特征工程矩阵处理常用方法

数据检测、筛选、处理是特征工程中比较常用的手段，常见的场景最终都可以归类为矩阵的处理，对矩阵的处理往往会涉及到

- 阈值处理
- 取top N的值
- 多条记录中筛选包含特定值的记录
- 特征拼接、记录拼接

对于矩阵的处理没有趁手的兵器可不行，python中比较强大的库numpy与pandas是最常用的两种。主要使用的函数有，`np.vstack,np.hstack,np.where,df.loc,heapq.nlargest`。这几个方法的应用已经基本上满足矩阵处理的大部分需求。本文将引入四个业务场景来介绍以上矩阵处理方法。

阈值处理

以单通道图片的提高背景亮度为例，把小于100的灰度值都设置为200。

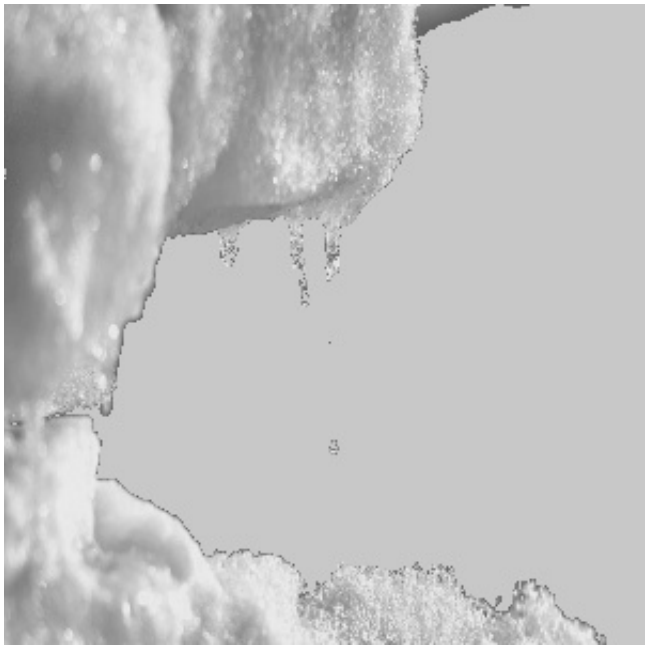
使用opencv接口读取一张图片的灰度图

```
import cv2
import matplotlib.pyplot as plt#约定俗成的写法plt

img = cv2.imread("timg.jpeg",0)
img=cv2.resize(img,(300,300))
cv2.imwrite("im.jpg",img)
```

```
img=np.atleast_2d(img)

img1=img.copy()
img1[np.where(img<100)]=200
cv2.imwrite("enhanced.jpg",img1)
```



二值化常用于目标检测，轮廓提取，或者其他应用

```
#二值化
img2=np.where(img>160,255,0)
cv2.imwrite("binary.jpg",img2)
```

特征拼接、记录拼接

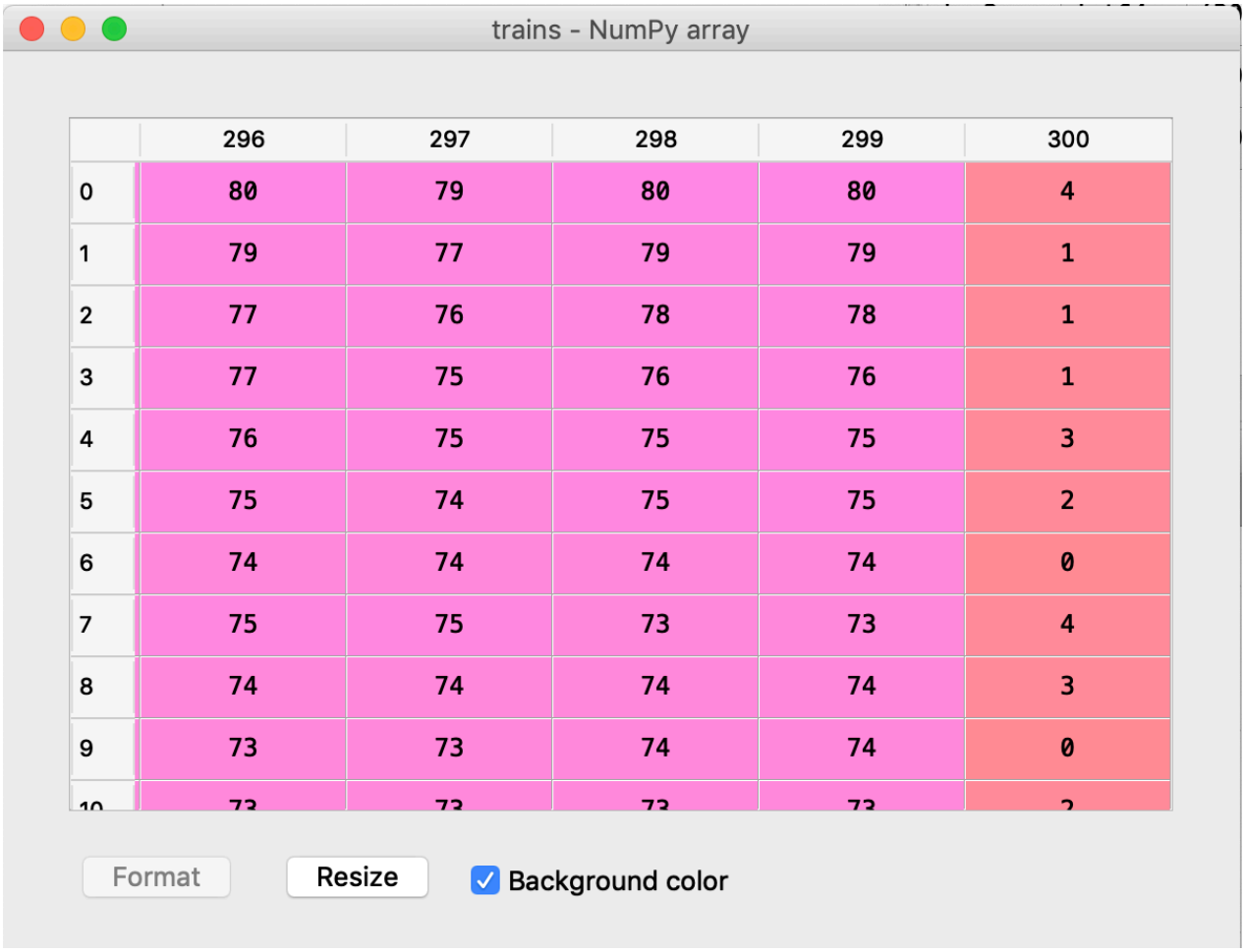
这个是最常用的处理方法，特征 X 与label Y 经常是分开存储的，在使用数据集之前经常需要shuffle操作，为了避免特征与Label混乱需要先拼接起来再shuffle。

以上文读取的图片特征数据作为例子，假设有300个样本，每个样本的特征维度为300，人工制造300个label用来做拼接工作。

```
labels=np.atleast_2d([random.randint(0,4) for i in
range(img.shape[1])]).reshape([-1,1])
```

```
trains=np.hstack((img,labels))
```

这样的话，特征就与标签拼接到一起，shuffle的时候也不会乱，对应不上。



	296	297	298	299	300
0	80	79	80	80	4
1	79	77	79	79	1
2	77	76	78	78	1
3	77	75	76	76	1
4	76	75	75	75	3
5	75	74	75	75	2
6	74	74	74	74	0
7	75	75	73	73	4
8	74	74	74	74	3
9	73	73	74	74	0
10	73	73	73	73	2

不仅可以水平拼接，numpy也提供了垂直拼接。这个函数经常用于，数据集扩充的时候，使用数组循环遍历一条条的加载到数据集比较麻烦，使用numpy提供的vstack方法会很方便的拼接到一起。

```
np.vstack()
```

```
In [13]: a=[1,2,3]

In [14]: b=[4,5,6]

In [15]: np.vstack([a,b])
Out[15]:
array([[1, 2, 3],
       [4, 5, 6]])

In [16]: np.vstack([a,b]).shape
Out[16]: (2, 3)
```

样本筛选

样本的筛选一般是挑选满足条件的行记录定位，再索引。引入一个场景，以`特征拼接、记录拼接`生成的数据为例，统计label==4的样本有多少个?

思路应该是:

- 定位label==4的分别在第几行，或者说index等于多少，获取这样一列数组
- 根据得到的index数据，分别从matrix中取出。

np.where函数能够得到满足条件的index.

```
np.where(trains[:,-1]==4)
```

```
In [18]: np.where(trains[:,-1]==4)
Out[18]:
(array([  0,   7,  12,  24,  30,  40,  41,  50,  58,  79,  95, 107, 111,
        124, 127, 141, 142, 148, 159, 160, 163, 176, 180, 183, 187, 193,
        196, 202, 206, 211, 213, 215, 216, 223, 226, 232, 236, 242, 244,
        256, 257, 258, 262, 264, 272, 275, 282, 283, 284, 294, 299]),)
```

从输出来看可以看到，第0行，7行，...299行的label等于4.

当然不仅仅可以用于一维的索引查找，二维矩阵依然能够定位特定值的位置。

```
np.where(trains==4)
```

```
In [22]: np.where(trains==4)
Out[22]:
(array([  0,   7,  12,  24,  30,  40,  41,  50,  58,  79,  95, 107, 111,
        124, 127, 141, 142, 148, 159, 160, 163, 176, 180, 183, 187, 193,
        196, 202, 206, 211, 213, 215, 216, 223, 226, 232, 236, 242, 244,
        256, 257, 258, 262, 264, 272, 275, 282, 283, 284, 294, 299]),
 array([300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300,
        300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300,
        300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300,
        300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300]))
```

可以看到返回了两个独立的数组，很明显第一个数组是坐标X,第二个数组是坐标Y。这样就能在二维空间中对某个特定值定位到具体的位置。

既然已经得到label等于4的行索引，那么就可以遍历行索引得到样本。除了遍历数组以外pandas提供了超级方便的接口。

```
import pandas as pd
df=pd.DataFrame(trains)
results=df.loc[np.where(trains[:,-1]==4)]
```

pandas中的loc接口，可以根据给定的行索引直接获取行数据。

Index	96	297	298	299	300
0		79	80	80	4
7		75	73	73	4
12		72	72	73	4
24		76	78	78	4
30		78	79	80	4
40		72	72	74	4
41		71	71	72	4
50		60	59	59	4
58		54	52	50	4
79		29	29	29	4
95		24	26	27	4
107		24	25	28	4
111		20	22	25	4

Top N方法

假设有下面一组字典集合，该集合是统计文本词的频率，我想找出文本中词频率的前两名的单词是什么?

```
list1={"numpy":8,"pandas":7,"python":6}
sorted(list1.items(),key=lambda item:item[1])[-2:]
```

```
In [33]: sorted(list1.items(),key=lambda item:item[1])[-2:]
Out[33]: [('pandas', 7), ('numpy', 8)]
```

不得不说可读性比较差，这里郭哥提供了一个非常简单的库。

```
import heapq
heapq.nlargest(2,list1,key=lambda item:item[1])
```

文本末为大家推荐一个巨好用的markdown编辑器，Typora,相信你会跟我一样用一次就喜欢上它，与绝大多数markdown编辑器不同，我用过csdn的markdown、vscode的markdown都是视图与编写环境分开，实话说不咋滴不怎么好用，然而Typora风格非常的简约，并且是边书写边同步视图,感觉就像在Txt文本中写内容一样，可以说它的视图与编写环境是一体的。

给你个链接自己体会下：https://www.typora.io/