2023/08/13 魏嘉星 陈彦臻汇报

EMD的概念

Signature: 图像的直方图就是把全部像素值量化为一系列bin，统计每个bin对应的像素个数；bin值(横轴)就是图片的特征，bin的高度(纵轴)可以看作该特征的重要程度。

signature写作s=(m, w)，m是某特征，w是它的权重。它的用处是可以较稀疏地表达直方图内容，节省空间。

EMD本身是一个线性规划问题。简单来说，等于把P位置上的m个坑的土用最小的代价(工作量)搬到Q位置的n个坑

假设:

$$P=(P_1,w_{p1},P_2,w_{p2}),\ldots,P_M,w_{pM}),Q=(q_1,w_{q1},q_2,w_{q2}),\ldots,(q_N,w_{qN})$$

$$\text{其中：}p_i\text{：图像的某个特征}(\text{如}bin\text{值});w_{pi}\text{：特征}p_i\text{的权重}(\text{例如}bin\text{像素数量})$$

问题：希望找到一个flow矩阵F = [f(i, j)]，每一项代表pi到qj的流动数量，dij是pi位置到qj位置的代价(距离)：

$$WORK(P,Q,F)=\sum_{i=1}^{m}\sum_{j=1}^{n}d_{ij}f_{ij}$$

$$EMD(P,Q)=\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}d_{ij}f_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n}f_{ij}}$$

把问题写成线性规划问题:

$$min\sum_{i=1}^{m}\sum_{j=1}^{n}d_{ij}f_{ij}$$

$$s.t$$

$$f_{ij}\geq 0,i=1,2,\ldots,M;j=1,2,\ldots,N\text{：从初始位置流动到目标位置}$$

$$\sum_{j=1}^{N}f_{ij}\leq w_{pi},i=1,2,\ldots,M\text{：根据权重限制从出发点移动的总土量}$$

$$\sum_{i=1}^{M}f_{ij}\leq w_{qj},j=1,2,\ldots,N\text{：根据权重限制移动到终点的总土量}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n}f_{ij}=min(\sum_{i=1}^{M}w_{pi},\sum_{j=1}^{N}w_{qj})\text{：从起点移动到终点的总土量上限是}P\text{和}Q\text{地容量的较小值}$$

求解要素

1.边界条件　　2 . 目标函数　　3. 距离矩阵　4. 流量矩阵　　5. 归一化EMD

求解代码

```
import numpy as np
import scipy.optimize
import matplotlib.pyplot as plt
```

```
# 限制条件1 -----> float
def positivity(f):
    '''
    确认土堆从起点流向终点且不可回头
    '''
    return f

# 限制条件2 -----> float
def fromSrc(f, wp, i, shape):
    """
    从P运出去的泥土总量不可能超过P中的全部泥土量
    """
    fr = np.reshape(f, shape)
    f_sumColi = np.sum(fr[i,:])
    return wp[i] - f_sumColi

# 限制条件3 -----> float
def toTgt(f, wq, j, shape):
    """
    仓库Q接收的泥土总量不可能超过Q的总容量
    """
    fr = np.reshape(f, shape)
    f_sumRowj = np.sum(fr[:,j])
    return wq[j] - f_sumRowj

# 限制条件4 -----> float
def maximiseTotalFlow(f, wp, wq):
    """
    总运输量的上限是P，Q总容量的较小值
    """
    return f.sum() - np.minimum(wp.sum(), wq.sum())
```

```
# 目标函数 -----> float
def flow(f, D):
    """
    flow表示泥土从P被运到Q的总量
    需要最小化运输过程总做功
    """
    f = np.reshape(f, D.shape)
    return (f * D).sum()
```

```
# 运输距离 -----> float
def groundDistance(x1, x2, norm = 2):
    """
    L-norm distance
    Default norm = 2
    """
    return np.linalg.norm(x1-x2, norm)

# 距离矩阵 -----> matrix
def getDistMatrix(s1, s2, norm = 2):
    """
    计算P，Q两地之间每一次运输所需距离组成的矩阵
    距离默认采用L2范数计算
    """
    # rows = s1 feature length
    # cols = s2 feature length
    numFeats1 = s1.shape[0]
    numFeats2 = s2.shape[0]
    distMatrix = np.zeros((numFeats1, numFeats2))

    for i in range(0, numFeats1):
        for j in range(0, numFeats2):
            distMatrix[i,j] = groundDistance(s1[i], s2[j], norm)

    # 这里提供了一种更快的方式(需要scipy.spatial)
    #import scipy.spatial
    #distMatrix = scipy.spatial.distance.cdist(s1, s2)

    return distMatrix

# 流量矩阵 -----> matrix
def getFlowMatrix(P, Q, D):
    """
    计算P，Q之间的流量(运输量)
    """
    numFeats1 = P[0].shape[0]
    numFeats2 = Q[0].shape[0]
    shape = (numFeats1, numFeats2)

    # Constraints
    cons1 = [{'type':'ineq', 'fun' : positivity},
             {'type':'eq', 'fun' : maximiseTotalFlow, 'args': (P[1], Q[1],)}]

    cons2 = [{'type':'ineq', 'fun' : fromSrc, 'args': (P[1], i, shape,)} for i
in range(numFeats1)]
    cons3 = [{'type':'ineq', 'fun' : toTgt, 'args': (Q[1], j, shape,)} for j in
range(numFeats2)]

    cons = cons1 + cons2 + cons3

    # Solve for F (solve transportation problem)
    F_guess = np.zeros(D.shape)
    F = scipy.optimize.minimize(flow, F_guess, args=(D,), constraints=cons)
    F = np.reshape(F.x, (numFeats1,numFeats2))

    return F
```

```
# 归一化EMD
def EMD(F, D):
    """
    归一化：避免总运量改变不一致导致计算出的总做功发生改变，使得EMD在P和Q结构保持不变的前提下可
以适配于不同大小的计算任务
    """
    return (F * D).sum() / F.sum()

# 运行函数
def getEMD(P,Q, norm = 2):
    """
    EMD computes the Earth Mover's Distance between
    the distributions P and Q
    P和Q是尺寸(2,N)的矩阵，第一行是N个特征，第二行是对应的比重
    采用L2范数作为距离的定义(默认的欧几里得距离)
    """
    D = getDistMatrix(P[0], Q[0], norm)
    F = getFlowMatrix(P, Q, D)

    return EMD(F, D)
```

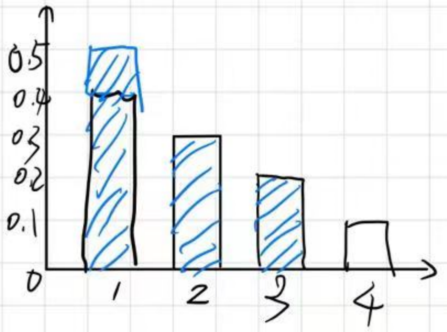
```
# 例子请见文末，此处提供计算代码
# 获取signature
def getExample_GaussianHistograms(N = 15, showPlot = True):
    """
    returns signature1[features][weights], signature2[features][weights]
    """
    x = np.linspace(-1,1,N)
    y1 = np.exp(-np.power(x - 0.0, 2.0) / (2 * np.power(0.2, 2.0)))
    y2 = np.exp(-np.power(x - 0.5, 2.0) / (2 * np.power(0.2, 2.0)))
    y1 /= np.sum(y1)
    y2 /= np.sum(y2)

    if showPlot:
        plt.bar(x, y1, width=0.1, alpha=0.5)
        plt.bar(x, y2, width=0.1, alpha=0.5)

    #features1 = y1.reshape((N,1))
    #weights1 = (1.0/N) * np.ones((N))

    #features2 = y2.reshape((N,1))
    #weights2 = (1.0/N) * np.ones((N))

    #signature1 = (features1, weights1)
    #signature2 = (features2, weights2)

    signature1 = (x.reshape((N,1)), y1.reshape((N,1)))
    signature2 = (x.reshape((N,1)), y2.reshape((N,1)))

    return signature1, signature2

# 做Rubner比较
def doRubnerComparisonExample():
    # Setup
    P, Q = getExampleSignatures1()
```

```python
    # Get EMD
    emd = getEMD(P, Q)

    # Output result
    print('We got: '+str(emd))
    print('Rubner C example got 160.54277') # 此处是作者使用Rubner C得到的结果

# 绘制bin直方图
def doGaussianHistogramExample():
    showPlot = True
    # Setup
    P, Q = getExample_GaussianHistograms(showPlot=showPlot)
    # Get EMD
    emd = getEMD(P, Q)
    # Output result
    print('EMD: '+str(emd))
    if showPlot:
        plt.show()


if __name__ == '__main__':
    print('EMD')

    doRubnerComparisonExample() # 与Rubner C得到的EMD结果对比
    #doGaussianHistogramExample() # 绘制高斯直方图

    print('Success')
```

```python
# 例子
def getExampleSignatures1():
    """
    returns signature1[features][weights], signature2[features][weights]
    """
    features1 = np.array([[100, 40, 22],
                          [ 211, 20, 2 ],
                          [ 32, 190, 150 ],
                          [ 2, 100, 100 ] ])
    weights1 = np.array([ 0.4, 0.3, 0.2, 0.1 ])


    features2 = np.array([ [ 0, 0, 0 ],
                           [ 50, 100, 80 ],
                           [ 255, 255, 255 ] ])
    weights2 = np.array([ 0.5, 0.3, 0.2 ])

    signature1 = (features1, weights1)
    signature2 = (features2, weights2)

    return signature1, signature2

def getExampleSignatures2():
    """
    returns signature1[features][weights], signature2[features][weights]
    """
    features1 = np.array([[100, 40, 22],
                            [ 211, 20, 2 ],
                            [ 32, 190, 150 ],
                            [ 2, 100, 100 ] ])
    weights1 = np.array([ 1.0,1.0,1.0,1.0 ])


    features2 = np.array([ [ 0, 0, 0 ],
                            [ 50, 100, 80 ],
                            [ 255, 255, 255 ] ])
    weights2 = np.array([ 1.0,1.0,1.0 ])

    signature1 = (features1, weights1)
    signature2 = (features2, weights2)

    return signature1, signature2
```

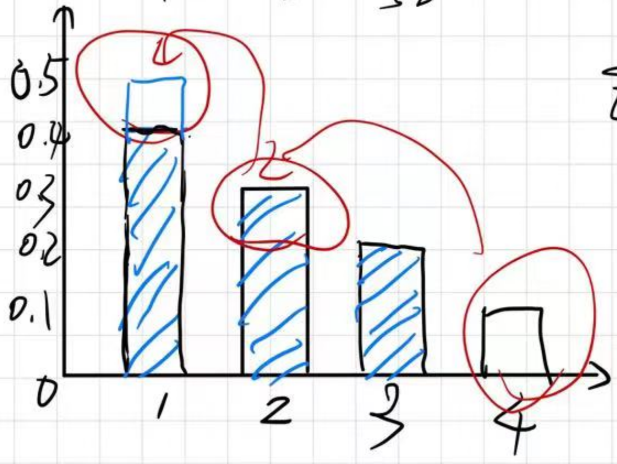
上述代码的运行结果如下： EMD=160.54277

```python
def getExampleSignatures1():
    """
    returns signature1[features][weights], signature2[features][weights]
    """
    features1 = np.array([[100, 40, 22],
                          [ 211, 20, 2 ],
                          [ 32, 190, 150 ],
                          [ 2, 100, 100 ] ])
    weights1 = np.array([ 0.4, 0.3, 0.2, 0.1 ])


    features2 = np.array([ [ 0, 0, 0 ],
                           [ 50, 100, 80 ],
                           [ 255, 255, 255 ] ])
    weights2 = np.array([ 0.5, 0.3, 0.2 ])

    signature1 = (features1, weights1)
    signature2 = (features2, weights2)

    return signature1, signature2
```



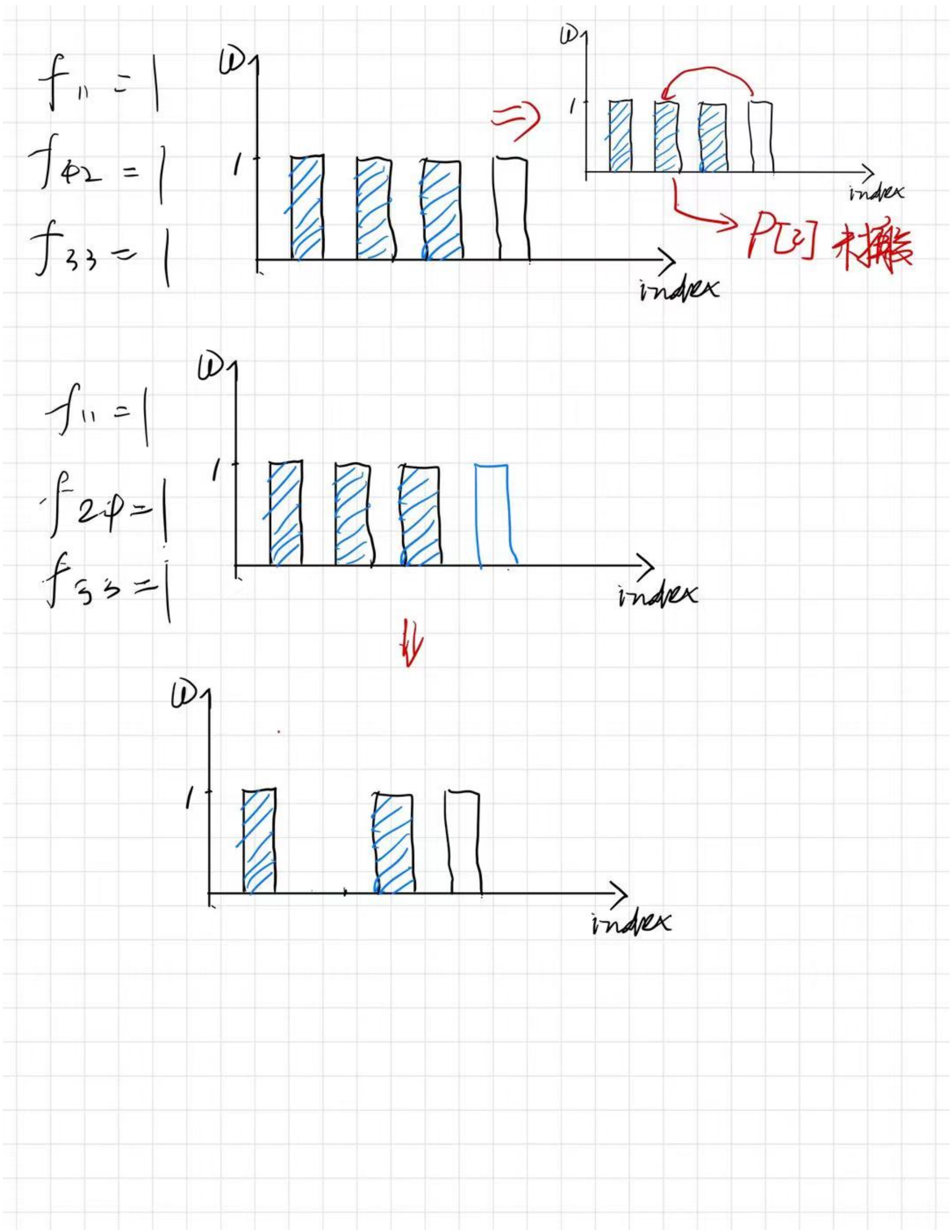
$$EMD = \frac{\sum_{ij} d_{ij} f_{ij}}{\sum f_{ij}}$$

$f_{11} = 0.4$	$d_{11} = 109.93$
$f_{21} = 0.1$	$d_{21} = 211.96$
$f_{22} = 0.2$	$d_{22} = 195.97$
$f_{33} = 0.2$	$d_{33} = 254.91$
$f_{42} = 0.1$	$d_{42} = 52.00$



$EMD \approx 160.54$

```
def getExampleSignatures2():
    """
    returns signature1[features][weights], signature2[features][weights]
    """
    features1 = np.array([[100, 40, 22],
                          [ 211, 20, 2 ],
                          [ 32, 190, 150 ],
                          [ 2, 100, 100 ] ])
    weights1 = np.array([ 1.0,1.0,1.0,1.0 ])



    features2 = np.array([ [ 0, 0, 0 ],
                          [ 50, 100, 80 ],
                              [ 255, 255, 255 ] ])
    weights2 = np.array([ 1.0,1.0,1.0 ])

    signature1 = (features1, weights1)
    signature2 = (features2, weights2)

    return signature1, signature2
```



f11 = 1
f42 = 1
f33 = 1
①
1
index
P[2] 未搬
f11 = 1
f2φ = 1
f33 = 1