CS168 Assignment [9]

SUNet ID(s): 16337266

Name(s): 徐原

Part 1

(a) Code:

```
#part1 a
file="wonderland-tree.txt"
data=open(file,'r').readlines()
n=1200

def part1_a():
    num_of_1=0
    total=0
    #注意最后一个换行符
    for row in data:
        for num in row[:-1]:
            if num=='1':
                num_of_1+=1
            total+=1
    print("k/n:",num_of_1/total)
part1_a()
```

结果:

$k/n = 252/1200 = 0.21$

(b) CS168 给出的参考网站http://www.cvxpy.org 中建模的参考代码为:

```
    import cvxpy as cp
    import numpy as np

    # Problem data.
    m = 30
    n = 20
    np.random.seed(1)
    A = np.random.randn(m, n)
    b = np.random.randn(m)
```

```
    # Construct the problem.
    x = cp.Variable(n)

    objective = cp.Minimize(cp.sum_squares(A*x - b))
    constraints = [0 <= x, x <= 1]
    prob = cp.Problem(objective, constraints)

    # The optimal objective value is returned by `prob.solve()`.
    result = prob.solve()

    # The optimal value for x is stored in `x.value`.
    print(x.value)

    # The optimal Lagrange multiplier for a constraint is stored in
    # `constraint.dual_value`.
    print(constraints[0].dual_value)
```

基于本问题的建模：

```
def part1_b(r=600):
    A=np.random.randn(n,n)
    xarray=[]
    for row in data:
        for item in row[:-1]:
            xarray.append(int(item))
    #600*1200
    xarray=np.array(xarray)
    #print(len(b))
    b=np.dot(A[:r],xarray)
    x=cp.Variable(n)
    objective=cp.Minimize(cp.norm(x, 1))
    constraints=[b==A[0:r]*x,x>=0,x<=1]
    prob=cp.Problem(objective,constraints)
    result=prob.solve(cp.ECOS_BB)
    #
    print("Validation:",np.allclose(x.value,xarray))
```

最终输出：Validation: True，即验证通过，两个解可以看作是相等的。

(c) Code:

```
def part1_c():
    mini=.001
    xarray=[]
    for row in data:
        for item in row[:-1]:
            xarray.append(int(item))
    xarray=np.array(xarray)
    similar=[]
    #use binary search
    begin=1
    end=n-1
    while begin<end:
        middle=begin+int((end-begin)/2)+1
        result=np.linalg.norm(get_x_value(middle)-xarray,1)
        if result<mini:
            similar.append(middle)
            end=middle
        else:
            if begin==middle:
                break
            begin=middle
    find=min(similar)
    print("min r is:",find)
```

结果：找到 r*=460

(d) Code:

```
getR=460
#part d
def part1_d(r=460):
    x=[r+i for i in range(-10,3)]
    x_value=[]
    for i in x:
        x_value.append(np.linalg.norm(get_x_value(i),1))
    plt.plot(x,x_value)
    plt.title("part1_d")
    plt.xlabel("i")
    plt.ylabel("norm(xi-x)")
```

```
    plt.savefig("part1_d.png",format="png")
    plt.show()
```

题目中，提示 c 画出的图会有一个急剧的下降，画出图来的确如此:

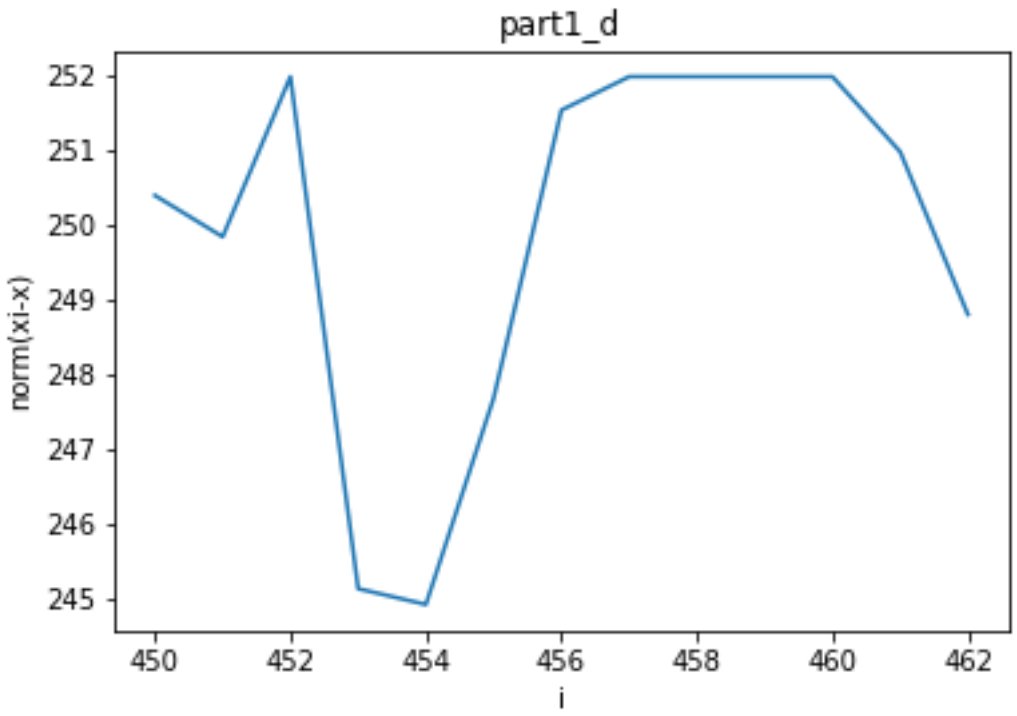

图 1: a sharp drop-off

Part 2

: http://nbviewer.jupyter.org/github/cvxgrp/cvxpy/blob/master/examples/notebooks/WWW/tv_inpainting.ipynbCode:

(a)

```
def part2_a():
    # Load the images.
    orig_img = Image.open("stanford-tree.png")
    corr_img = Image.open("corrupted.png")

    # Convert to arrays.
```

```
    Uorig = np.array(orig_img)[:,:,0]
    Ucorr = np.array(corr_img)[:,:,0]
    rows,cols=Uorig.shape
    # Known is 1 if the pixel is known,
    # 0 if the pixel was corrupted.
    Known = np.zeros((rows, cols))
    for i in range(rows):
        for j in range(cols):
             if Uorig[i, j] == Ucorr[i, j]:
                Known[i, j] = 1
    return Uorig,Ucorr,Known
```

由此读入图片且处理为矩阵形式。

(b) 在进行替换时注意边缘检测：

```
def part2_b():
    Uorig,Ucorr,Known=part2_a()
    m,n=Uorig.shape
    result=np.zeros((m,n))
    for i in range(m):
        for j in range(n):
            if Known[i][j]=='1':
                result[i][j]=int(Uorig[i][j])
            else:
                neighbors=[]
                if i-1>=0:
                    neighbors.append(int(Uorig[i-1][j]))
                if i+1<=m-1:
                    neighbors.append(int(Uorig[i+1][j]))
                if j-1>=0:
                    neighbors.append(int(Uorig[i][j-1]))
                if j+1<=n-1:
                    neighbors.append(int(Uorig[i][j+1]))
                ave=sum(neighbors)/len(neighbors)
                result[i][j]=ave
    plt.imshow(result)
    plt.savefig("part2_b.png", format = 'png')
part2_b()
```

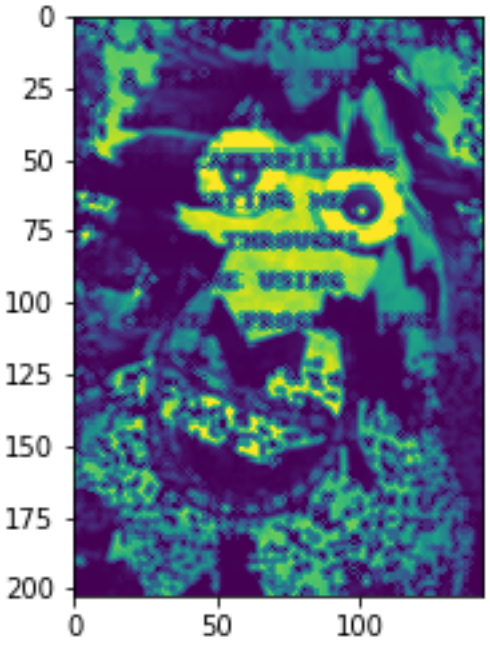

图 2: part2b 处理后的结果

分析：采用 4 近邻的方法重建图片，一定程度上是可取的，比如此处大概轮廓已恢复，但是当损失的特征相对重要时，无法确定填充对象，此时用 4 邻域估计不见得是一个好的选择。且可以看到图片被污染的痕迹依然十分明显。

(c) 参照模型建模，Code:

```
def part2_c():
    Uorig,Ucorr,Known=part2_a()
    U=cp.Variable(Uorig.shape)
    obj=cp.Minimize(cp.tv(U))
    constraints = [cp.multiply(Known, U) == cp.multiply(Known, Uorig)]
    prob=cp.Problem(obj, constraints)
    prob.solve(verbose = True)
    plt.imshow(U.value)
    plt.savefig("part2_c.png", format = "png")
part2_c()
```

```
ECOS 2.0.7 - (C) embotech GmbH, Zurich Switzerland, 2012-15. Web:
www.embotech.com/ECOS

It     pcost       dcost      gap   pres   dres    k/t    mu
step   sigma     IR    |   BT
 0  +0.000e+00  -0.000e+00  +1e+07  5e-01  1e-03  1e+00  3e+02
---    ---    1  1  - |  -  -
 1  +3.110e+05  +3.110e+05  +2e+06  2e-01  3e-04  4e+01  8e+01
0.8826  1e-01   1  1  1 |  0  0
 2  +5.481e+05  +5.481e+05  +7e+05  6e-02  8e-05  1e+01  3e+01
0.6942  5e-02   1  1  1 |  0  0
 3  +6.756e+05  +6.756e+05  +2e+05  2e-02  2e-05  5e+00  8e+00
0.7359  5e-02   1  1  1 |  0  0
 4  +7.225e+05  +7.226e+05  +8e+04  7e-03  8e-06  2e+00  3e+00
0.7112  1e-01   1  1  1 |  0  0
 5  +7.411e+05  +7.411e+05  +3e+04  2e-03  3e-06  9e-01  1e+00
0.8311  2e-01   1  1  1 |  0  0
 6  +7.491e+05  +7.491e+05  +8e+03  6e-04  7e-07  3e-01  3e-01
0.8316  1e-01   1  1  1 |  0  0
 7  +7.510e+05  +7.510e+05  +3e+03  2e-04  3e-07  9e-02  1e-01
0.7177  1e-01   1  1  1 |  0  0
 8  +7.518e+05  +7.518e+05  +7e+02  5e-05  6e-08  2e-02  2e-02
0.8790  1e-01   2  1  1 |  0  0
 9  +7.520e+05  +7.520e+05  +2e+02  2e-05  2e-08  8e-03  8e-03
```

图 3: part2c 生成序列

分析：建模结果和生成图片如图所示，Part2c 恢复的图像质量远高于 Part2b 种的重建效果。重建后不再有许多黑色像素点，图像看起来更像原始图像。

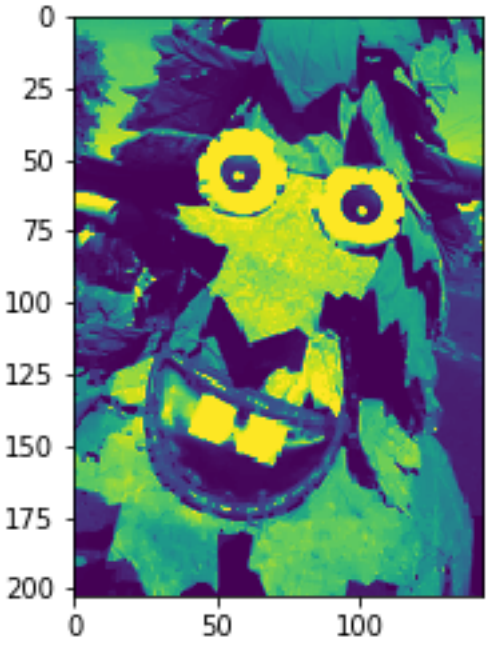

图 4: part2c 处理后的结果

(d) 压缩感知理论指出：只要信号是可压缩的或在某个变换域是稀疏的，那么就可以用一个与变换基不相关的观测矩阵将变换所得高维信号投影到一个低维空间上，然后用一个优化问题就可以从这些少量投影中以高概率重构出原信号。且这样的投影包含重构信号的足够信息。证明源于：https://blog.csdn.net/jbb0523/article/details/52013669