数据可视化

作业一

姓 名: 冉 诗 菡
学 号: 15307130424
计算机科学与技术 (数据科学方向)

复旦大学
大数据学院

2018 年 3 月 21 日

1

1.1 题目

Design a similarity metric or distance metric for student information data formatted as follows:

表 1: Data format

Value	Description	Unit
V1	height of the student	m
V2	Weight of the student	kg
V3	Place of Birth	province/city
V4	Grade	grade 1-5

1.2 解答

在定义相似度的时候，V1，V2 为数值型数据，可以先标准化再归一化处理，然后运用欧式距离定义不同二维数组 (V1,V2) 之间的相似度。

$$d_{1,2} = d_{1,2}(V_p, V_q) = \sqrt{\sum_{i=1}^{2}(\frac{V_{i,p} - \mu_i}{\sigma_i} + \frac{V_{i,q} - \mu_i}{\sigma_i})^2}$$

再对 $d_{1,2}$ 做归一化，即：

$$d_{1,2} = \frac{d_{1,2}}{\max d_{1,2}}$$

V3 的处理可以考虑两种方案，一种是简单的认为 V3 是类别型数据，在同一个城市则为 1，在不同的城市则为 0。该方法简单明了，不用额外添加信息。另一种方案是将其视为区间型数据处理，需要额外的地理距离信息，将出生地信息转换成城市与城市间的距离信息 V3'，此时 V3' 变为数值型数据，处理方式如 V1 和 V2。这里采用第一种处理方法。

$$d_3(V_p, V_q) = \begin{cases} 0 & V_{3,p} = V_{3,q} \\ 1 & \text{otherwise} \end{cases}$$

V4 为序数型数据，直观上我们可以理解 “大一与大一的相似度高于大一与大二的相似度”，所以我们先将数据进行归一化。再用绝对差值定义两

表 2: 转换公式

RGB to CMYK	CMYK to RGB
C = 255 - R	R = 255 - C
M = 255 - G	G = 255 - M
Y = 255 - B	B = 255 - Y
K = 0	None

个 V4 之间的距离即可。

$$d_4 = d_4(V_p, V_q) = | \frac{V_{4,p} - 1}{\max V_4 - 1} - \frac{V_{4,q} - 1}{\max V_4 - 1} |$$

总距离指标可以用均值混合以上三种距离。

$$d = \frac{1}{3} \times (d_{1,2} + d_3 + d_4)$$

2

2.1 题目

在进行颜色的计算时通常需要在不同的颜色空间中进行数值转换，请用编程语言（建议 C/C++ 或 Python）实现 2 个常用的颜色空间的相互转换算法。请做好代码的注释。

2.2 解答

我选择的两个颜色空间是最常见的 RGB 和 CMYK，转换公式如表2。

代码如下：

```
from PIL import Image
import numpy as np

# RGB to CMY
def rgbtocmy (file):
    RGB_img = np.asarray(file)
    CMYK_img = 255 - RGB_img
    CMYK = Image.fromarray(np.uint8(CMYK_img))
    CMYK.mode = 'CMYK'
    return CMYK

# CMY to RGB
def cmytorgb (file):
    CMYK_img = np.asarray(file)
    RGB_img = 255 - CMYK_img[:,:,:3]
    RGB = Image.fromarray(np.uint8(RGB_img))
```

```
    RGB.mode = 'RGB'
    return RGB


if __name__ == '__main__':
    RGB =Image.open('./photo.JPG')
    CMYK = rgbtocmy(RGB)
    RGB = cmytorgb(CMYK)
    RGB.show()
```

其实 Python 中有一种更简单直接的命令，即是直接利用 convert 函数

```
CMYK = RGB.convert('CMYK')
```

由于在写代码的过程中发现 PIL 包中的 show 命令并不会以 CMYK 的形式展现出图片的样子，而是仍然会以 RGB 的形式来展示 (于是看到的图片就是与原图片互补的颜色)。所以我用的方法是先用内置的 convert 函数将读入的 RGB 转成 CMYK 形式，再用我自己的写的函数将 CMYK 复原成 RGB，即：

```
CMYK = RGB.convert('CMYK')
RGB = cmytorgb(CMYK)
RGB.show()
```

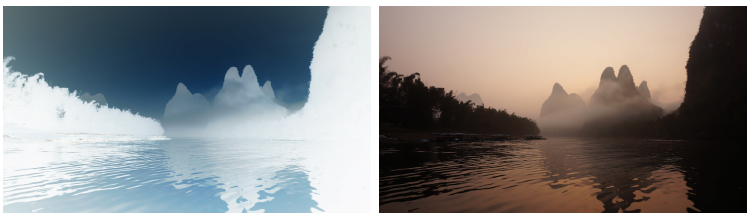

图 1: 转换成 CMYK 与复原 RGB 的输出实例

3

3.1 题目

视觉感知的相对判断原则在网络上常被用于制作一些包含错觉的图片，请查找两幅这样的图片，并说明所理解的其背后用到的视觉感知的相对性原理。

3.2 解答

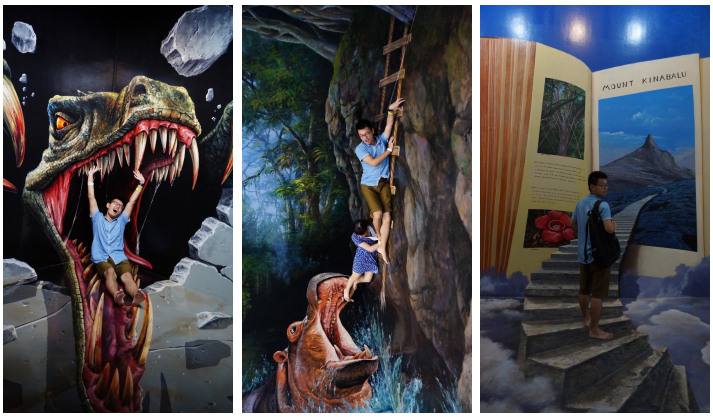

图 2: 三张在 3D 错觉馆中拍摄的照片

由于感知系统是基于相对判断而非绝对判断，又由格式塔贴近原则可知：人在获取视觉感知时会倾向于将事物理解成一个整体而不是部分之和。当我们在看这几张图片的时候，我们会以人为参照物去理解整张图片，从而有了“立体”感，我们倾向于将人融入 3D 画构造的背景中。但若是当时在现场换一个角度看，当人不能融入背景，则会有一种突兀的感觉，导致人去观察“部分”，而不是“整体”。

比如第二张图片，我们由于日常经验的影响，主观认为人是立体的，于是对于这张图片中的犀牛，由于它刚好在我们脚下，于是我们也通过类比“人”而认为它也是立体的，由此就伪装出了一副 3D 图画。同理可解释第一张图片，我们以认为参照，造成了一副人是被恐龙所食的假象。再比如第三张图片，因为我们一眼看过去，认为人是立体的站着的，于是便会相对的觉得台阶也是立体的，人仿佛是在“走路”一样。但其实都是假象。

4

4.1 题目

Design an algorithm which returns the n-th largest number from an array of unsorted numbers, submit your code (in python or c/c++) and test data.

4.2 解答

利用在数据结构上学到的内容，采用堆排序来解决这个问题。由于题目只要求返回第 n 大的数字，而非返回前 n 大的数字，所以对于排序过程进行了提前终止。

堆排序的算法实质上很简单，主要是利用了“完全二叉树”的一些性质，由此才能直接用数字索引代表左右孩子结点。中心思想是先建立最大堆，即父节点一定是比子结点大的完全二叉树，然后再通过递归把最大的调整到堆顶。

```
# 该函数将父节点与子结点中的最大值换到父节点
def MAX_Heapify(heap, HeapSize, root):
    left = 2*root + 1                    # 左孩子结点
    right = left + 1                     # 右孩子结点
    larger = root                        # 假设父节点是三者中最大的
    # 如果左结点的下标没有越界且左结点比父节点大
    if left < HeapSize and heap[larger] < heap[left]:
        larger = left
    # 如果右结点的下标没有越界且右结点比父节点大
    if right < HeapSize and heap[larger] < heap[right]:
        larger = right
    # 如果做了堆调整，则对结点的值做相应的更换，将父节点变为三者中最大的
    if larger != root:
        heap[larger], heap[root] = heap[root],heap[larger]
        # 不断递归
        MAX_Heapify(heap, HeapSize, larger)

# 构造一个堆
def Build_MAX_Heap(heap):
    HeapSize = len(heap)     # HeapSize用于比较是否已经没有孩子结点
    # 从倒数第二层的最后一个结点开始往前
    for i in range((HeapSize -2)//2, -1, -1):
        MAX_Heapify(heap, HeapSize, i)

# 将根节点（最大值）取出与此时的最后一位做对调
# 然后对前面len-1个节点继续进行对调整过程。
def HeapSort(heap, n):
    Build_MAX_Heap(heap)   # 将数组建成最大堆即保证所有的子结点比父节点小
    for i in range(len(heap)-1, len(heap)-1-n, -1):
        heap[0], heap[i] = heap[i], heap[0]
        MAX_Heapify(heap, i, 0)
    return heap

if __name__ == '__main__':
    tst = [5,6,2,3,8,1,8,7,9,0,10,4]
    print(HeapSort(tst, n=5))   # 找第n大的
```