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

1.1 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')
```

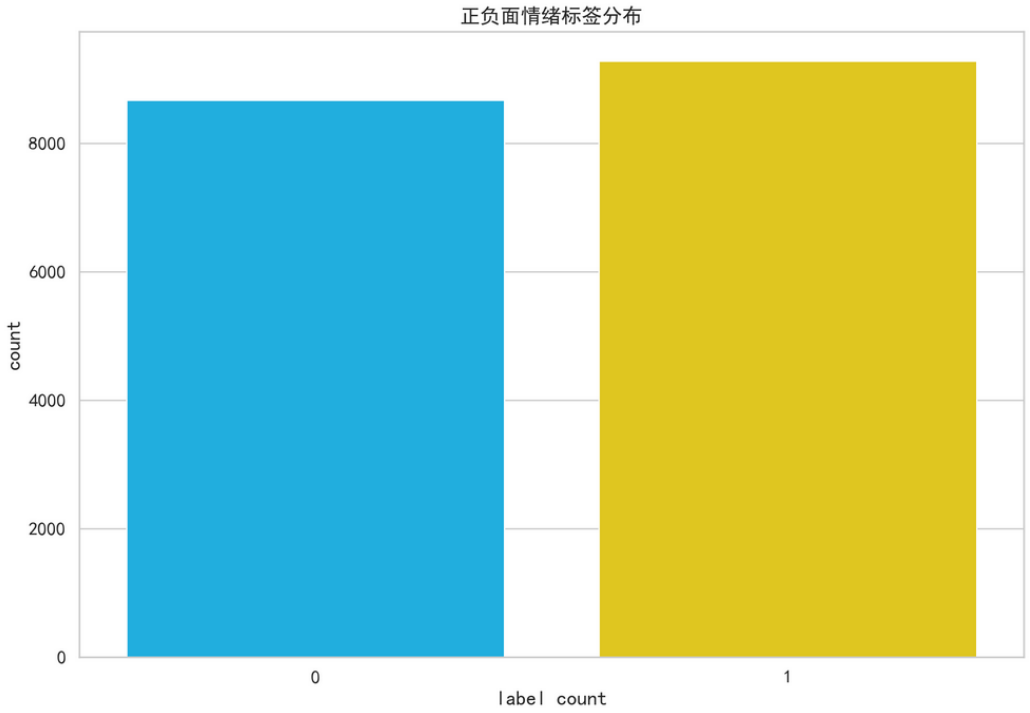

图 1 标签的分布

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')
```

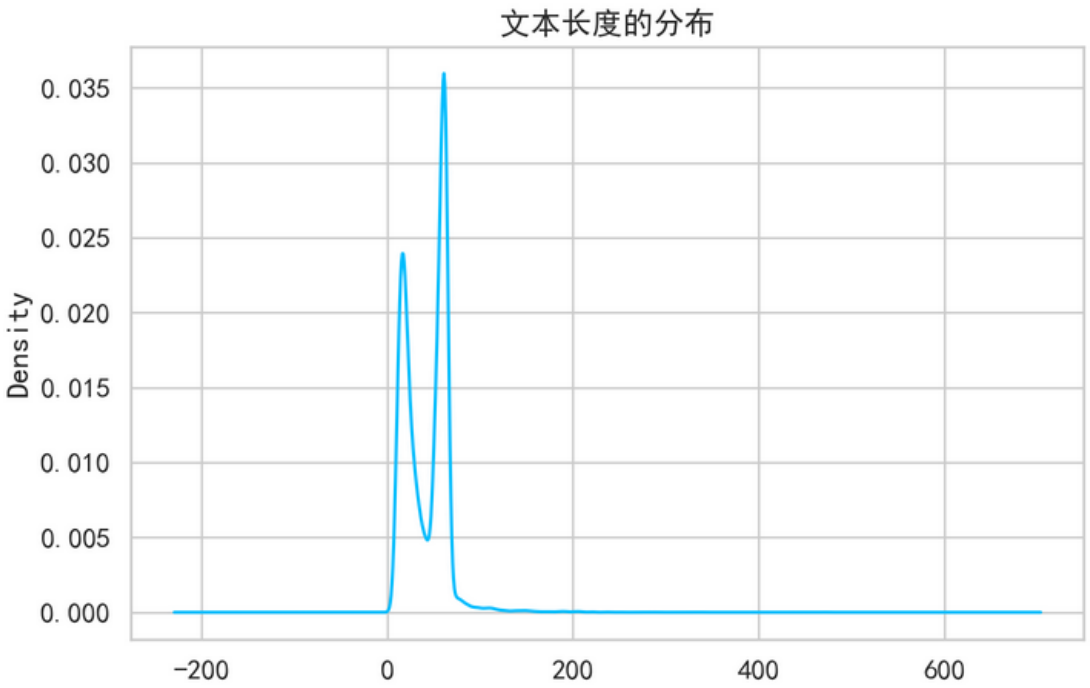

图 2 文本长度整体分布

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.ilo
c[:10])
plt.grid()
```

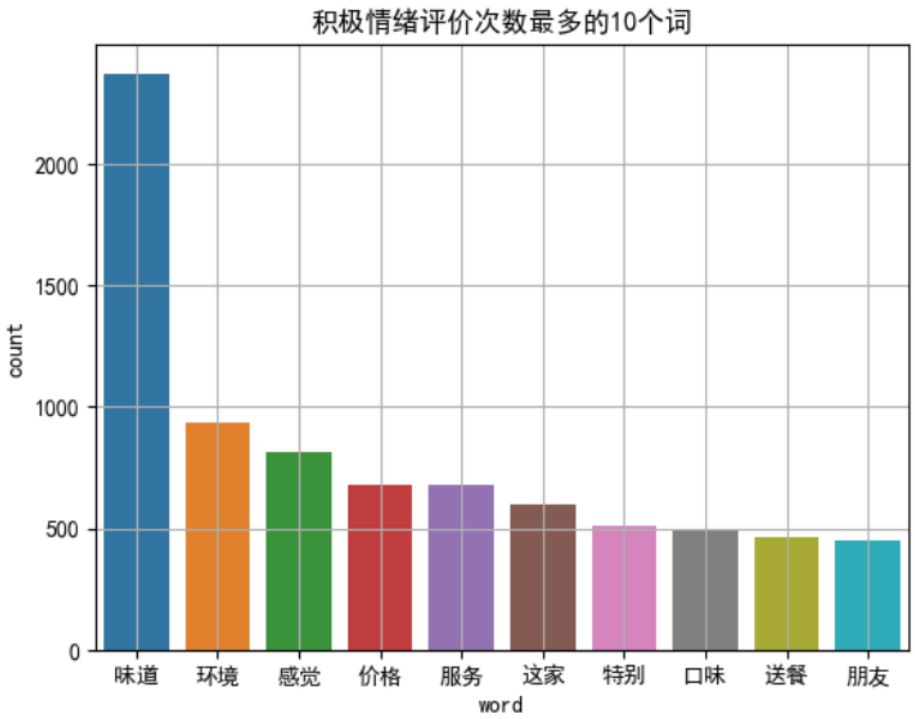

图 4 积极情绪评价次数最多的 10 个词

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()
```

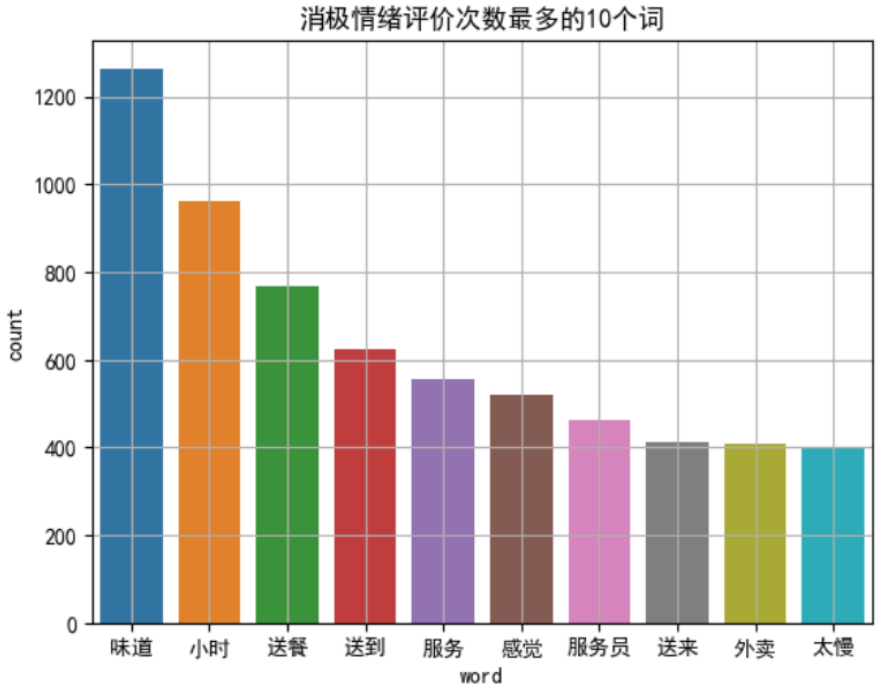

图 4 消极情绪评价次数最多的 10 个词

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().so
rt_index().tolist()[:24]
data['negative'] = train.groupby('target')['hour'].value_counts().so
rt_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()
```

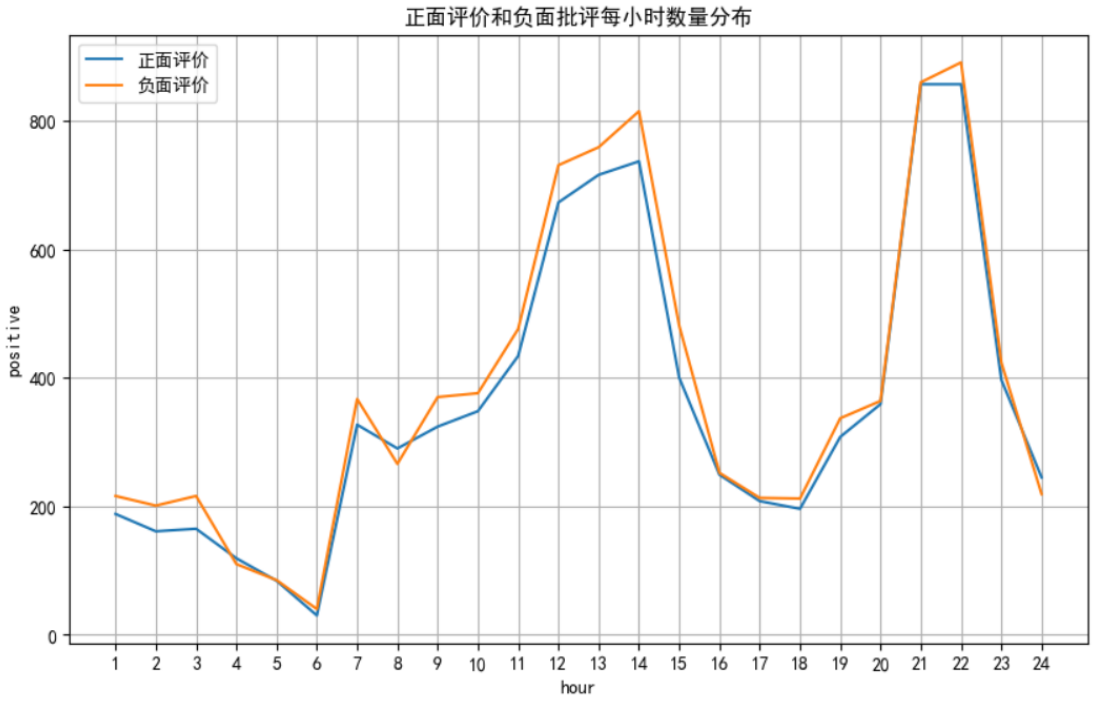

图 5 评价数量随小时的变化

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().sor
t_index().tolist()[:31]
data['negative'] = train.groupby('target')['day'].value_counts().sor
t_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()
```

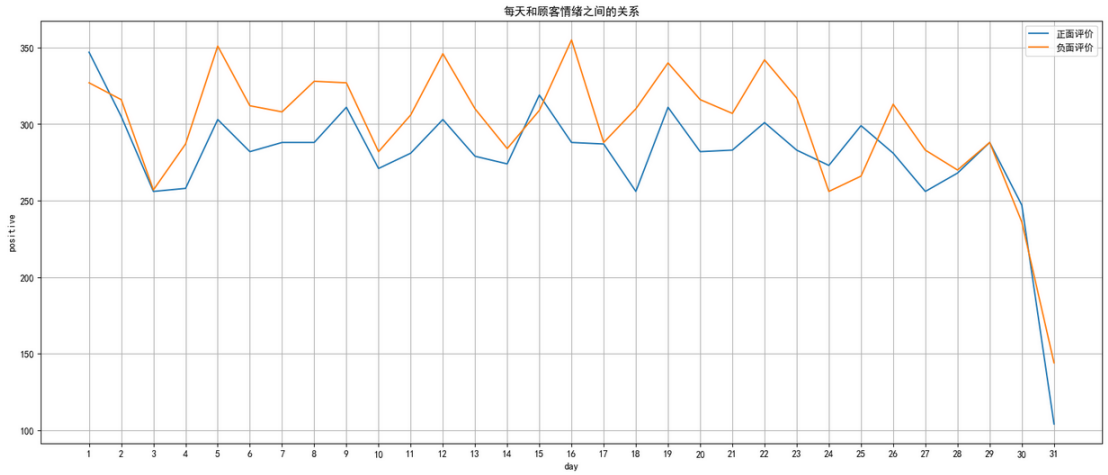

图 6 评论数量随天的变化

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 = nam
e_list,fc = 'b')
plt.legend()
plt.show()
```

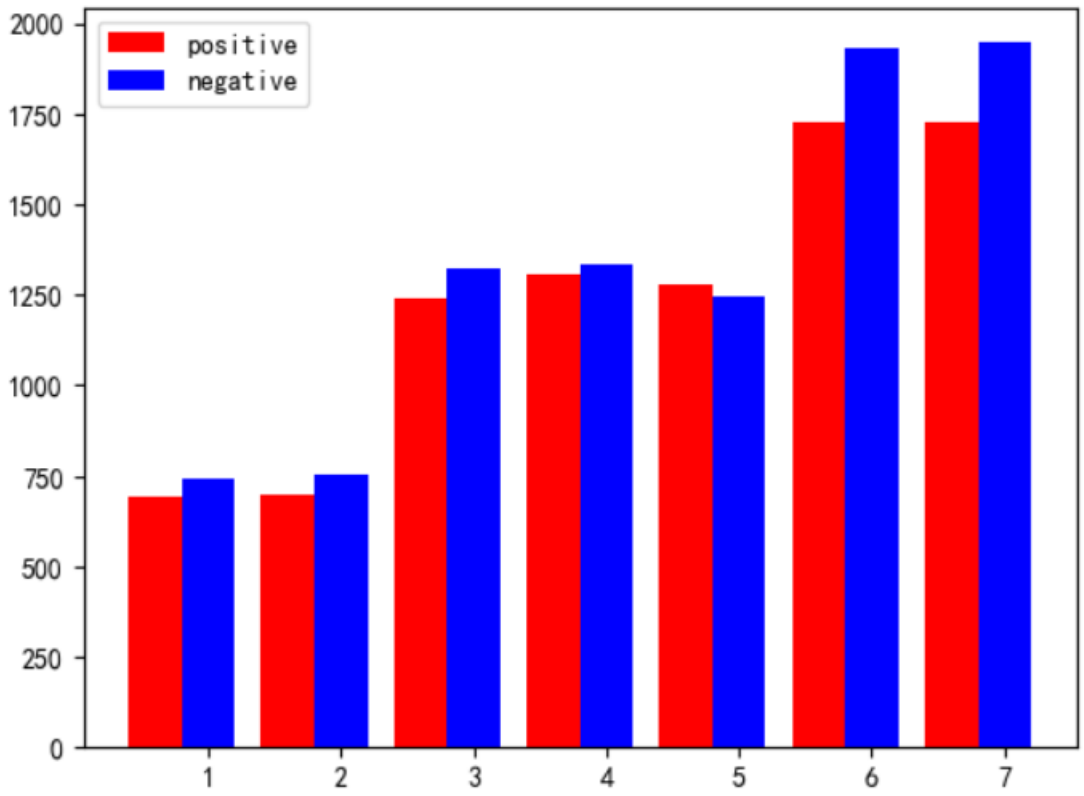

图 7 评论数量随星期的变化

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()
```

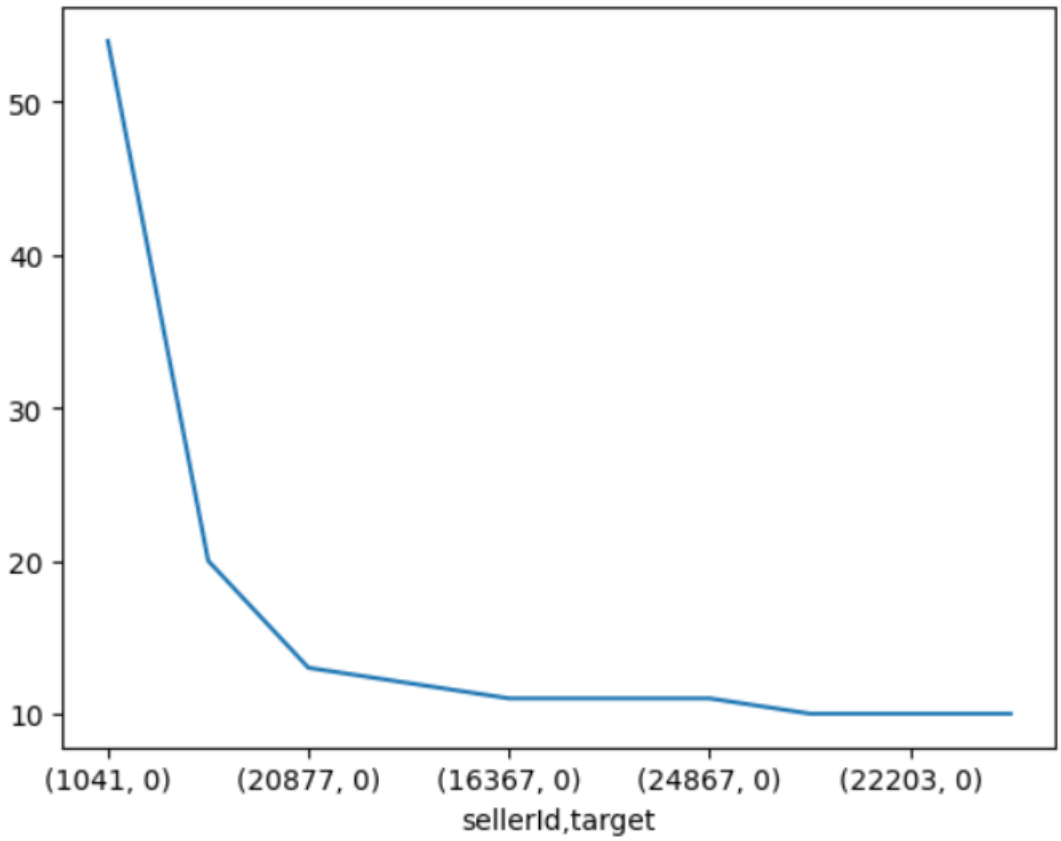

图 8 积极情绪商家排名展示

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：

$$\text{TF-IDF}(t,d,D) = \text{TF}(t,d) \times \text{IDF}(t,D)$$

其中，t表示一个词或词语，d表示文档，D表示文档集合，$\text{TF}(t,d)$表示词t在文档d中出现的频率，$\text{IDF}(t,D)$表示词t在文档集合D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：

$$\text{TF}(t,d) = \frac{n_t}{\sum_{i=1}^{n} n_i}$$

其中，n_t表示词t在文档d中出现的次数，n_i表示词i在文档d中出现的次数，n表示文档d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：

$$\text{TF-IDF}(t,d,D) = \frac{n_t}{\sum_{i=1}^{n} n_i} \times \log\left(\frac{|D|}{|d \in D: t \in d|}\right)$$

其中$|D|$表示文档集合D中的文档数量，$|d \in D: t \in d|$表示在文档集合D中包含词t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()
```

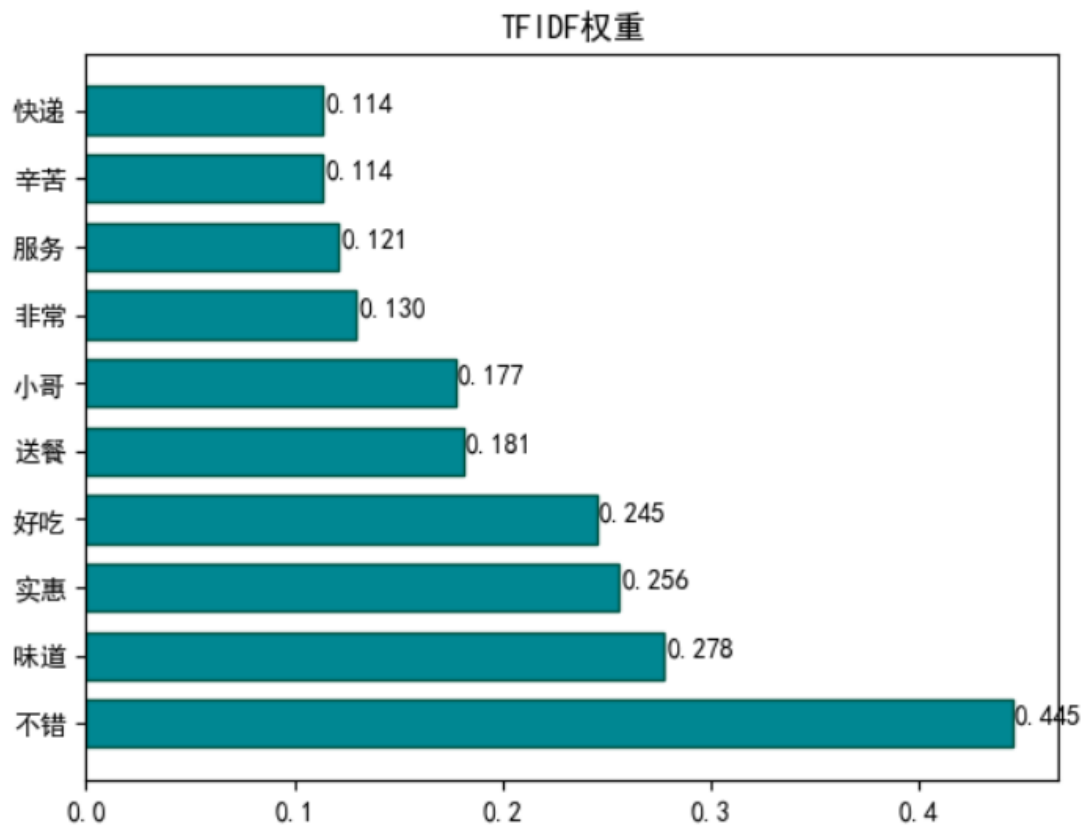

图 8 积极情绪商家排名展示

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()
```

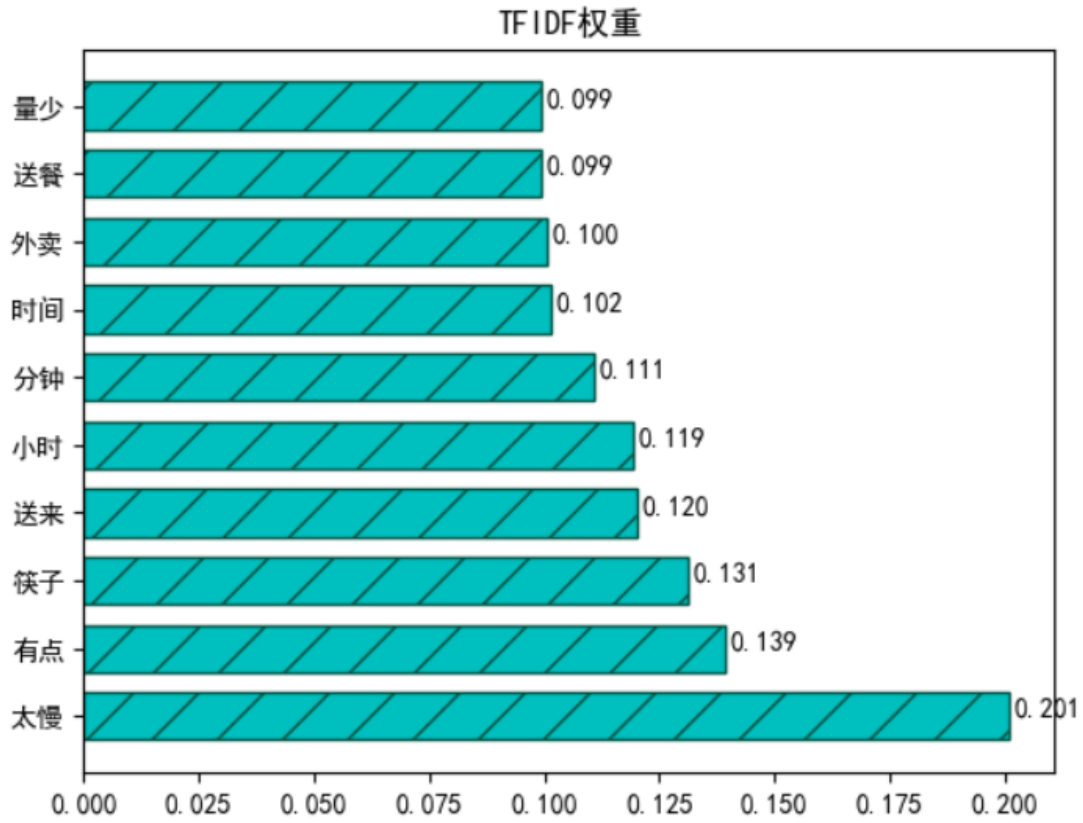

图 9 消极情绪商家排名展示

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');
```

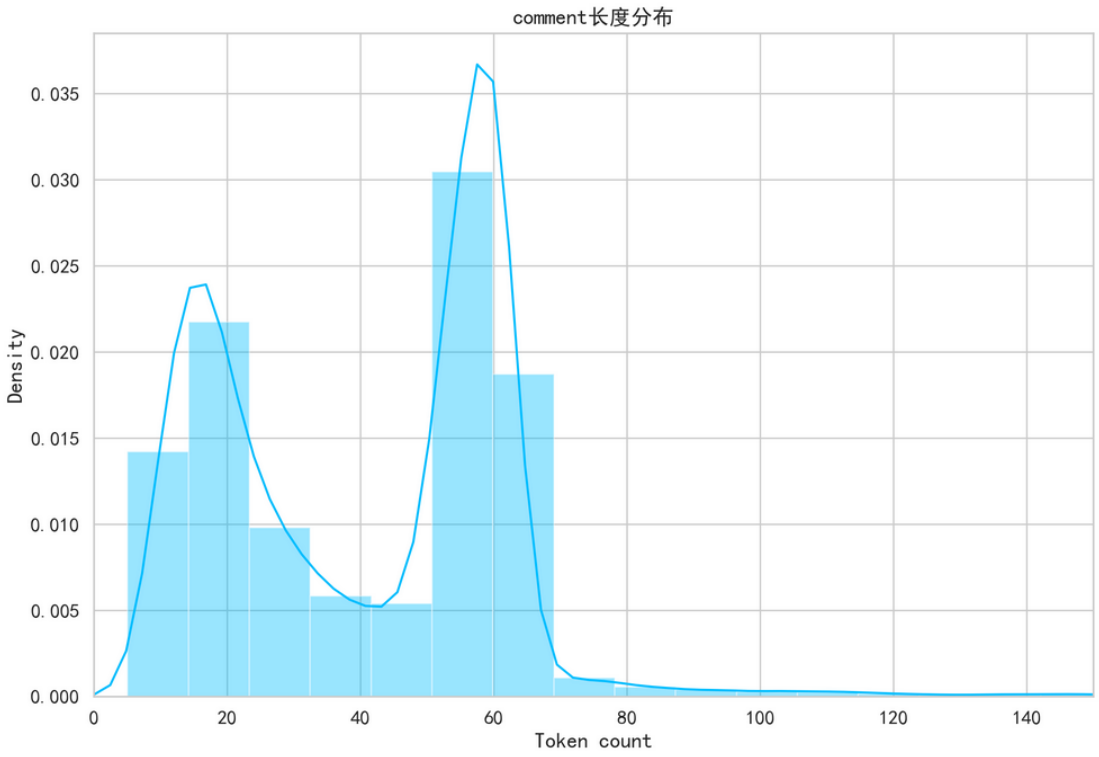

图 10 comment 长度分布

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_st
ate=RANDOM_SEED)
df_val, df_test = train_test_split(df_test, test_size=0.5, random_st
ate=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L_2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='tr
ain accuracy')
plt.plot(torch.tensor(history['val_acc'], device='cpu'), label='vali
dation accuracy')
plt.title('Training history')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
```

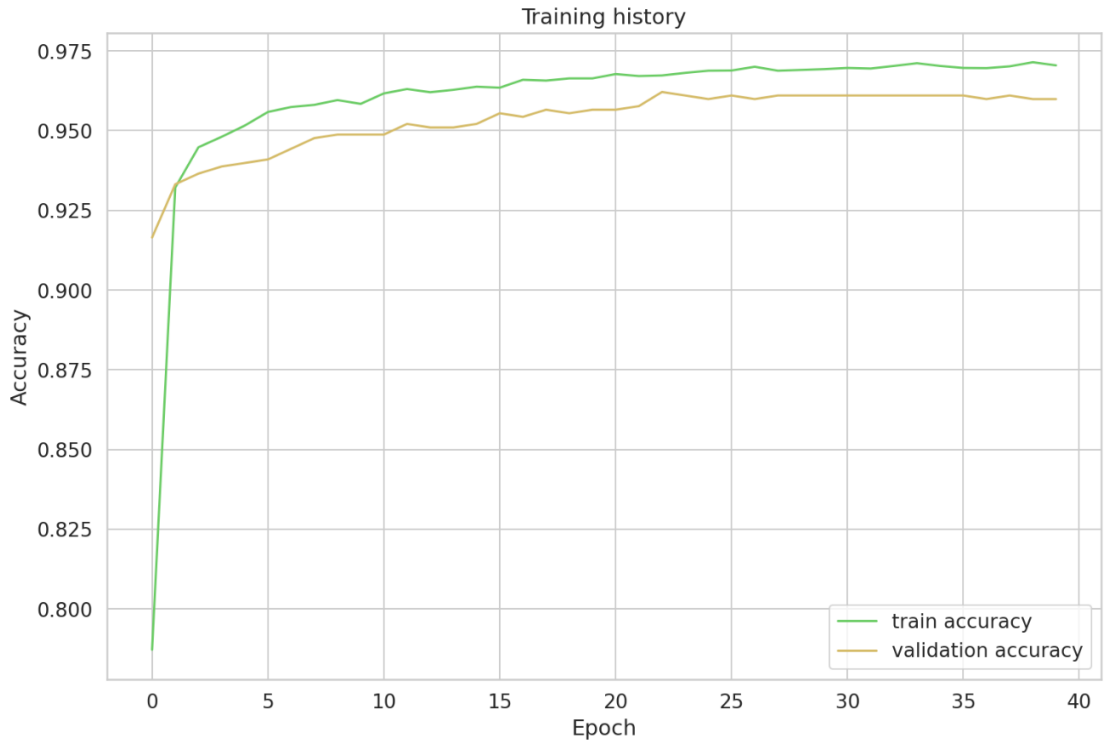

图 11 模型准确率曲线

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)
```

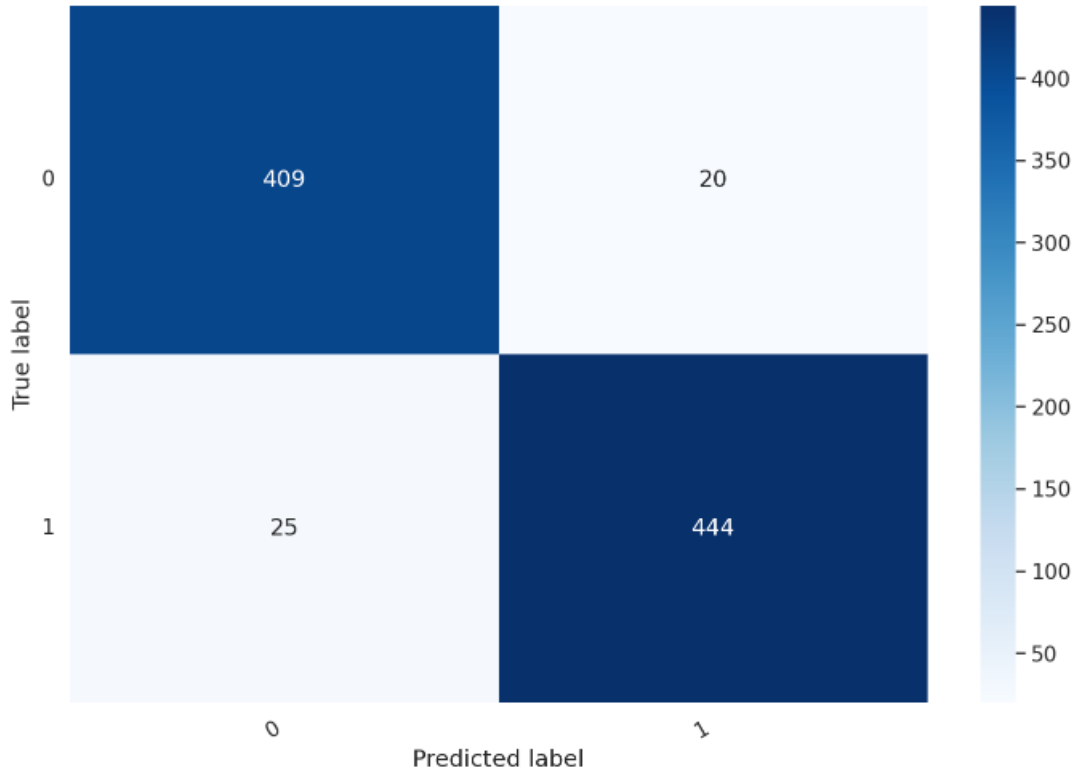

图 12 混淆矩阵

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_val
ues(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]);
```