《流浪地球》豆瓣影评分析实训报告

1.1 背景与挖掘目标

本案例以国内提供最新的电影介绍及评论包括上映影片的影讯查询及购票服务的豆瓣电影为数据源，获取了 2019 年初争议比较大的《流浪地球》豆瓣短评，并对获取的短评数据进行了预处理和评分分析与挖掘。

1.2 数据预处理

1.2.1 文件读取

首先读取保存到本地的短评数据，如代码 1 所示。

代码 1 读取文件

```
import pandas as pd
import os
os.chdir('/course/《流浪地球》豆瓣影评分析/data')

f = open('./流浪地球.csv')
data = pd.read_csv(f)
f.close()
data.head()
```

1.2.2 文本数据预处理

从上面的表格可以看到用户城市（citys）和评分（scores）两栏的数据放置在方框里面，需要处理。同时用户评论里面包含一些特殊字符，也需要进一步处理。最终处理后的效果是数据处理后可以去除多余符号。实现代码见代码 2。

代码 2 预处理

```
import re


data['citys'] = data['citys'].apply(
        lambda x: re.findall("[^'\[\]]+", x)[0] if len(re.findall('[^'[]]+', x))!=0 else None
)
```

```
data['scores'] = data['scores'].apply(
    lambda x: int(re.findall('[0-9]+', x)[0]) if len(re.findall('[0-9]+', x))!=0 else None
)
data['content'] = data['content'].apply(
    lambda x: re.sub('[^\u4E00-\u9FD5,.?!，。！ ？ 、；；:：0-9]+', '', x)
)

data.head()
```

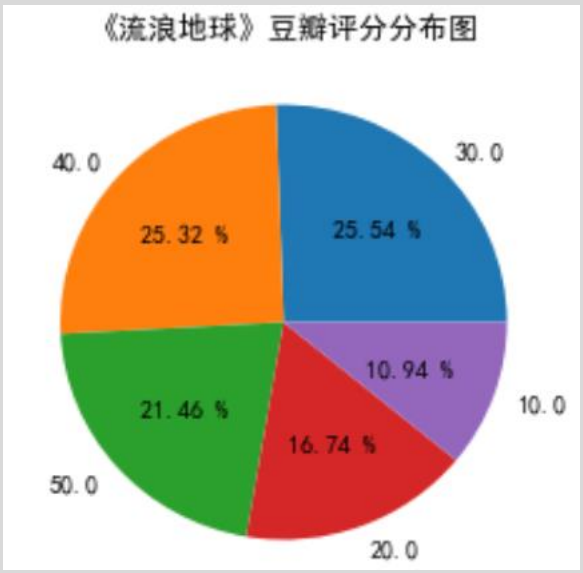
1.3 评分统计分析

先进行评分统计再绘制饼图分析评分分布情况，实现如代码 3 所示。

代码 3 评分统计

```
num = data['scores'].value_counts()  # 统计词频

plt.figure(figsize=(4, 4))
plt.rcParams['font.sans-serif'] = 'Simhei'
plt.pie(num, autopct="%.2f %%", labels=num.index)
plt.title('《流浪地球》豆瓣评分分布图')
plt.show()
```



评分统计结果如下：

评分	数量
★☆☆☆☆	49
★★☆☆☆	77

★★★☆☆	124
★★★★☆	117
★★★★★	98

从统计表和饼图可以看到用户对《流浪地球》的评价两极分化，但大多数倾向 3 星到 4 星。

1.4 评论时间分布统计

1.4.1 评论随日期变化

接下来统计随着时间的推进，豆瓣用户对《流浪地球》发表评论的数量变化情况，实现代码见代码 4。

代码 4 评论随日期变化

```
num = data['times'].apply(lambda x: x.split()[0]).value_counts()
num2 = num.sort_index()

plt.figure(figsize=(8,5))
plt.plot(range(len(num2)), num2)
plt.xticks(range(len(num2)), num2.index, rotation=45)
plt.title('评论数量随日期的变化图')
plt.grid()
plt.show()
```

在这里涉及到《流浪地球》的三个时间点：2019-1-20、2019-1-28 进行了点映，而 2019-2-5 影片正式上映。从折线图可以看到点映后会有小幅的评论数量增加，正式上映后，评论数据大幅上涨，达到了高峰。正式上映后，每日发布的评论数量逐渐减少。

在影片上映 3 天内为评论高峰，这符合常识，但是也可能有偏差，因为爬虫获取的数据是经过豆瓣电影排序的，倘若数据量足够大得出的趋势可能更接近真实情况。

影片在上映前也有部分评论，分析可能是影院公映前的小规模试映，且这些提前批的用户的评分均值，差不多接近影评上映后的大规模评论的最终评分 ，从这些细节中，我们或许可以猜测，这些能提前观看影片的，可能是资深影迷或者影视从业人员，他们的评论有着十分不错的参考价值。

1.4.2 评论数量随时刻的变化

接下来对分析评论数量随时刻的变化情况。实现代码见代码 5。

代码 5 评论数量随时刻的变化

```
num = pd.to_datetime(data['times']).apply(lambda x: x.hour).value_counts()
num2 = num.sort_index()
plt.plot(num2.index, num2)
plt.grid()
plt.xticks(num2.index, num2.index)
plt.title('评论数量随时刻的变化图')
plt.show()
```

豆瓣用户发布短评的时间主要集中在晚上，17 点至凌晨 0 点比例尤为明显。随着时间

向深夜推进，比例逐渐下降，凌晨 4 点达到最低值。这主要与用户的作息生活有关系。

同时短评一般在在观看完电影后发布的，所以用户可能偏向于观影结束回到家之后再进行对影片的评价行为。

1.4.3 豆瓣评分的时间趋势分析

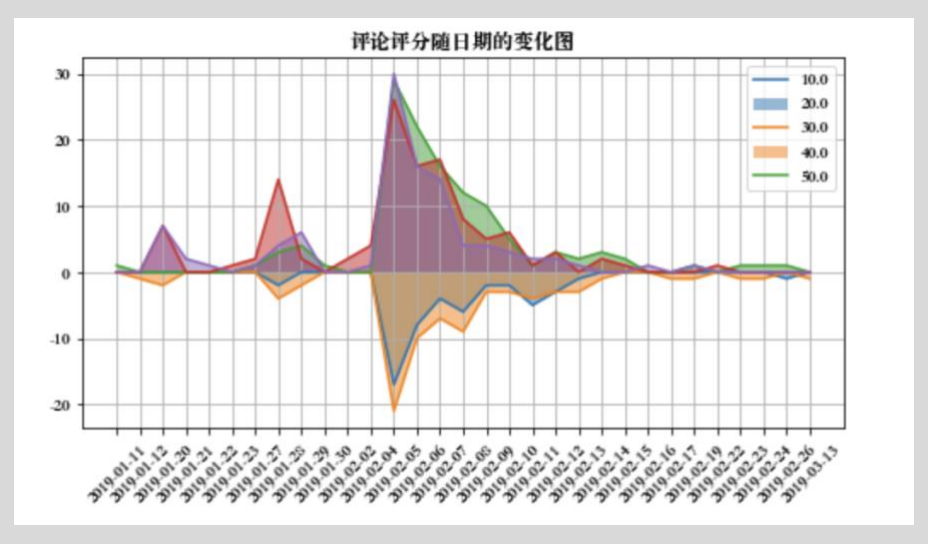
对《流浪地球》短评数据再进行豆瓣评分随时间变化而变化的情况分析，具体实现见代码 6。

代码 6 豆瓣评分的时间趋势分析

```
data['date'] = data['times'].apply(lambda x: x.split()[0])
tmp = pd.DataFrame(0, columns=data['scores'].astype(str).drop_duplicates().sort_values(),
                                index=data['date'].drop_duplicates())
tmp.sort_index(inplace=True)

for i, j, k in zip(data['scores'], data['date'], data['votes']):
#         print(i, j, k)
    tmp.loc[j, str(i)] += 1
print(tmp)

plt.figure(figsize=(8, 4))
(n, m) = tmp.shape
plt.rcParams['axes.unicode_minus'] = False
for i in range(0, m-1):
    plt.plot(range(n), (1 if i>=2 else -1)*tmp.iloc[:, i])
    plt.fill_between(range(n), (1 if i>=2 else -1)*tmp.iloc[:, i], alpha=0.5)
plt.legend(tmp.columns[:-1])
plt.xticks(range(n), tmp.index, rotation=45)
plt.title('评论评分随日期的变化图')
plt.grid()
plt.show()
```



从 **Error! Reference source not found.**得到的折线图可以得到以下结论。

（1） 在点映期间，对电影的评价大部分是正面评价，但是电影上映后用户对《流浪地球》的评价开始两极分化。

（2） 一星评价中，2019-2-11 有个小高峰，而当天是星期一，好评的数量是小低谷，可能是刷负分的评价。

1.5 分析评论者的城市分布情况

1.5.1 评论数量最多的前 10 个城市排名

先统计各个城市的评论数量，再绘制柱状图分析用户城市分布情况，实现如代码 7 所示。

代码 7 城市排名

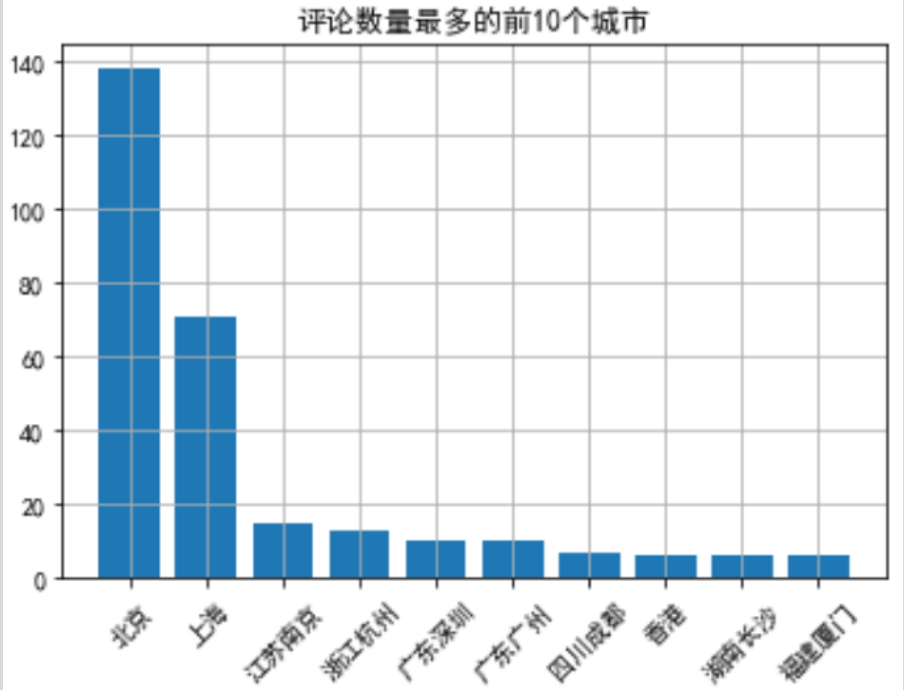
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import os

os.chdir('/course/《流浪地球》豆瓣影评分析/data')
f = open('./流浪地球.csv')
data = pd.read_csv(f)
f.close()

data['citys'] = data['citys'].apply(
    lambda x: re.findall("[^'\[\]]+", x)[0] if len(re.findall('[^'[]]+', x))!=0 else None
```

```
)
data['scores'] = data['scores'].apply(
        lambda x: int(re.findall('[0-9]+', x)[0]) if len(re.findall('[0-9]+', x))!=0 else None
)

num = data['citys'].value_counts()
plt.rcParams['font.sans-serif'] = 'Simhei'
plt.bar(range(len(num[:10])), num[:10])
plt.xticks(range(len(num[:10])), num[:10].index, rotation=45)
plt.title('评论数量最多的前 10 个城市')
plt.grid()
plt.show()
```



可以看出北京、上海的用户是最多的，有将近一半的评论用户来自这两个城市。

1.5.2 不同城市对影片的评分情况分析

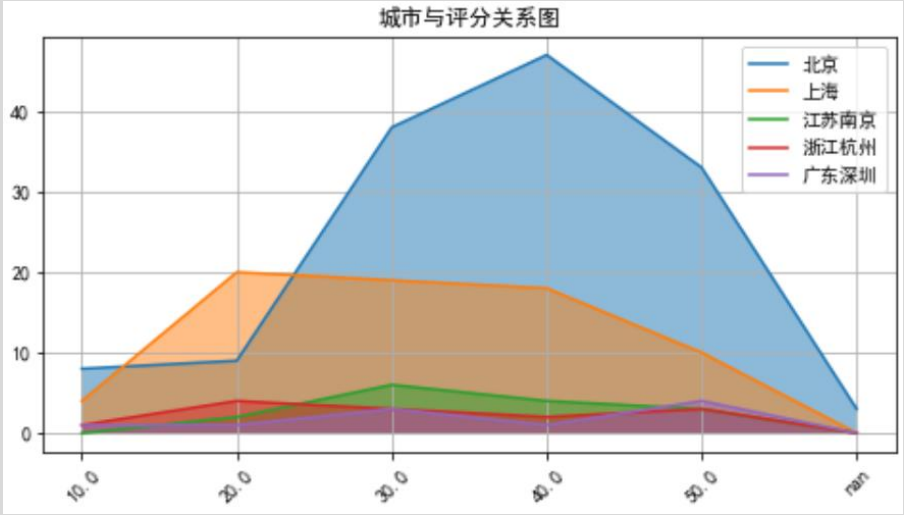
接下来对城市信息做进一步分析，分析各个城市的评分情况，，实现代码见代码 8。

代码 8 不同城市对影片的评分情况分析

```
tmp    = pd.DataFrame(0, index=data['scores'].astype(str).drop_duplicates().sort_values(),
                                columns=num[:5].index)
tmp.sort_index(inplace=True)
for i, j, k in zip(data['scores'], data['citys'], data['votes']):
#         print(i, j, k)
        if j in num[:5].index:
```

```
            tmp.loc[str(i), j] += 1


plt.figure(figsize=(8, 4))
(n, m) = tmp.shape
plt.rcParams['axes.unicode_minus'] = False
for i in range(0, m):
    plt.plot(range(n), tmp.iloc[:, i])
    plt.fill_between(range(n), tmp.iloc[:, i], alpha=0.5)
plt.legend(tmp.columns)
plt.xticks(range(n), tmp.index, rotation=45)
plt.title('城市与评分关系图')
plt.grid()
plt.show()
```



对两个城市的数据统计时发现，北京的用户倾向于给该类型主旋律影片四星的评价，而上海地区打差评的评价更多一些。

1.5.3 评论数量最多的前 10 个省份

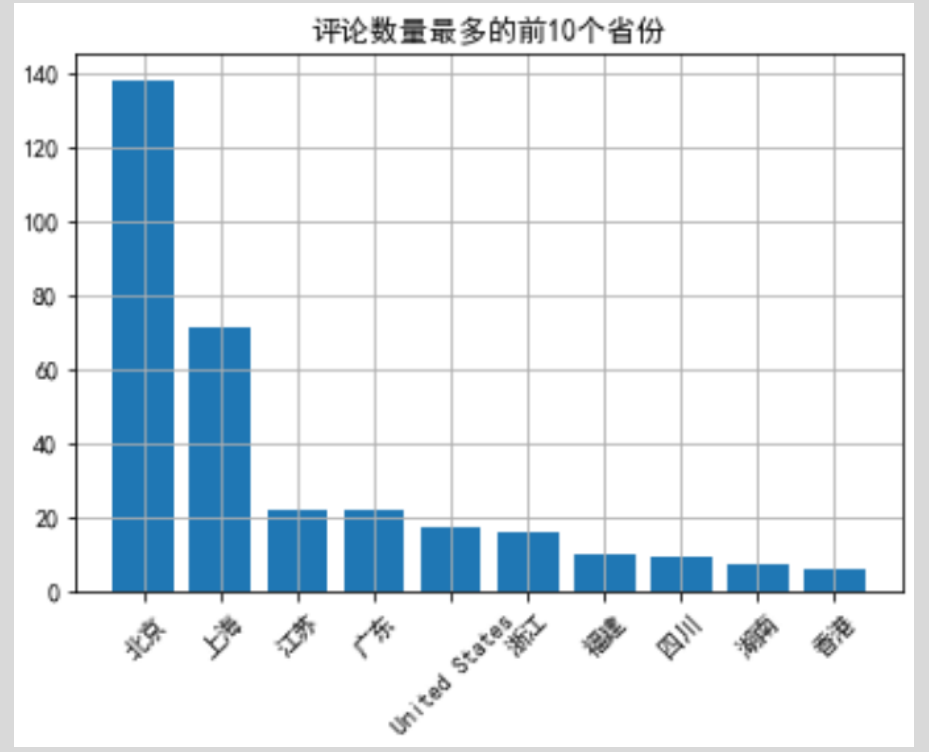
最后对用户所在的省份进行统计分析，实现代码见代码 9。

代码 9 评论数量最多的前 10 个省份

```
tmp = data.loc[data['citys'].notnull(), 'citys']
provice = tmp.apply(lambda x: re.findall('[a-zA-Z ]+', x)[-1] if re.findall('[a-zA-Z]+', x)!=[]
else x[:2])
num = provice.value_counts()


plt.bar(range(len(num[:10])), num[:10])
plt.xticks(range(len(num[:10])), num[:10].index, rotation=45)
plt.title('评论数量最多的前 10 个省份')
```

```
plt.grid()
plt.show()
```



从 **Error! Reference source not found.**得到的柱状图可以看出，北上广使用豆瓣进行评价的记录更多一下，可能是豆瓣的人文、企业文化受众多为一线城市的民众或受教育水平影响。

1.6 分词并去除停用词

接下来对预处理后的短评数据进行文本分词以及去除停用词操作，主要涉及涉及到中文分词库 jieba 以及停用词表。实现代码见代码 10。

代码 10 分词并去除停用词

```
import jieba
import itertools
from wordcloud import WordCloud
import matplotlib.pyplot as plt

with open('./stoplist.txt', 'r', encoding='utf-8') as f:
    stop = f.read()
stop = stop.split()
stop = [' ', '\n', '这部'] + stop

data_cut = data['content'].apply(jieba.lcut)  # 分词
```

```
# 去除停用词
data_after = data_cut.apply(
    lambda x: [i for i in x if i not in stop]
)

print(data_cut.head())
print(data_after.head())
```

1.7 统计词频与绘制词云图

对去除完停用词后的数据进行词频统计并绘制词云的操作，分析一下对于《流浪地球》这部电影，豆瓣的用户主要探讨的是什么主题内容。具体实现见代码 11。

代码 11 统计词频并绘制词云

```
num = pd.Series(list(itertools.chain(*list(data_after)))).value_counts()   # 统计词频

pic = plt.imread('./aixin.jpg')
wc = WordCloud(font_path='./data/simhei.ttf', background_color='White', mask=pic)
wc2 = wc.fit_words(num)

plt.imshow(wc2)
plt.axis('off')
plt.show()
```

从 **Error! Reference source not found.**得到的词云图可以看到高频词“中国”、 “地球”、“人类”表现出该片的主要人文思想，“特效”体现出特效镜头对科幻片的重要性，“科幻电影”

体现出影迷对科幻类电影的浓厚兴趣。

同样的可以选择分别对好评与差评的短评数据进行词云图的绘制，如代码 12 所示。

代码 12 分别绘制好评与差评的词云图

```
import itertools
import matplotlib.pyplot as plt
data_good = data_after.loc[data['scores'] >= 30]
data_bad = data_after.loc[data['scores'] < 30]

def my_wc(data_after):
    tmp = list(itertools.chain(*list(data_after)))
    num = pd.Series(tmp).value_counts()

    import matplotlib.pyplot as plt
    from wordcloud import WordCloud

    pic = plt.imread('./aixin.jpg')
    wc = WordCloud(background_color='white', font_path='./data/simhei.ttf', mask=pic)
    wc2 = wc.fit_words(num)
    plt.imshow(wc2)
    plt.axis('off')
    plt.show()
my_wc(data_bad)
my_wc(data_good)
```

从好评词云与差评词云中可以得到，好评与差评探讨的主题（出现的高频词）都是类似的，只是个人观点导致在对同一部影片的同一话题出现两极分化的观点。

1.8 模型构建与评估

1.8.1 划分数据集

评分小于 30 为差评，标记为 0；反之则为好评，标记为 1。将原始数据划分为训练集和测试集，划分比例为 4:1。具体实现见代码 13。

代码 13 划分数据集

```
data_new_good=pd.DataFrame()
data_new_good['text']=data_good.apply(lambda x:' '.join(x))
data_new_good['label']=1
data_new_good.reset_index(inplace=True,drop=True)
data_new_bad=pd.DataFrame()
data_new_bad['text']=data_bad.apply(lambda x:' '.join(x))
data_new_bad['label']=0
data_new_bad.reset_index(inplace=True,drop=True)
data_new_bad

data_new=pd.concat([data_new_bad,data_new_good],axis=0)
data_new

#划分数据集
from sklearn.model_selection import train_test_split
test_ratio = 0.2
```

```
src_training, src_testing = train_test_split(data_new, test_size=test_ratio, stratify=data_new['label'],random_state=123)

comments_train, comments_test = src_training['text'].values, src_testing['text'].values
y_train, y_test = src_training['label'].values, src_testing['label'].values
```

1.8.2 文本向量化

由于文本数据无法直接用于建模，因此需要将文本表示成计算机能够直接处理的形式，即文本数字化。具体实现如代码 14 所示。

代码 14 文本向量化

```
# 向量化：把训练文本和测试文本转换成 tf-idf 向量。
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

count_vectorizer = CountVectorizer()
tfidf_transformer = TfidfTransformer()

word_count_train = count_vectorizer.fit_transform(comments_train)
tfidf_train = tfidf_transformer.fit_transform(word_count_train)

word_count_test = count_vectorizer.transform(comments_test)
tfidf_test = tfidf_transformer.transform(word_count_test)
print(tfidf_train.shape, tfidf_test.shape)
```

1.8.3 分类模型与评估

经过上述处理之后，数据已经变成符合我们建模需要的词条-文档矩阵类型的数值矩阵，在 Python 里使用 sklearn.Neighbors 中的 KNeighborsClassifier 函数即可搭建一个 K 近邻分类模型。构建分类模型并进行模型评估，具体实现如 代码 15 所示。

代码 15 分类模型与评估

```
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score
knn = KNeighborsClassifier(n_neighbors=8, weights= 'distance')
```

```
knn.fit(tfidf_train, y_train)
tfidf_y_pred = knn.predict(tfidf_test)


print('tfidf_ KNeighborsClassifier test accuracy %s' % accuracy_score(y_test, tfidf_y_pred))
```