Data Visualization Homework 1

Linyang He(15307130240)

1

首先关注V3. 可以发现，V3是类别型属性. 考虑公式

$$d(\mathbf{X},\mathbf{Y})_3 = \frac{p-m}{p} = \frac{2-m}{2} \in [0,1]$$

这里$p=2$, m = 相同的属性个数。

考虑V1，V2 是比值型属性。不妨设$\boldsymbol{X}=(v_{1x},v_{2x}),\boldsymbol{Y}=(v_{1y},v_{2y})$, 可以考虑直接用欧式距离。 为了让不同的属性的差异造成的影响平衡，我们主要采用标准化欧式距离。标准差分别为σ_1,σ_2

$$d(\mathbf{X},\mathbf{Y})_{12} = \sqrt{[\frac{(v_{1x}-v_{1y})}{\sigma_1}+\frac{(v_{2x}-v_{2y})}{\sigma_2}]^2}$$

最后考虑V4. V4显然是序数型属性，我们将grade1-5归一化，分别为$0,0.25,0.5,0.75,1$. 即是：

$$v_4 = \frac{k-1}{4}$$

这里k表示grade k。

于是采用欧式距离。

$$d(\mathbf{X},\mathbf{Y})_4 = |v_{4x}-v_{4y}| \in [0,1]$$

最后有

$$d(\mathbf{X},\mathbf{Y}) = \sqrt{d(\mathbf{X},\mathbf{Y})_{12}^2 + d(\mathbf{X},\mathbf{Y})_3^2 + d(\mathbf{X},\mathbf{Y})_4^2}$$

2

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。

解答：

原始图片如图

我们用Python和opencv库来实现RGB颜色空间和HSV颜色空间的互相转换。两种颜色空间的互相转换的算法来自网络：

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1678 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.1772 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

算法一(从RGB到HSV)，

```
def bgr2yuv(img):
    bgrcolors = cv2.split(img)
    # 在opencv中默认是brg
    B, G, R = bgrcolors[0], bgrcolors[1], bgrcolors[2]
    # 注意由于图像是uint8格式，需要进行数据的整理
    Y = 0.299 * R + 0.587 * G + 0.114 * B
    U = - 0.1687 * R - 0.3313 * G + 0.5 * B + 128
    V = 0.5 * R - 0.4187 * G - 0.0813 * B - 128
    yuvcolors = [Y, U, V]
    new_img = cv2.merge(yuvcolors)
    #数据类型的转换
    new_img = np.asarray(new_img,dtype=np.uint8)
    return new_img
```

输出的图像如图

算法二，

```
def yuv2bgr(img):
    yuvcolors = cv2.split(img)
    Y, U, V = yuvcolors[0], yuvcolors[1], yuvcolors[2]
    # 注意由于图像是uint8格式，需要进行数据的整理
    R = Y + 1.14 * V - 128
    G = Y - 0.39 * U - 0.58 * V - 128
    B = Y + 2.03 * U
    bgrcolors = [B, G, R]
    new_img = cv2.merge(bgrcolors)
    #数据类型的转换
    new_img = np.asarray(new_img,dtype=np.uint8)
    return new_img
```

把用算法一得到的图像再转回rgb, 输出的图像如图

可以发现，由于我们的算法中计算的时候采取的都是近似的方法，图像有一点失真。

而opencv中库函数

```
img1_lib = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
img2_lib = cv2.cvtColor(img1_lib, cv2.COLOR_YUV2BGR)
```

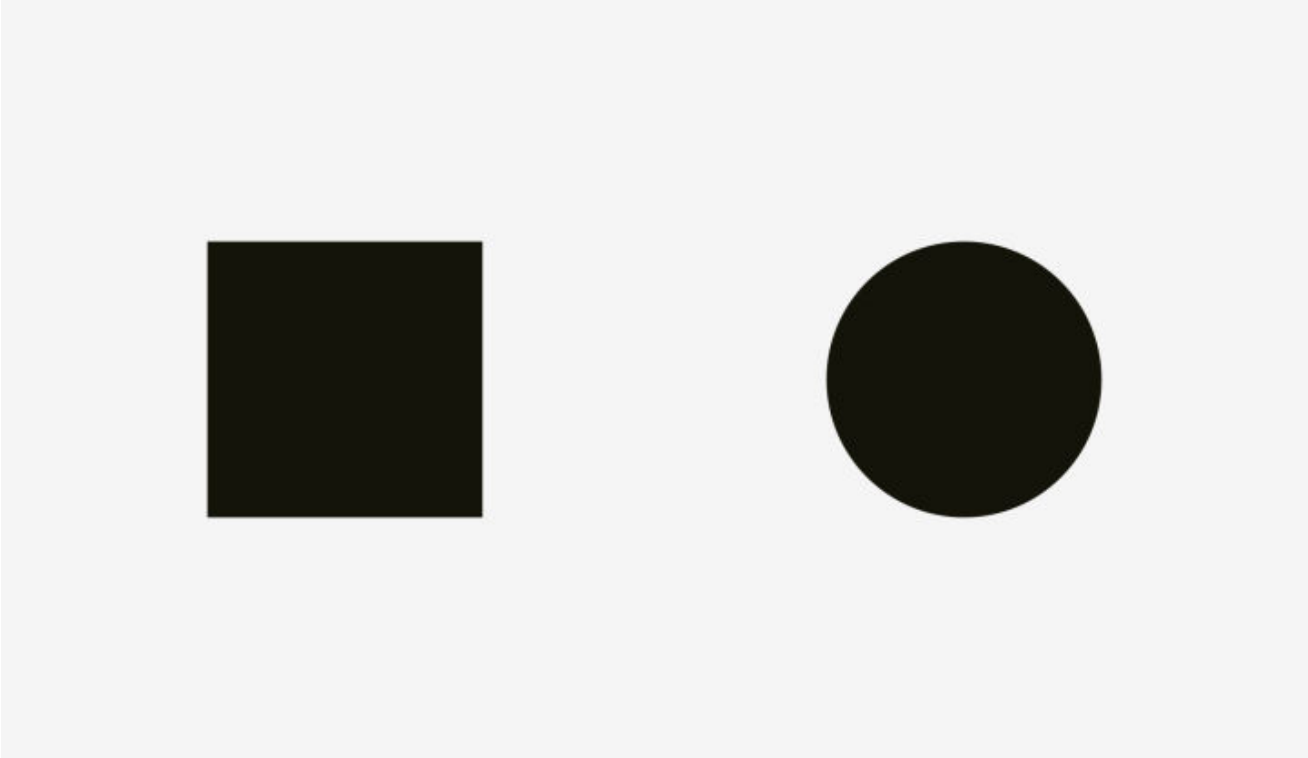
得到的图像分别如下图所示：


photo by Christy

发现库函数的要更为精准。

3

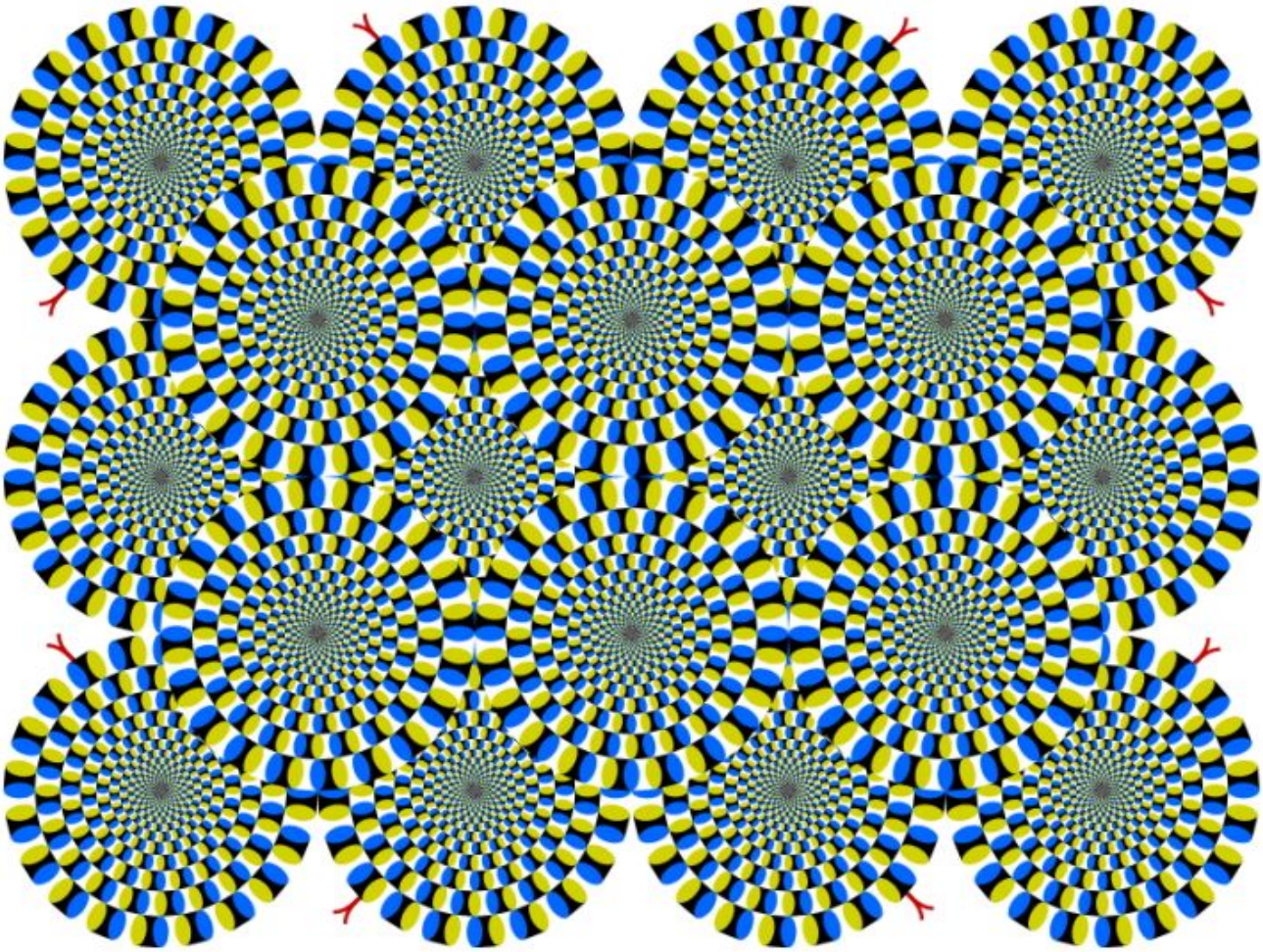
图像：



视觉错误：长宽各 400px 的两个图形看起来并不一样大。会觉得左边的正方形大一些

认知上的视觉相对原理：400px 的两个图形叠在一起，整个圆形都被包裹在了正方形之内，而正方形多出的四个面积区域就是造成这种视觉误差的原因。

图像：



视觉错误：会觉得图像在旋转，但是其实这是静止的图像。

认知上的视觉相对原理：根据网络资料显示，这是一种周边漂移错觉现象。至于原因，这个图像背后涉及到的眼球运动和神经生物学机制还存在着争议。从网上找到了一种解释：

> 外界信息传递到眼睛里，经过大脑处理，让人意识到看到了东西，这需要一段时间，这段时间是Response Latency（反应潜伏期），代表了人的视觉对外界信息刺激的反应过程。而人识别高对比度（比如黑与白），比识别低对比度（比如深灰与浅灰）的潜伏期更短，当高对比画面与低对比画面同时出现时，就显得我们**先看到了高对比画面**，如上图中的黑色和白色，**后看到了低对比画面**，如上图的蓝色和黄色。**接收信息的先后，是人们观察物体运动的基础**，如此一来，在Rotating Snakes中，我们先看到黑-白，后看到蓝-黄，就显得黑-白向蓝-黄的方向运动，于是，整个圆环就转起来了。

4

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

```
import numpy as np

import copy
```

```
def nlargest(array, n):
    '''
    We will use selection sort algorithm.
    '''
    length = len(array)
    if n > length or n < 0:
        raise Exception("Error!")

    new_array = copy.deepcopy(array)
    for i in range(length):
        for j in range(i+1,length):
            if new_array[j] > new_array[i]:
                new_array[j], new_array[i] = new_array[i], new_array[j]
        if i == n-1:
            result = new_array[i]
            break

    return result

test_data = np.random.randint(1,100,10)
print(test_data)
answer = nlargest(test_data, 3)
print(answer)
```

我们选择了选择排序算法，因为这样我们只用排序前n个数字即可，就可以找到第n大的数字了，相对来说算法效率比较高。运行结果入下

```
[90  3 82 39 71 32 37 63 62  2]
71
```

可见，上面数组的第3大的数字是71.