Pandas 实战 Kagge 百万级影评数据集之数据清洗和特征...

split 本项目基于 Kaggle 电影影评数据集，与大家一起实战，包括：

- 如何使用 Pandas 做数据清洗和特征工程；
- 如何进行数据探索性分析（`EDA`）；

学会数据分析的基本思维、基本技能和工具。包括：使用数据分析常用工具 `numpy` 和 `pandas`，绘图工具 `matplotlib` 和 `pyecharts`.

本项目需要导入的包：

复制
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pyecharts.charts import Bar,Grid,Line,Pie
import pyecharts.options as opts
from pyecharts.globals import ThemeType
```

1 导入数据

数据来自 Kaggle，共包括如下三个文件：

1. movies.dat ，共有 34000+ 行记录
2. ratings.dat，共有 810000+ 行记录
3. users.dat，共有 60000+ 行记录

百度网盘的下载链接:

> https://pan.baidu.com/s/1Na8RCfpnyFrm1aTtMgiDuQ 提取码: wvnx

首先，导入电影数据文件 `movies.dat`，它共包括 3 个字段：Movie ID, Movie Title, Genre，分别表示电影 ID、电影名称、题材（可能属于多个题材，中间用 | 分割），使用 pandas 导入此文件：

复制
```
import pandas as pd
movies = pd.read_csv('../dataset/movietweetings/movies.dat', delimiter='::',
engine='python', header=None, names = ['Movie ID', 'Movie Title', 'Genre'],encoding='utf-8')
movies.head()
```

导入后的数据，前 5 行显示如下：

其次，导入用户相关的数据文件 `users.dat`：

复制
```
users = pd.read_csv('../dataset/movietweetings/users.dat', delimiter='::',
engine='python', header=None, names = ['User ID', 'Twitter ID'], encoding='utf-8')
users.head()
```

它一共有 2 列，分别表示用户 ID, Twitter ID, 前 5 行数据显示结果：

同样方法导入 rating.data，关于评分记录：

复制
```
ratings = pd.read_csv('../dataset/movietweetings/ratings.dat', delimiter='::',
engine='python', header=None, names = ['User ID', 'Movie ID', 'Rating', 'Rating Timestamp'], encoding='utf-8')

ratings.head()
```

前 5 行结果显示如下，一共有 4 列。分别表示用户 ID, 电影 ID，电影得分，评分时间戳。

read_csv 使用说明

- 第一个参数表示文件的相对路径
- 第二个关键字参数：delimiter='::'，表示文件分隔符使用::
- 后面几个关键字参数分别代表使用的引擎，文件没有表头，所以 header 为None;
- 导入后 DataFrame 的列名，使用 `names` 关键字设置，这个参数大家可以记住，比较有用。

2 数据预览

前面提到过，Pandas 提供 2 个很好用的方法：`info`、`describe`。

`info` 统计出数据的每一列类型，是否为 null 和个数；

`describe` 描述出数据每一列的统计学属性信息，包括常见的 `平均值`、`方差`、`中位数`、`分位数` 等。

复制
```
movies.info()
```

一共有 3万4千多部电影，只有 `Genre` 列存在空值，因为总共有 34437 行，其中有 34159 行为非空。 `Movie ID` 列的数据类型为 `int64`，其他两列都为 `object`：

复制
```
users.info()
```

共有 6万多观看电影用户记录，所有列都不为空，2 个列的数据类型都为 int64:

```
ratings.info()
```

电影评论记录共有8万多，没有空值，4 个列的数据类型都为 int64，占用内存 24.9 MB:

```
ratings.describe()
```

看到电影评论的平均得分为 `7.30`，方差大约 `1.86`

电影评论得分是一个重要特征列，绘制频率分布直方图和箱型图，直观感受评论得分的分布情况。

```
plt.figure(figsize=[10,8])
plt.subplot(221)
plt.hist(x = ratings['Rating'], color = ['orange'])
plt.subplot(222)
plt.boxplot(x = ratings['Rating'], showmeans = True, meanline = True)
plt.grid()
plt.show()
```

25% 的电影得分在 0~6 分，6~7 分的电影又有 25%， 7~9 分的电影又有 25%，9 分以上的电影占有 25%，如下两幅图所示：

3 补全空值

3 个数据文件，只有 movies.dat 文件的 `Genre` 列存在空值，我们打印前 5 行空值：

```
movies[movies['Genre'].isnull()].head()
```

`Genre` 列表示电影的类型，是一个分类列。根据常识，常见电影的分类可能也就几十种。使用 `value_counts` 方法统计此列的取值种类数：

```
movies['Genre'].value_counts()
```

结果显示有 2693 种，超乎想象。原来就像文章开始提到那样，一部电影可能属于多种类型，中间用 `|` 分割，如下所示：

思路：根据分隔符 `|` 解析字符串，然后统计每个子串出现次数。这个地方的处理需要一定技巧，也有一定困难。

首先填充 `Genre` 列的空值，填充为 `others` 类型：

```
movies['Genre'].isnull().sum() / len(movies['Genre']) # 打印下空值比例
movies['Genre'].fillna('others',inplace=True)
```

4 特征工程

接着上面，处理 `Genre` 列，这一列处理相对繁琐，一步一步完成特征工程。

根据分割符 `|` 解析此列，并将此列转换为 NumPy 对象：

```
genres = movies['Genre'].str.split('|').to_numpy()
```

使用 `defaultdict` 统计出电影种类和对应电影数。

```
from collections import defaultdict
counter = defaultdict(int)
for genre in genres:
    for e in genre:
        counter[e] += 1
counter
```

统计结果：

一共有 29 种电影。对上面字典按种类数排序：

```
counter_sorted = sorted(counter.items(),key=lambda x: x[1])
counter_sorted
```

只取电影种类数最多的前10进行分析：

```
top10 = counter_sorted[-10:]
```

绘制前 10 最多种类数的柱状图，使用 pyecharts 绘制：

```
x = [ x for x,y in top10]
y = [ y for x,y in top10]
bar = (
    Bar(init_opts= opts.InitOpts(height='1200px'))
    .add_xaxis(x)
    .add_yaxis('电影种类名',y,category_gap='50%')
```

```
    .reversal_axis()
    .set_global_opts(title_opts=opts.TitleOpts(title="电影种类及影片数"),
                     toolbox_opts=opts.ToolboxOpts())
)
grid = (
    Grid(init_opts=opts.InitOpts(theme=ThemeType.LIGHT))
    .add(bar, grid_opts=opts.GridOpts(pos_left="30%"))
)
grid.render_notebook()
```

进一步观察这些种类所占比率，想到饼状图，使用 pyecharts 绘制：

复制

```
c = (
    Pie()
    .add(
        "",
        [list(z) for z in top10 ],
        radius=["40%", "55%"],
        label_opts=opts.LabelOpts(
            position="outside",
            formatter="{a|{a}}{abg|}\n{hr|}\n {b|{b}: }{c}  {per|{d}%}  "
,
            background_color="#eee",
            border_color="#aaa",
            border_width=1,
            border_radius=4,
            rich={
                "a": {"color": "#999", "lineHeight": 22, "align": "center"
},
                "abg": {
                    "backgroundColor": "#e3e3e3",
                    "width": "100%",
                    "align": "right",
                    "height": 22,
                    "borderRadius": [4, 4, 0, 0],
                },
                "hr": {
                    "borderColor": "#aaa",
                    "width": "100%",
                    "borderWidth": 0.5,
                    "height": 0,
                },
                "b": {"fontSize": 16, "lineHeight": 33},
                "per": {
                    "color": "#eee",
                    "backgroundColor": "#334455",
                    "padding": [2, 4],
                    "borderRadius": 2,
                },
            },
        ),
    )
)

c.render_notebook()
```

看到前三类已占到 50% 以上，最多的 `Drame` 占比 27%.

应用到 `movies` 中，只保留 `Genre` 取值位于前 10 的电影类型中：

复制

```
top10_genre = [x for x,y in top10]
pat = '|'.join(top10_genre)
mask1  = movies['Genre'].str.contains(pat)
mask1
```

此处使用一个有用的小技巧，符号 `|` 在正则表达式中表示 `或`，使用 `|` 连接 `top10_genre` 得到一个正则字符串 `pat`，使用 `contains` 方法判断 `Genre` 列的取值（每个单元格值是个 list）是否匹配串 `pat`，得到掩码 `mask1`：

复制

```
0         True
1         True
2         True
3        False
4         True
         ...
34432     True
34433     True
34434     True
34435     True
34436     True
Name: Genre, Length: 34437, dtype: bool
```

将掩码应用到 `movies`

复制

```
movies1 = movies[mask1]
movies1
```

结果：

根据 `Movie ID` 列去重，并且重置索引列（从0开始依次递增1）

复制

```
movies2 = movies.drop_duplicates(['Movie ID']).reset_index()
movies2
```

观察列 `Movie Title` ，括号里的数字表示电影的出场日期，把它提取到单独一列里。

使用根据特定字符的分列方法 `partition`：

复制

```
parts = movies2['Movie Title'].str.partition('(')
parts
```

结果如下，创建出 3 列：

最后一列是我们想要的，只需剔除多余的右括号字符：

```
myear = parts[2].str.replace(')','')
myear
```

得到结果：

注意此列的类型还不是 int 型：

```
myear.apply(type)
```

结果：

使用 `astype` 方法，转化为 int 型：

```
myear = myear.astype(int)
myear
```

最后，赋值给新创建的列 `year` 上：

```
movies2.loc[:,'year'] = myear
movies2
```

结果如下：

```
    index   Movie ID    Movie Title Genre   year
0    0   8   Edison Kinetoscopic Record of a Sneeze (1894)   Documentary|
Short   1894
1    1   10  La sortie des usines Lumière (1895) Documentary|Short   1895
2    2   12  The Arrival of a Train (1896)   Documentary|Short   1896
3    3   25  The Oxford and Cambridge University Boat Race ...   others
1895
4    4   91  Le manoir du diable (1896)  Short|Horror    1896
...    ... ... ... ... ... ...
34430    34432   10977680    Deon Cole: Cole Hearted (2019)  Comedy  2019
34431    34433   10987544    The Forest of Love (2019)   Crime   2019
34432    34434   11033952    Square One (2019)   Documentary 2019
34433    34435   11064486    Puppy (2019)    Drama   2019
34434    34436   11066130    Upstarts (2019) Drama   2019
34435 rows × 5 columns
```

最后预览下 movies2:

```
movies2.info()
```

结果：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34435 entries, 0 to 34434
Data columns (total 5 columns):
index          34435 non-null int64
Movie ID       34435 non-null int64
Movie Title    34435 non-null object
Genre          34435 non-null object
year           34435 non-null int32
dtypes: int32(1), int64(2), object(2)
memory usage: 1.2+ MB
```

至此得到 `movies2` 就是我们清洗后的 DataFrame.

`users` 两列都是 ID, 并且无空值，不用做清洗。

`ratings` 有一个时间戳列，是用偏移值表示，使用起来不方便，想办法转化构造出年、月、日三列。

Pandas 方法 `to_datetime` 能实现按列转化时间戳为 `datetime` 对象，省去 for 循环，代码可读性和执行效率都会变好。

```
rating_dt = pd.to_datetime(ratings['Rating Timestamp'],unit='s')
rating_dt
```

使用 `to_datetime` 方法需要注意参数 `unit` 默认为 ns, 此列的时间戳单位为 s，所以要重新设置一下。结果显示如下：

这样构造出列 `rating_dt`，并且类型为 `datetime`，其实只要到日期就行。我们把整数型的时间戳转化为 `datetime` 类型后，就可以直接在 `rating_dt` 上访问 `dt` ，进而拿到 `date`, `year`, `month` 等各种好用的属性。

同时我们基于 `datetime` 构造出几列：`year`, `month`, `day`，代码如下：

```
ratings.loc[:,'rating_dt'] = rating_dt.dt.date
ratings.loc[:,'year'] = rating_dt.dt.year
ratings.loc[:,'month'] = rating_dt.dt.month
ratings.loc[:,'day'] = rating_dt.dt.day
ratings
```

这样 `Rating Timestamp` 列就可以删除了：

```
ratings.drop('Rating Timestamp', axis=1, inplace = True)
ratings
```

结果：

小结

今天，主要使用 Pandas 完成百万级影评数据集，首先导入数据，然后使用 2 个方法快速完成数据预览，补全空值，最后做特征工程，创建某些特征，删除某些特征。

可以看到 Pandas 数据分析非常便捷，几乎不用 for 循环，今天使用的方法也都是 Pandas 中经常使用的，这其中主要包括：

- 读入数据的方法 read_csv, 参数 header, names
- 预览数据的方法 info, describe
- 与补全数据相关的方法 isnull， value_counts, fillna
- 特征工程频繁使用的方法 split, contains, replace，正则方法，drop*duplicates*，*partition*，*to*datetime， drop， str 和 dt 访问器
- Python 内置的 defaultdict, sorted 等

绘图工具 matplotlib 和 pyecharts，后者官方资料详细，给出大量的 demo 实例，大家可参考：http://gallery.pyecharts.org/#/Pie/pie*rich*label

希望大家完整敲一遍以上所有代码，在这里也提供下今天课程的完整 notebook 代码。

下载地址：https://pan.baidu.com/s/10ag5JxlZr4r6-HUTLe3soA 提取码: imbt

下一章