数据分析之情感分析

案例背景

情感分析是分本分类中常见的应用场景。简单来说，就是从一段文本描述中，理解文本的感情色彩。常见的情感分析就是客户对商品或者服务的反馈，例如顾客对商品，酒店的评价等场景。在传统模式下，往往是通过人工对文本内容进行核对，从而将文本分为褒义，中性，贬义等。这种方式会消耗大量的人力资源，并且效率不高。

任务与实现

我们以分析电影评论为例。我们的任务在于，根据电影评论中的内容，进行文本预处理，建模等操作，从而可以识别评论的感情色彩，节省人力资源。

具体实现内容包括：

- 能够对文本数据进行预处理。【文本清洗，分词，去除停用词，文本向量化等操作。】
- 能够通过Python统计词频，生成词云图。【描述性统计分析】
- 能够通过方差分析，进行特征选择。【推断性统计分析】
- 能够根据文本向量，对文本数据进行分类。【朴素贝叶斯算法】

任务扩展

情感分析是文本分类的一种常见场景，因此，本案例的实现也可以应用到其他根据文本内容来实现分类的场景，例如，垃圾邮件过滤，新闻分类等。

数据集描述

数据集为电影《哪吒》的影评数据，包括如下信息：

- time：评论时间。
- city：城市。
- gender：性别。0：未知，1：男，2：女。
- name：昵称。
- level：评论者等级。
- score：评分，取值范围为[0, 5]。
- comment：评论内容。

加载数据

准备工作

- 下载Anaconda软件，请点击这里进行下载。
- 安装jieba库，命令如下：
  `pip install jieba`
- 安装wordcloud库，命令如下：
  `pip install wordcloud`

加载数据集

导入相关的库。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="darkgrid", font_scale=1.2)
plt.rcParams["font.family"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False
```

```
# 读取数据，昵称列对我们分析没有意义，直接去掉。
data = pd.read_csv("comment.csv", usecols=["time", "city", "gender", "level", "score", "comment"])
print(data.shape)
display(data.head())
```

```
(578760, 6)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	time	city	gender	level	score	comment
0	2019-08-13 14:53:57	杭州	1	1	5.0	很不错的电影
1	2019-08-13 14:53:49	上海	0	4	5.0	人物丰满 情节紧凑 难得的好片 喜欢!
2	2019-08-13 14:53:48	黔南	0	1	5.0	超级好看，哪吒和敖丙超级帅
3	2019-08-13 14:53:44	郑州	2	3	4.5	好看好看好看
4	2019-08-13 14:53:43	信宜	0	2	5.0	为国产动画电影打call，第一次给电影写评论，十分好评！哪吒给了我特效的惊喜，也给了我情感上...

数据预处理

文本数据

结构化数据与非结构化数据

结构化数据，是可以表示成多行多列的形式，并且，每行（列）都有着具体的含义。非结构化数据，无法合理的表示为多行多列的形式，即使那样表示，每行（列）也没有具体的含义。

文本数据预处理

文本数据，是一种非结构化数据。因此，其预处理的步骤与方式也会与结构化数据有所差异。文本数据预处理主要包含：

- 缺失值处理
- 重复值处理
- 文本内容清洗
- 分词
- 停用词处理

缺失值处理

检测缺失值，并对缺失值进行处理。

```
data.isnull().sum()
# data.info()
```

```
time         0
city       226
gender       0
level        0
score        0
comment      4
dtype: int64
```

“city”与“comment”存在缺失值，我们需要进行处理。

```
data["city"].fillna("未知", inplace=True)
# 删除没有评论的数据。
data.dropna(inplace=True)
data.isnull().sum()
```

```
time       0
city       0
gender     0
level      0
score      0
comment    0
dtype: int64
```

重复值处理

重复的数据，对文本分析与建模没有帮助，我们可以直接删除重复记录。

```
print(data.duplicated().sum())
display(data[data.duplicated()].iloc[:5])
```

```
224
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	time	city	gender	level	score	comment
2141	2019-08-13 11:53:54	佛山	2	1	5.0	好看，带儿子一起看的。画面制作精良，场面震撼。看到李靖愿拿换灵符替哪吒遭天雷劫，父母的爱终感...
6889	2019-08-12 21:43:45	汕头	1	1	5.0	***好看!
8218	2019-08-12 20:49:37	慈利	0	2	5.0	真的很满足 多次戳中泪点 真的很震撼 棒棒棒!👍
8897	2019-08-12 20:06:29	九江	2	2	5.0	二刷了！！！巨好看！！！！
9629	2019-08-12 19:19:28	东方	0	3	5.0	我看了两次!

删除重复的记录，并检测是否删除成功。

```
data.drop_duplicates(inplace=True)
print(data.duplicated().sum())
```

```
0
```

文本内容清洗

文本中的标点符号，与一些特殊字符，在我们当前的环境中，这些内容对文本分析的作用是不大的，可以将其去除。

去除文本中指定的字符，我们可以使用正则匹配的方式。给定如下的两种方案，我们会选择（ ）。

```
import re

pattern = r"[!\"#$%&'()*+,-./:;<=>?@[\\\]^_`{|}~—！，。？、￥…（）：【】《》‘’“”\s]+"
re_obj = re.compile(pattern)

# A方案
def clear(text):
    return re.sub(pattern, "", text)

# B方案
def clear(text):
    return re_obj.sub("", text)

data["comment"] = data["comment"].apply(clear)
```

A A方案，B方案无法满足需求。
B B方案，A方案无法满足需求。
C A与B两个方案都可以，没有谁优先。
D A与B两个方案都可以，但优先选择A方案。
E A与B两个方案都可以，但优先选择B方案。

```
import re

re_obj = re.compile(
    r"[!\"#$%&'()*+,-./:;<=>?@[\\\]^_`{|}~—！，。？、￥…（）：【】《》‘’“”\s]+")


def clear(text):
    return re_obj.sub("", text)


data["comment"] = data["comment"].apply(clear)
data.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	time	city	gender	level	score	comment
0	2019-08-13 14:53:57	杭州	1	1	5.0	很不错的电影
1	2019-08-13 14:53:49	上海	0	4	5.0	人物丰满情节紧凑难得的好片喜欢
2	2019-08-13 14:53:48	黔南	0	1	5.0	超级好看哪吒和敖丙超级帅
3	2019-08-13 14:53:44	郑州	2	3	4.5	好看好看好看
4	2019-08-13 14:53:43	信宜	0	2	5.0	为国产动画电影打call第一次给电影写评论十分好评哪吒给了我特效的惊喜也给了我情感上的感动我...

分词

分词是将连续的文本，分割成语义合理的若干词汇序列。对于英文来说，分词是非常容易的，但是中文分词会有一定的难度。我们可以通过jieba来实现分词的功能。

```
import jieba

s = "今天，外面下了一场很大的雨。"
# cut与lcut的区别：前者返回生成器，后者返回列表。
words = jieba.cut(s)
print(words)
print(list(words))
words = jieba.lcut(s)
print(words)
```

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\77981\AppData\Local\Temp\jieba.cache
```

```
<generator object Tokenizer.cut at 0x000002597D833C48>
```

```
Loading model cost 0.610 seconds.
Prefix dict has been built successfully.
```

```
['今天', '，', '外面', '下', '了', '一场', '很大', '的', '雨', '。']
['今天', '，', '外面', '下', '了', '一场', '很大', '的', '雨', '。']
```

我们对评论内容列（comment）进行分词处理，给定如下两种方案，我们应该选择（ ）。

```python
# A方案
def cut_word(text):
    return jieba.cut(text)

# B方案
def cut_word(text):
    return jieba.lcut(text)


data["comment"] = data["comment"].apply(cut_word)
```

A A方案，B方案无法满足需求。
B B方案，A方案无法满足需求。
C A与B两个方案都可以，没有谁优先。
D A与B两个方案都可以，但优先选择A方案。
E A与B两个方案都可以，但优先选择B方案。

```python
def cut_word(text):
    return jieba.cut(text)


data["comment"] = data["comment"].apply(cut_word)
data.sample(5)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	time	city	gender	level	score	comment
440499	2019-07-29 13:51:43	商丘	2	3	4.5	<generator object Tokenizer.cut at 0x000002591...
226022	2019-08-04 12:47:10	成都	2	2	5.0	<generator object Tokenizer.cut at 0x000002591...
518287	2019-07-27 19:59:40	东阳	0	1	4.5	<generator object Tokenizer.cut at 0x000002591...
22907	2019-08-11 22:02:38	汕头	0	2	4.5	<generator object Tokenizer.cut at 0x000002597...
248068	2019-08-03 20:01:23	泸州	0	2	5.0	<generator object Tokenizer.cut at 0x000002591...

停用词处理

在讲解停用词之前，我们来看这样的一个练习。

“然而，无论如何，我写作业了。”最能体现这句话核心含义的词语是什么?
A 然而 了
B 无论如何 写 作业
C 无论如何 写 了
D 我 写 作业

停用词，指的是在我们语句中大量出现，但却对语义分析没有帮助的词。对于这样的词汇，我们通常可以将其删除，这样的好处在于：

1. 可以降低存储空间消耗。
2. 可以减少计算时间消耗。

对于哪些词属于停用词，已经有统计好的停用词列表，我们直接使用就好。

我们对评论内容列（comment）去除停用词，对比以下两种方案，我们会选择（ ）。

```python
# A方案，使用set。
def get_stopword():
    s = set()
    with open("stopword.txt", encoding="UTF-8") as f:
        for line in f:
            s.add(line.strip())
    return s

# B方案，使用list。
def get_stopword():
    s = list()
    with open("stopword.txt", encoding="UTF-8") as f:
        for line in f:
            s.append(line.strip())
    return s

def remove_stopword(words):
    return [word for word in words if word not in stopword]

stopword = get_stopword()
data["comment"] = data["comment"].apply(remove_stopword)
```

A A方案，B方案无法满足需求。
B B方案，A方案无法满足需求。
C A与B两个方案都可以，没有谁优先。
D A与B两个方案都可以，但优先选择A方案。
E A与B两个方案都可以，但优先选择B方案。

```python
def get_stopword():
    s = set()
    with open("stopword.txt", encoding="UTF-8") as f:
        for line in f:
            s.add(line.strip())
    return s


def remove_stopword(words):
    return [word for word in words if word not in stopword]


stopword = get_stopword()
data["comment"] = data["comment"].apply(remove_stopword)
data.sample(5)
```

```css
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

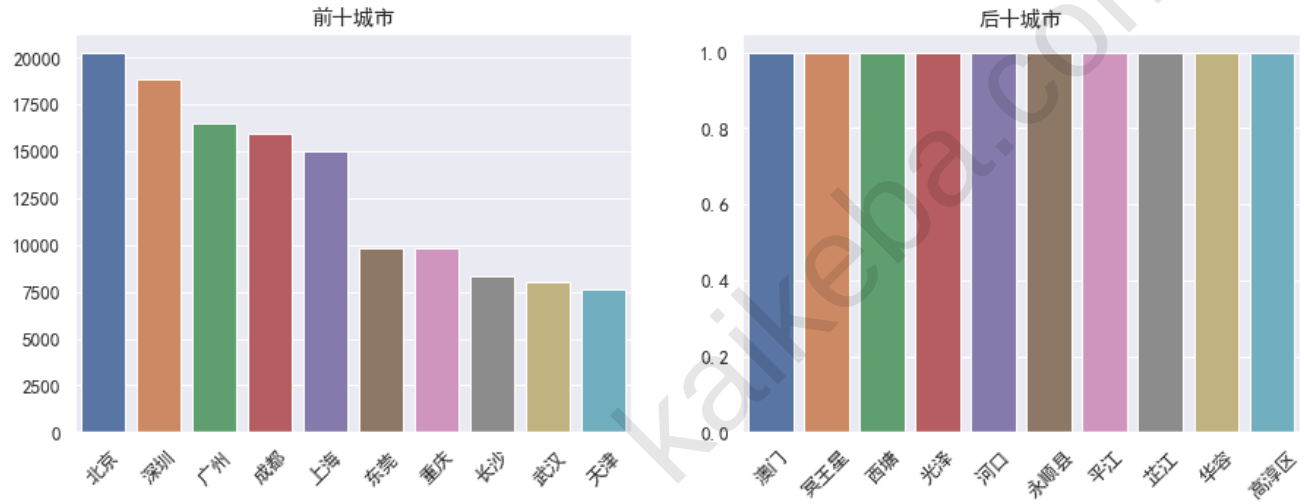
	time	city	gender	level	score	comment
327007	2019-08-01 15:36:44	西安	0	1	5.0	[敖丙好帅]
281832	2019-08-02 21:39:28	于都	0	2	5.0	[全程, 飙泪, 太, 感动]
502078	2019-07-28 08:37:51	驻马店	0	2	5.0	[天呀, 看太, 好看, 想二刷]
31347	2019-08-11 17:26:35	当阳	0	1	5.0	[天, 太, 好看, 辽, 😍, 👍, 哪吒, 敖丙是, 同人, cp, 真的, 兄弟, 李...
426844	2019-07-29 19:18:06	长春	1	4	5.0	[好看, 岁, 小孩, 小家伙, 看过, 哪吒, 传奇, 第一次, 接触, 哪吒, 尾, 一...

数据基本分析

用户所在城市

统计在所包含的用户中，数量最多与最少的10个城市。

```
fig, ax = plt.subplots(1, 2)
fig.set_size_inches(15, 5)
count = data["city"].value_counts()
top = count.iloc[:10]
bottom = count.iloc[-10:]
for index, d, title in zip(range(2), [top, bottom], ["前十城市", "后十城市"]):
    a = sns.barplot(d.index, d.values, ax=ax[index])
    # 旋转45度，避免字体重叠。
    a.set_xticklabels(a.get_xticklabels(), rotation=45)
    a.set_title(title)
```

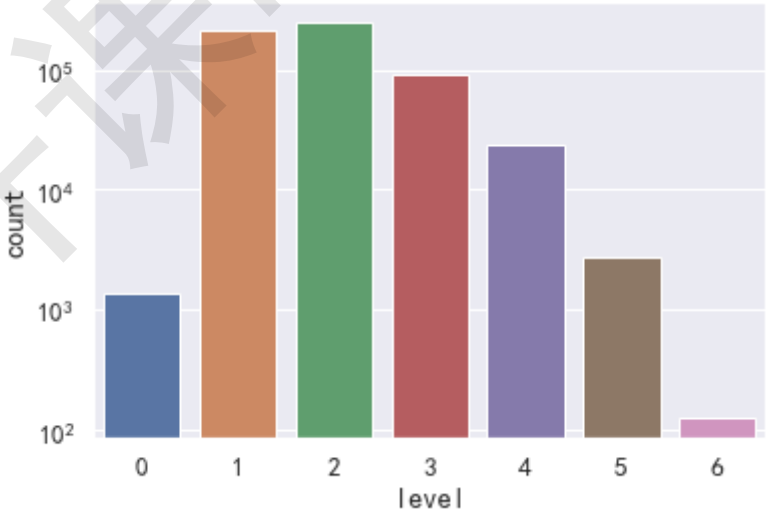


用户等级

绘制用户每个等级的数量。

```
sns.countplot(x="level", data=data, log=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x259779033c8>
```



课堂练习

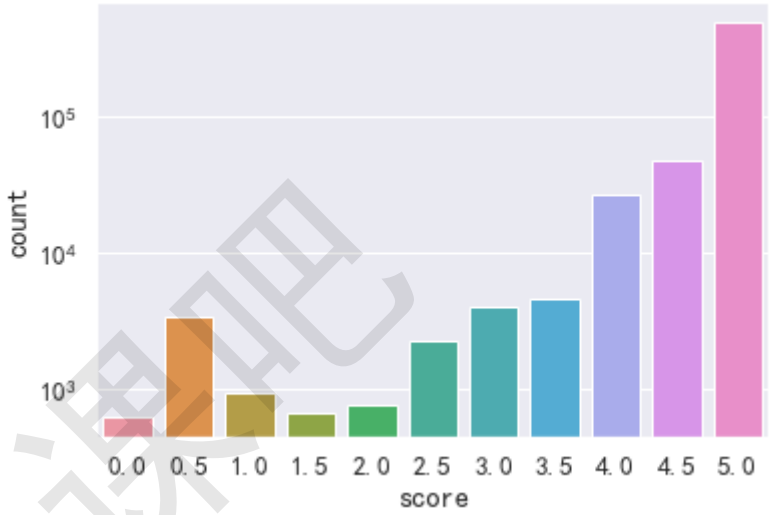
在上例中，我们为什么要使用log参数?
A 不使用也可以，没有什么影响。
B 避免数值的数量级过大。
C 避免数值相差很大时，显示效果较差。
D 如果不使用，运行会产生错误。

评分

绘制每个评分的数量。

```
sns.countplot(x="score", data=data, log=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2590072fd88>
```

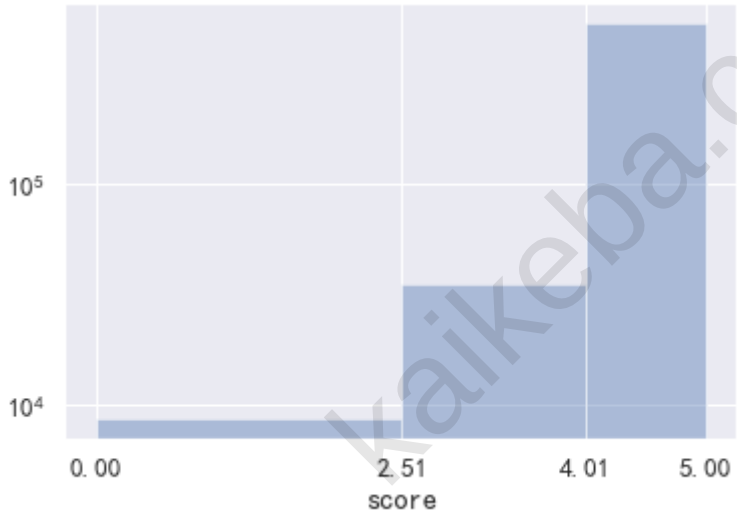


将评分分为三个等级，绘制直方图。

- 好评[4.5, 5]。
- 中评[3, 4]。
- 差评[0, 2.5]。

```
data_period = [0, 2.51, 4.01, 5]
# 也可以使用matplotlib来绘制直方图。
# plt.hist(data["score"], bins=data_period, log=True)
# bins指定的区间为左闭右开，最后一个区间为双闭。
ax = sns.distplot(data["score"], bins=data_period, hist_kws={"log": True}, kde=False)
# 让x轴显示的数值与bins指定的区间一致。
ax.set_xticks(data_period)
```

```
[<matplotlib.axis.XTick at 0x2590040e908>,
 <matplotlib.axis.XTick at 0x25900411f48>,
 <matplotlib.axis.XTick at 0x259004111c8>,
 <matplotlib.axis.XTick at 0x25900434648>]
```

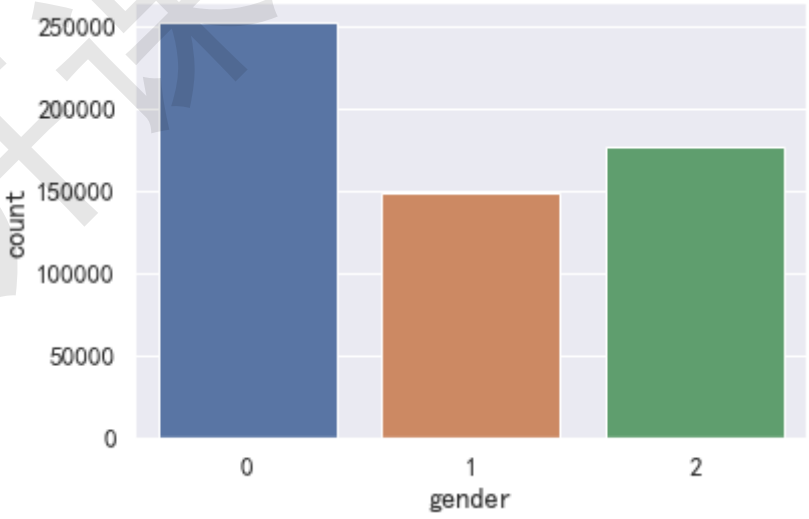


性别对比

统计评论用户中，性别的对比。

```
# 0：未知，1：男，2：女。
sns.countplot(x="gender", data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25907228888>
```

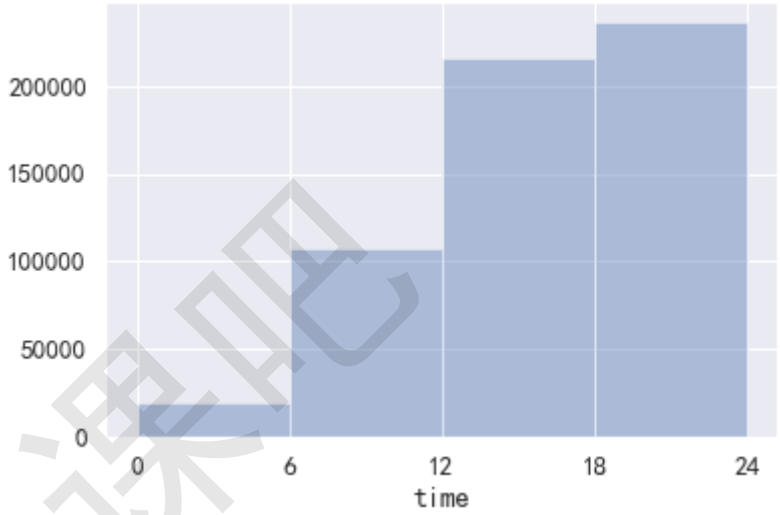


评论时间分布

我们提取用户在写影评时的时间，分析影评用户主要活跃的时间段。

```
# 时间列的格式：2019-07-31 21:21:02
hour = data["time"].str.extract(r" (\d{2}):", expand=False)
hour = hour.astype(np.int32)
# 直方图的区间的前闭后开，最后一个区间双闭。
time_period = [0, 6, 12, 18, 24]
ax = sns.distplot(hour, bins=time_period, kde=False)
ax.set_xticks(time_period)
```

```
[<matplotlib.axis.XTick at 0x259071f7448>,
 <matplotlib.axis.XTick at 0x259071f1c48>,
 <matplotlib.axis.XTick at 0x259003f3388>,
 <matplotlib.axis.XTick at 0x25919897e08>,
 <matplotlib.axis.XTick at 0x2591989d248>]
```



词汇统计

词汇频数统计

统计在所有评论中，出现频数最多的N个词汇。

```
from itertools import chain
from collections import Counter

li_2d = data["comment"].tolist()
# 将二维列表扁平化为一维列表。
li_1d = list(chain.from_iterable(li_2d))
print(f"总词汇量：{len(li_1d)}")
c = Counter(li_1d)
print(f"不重复词汇数量：{len(c)}")
common = c.most_common(15)
print(common)
```

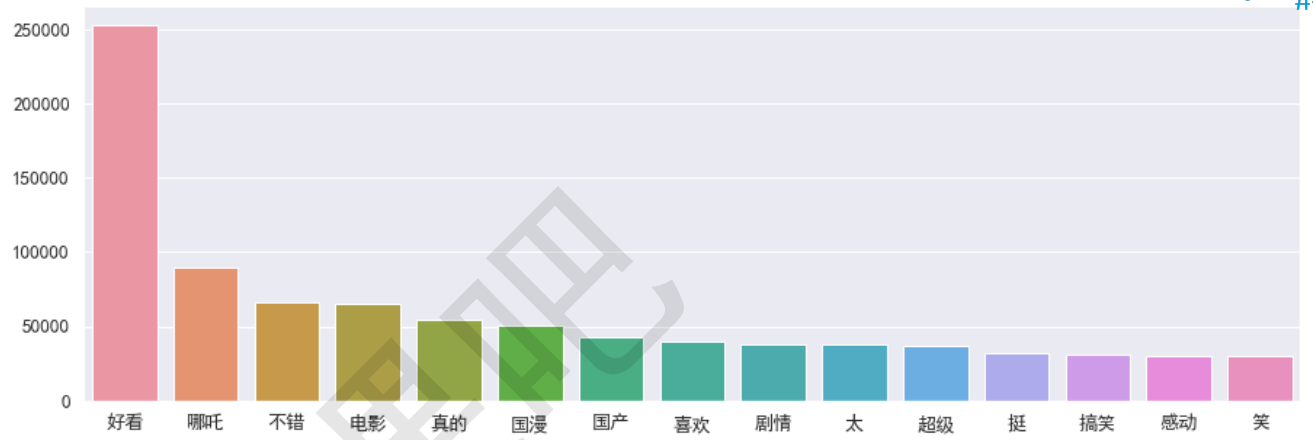
```
总词汇量：4107951
不重复词汇数量：86474
[('好看', 253127), ('哪吒', 89622), ('不错', 66558), ('电影', 64864), ('真的', 54514),
('国漫', 50663), ('国产', 43181), ('喜欢', 39873), ('剧情', 37878), ('太', 37583), ('超
级', 36612), ('挺', 31547), ('搞笑', 31235), ('感动', 30276), ('笑', 29937)]
```

可视化

将之前统计词汇的频数，以柱形图的形式可视化。

```
d = dict(common)
plt.figure(figsize=(15, 5))
# seaborn内部不支持dict_keys与dict_values类型，maplotlib.bar可以。
sns.barplot(list(d.keys()), list(d.values()))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x259072003c8>
```



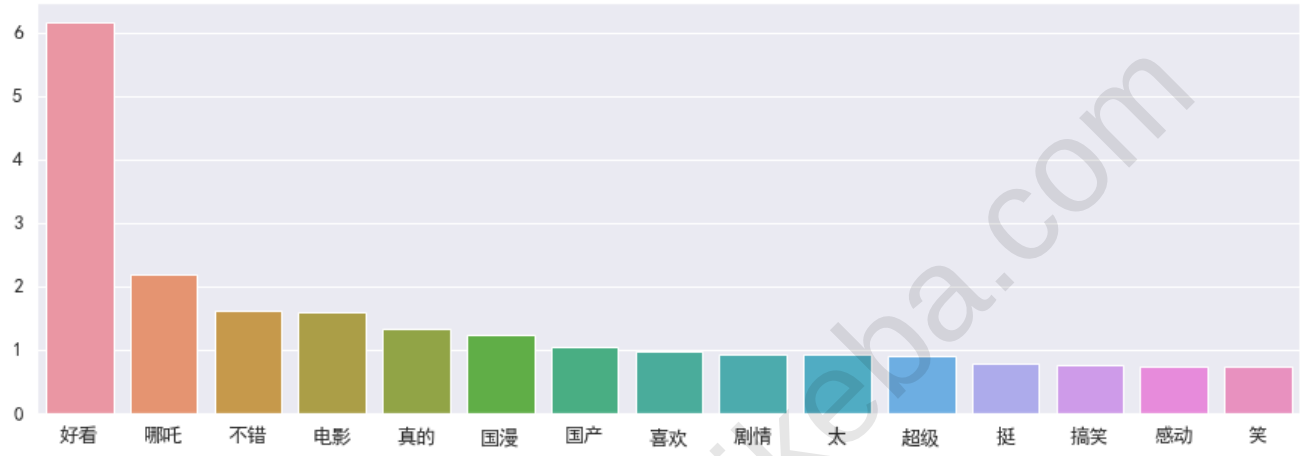
频率统计

将之前统计词汇的频数，转换成频率显示。

```
total = len(li_1d)
percentage = [v * 100 / total for v in d.values()]
print([f"{v:.2f}%" for v in percentage])
plt.figure(figsize=(15, 5))
sns.barplot(list(d.keys()), percentage)
```

```
['6.16%', '2.18%', '1.62%', '1.58%', '1.33%', '1.23%', '1.05%', '0.97%', '0.92%',
'0.91%', '0.89%', '0.77%', '0.76%', '0.74%', '0.73%']
```

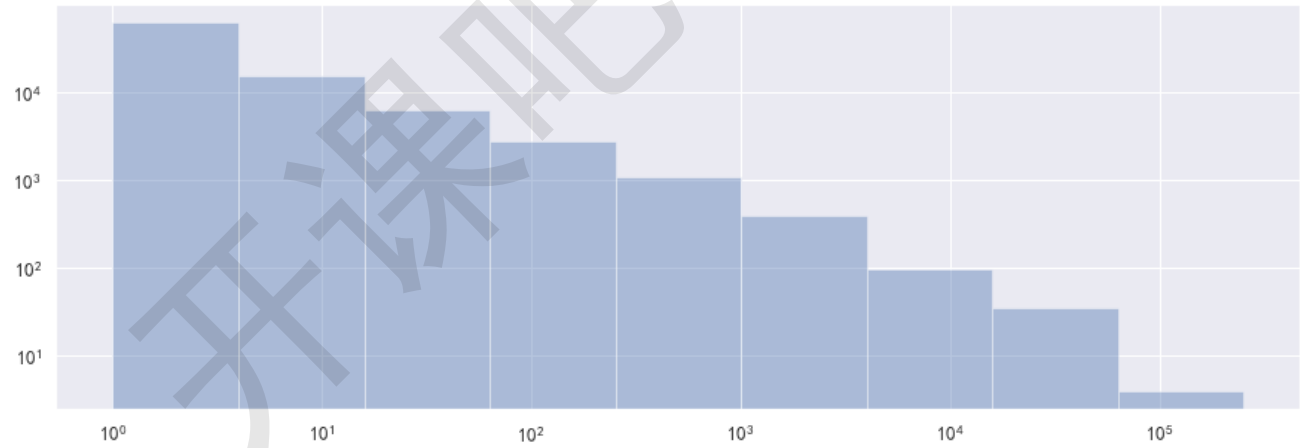
```
<matplotlib.axes._subplots.AxesSubplot at 0x259071f7c48>
```



频数分布统计

绘制所有词汇的频数分布直方图。

```
plt.figure(figsize=(15, 5))
v = list(c.values())
end = np.log10(max(v))
ax = sns.distplot(v, bins=np.logspace(0, end, num=10), hist_kws={"log": True},
kde=False)
ax.set_xscale("log")
```

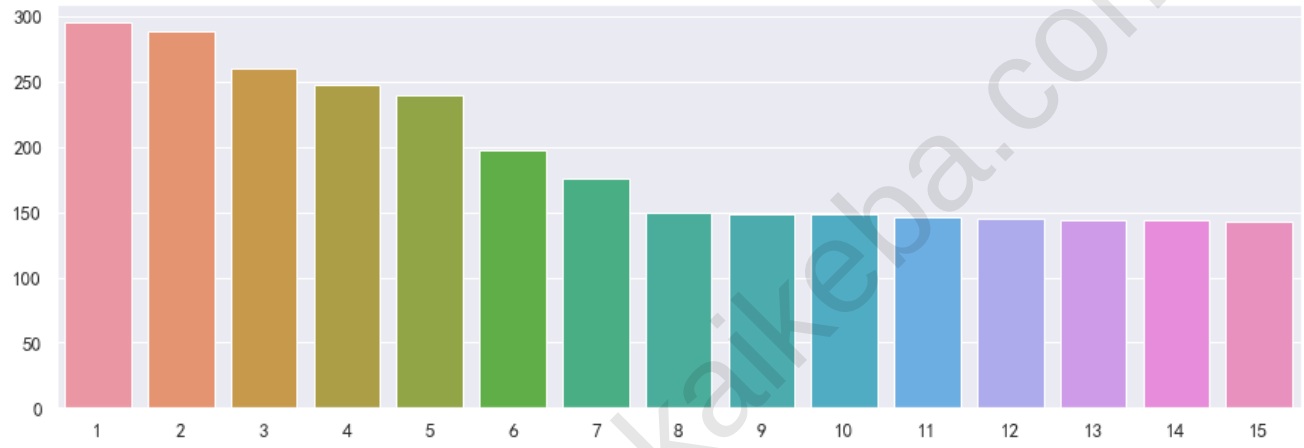


评论词汇长度统计

统计每个用户评论使用的词汇数量。并绘制用词最多的前N个评论。

```
plt.figure(figsize=(15, 5))
# 计算每个评论的用词数。
num = [len(li) for li in li_2d]
# 统计用词最多的前15个评论。
length = 15
sns.barplot(np.arange(1, length + 1), sorted(num, reverse=True)[:length])
```

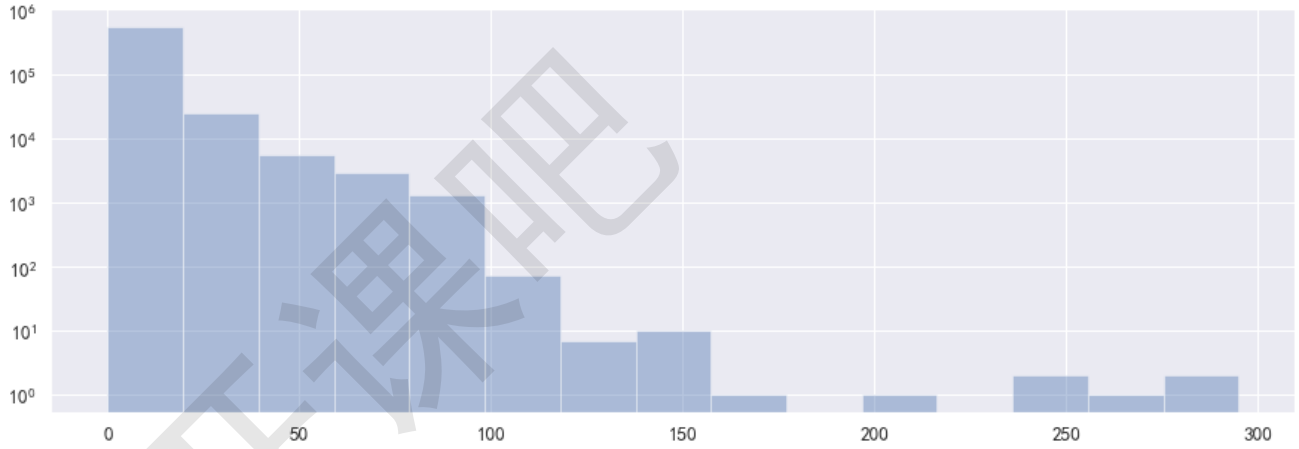
```
<matplotlib.axes._subplots.AxesSubplot at 0x25912360708>
```



评论词汇长度分布统计

统计所有用户在评论时所使用的词汇数，绘制直方图。

```
plt.figure(figsize=(15, 5))
ax = sns.distplot(num, bins=15, hist_kws={"log": True}, kde=False)
```



生成词云图

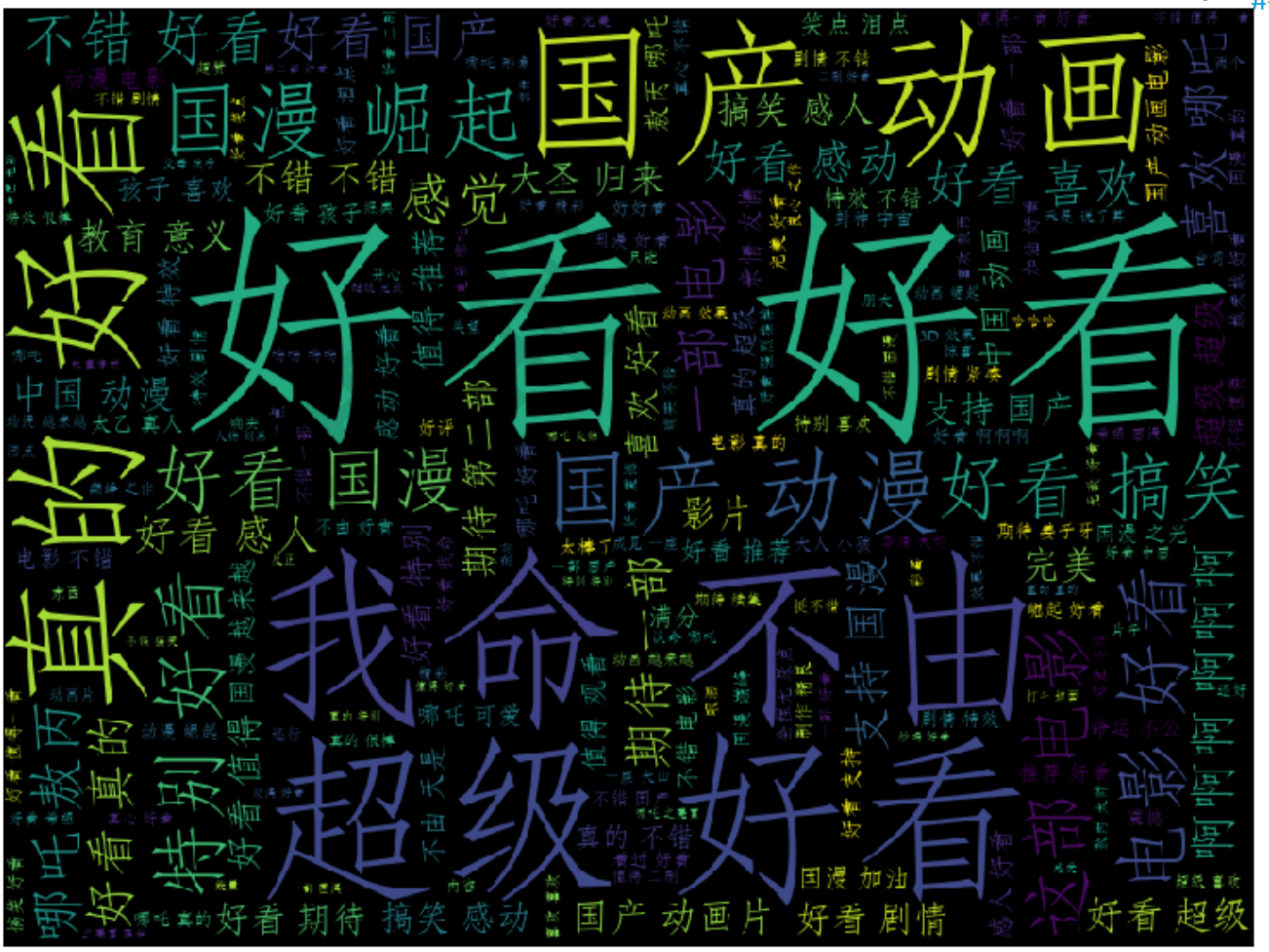
Python中，wordcloud模块提供了生成词云图的功能，我们可以使用该模块生成词云图。该模块并非Anaconda默认的模块，需要独立安装后才能使用。

标准词云图

```
from wordcloud import WordCloud

# 需要指定字体的位置，否则中文无法正常显示。
wc = WordCloud(font_path=r"C:/Windows/Fonts/STFANGSO.ttf", width=800, height=600)
# WordCloud要求传递的词汇使用空格分开的字符串。
join_words = " ".join(li_1d)
img = wc.generate(join_words)
plt.figure(figsize=(15, 10))
plt.imshow(img)
plt.axis('off')
# 将图像保存到本地。
wc.to_file("wordcloud.png")
```

```
<wordcloud.wordcloud.WordCloud at 0x259137e6d48>
```



自定义背景

此外，我们还可以使用指定的图片作为背景，生成词云图。

```
wc = WordCloud(font_path=r"C:/Windows/Fonts/STFANGSO.ttf",
mask=plt.imread("../imgs/map.jpg"))
img = wc.generate(join_words)
plt.figure(figsize=(15, 10))
plt.imshow(img)
plt.axis('off')
```

```
(-0.5, 1169.5, 965.5, -0.5)
```

课堂练习

之前生成的词云图，有什么问题吗?

A 没有问题。

B 好像有点问题。

另类词云图

通过WordCloud对象的generate方法生成的词云图，词汇的大小并不是严格根据词频数量显示的。如果需要传统意义的词云图，可以调用generate_from_frequencies方法。

```
plt.figure(figsize=(15, 10))
img = wc.generate_from_frequencies(c)
plt.imshow(img)
plt.axis('off')
```

```
(-0.5, 1169.5, 965.5, -0.5)
```

文本向量化

对文本数据进行建模，有两个的问题需要解决：

1. 模型进行的是数学运算，因此需要数值类型的数据，而文本不是数值类型数据。
2. 模型需要结构化数据，而文本是非结构化数据。

将文本转换为数值特征向量的过程，称为**文本向量化**。将文本向量化，可以分为如下步骤：

1. 对文本分词，拆分成更容易处理的单词。
2. 将单词转换为数值类型，即使用合适的数值来表示每个单词。

同样，需要注意的是，文本是非结构化数据，在向量化过程中，需要将其转换为结构化数据。

词袋模型

词袋模型，直观上理解，就是一个装满单词的袋子。实际上，词袋模型是一种能够将文本向量化的方式。在词袋模型中，每个文档为一个样本，每个不重复的单词为一个特征，单词在文档中出现的次数作为特征值。例如，给定如下的文档集：

文档
Where there is a will, there is a way.
There is no royal road to learning.

如果转换为词袋模型，则结果为：

is	learning	no	road	royal	there	to	way	where	will
2	0	0	0	0	2	0	1	1	1
1	1	1	1	1	1	1	0	0	0

这样，我们就成功对文档实现了向量化的操作，同时，运用词袋模型，我们也将文本数据转换为结构化数据。

```
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer()
docs = [
    "Where there is a will, there is a way.",
    "There is no royal road to learning.",
]
bag = count.fit_transform(docs)
# bag是一个稀疏的矩阵。
print(bag)
# 调用稀疏矩阵的toarray方法，将稀疏矩阵转换为ndarray对象（稠密矩阵）。
print(bag.toarray())
```

```
  (0, 8)    1
  (0, 5)    2
  (0, 0)    2
  (0, 9)    1
  (0, 7)    1
  (1, 5)    1
  (1, 0)    1
  (1, 2)    1
  (1, 4)    1
  (1, 3)    1
  (1, 6)    1
  (1, 1)    1
[[2 0 0 0 0 2 0 1 1 1]
 [1 1 1 1 1 1 1 0 0 0]]
```

这里需要留意的是，默认情况下，CountVectorizer只会对字符长度不小于2的单词进行处理，如果单词长度小于2（单词仅有一个字符），则会忽略该单词，例如，上例中的单词“a”，并没有作为特征进行向量化。

```
# 获取每个特征对应的单词。
print(count.get_feature_names())
# 输出单词与编号的映射关系。
print(count.vocabulary_)
```

```
['is', 'learning', 'no', 'road', 'royal', 'there', 'to', 'way', 'where', 'will']
{'where': 8, 'there': 5, 'is': 0, 'will': 9, 'way': 7, 'no': 2, 'royal': 4, 'road': 3, 'to': 6, 'learning': 1}
```

经过训练后，CountVectorizer就可以对未知文档（训练集外的文档）进行向量化。当然，向量化的特征仅为训练集中出现的单词特征，如果未知文档中的单词不在训练集中，则在词袋模型中无法体现。

```
test_docs = ["While there is life there is hope.", "No pain, no gain."]
t = count.transform(test_docs)
print(t.toarray())
```

```
[[2 0 0 0 0 2 0 0 0 0]
 [0 0 2 0 0 0 0 0 0 0]]
```

从结果可知，像第2个文档中，“pain”等词汇，在训练集中是没有出现的，而文本向量化是根据训练集中出现的单词作为特征，因此，这些词汇在转换的结果中无法体现。

CountVectorizer在实现上，将文档向量化为稀疏矩阵，那是否可以将文档向量化为ndarray数组（稠密矩阵）呢?
A 可以，这样做与稀疏矩阵差不多。
B 可以，但是这样做没有稀疏矩阵效果好。
C 不可以，内部实现的限制，无法完成。
D 不可以，这样做会产生安全隐患。

TF-IDF

通过CountVectorizer类，我们能够将文档向量化处理。在向量化过程中，我们使用每个文档中单词的频数作为对应特征的取值。这是合理的，因为，单词出现的次数越多，我们就认为该单词理应比出现次数少的单词更加重要。

然而，这是相对的，有些单词，我们不能仅以当前文档中的频数来进行衡量，还要考虑其在语料库中，在其他文档中出现的次数。因为有些单词，确实是非常常见的，其在语料库所有的文档中，可能都会频繁出现，对于这样的单词，我们就应该降低其重要性。例如，在新闻联播中，“中国”，“发展”等单词，在语料库中出现的频率非常高，因此，即使这些词在某篇文档中频繁出现，也不能说明这些词对当前文档是非常重要的，因此这些词并不含有特别有意义的信息。

TF-IDF可以用来调整单词在文档中的权重。其由两部分组成：

1. TF（Term-Frequency）词频，指一个单词在文档中出现的次数。
2. IDF（Inverse Document-Frequency）逆文档频率。

计算方式为：

$$idf(t) = \log \frac{n}{1+df(t)}$$

$$tf-idf(t,d) = tf(t,d) \times idf(t)$$

- t：某单词。
- n：语料库中文档的总数。
- $df(t)$：语料库中含有单词t的文档个数。

说明：scikit-learn库中实现的tf-idf转换，与标准的公式略有不同。并且，tf-idf结果会使用L1或L2范数进行标准化（规范化）处理。

```
from sklearn.feature_extraction.text import TfidfTransformer

count = CountVectorizer()
docs = [
    "Where there is a will, there is a way.",
    "There is no royal road to learning.",
]
bag = count.fit_transform(docs)
tfidf = TfidfTransformer()
t = tfidf.fit_transform(bag)
# TfidfTransformer转换的结果也是稀疏矩阵。
print(t.toarray())
```

```
[[0.53594084 0.         0.         0.         0.         0.53594084
  0.         0.37662308 0.37662308 0.37662308]
 [0.29017021 0.4078241  0.4078241  0.4078241  0.4078241  0.29017021
  0.4078241  0.         0.         0.        ]]
```

此外，scikit-learn中，同时提供了一个类TfidfVectorizer，其可以直接将文档转换为TF-IDF值，也就是说，该类相当于集成了CountVectorizer与TfidfTransformer两个类的功能，这对我们实现上提供了便利。

```
from sklearn.feature_extraction.text import TfidfVectorizer

docs = [
    "Where there is a will, there is a way.",
    "There is no royal road to learning.",
]
tfidf = TfidfVectorizer()
t = tfidf.fit_transform(docs)
print(t.toarray())
```

```
[[0.53594084 0.         0.         0.         0.         0.53594084
  0.         0.37662308 0.37662308 0.37662308]
 [0.29017021 0.4078241  0.4078241  0.4078241  0.4078241  0.29017021
  0.4078241  0.         0.         0.        ]]
```

可以看出，这与我们之前使用CountVectorizer与TfidfTransformer两个类转换的结果是一样的。

建立模型

构建训练集与测试集

我们需要将每条评论的词汇进行整理。目前，我们文本内容已经完成了分词处理，但词汇是以列表类型呈现的，为了方便后续的向量化操作（文本向量化需要传递空格分开的字符串数组类型），我们将每条评论的词汇组合在一起，成为字符串类型，使用空格分隔。

```
def join(text_list):
    return " ".join(text_list)


data["comment"] = data["comment"].apply(join)
data.sample(5)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	time	city	gender	level	score	comment
111741	2019-08-07 22:55:17	湘潭	2	1	4.0	好看 笑 点 泪点
540780	2019-07-27 11:37:36	中山	1	4	5.0	真的 超级 超级 好看 满分 推荐
347143	2019-07-31 21:20:46	黔南	1	1	5.0	国漫 崛起
251949	2019-08-03 18:36:51	南京	0	2	5.0	确实 好看
169963	2019-08-05 19:39:02	江都	0	1	4.5	国产 动漫 赞 👍

然后，我们需要构造目标值，这里我们增加一个目标列（y）：

- 好评 2
- 中评 1
- 差评 0

```
data["target"] = np.where(data["score"] >= 4.5, 2, np.where(data["score"] >= 3, 1, 0))
data["target"].value_counts()
```

```
2    534733
1     35260
0      8539
Name: target, dtype: int64
```

```
p = data[data["target"] == 2]
m = data[data["target"] == 1]
n = data[data["target"] == 0]
p = p.sample(len(m))
data2 = pd.concat([p, m, n], axis=0)
data2["target"].value_counts()
```

```
2    35260
1    35260
0     8539
Name: target, dtype: int64
```

这样，我们就可以来对样本数据进行切分，构建训练集与测试集。

```
from sklearn.model_selection import train_test_split

X = data2["comment"]
y = data2["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
print("训练集样本数：", y_train.shape[0], "测试集样本数：", y_test.shape[0])
```

```
训练集样本数： 59294 测试集样本数： 19765
```

特征选择

特征维度

大家需要注意，到目前为止，数据集X还是文本类型，我们需要对其进行向量化操作。这里，我们使用TfidfVectorizer类，在训练集上进行训练，然后分别对训练集与测试集实施转换。

```
vec = TfidfVectorizer(ngram_range=(1, 2), max_df=0.5, min_df=1)
X_train_tran = vec.fit_transform(X_train)
X_test_tran = vec.transform(X_test)
display(X_train_tran, X_test_tran)
```

```
<59294x214002 sparse matrix of type '<class 'numpy.float64'>'
    with 685544 stored elements in Compressed Sparse Row format>
```

```
<19765x214002 sparse matrix of type '<class 'numpy.float64'>'
    with 173417 stored elements in Compressed Sparse Row format>
```

课堂练习

我们已经成功进行了转换，只不过，数据存储在稀疏矩阵中，如果我们调用稀疏矩阵的$toarray$方法，能够查看到TF-IDF值吗?

A 可以。

B 不能，但运行时不会产生错误。

C 不能，运行时会产生错误。

方差分析

使用词袋模型向量化后，会产生过多的特征，这些特征会对存储与计算造成巨大的压力，同时，并非所有的特征都对建模有帮助，基于以上原因，我们在将数据送入模型之前，先进行特征选择。

这里，我们使用方差分析（ANOVA）来进行特征选择，选择与目标分类变量最相关的20000个特征。方差分析用来分析两个或多个样本（来自不同总体）的均值是否相等，进而可以用来检验分类变量与连续变量之间是否相关。检验方式为，根据分类变量的不同取值，将样本进行分组。首先计算组内差异（SSE）与组间差异（SSM）：

$$SSE = \sum_{i=1}^{m} \sum_{j=1}^{n_i} (x_{ij} - \bar{x_i})^2$$

$$SSM = \sum_{i=1}^{m} n_i (\bar{x_i} - \bar{x})^2$$

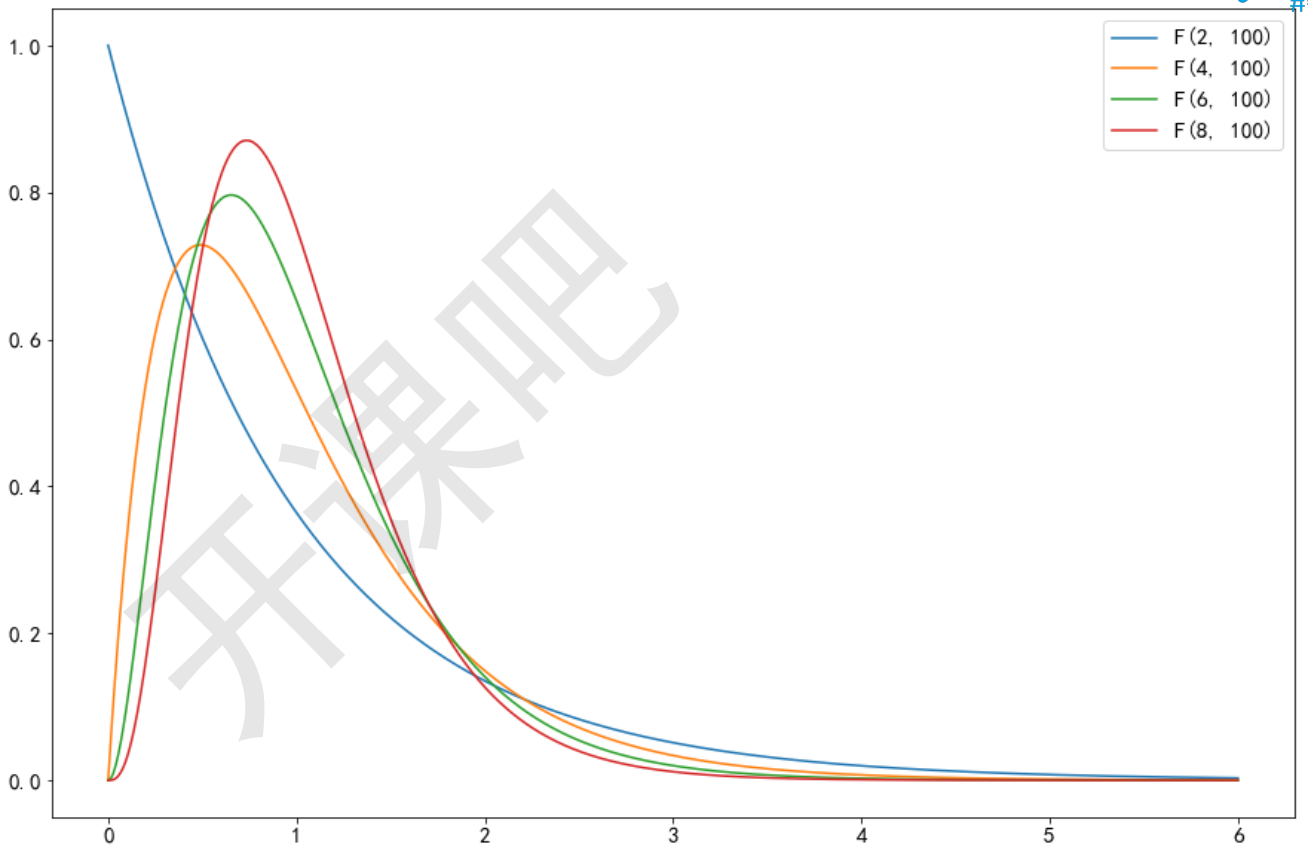
- m：组的数量（每个类别一个分组）。
- n_i：第i组中含有的观测值数量。
- x_{ij}：第i组中第j个观测值。
- $\bar{x_i}$：第i组中所有观测值的均值。
- $\bar{x}$：所有观测值的均值。

然后我们就可以构造F统计量：

$$F = \frac{SSM/(m-1)}{SSE/(n-m)}$$

- m：组的数量。
- n：观测值的数量。

该统计量服从自由度为$(m-1, n-m)$的F分布。F检验的原假设为各组的均值均相等，备则假设为至少存在两组数据均值不相等。




```python
from sklearn.feature_selection import f_classif

# 根据y进行分组，计算X中，每个特征的F值与P值。
# F值越大，P值越小。
f_classif(X_train_tran, y_train)
```

```
(array([1.21521908, 4.19737913, 0.61895789, ..., 0.61895789, 0.61929586,
        0.61929586]),
 array([0.29665241, 0.0150394 , 0.53850881, ..., 0.53850881, 0.53832684,
        0.53832684]))
```

```python
from sklearn.feature_selection import SelectKBest

# tf-idf值不需要太多的精度，使用32位的浮点数表示足矣。
X_train_tran = X_train_tran.astype(np.float32)
X_test_tran = X_test_tran.astype(np.float32)
# 定义特征选择器，用来选择最好的k个特征。
selector = SelectKBest(f_classif, k=min(20000, X_train_tran.shape[1]))
selector.fit(X_train_tran, y_train)
# 对训练集与测试集进行转换（选择特征）。
X_train_tran = selector.transform(X_train_tran)
X_test_tran = selector.transform(X_test_tran)
print(X_train_tran.shape, X_test_tran.shape)
```

```
(59294, 20000) (19765, 20000)
```

朴素贝叶斯

```
from sklearn.metrics import classification_report
from sklearn.naive_bayes import ComplementNB

gnb = ComplementNB()
gnb.fit(X_train_tran, y_train)
y_hat = gnb.predict(X_test_tran)
print(classification_report(y_test, y_hat))
```

```
              precision    recall  f1-score   support

           0       0.40      0.41      0.41      2227
           1       0.67      0.63      0.65      8765
           2       0.71      0.74      0.73      8773

    accuracy                           0.66     19765
   macro avg       0.59      0.60      0.59     19765
weighted avg       0.66      0.66      0.66     19765
```

总结

- 文本数据的预处理。
- TF-IDF文本向量化。
- 词云图的生成。
- 方差分析实现特征选择。
- 朴素贝叶斯实现文本分类。

扩展点与后续优化

- 其他分类算法。
- 算法超参数调整。
- 样本不均衡的处理方式。