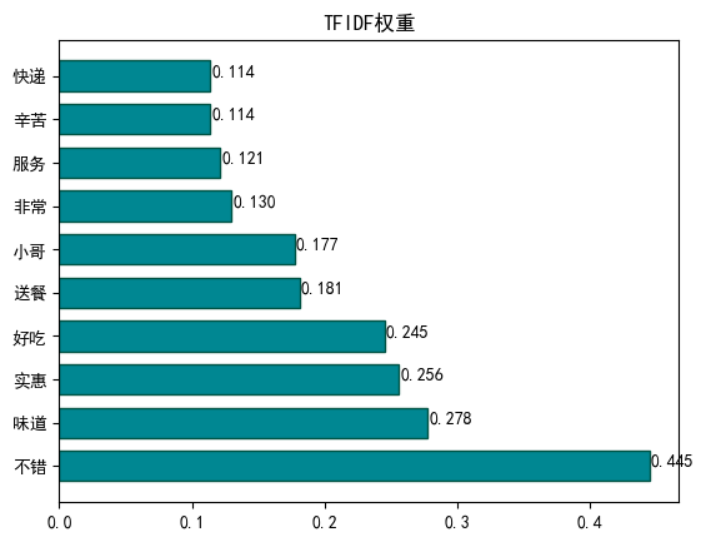
plt.title('TFIDF权重') # 标题

**for** a,b **in** zip( score['权重'],score['关键词']):

   print(a,b)

   plt.text(a**+**0.001, b,'%.3f'**%**float(a))

plt.show()



**图8 积极情绪商家排名展示**

通过以上图片可以看出，‘不错’一词分值较高，可以得出该商家给顾客的第一感觉很好。其次‘味道’一词权重占比第二，说明顾客对该商家的味道非常的满意。关键词权重排名第三的词为实惠，说明了该商家出餐分量足，深受顾客喜爱。

# 5 问题四分析消极情绪商家以及改进策略

## 5.1 分析消极情绪最多的商家

首先应选择消极情绪的评论，使用分组聚合操作进行商家数量的统计，最后进行排序操作。如代码20所示：

**代码20 计算消极评论数量排名前3的商家**

train[train['target']**==**1].groupby('sellerId')['target'].value\_counts().sort\_values(ascending **=** False)[:3]

**表3 消极评论商家前3名**

|  |  |  |
| --- | --- | --- |
| sellerId | Target | 消极评论数量 |
| 971 | 1 | 44 |
| 1173 | 1 | 16 |
| 961 | 1 | 14 |

可以看出id为971的商家消极评论最多，一共为44条评论。接下来选取id为1041的商家信息，并进行分析，首先查看一下该商家的评论分布，通过观察，得出商家积极评论有6条，消极评论只有44条，好评率为12%。

## 5.2 分析消极情绪最多的商家

接下来选取id为971商家数据，使用TFIDF进行关键词权重的提取。如代码21所示：

**代码21 使用jieba进行分词**

lis **=** jieba.analyse.extract\_tags(text, withWeight **=** True, topK**=**10)

通过以上代码可以提取相应的关键词，使用画图函数可以更加直观的展示每个词的权重。如代码22所示：

**代码22 商家评论最重要的10个词**

core **=** pd.DataFrame(columns**=**['名词','重要性'])

score['名词'] **=** [i[0] **for** i **in** lis]

score['重要性'] **=** [i[1] **for** i **in** lis]

plt.barh(score['名词'], score['重要性'], height**=**0.7, color**=**"c",hatch**=**"/",edgecolor**=**'#005344')

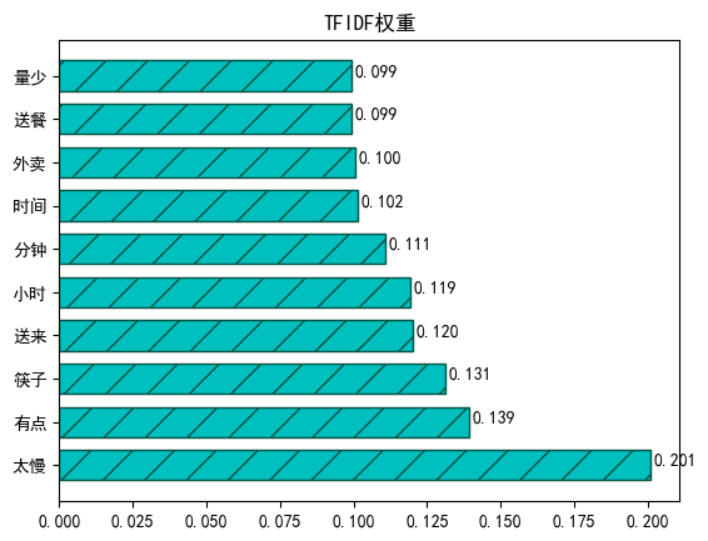
plt.title('TFIDF权重') # 标题

**for** a,b **in** zip( score['重要性'],score['名词']):

   print(a,b)

   plt.text(a**+**0.001, b,'%.3f'**%**float(a))

plt.show()



**图9 消极情绪商家排名展示**

通过上图可以看出，在商家id为971的评论中，‘太慢’一次占据较大的权重，说明送餐超时问题。‘筷子’一词占据第三，说明商家未注重顾客的备注。

## 5.3 改进策略

通过分析，该商家主要有三方面的问题：送餐超时、餐具、量少三个方面。需进行改进。首先应该是解决出餐、送餐慢的问题，商家应该提高出餐效率，加快出餐进度，以保证送餐员能够按时送到顾客手中，避免顾客长时间的等待。然后就是商家应该注重顾客备注的餐具数量，按照顾客的需求而进行餐具增减，这样才能保证顾客能够顺利的食用。最后就是在保证盈利的情况下，适当的增加份量，使每位顾客能够吃饱，不会因为量少而导致顾客产生消极情绪。

综上所述，对该商家提出的改进策略：提高出餐速度、合理分配餐具数量、在保证盈利的情况下加大份量。该商家如果能够按照此策略进行改进，对提高顾客积极情绪有着很大的帮助。

# 6 问题五模型的建立和评估测试

## 6.1 建立餐饮评论情感倾向模型

在建立模型前，需对数据进行分析，由于每个商家的评论长度长短不一，所以应该选取一个合适的长度，大于改长度的进行截断，小于该长度的进行填充，评论长度查看如代码23所示：

**代码23 查看评论长度分布**

token\_lens **=** []

**for** txt **in** train.comment:

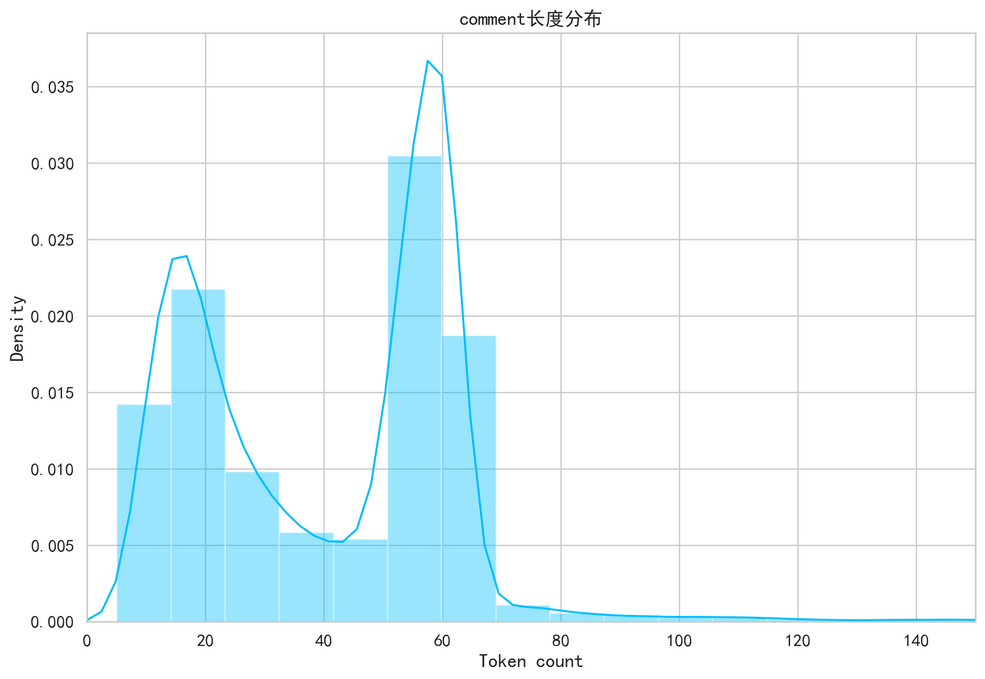
    tokens **=** tokenizer.encode(txt, max\_length**=**512)

token\_lens.append(len(tokens))

sns.distplot(token\_lens)

plt.xlim([0, 150]);

plt.xlabel('Token count');



**图10 comment长度分布**

通过上图可以看出文本每条评论的长度绝大多数都在80以内，所以可以选取80为合适的长度，既不会丢失太多信息，也不会填充太多无用的信息。接下来进行数据集的分割，将训练数据分割成训练集和验证集，训练集用来训练模型，验证集用来评估模型的好坏，将数据的9份用来训练，1份用来验证。如代码24所示。

**代码24 数据集分割**

df\_train, df\_test **=** train\_test\_split(train, test\_size**=**0.1, random\_state**=**RANDOM\_SEED)

df\_val, df\_test **=** train\_test\_split(df\_test, test\_size**=**0.5, random\_state**=**RANDOM\_SEED)

由于comment内容全为文字内容，无法直接将其输入网络，故应进行数据预处理操作，即文本向量化操作，使用预训练roberta对文本进行embedding，如代码25所示：

**代码25 文本向量化**

**class** EnterpriseDataset(Dataset):

**def** \_\_init\_\_(self,texts,labels,tokenizer,max\_len):

        self.texts**=**texts

        self.labels**=**labels

        self.tokenizer**=**tokenizer

        self.max\_len**=**max\_len

**def** \_\_len\_\_(self):

**return** len(self.texts)

**def** \_\_getitem\_\_(self,item):

        text**=**str(self.texts[item])

        label**=**self.labels[item]

        encoding**=**self.tokenizer.encode\_plus(

            text,

            add\_special\_tokens**=**True,

            max\_length**=**self.max\_len,

            return\_token\_type\_ids**=**True,

            pad\_to\_max\_length**=**True,

            return\_attention\_mask**=**True,

            return\_tensors**=**'pt',

        )

**return** {

            'texts':text,

            'input\_ids':encoding['input\_ids'].flatten(),

            'attention\_mask':encoding['attention\_mask'].flatten(),

            'labels':torch.tensor(label,dtype**=**torch.long)

        }

上述编码的结果包含：input\_ids和attention\_mask，其中input\_ids为编码的结果，attention\_mask为可以保证模型在做attention时，有效数据不会被mask。

接下来是模型的搭建，通过搭建神经网络模型，进行数据的预测，本赛题使用的模型是chinese-roberta-wwn模型，该模型在情感分类任务上较为优越，下面是模型的加载，如代码26所示。

**代码26 加载预训练语言模型**

PRE\_TRAINED\_MODEL\_NAME **=** 'hfl/chinese-roberta-wwm-ext'

tokenizer **=** AutoTokenizer.from\_pretrained(PRE\_TRAINED\_MODEL\_NAME)

在chinese-roberta-wwn模型的后添加全连接层进行二分类处理，即积极情绪和消极情绪的分类，同时，为防止过拟合，选择神经网络的丢弃率为0.3，网络搭建的实现如代码27所示。

**代码27 模型的构建**

**class** EnterpriseDangerClassifier(nn.Module):

**def** \_\_init\_\_(self, n\_classes):

        super(EnterpriseDangerClassifier, self).\_\_init\_\_()

        self.bert **=** BertModel.from\_pretrained(PRE\_TRAINED\_MODEL\_NAME)

        self.drop **=** nn.Dropout(p**=**0.3)

        self.out **=** nn.Linear(self.bert.config.hidden\_size, n\_classes) # 两个类别

**def** forward(self, input\_ids, attention\_mask):

        \_, pooled\_output **=** self.bert(

            input\_ids**=**input\_ids,

            attention\_mask**=**attention\_mask,

            return\_dict **=** False

        )

        output **=** self.drop(pooled\_output) # dropout

**return** self.out(output)

对于神经网络模型，需要选择合适的优化器，以及损失函数的选取。这里选择的优化器为AdamW，其优点是在Adam优化器的基础上加入正则化，有效避免了过拟合问题。损失函数选取的是交叉熵损失函数，用于评估分类问题。如代码28所示：

**代码28 定义优化器和损失函数**

optimizer **=** AdamW(model.parameters(), lr**=**2e**-**3, correct\_bias**=**False)

scheduler **=** get\_linear\_schedule\_with\_warmup(

  optimizer,

  num\_warmup\_steps**=**0,

  num\_training\_steps**=**total\_steps

)

loss\_fn **=** nn.CrossEntropyLoss().to(device)

## 6.2 情感倾向模型的训练和评估

在神经模型建立完成之后，需要进行模型的训练，如代码29所示。

**代码29 训练模型**

**def** train\_epoch(

  model,

  data\_loader,

  loss\_fn,

  optimizer,

  device,

  scheduler,

  n\_examples

):

    model **=** model.train() # train模式

    losses **=** []

    correct\_predictions **=** 0

**for** d **in** data\_loader:

        input\_ids **=** d["input\_ids"].to(device)

        attention\_mask **=** d["attention\_mask"].to(device)

        targets **=** d["labels"].to(device)

        outputs **=** model(

            input\_ids**=**input\_ids,

            attention\_mask**=**attention\_mask

        )

        \_, preds **=** torch.max(outputs, dim**=**1)

        loss **=** loss\_fn(outputs, targets)

        correct\_predictions **+=** torch.sum(preds **==** targets)

        losses.append(loss.item())

        loss.backward()

        nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm**=**1.0)

        optimizer.step()

        scheduler.step()

        optimizer.zero\_grad()

**return** correct\_predictions.double() **/** n\_examples, np.mean(losses)

训练完成后需进行模型的评估，选择所给数据的五分之一用来模型的评估，并计算相关的准确率，进行可视化展示，如代码30所示。

**代码30 绘制准确率曲线**

plt.plot(torch.tensor(history['train\_acc'], device**=**'cpu'), label**=**'train accuracy')

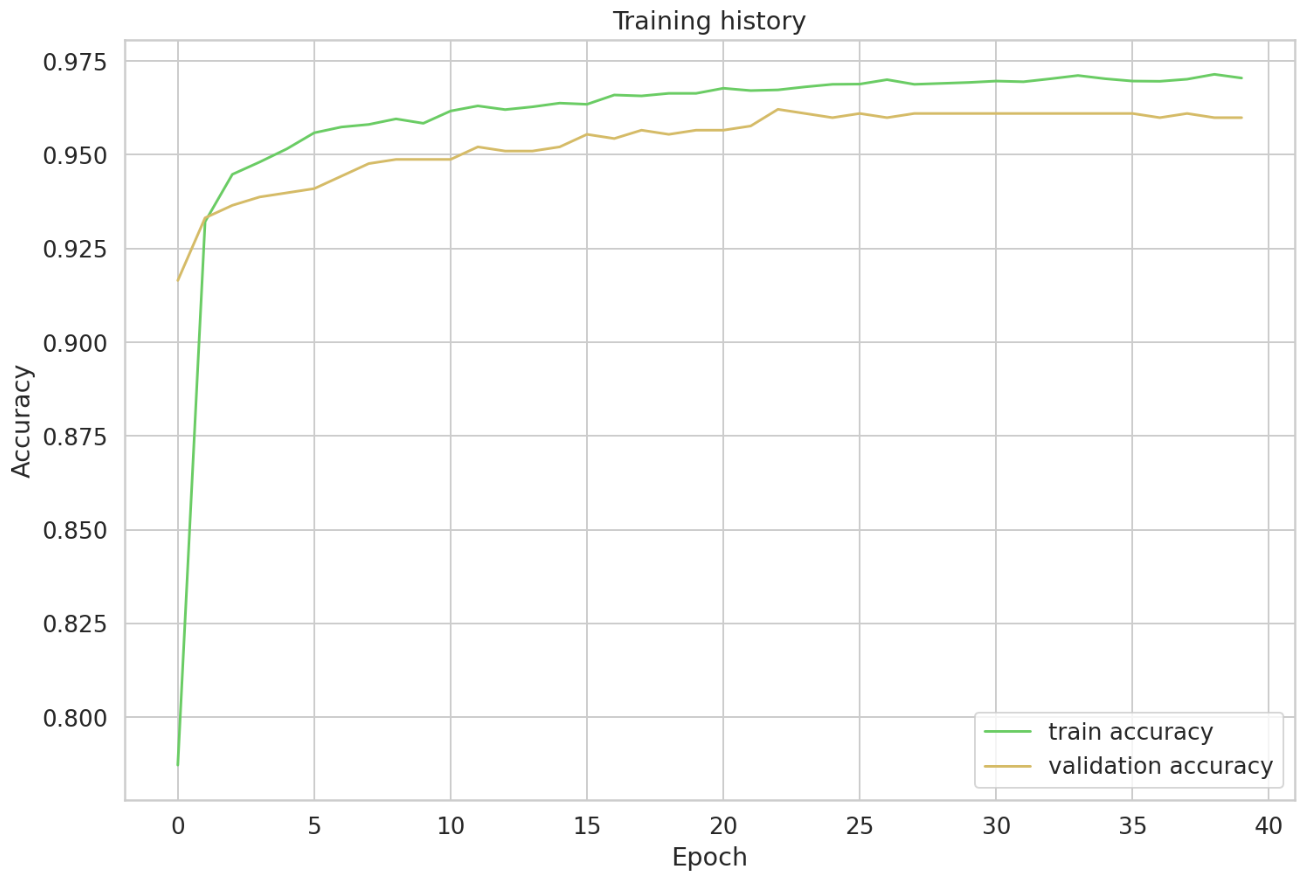
plt.plot(torch.tensor(history['val\_acc'], device**=**'cpu'), label**=**'validation accuracy')

plt.title('Training history')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend()



**图11 模型准确率曲线**

从上图可以看出，训练准确率和验证准确率都在增高，最终训练准确率收敛在97.2%左右

接下来查看一下模型的混淆矩阵，如代码31所示。

**代码31 绘制混淆矩阵**

**def** show\_confusion\_matrix(confusion\_matrix):

    hmap **=** sns.heatmap(confusion\_matrix, annot**=**True, fmt**=**"d", cmap**=**"Blues")

    hmap.yaxis.set\_ticklabels(hmap.yaxis.get\_ticklabels(), rotation**=**0, ha**=**'right')

    hmap.xaxis.set\_ticklabels(hmap.xaxis.get\_ticklabels(), rotation**=**30, ha**=**'right')

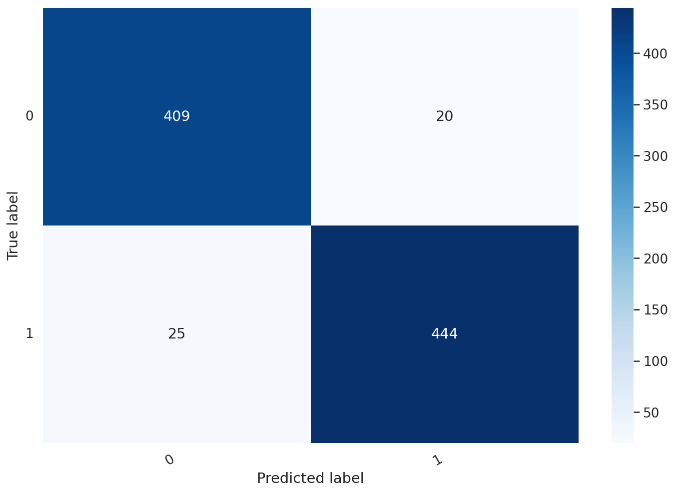
    plt.ylabel('True label')

    plt.xlabel('Predicted label');

cm **=** confusion\_matrix(y\_test, y\_pred)

df\_cm **=** pd.DataFrame(cm, index**=**class\_names, columns**=**class\_names)

show\_confusion\_matrix(df\_cm)



**图12 混淆矩阵**

从以上混淆矩阵可以观察出，TF为409，TP为444，在验证数据中，共有853条样本预测正确，45条样本是预测错误的，准确率在97%左右。接下来看一下模型各方面的评估。如代码32所示。

**代码32 模型的评估**

print(classification\_report(y\_test, y\_pred, target\_names**=**[str(label) **for** label **in** class\_names]))

**表4 评估指标**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| 0 | 0.94 | 0.95 | 0.95 | 429 |
| 1 | 0.96 | 0.95 | 0.95 | 469 |
| Weighted avg | 0.95 | 0.95 | 0.95 | 898 |
| Maacro avg | 0.95 | 0.95 | 0.95 | 898 |

## 6.3 对附件test.xlsx进行预测

首先对测试集进行读取，发现测试集共含有1500条样本，由于已经完成了模型的训练和评估步骤，接下来可以进行模型的预测，并将结果补充到文件的第一列。如代码33所示。

**代码33 模型的预测**

**def** con(sample\_text):

    encoded\_text **=** tokenizer.encode\_plus(

    sample\_text,

    max\_length**=**MAX\_LEN,

    add\_special\_tokens**=**True,

    return\_token\_type\_ids**=**False,

    pad\_to\_max\_length**=**True,

    return\_attention\_mask**=**True,

    return\_tensors**=**'pt',

    )

    input\_ids **=** encoded\_text['input\_ids'].to(device)

    attention\_mask **=** encoded\_text['attention\_mask'].to(device)

    output **=** model(input\_ids, attention\_mask)

    \_, prediction **=** torch.max(output, dim**=**1)

**return** class\_names[prediction]

test['target'] **=** test['comment'].apply(**lambda** x:con(x))

最后将文件进行保存

test.to\_excel('./working/test.xlsx',index = False)

# 7 附录

注：本论文进行数据分析相关操作均使用Python代码编写，下面是关键代码，完整代码放在支撑材料里面。

**问题一：问题一制作词云和查找评论前10词**

# 导入所需要的包

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import jieba

import jieba

import matplotlib.pyplot as plt

import pandas as pd

from imageio import imread # pip install pillow

from wordcloud import WordCloud

import warnings

warnings.filterwarnings('ignore')

plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体

plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题

# 读取数据

train = pd.read\_excel('./data/data.xlsx')

test = pd.read\_excel('./data/test.xlsx')

data\_数据说明 = pd.read\_excel('./data/字段说明.xlsx')

text = ''

for i in train['comment'].tolist():

text+=i.replace('text','')

# 停用词 简单理解：无视这些烂大街的东西

# 停用词是指在信息检索中，为节省存储空间和提高搜索效率，

# 在处理自然语言数据（或文本）之前或之后会自动过滤掉某些字或词

stop = pd.read\_csv('./stoplist.txt', header=None, encoding='utf-8',engine='python', sep='limh')

b = stop.drop\_duplicates()

stop = [' ', '　'] + list(stop[0])

# stop1 = [chr(32), chr(12288)] + list(stop[0])

stop = set(stop)

# 去停用词

words = [word for word in words if word not in stop]

word\_num = pd.DataFrame(words, columns=['word'])

word\_num = word\_num[word\_num['word'] != '\n']

word\_num['count'] = 1

word\_num = word\_num.groupby('word').sum()

word\_many = word\_num[word\_num['count'] > 50]

# 背景图片

back\_pic = imread("E:\\222.png") # aixin.jpg # 设置背景图片

wc = WordCloud(font\_path='C:\\Windows\\Fonts\\simkai.TTF', # 设置字体 使用的 windows 自带的字体

background\_color="white", # ="white", #背景颜色

max\_words=2000, # 词云显示的最大数

mask=back\_pic, # 设置背景图片

max\_font\_size=200, # =200, #字体最大值

random\_state=42, )

# 生成词云

wc.fit\_words(word\_many['count'])

# 绘图

plt.figure(figsize=(30, 15))

plt.imshow(wc)

plt.axis('off')

**问题二：分析用户评论情绪与时间的关系**

# 读取数据

train = pd.read\_excel('./data/data.xlsx')

test = pd.read\_excel('./data/test.xlsx')

data\_数据说明 = pd.read\_excel('./data/字段说明.xlsx')

train['year'] = train['timestamp'].apply(lambda x:str(x)[:4])

train['month'] = train['timestamp'].apply(lambda x:str(x)[5:7])

train['day'] = train['timestamp'].apply(lambda x:str(x)[8:10])

train['hour'] = train['timestamp'].apply(lambda x:str(x)[11:13])

train['muin'] = train['timestamp'].apply(lambda x:str(x)[14:16])

train['sec'] = train['timestamp'].apply(lambda x:str(x)[17:19])

data = pd.DataFrame(columns=['hour','positive','negative'])

data['hour'] = range(1,25)

data['positive'] = train.groupby('target')['hour'].value\_counts().sort\_index().tolist()[:24]

data['negative'] = train.groupby('target')['hour'].value\_counts().sort\_index().tolist()[24:]

plt.figure(figsize=(10,6))

sns.lineplot(x='hour',y='positive',data=data,label='正面评价')

sns.lineplot(x='hour',y='negative',data=data,label='负面评价')

plt.xticks(data['hour'])

plt.title('正面评价和负面批评每小时数量分布')

plt.grid()

data = pd.DataFrame(columns=['day','positive','negative'])

data['day'] = range(1,32)

data['positive'] = train.groupby('target')['day'].value\_counts().sort\_index().tolist()[:31]

data['negative'] = train.groupby('target')['day'].value\_counts().sort\_index().tolist()[31:]

plt.figure(figsize=(20,8))

sns.lineplot(x='day',y='positive',data=data,label='正面评价')

sns.lineplot(x='day',y='negative',data=data,label='负面评价')

plt.xticks(data['day'])

plt.title("每天和顾客情绪之间的关系")

plt.grid()

data = pd.DataFrame(columns=['flag','positive','negative'])

data['flag'] = range(1,min+1)

data['positive'] = train.groupby('target')['flag'].value\_counts().sort\_index().tolist()[:min]

data['negative'] = train.groupby('target')['flag'].value\_counts().sort\_index().tolist()[min:]

total\_width, n = 0.8, 2

width = total\_width / n

name\_list = data['weekday'].tolist()

num\_list = data['positive'].tolist()

num\_list1 = data['negative'].tolist()

x =list(range(len(num\_list)))

total\_width, n = 0.8, 2

width = total\_width / n

plt.bar(x, num\_list, width=width, label='positive',fc = 'r')

for i in range(len(x)):

x[i] = x[i] + width

plt.bar(x, num\_list1, width=width, label='negative',tick\_label = name\_list,fc = 'b')

plt.legend()

plt.show()

**问题三：分析积极情绪商家及其优点**

# 读取数据

train = pd.read\_excel('./data/data.xlsx')

test = pd.read\_excel('./data/test.xlsx')

data\_数据说明 = pd.read\_excel('./data/字段说明.xlsx')

train[train['sellerId']==1041]['target'].value\_counts()

train[train['sellerId']==1041]['comment'].apply(lambda x:jieba.lcut(x))

from jieba import analyse

lis = jieba.analyse.extract\_tags(text, withWeight = True, topK=10) # 返回权重

score = pd.DataFrame(columns=['关键词','权重'])

score['关键词'] = [i[0] for i in lis]

score['权重'] = [i[1] for i in lis]

from matplotlib import pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.barh(score['关键词'], score['权重'], height=0.7, color='#008792', edgecolor='#005344')

plt.title('TFIDF权重') # 标题

for a,b in zip( score['权重'],score['关键词']):

print(a,b)

plt.text(a+0.001, b,'%.3f'%float(a))

plt.show()

**问题四：分析消极情绪商家以及改进策略**

# 读取数据

train = pd.read\_excel('./data/data.xlsx')

test = pd.read\_excel('./data/test.xlsx')

data\_数据说明 = pd.read\_excel('./data/字段说明.xlsx')

train[train['target']==1].groupby('sellerId')['target'].value\_counts().sort\_values(ascending = False)[:3]

train[train['sellerId']==971].shape

train[train['sellerId']==971]['target'].value\_counts()

from jieba import analyse

lis = jieba.analyse.extract\_tags(text, withWeight = True, topK=11)

score = pd.DataFrame(columns=['名词','重要性'])

score['名词'] = [i[0] for i in lis]

score['重要性'] = [i[1] for i in lis]

score = score[1:]

from matplotlib import pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei'] # 显示中文黑体

# plt.rcParams['axes.unicode\_minus'] = False # 负值显示

plt.barh(score['名词'], score['重要性'], height=0.7, color="c",hatch="/",edgecolor='#005344') # 更多颜色可参见颜色大全

# plt.xlabel('feature importance') # x 轴

# plt.ylabel('features') # y轴

plt.title('TFIDF权重') # 标题

for a,b in zip( score['重要性'],score['名词']): # 添加数字标签

print(a,b)

plt.text(a+0.001, b,'%.3f'%float(a)) # a+0.001代表标签位置在柱形图上方0.001处

plt.show()

**问题五：模型的建立和评估测试**

# 导入所需要的包

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import jieba

import jieba

import matplotlib.pyplot as plt

import pandas as pd

from imageio import imread # pip install pillow

# 导入transformers

import transformers

from transformers import BertModel, BertTokenizer,BertConfig, AdamW, get\_linear\_schedule\_with\_warmup

from transformers import AutoModel, AutoTokenizer,AutoConfig, AdamW, get\_linear\_schedule\_with\_warmup

# 导入torch

import torch

from torch import nn, optim

from torch.utils.data import Dataset, DataLoader

import torch.nn.functional as F

# 常用包

import re

import numpy as np

import pandas as pd

import seaborn as sns

from pylab import rcParams

import matplotlib.pyplot as plt

from matplotlib import rc

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix, classification\_report

from collections import defaultdict

from textwrap import wrap

import warnings

warnings.filterwarnings('ignore')

%matplotlib inline

%config InlineBackend.figure\_format='retina' # 主题

# 读取数据

train = pd.read\_excel('./data/data.xlsx')

test = pd.read\_excel('./data/test.xlsx')

data\_数据说明 = pd.read\_excel('./data/字段说明.xlsx')

sns.set(style='whitegrid', palette='muted', font\_scale=1.2)

HAPPY\_COLORS\_PALETTE = ["#01BEFE", "#FFDD00", "#FF7D00", "#FF006D", "#ADFF02", "#8F00FF"]

sns.set\_palette(sns.color\_palette(HAPPY\_COLORS\_PALETTE))

rcParams['figure.figsize'] = 12, 8

plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体

plt.rcParams["axes.unicode\_minus"]=False #该语句解决图像中的“-”负号的乱码问题

# 固定随机种子

RANDOM\_SEED = 42

np.random.seed(RANDOM\_SEED)

torch.manual\_seed(RANDOM\_SEED)

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

sns.countplot(x="target", data=train)

plt.title("正负面情绪标签分布")

plt.xlabel('label count')

PRE\_TRAINED\_MODEL\_NAME = 'bert-base-chinese'

PRE\_TRAINED\_MODEL\_NAME = 'hfl/chinese-roberta-wwm-ext'

# PRE\_TRAINED\_MODEL\_NAME = 'hfl/chinese-roberta-wwm'

# tokenizer = BertTokenizer.from\_pretrained(PRE\_TRAINED\_MODEL\_NAME)

tokenizer = AutoTokenizer.from\_pretrained(PRE\_TRAINED\_MODEL\_NAME)

token\_lens = []

for txt in train.comment:

tokens = tokenizer.encode(txt, max\_length=512)

token\_lens.append(len(tokens))

sns.distplot(token\_lens)

plt.xlim([0, 150]);

plt.title('comment长度分布')

plt.xlabel('Token count');

# 自定义数据集

class EnterpriseDataset(Dataset):

def \_\_init\_\_(self,texts,labels,tokenizer,max\_len):

self.texts=texts

self.labels=labels

self.tokenizer=tokenizer

self.max\_len=max\_len

def \_\_len\_\_(self):

return len(self.texts)

def \_\_getitem\_\_(self,item):

"""

item 为数据索引，迭代取第item条数据

"""

text=str(self.texts[item])

label=self.labels[item]

encoding=self.tokenizer.encode\_plus(

text,

add\_special\_tokens=True,

max\_length=self.max\_len,

return\_token\_type\_ids=True,

pad\_to\_max\_length=True,

return\_attention\_mask=True,

return\_tensors='pt',

)

return {

'texts':text,

'input\_ids':encoding['input\_ids'].flatten(),

'attention\_mask':encoding['attention\_mask'].flatten(),

'labels':torch.tensor(label,dtype=torch.long)

}

# 划分数据集并创建生成器

df\_train, df\_test = train\_test\_split(train, test\_size=0.1, random\_state=RANDOM\_SEED)

df\_val, df\_test = train\_test\_split(df\_test, test\_size=0.5, random\_state=RANDOM\_SEED)

df\_train.shape, df\_val.shape, df\_test.shape

def create\_data\_loader(df,tokenizer,max\_len,batch\_size):

ds=EnterpriseDataset(

texts=df['comment'].values,

labels=df['target'].values,

tokenizer=tokenizer,

max\_len=max\_len

)

return DataLoader(

ds,

batch\_size=batch\_size,

)

BATCH\_SIZE = 4

train\_data\_loader = create\_data\_loader(df\_train, tokenizer, MAX\_LEN, BATCH\_SIZE)

val\_data\_loader = create\_data\_loader(df\_val, tokenizer, MAX\_LEN, BATCH\_SIZE)

test\_data\_loader = create\_data\_loader(df\_test, tokenizer, MAX\_LEN, BATCH\_SIZE)

# 定义模型

class EnterpriseDangerClassifier(nn.Module):

def \_\_init\_\_(self, n\_classes):

super(EnterpriseDangerClassifier, self).\_\_init\_\_()

self.bert = BertModel.from\_pretrained(PRE\_TRAINED\_MODEL\_NAME)

self.drop = nn.Dropout(p=0.3)

self.out = nn.Linear(self.bert.config.hidden\_size, n\_classes) # 两个类别

def forward(self, input\_ids, attention\_mask):

\_, pooled\_output = self.bert(

input\_ids=input\_ids,

attention\_mask=attention\_mask,

return\_dict = False

)

output = self.drop(pooled\_output) # dropout

return self.out(output)

# 定义训练函数

def train\_epoch(

model,

data\_loader,

loss\_fn,

optimizer,

device,

scheduler,

n\_examples

):

model = model.train() # train模式

losses = []

correct\_predictions = 0

for d in data\_loader:

input\_ids = d["input\_ids"].to(device)

attention\_mask = d["attention\_mask"].to(device)

targets = d["labels"].to(device)

outputs = model(

input\_ids=input\_ids,

attention\_mask=attention\_mask

)

\_, preds = torch.max(outputs, dim=1)

loss = loss\_fn(outputs, targets)

correct\_predictions += torch.sum(preds == targets)

losses.append(loss.item())

loss.backward()

nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm=1.0)

optimizer.step()

scheduler.step()

optimizer.zero\_grad()

return correct\_predictions.double() / n\_examples, np.mean(losses)

# 定义评估函数

def eval\_model(model, data\_loader, loss\_fn, device, n\_examples):

model = model.eval() # 验证预测模式

losses = []

correct\_predictions = 0

with torch.no\_grad():

for d in data\_loader:

input\_ids = d["input\_ids"].to(device)

attention\_mask = d["attention\_mask"].to(device)

targets = d["labels"].to(device)

outputs = model(

input\_ids=input\_ids,

attention\_mask=attention\_mask

)

\_, preds = torch.max(outputs, dim=1)

loss = loss\_fn(outputs, targets)

correct\_predictions += torch.sum(preds == targets)

losses.append(loss.item())

return correct\_predictions.double() / n\_examples, np.mean(losses)

# 训练

history = defaultdict(list) # 记录10轮loss和acc

best\_accuracy = 0

for epoch in range(EPOCHS):

print(f'Epoch {epoch + 1}/{EPOCHS}')

print('-' \* 10)

train\_acc, train\_loss = train\_epoch(

model,

train\_data\_loader,

loss\_fn,

optimizer,

device,

scheduler,

len(df\_train)

)

print(f'Train loss {train\_loss} accuracy {train\_acc}')

val\_acc, val\_loss = eval\_model(

model,

val\_data\_loader,

loss\_fn,

device,

len(df\_val)

)

print(f'Val loss {val\_loss} accuracy {val\_acc}')

print()

history['train\_acc'].append(train\_acc)

history['train\_loss'].append(train\_loss)

history['val\_acc'].append(val\_acc)

history['val\_loss'].append(val\_loss)

if val\_acc > best\_accuracy:

torch.save(model.state\_dict(), 'best\_model\_state.bin')

best\_accuracy = val\_acc

# 评估

plt.plot(history['train\_acc'], label='train accuracy')

plt.plot(history['val\_acc'], label='validation accuracy')

plt.title('Training history')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend()

plt.ylim([0, 1]);