Lab4 实验报告

李青雅 523030910004 电院 2301

2024 年 12 月 22 日

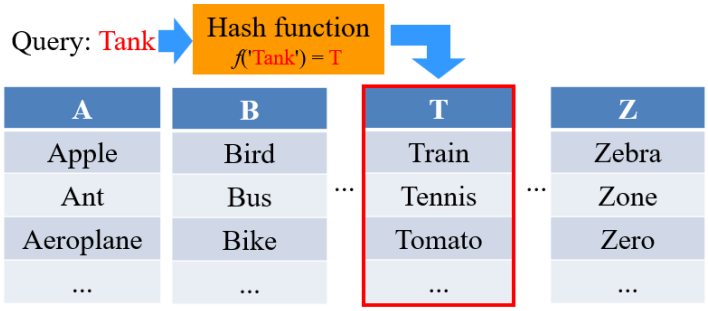
目录

1 实验概览

用 Nearest neighbor (NN) 或 k-nearest neighbor (KNN) 在数据库中检索和输入数据距离最近的 1 个或 k 个数据，一般情况下算法复杂度为 O(n)（例如暴力搜索），优化情况下可达到 O(log n)（例如二叉树搜索），其中 n 为数据库中的数据量。当数据库很大（即 N 很大时），搜索速度很慢。因此我们使用局部敏感哈希（Locality-Sensitive Hashing，LSH）的方法进行图像匹配。

LSH 是一种有效解决高维数据检索问题的算法，通过将高维向量投影到低维哈希空间中，使得相似的数据点以高概率映射到相同的哈希桶内，从而显著提升检索效率。哈希的基本思想是按照某种规则（Hash 函数）把数据库中的数据分类，对于输入数据，先按照该规则找到相对应的类别，然后在其中进行搜索。由于某类别中的数据量相比全体数据少得多，因此搜索速度大大加快。

下图为体现哈希思想的查字典类比：



2 实验环境

Python, OpenCV, Numpy, time, Matplotlib

3 实验流程

这里以使用图像的颜色直方图作为特征信息，使用 LSH 算法进行图像匹配。文末会分别给出以灰度和梯度作为信息的 LSH 算法实现。

3.1 数据表示

图像、视频、音频等数据都可以表示成一个 d 维的正数向量 $p = (p_1, p_2 ... p_d)$，其中 p_i 是满足 $0 \leq p_i \leq C$ 的整数，C 是整数的上限。在本实验中，我们将每幅图像按下图分为四个区域：

接下来，我们对每个区域的颜色直方图提取特征，归一化，并量化为向量，再将这四个向量合并为这幅图像的颜色直方图特征向量。根据以下条件，将 p 的每个分量投影到 0、1 或 2：

$$p_i = \begin{cases} 0, & \text{if } 0 \le p_i < 0.3 \\ 1, & \text{if } 0.3 \le p_i < 0.6 \\ 2, & \text{if } 0.6 \le p_i \end{cases}$$

于是最终得到的特征向量的每个元素满足 $p_i \in 0, 1, 2$。下图为部分图片的颜色直方图特征向量：

```
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2]
[0, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 0]
[2, 0, 2, 2, 2, 0, 1, 2, 2, 2, 1, 2]
[2, 0, 2, 1, 1, 0, 2, 2, 2, 1, 2, 2]
[0, 2, 0, 2, 2, 2, 2, 2, 2, 0, 0, 2]
[2, 2, 2, 2, 2, 2, 0, 2, 0, 2, 2, 1]
[0, 2, 2, 0, 1, 0, 2, 0, 2, 2, 2, 2]
[2, 2, 1, 2, 2, 0, 2, 2, 2, 2, 2, 2]
[0, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2]
[2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 1, 2]
[2, 2, 1, 1, 1, 0, 1, 2, 2, 1, 2, 2]
[2, 1, 2, 0, 2, 2, 0, 2, 0, 2, 2, 0]
[2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2]
```

3.2 LSH 预处理

这里的预处理需要我们将特征向量通过哈希函数映射到高维稀疏空间中，从而为 LSH 提供更有效的哈希值计算。

3.2.1 利用汉明码预处理

我们可以通过汉明码的方式将特征向量映射到高维空间中，由于这里 p 中的每个元素都是 0、1、2 中的一个，我们只需要关心 0、1、2 所对应的汉明码：

$$0 \to 00, 1 \to 10, 2 \to 11$$

例如，对于一个长度为 4 的特征向量 $p = (0, 1, 2, 1, 0, 2)$，其映射后的汉明码表示为 $v(p) =$ 001011100011。

但是在 LSH 中，不直接使用完整的高维表示 $v(p)$，而是选择某些下标进行投影，用于生成哈希值，其中投影集合用 I_i 表示。我们定义 $v(p)$ 在集合 $I_i = i_1, i_2, ...i_m, 1 \le i_1 < i_2 < ... < i_m \le d'$ 上的投影为 $g_i(p) = (p_i1, p_i2, ...p_im)$，其中 p_ij 为 v(p) 的第 i_j 个元素。对于上述 p，它在 {1,3,7,8} 上的投影为 (0,1,1,0)。

3.2.2 利用哈希函数计算

我们也可以不必显式地将 d 维空间中的点 p 映射到 d' 维 Hamming 空间向量 $v(p)$，而是采用哈希函数族的映射方法。这里我们通过定义 $I = 1, 2, ..., d'$，且定义 p|I 为向量 p 在坐标集 I 上的投影，即以坐标集 I 中每个坐标为位置索引，取向量 p 对应位置的比特值并将结果串联起来。

$I|i$ 表示 I 中范围在 $(i-1)*C+1$ $i*C$ 中的坐标。例如，$I = \{1,3,7,8\}, I|1 = \{1\}, I|2 = \{3\}, I|3 = \emptyset, I|4 = \{7,8\}, I|5 = I|6 = \emptyset$。$v(p)$ 在 I 上的投影即是 $v(p)$ 在 $I|i(i = 1, 2, ..., d)$ 上的投影串联，$v(p)$ 在 I|i 上的投影是一串 1 紧跟一串 0 的形式，需要求出 1 的个数。1 的个数由以下函数计算：

$$|I|i - C * (i-1) \leq x_i|$$

在上文的例子中，$p = (0, 1, 2, 1, 0, 2)$，因为 $I|3 = I|5 = I|6 = \emptyset$，所以这里只需要考虑 $i = 1, 2, 4$ 的情况。

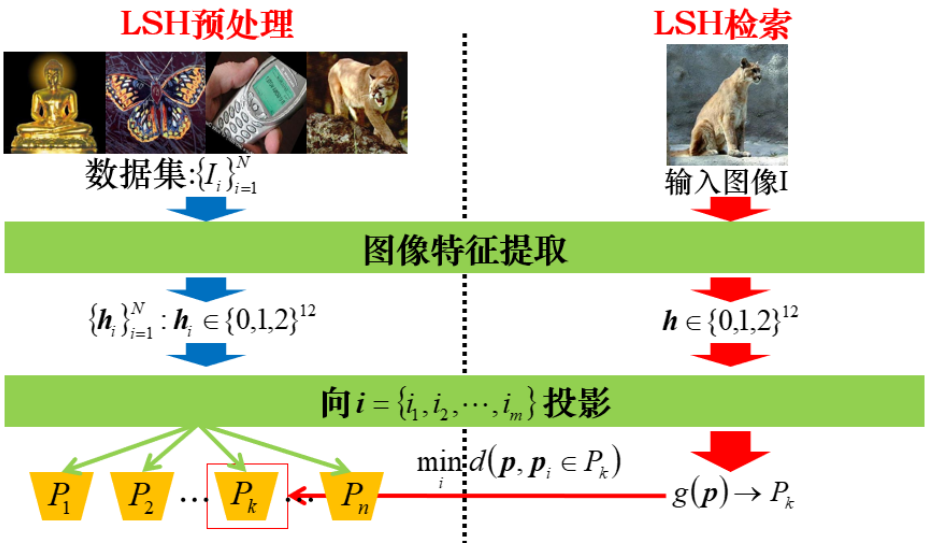
- $i = 1$ 时，$I|1 = \{1\}$，$|I|1 - C * (1-1) \leq 0| = 0$，$I|1$ 上的投影为 0。
- $i = 2$ 时，$I|2 = \{3\}$，$|I|2 - C * (2-1) \leq 1| = 1$，$I|2$ 上的投影为 1。
- $i = 4$ 时，$I|4 = \{7,8\}$，$|I|4 - C * (4-1) \leq 2| = 2$，$I|4$ 上的投影分别为 1、0。

所以将它们串联得到 $g(p) = (0, 1, 1, 0)$，即为 $v(p)$ 在 I 上的投影。

3.3 LSH 检索

$g(p)$ 被称作 Hash 函数，对于容量为 N 的数据集 $P = \{p_i\}_{i=1}^N$，$g(p)$ 可能的输出有 n 个，n 远小于 N，这样就将原先的 N 个数据分成了 n 个类别，其中每个类别中的数据具有相同的 Hash 值，不同类别的数据具有不同的 Hash 值。

对于待检索的输入 p，先计算 $g(p)$，找到其对应的类别，然后在该类别的数据集中进行搜索，能够大大加快检索速度。下图为检索算法的流程示意图：



4 实验结果

这里我们分别改变 I 的值和获得 $g(p)$ 的方法（见 3.2），对比两种方法的检索时间和检索效果。总的来说，检索的效果都很精准，主要是检索时间的差异。

4.1 改变 I 的值

下图我们均使用汉明码的方式获得 $g(p)$,但是改变 I 的值,分别取 I={1,7,8,10,20}、I={1,7,10,15,20}，I={1,7,8,10,15,20} 和 I={1,3,7,8,17,19,21}。如下图所示：

```
When using lsh_search......
The efficiency of lsh_search is 254.990816116333ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
When using nn_search......
The efficiency of nn_search is 255.66959381103516ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
```

图 1: I={1,7,8,10,20}

```
When using lsh_search......
The efficiency of lsh_search is 260.98179817199707ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
When using nn_search......
The efficiency of nn_search is 261.5551948547363ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
```

图 2: I={1,7,10,15,20}

```
When using lsh_search......
The efficiency of lsh_search is 255.0206184387207ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
When using nn_search......
The efficiency of nn_search is 257.9345703125ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
```

图 3: I={1,7,8,10,15,20}

```
When using lsh_search......
The efficiency of lsh_search is 256.1812400817871ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
When using nn_search......
The efficiency of nn_search is 258.6369514465332ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
```

图 4: I={1,3,7,8,17,19,21}

可以看出，LSH 算法比暴力搜索的速度稍快，当待检索的数据量较大时，LSH 算法的优势将更加明显。这里对于不同的 I 值，检索时间也有所不同，但是几乎没有太大差别，笔者也并没有发现以何种规律取 I 的值会使检索时间最短。但从结果可以看出，I={1,7,8,10,15,20} 的检索时间最短。

4.2 改变获得 $g(p)$ 的方法

我们以 I={1,7,8,10,15,20} 为例，分别使用汉明码和哈希函数的方式获得 $g(p)$，如下图所示：

```
When using lsh_search......
The efficiency of lsh_search is 256.1812400817871ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
When using nn_search......
The efficiency of nn_search is 258.6369514465332ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
```

图 5: 汉明码

```
When using lsh_search......
The efficiency of lsh_search is 261.87872886657715ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
When using nn_search......
The efficiency of nn_search is 260.62583923339844ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
```

图 6: 哈希函数

我们能发现，使用汉明码的方式获得 $g(p)$ 的检索时间要比使用哈希函数的方式短，这可能是因为汉明码的方式更加简单，计算量更小。也有可能是图片集较小，加上每次代码运行的时间都会有所不同。当 C 较大时，哈希函数的方式可能会更加快速。

5 拓展思考

5.1 检索效果思考

使用颜色分布直方图作为特征信息的检索效果非常好，符合笔者的实验预期。检索出的图像与输入图像的相似性主要体现在颜色分布上，其中图库中的 12 号图片与输入图像的颜色分布较为相

似，但程序仍然将其摒弃，检索出与目标图像完全一致的图像，说明程序的检索效果非常精准。

5.2 设计其他特征信息进行检索

除了颜色直方图，我们还可以使用灰度直方图、梯度直方图等信息作为特征信息进行检索。这里我们分别以灰度和梯度作为特征信息，实现 LSH 算法。统一使用哈希函数的方式获得 $g(p)$，取 I={1,7,8,10,20} 进行检索。

```
When using lsh_search......
The efficiency of lsh_search is 28.514623641967773ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
When using nn_search......
The efficiency of nn_search is 28.442859649658203ms
There is(are) 1 result(s), and they are(it is):
        dataset/38.jpg
```

图 7: 灰度直方图

上图为灰度直方图的检索结果，可以看出，两种算法的时间差异不大，准确率都很高。下图为梯度直方图的检索结果，检索的效果明显比彩色直方图和灰度直方图差，这可能是因为梯度直方图的特征信息不够明显，也可能是因为梯度直方图的特征信息不够稳定，导致检索效果不佳。

```
When using lsh_search......
The efficiency of lsh_search is 234.53044891357422ms
There is(are) 18 result(s), and they are(it is):
        dataset/1.jpg
        dataset/4.jpg
        dataset/5.jpg
        dataset/6.jpg
        dataset/7.jpg
        dataset/8.jpg
        dataset/10.jpg
        dataset/14.jpg
        dataset/17.jpg
        dataset/18.jpg
        dataset/20.jpg
        dataset/23.jpg
        dataset/25.jpg
        dataset/26.jpg
        dataset/33.jpg
        dataset/36.jpg
        dataset/38.jpg
        dataset/40.jpg
```

图 8: LSH 算法实现

```
When using nn_search......
The efficiency of nn_search is 236.5090847015381ms
There is(are) 18 result(s), and they are(it is):
        dataset/1.jpg
        dataset/4.jpg
        dataset/5.jpg
        dataset/6.jpg
        dataset/7.jpg
        dataset/8.jpg
        dataset/10.jpg
        dataset/14.jpg
        dataset/17.jpg
        dataset/18.jpg
        dataset/20.jpg
        dataset/23.jpg
        dataset/25.jpg
        dataset/26.jpg
        dataset/33.jpg
        dataset/36.jpg
        dataset/38.jpg
        dataset/40.jpg
```

图 9: NN 算法实现

图 10: 梯度直方图

6 实验心得与体会

这次 LSH 的实验比上次的 SIFT 实验更清晰，更容易理解。LSH 算法的思想很简单，但是实现起来却有很多细节需要注意，比如哈希函数的设计、I 的取值等。以及每次实验都会碰到的数组越界问题，这次和室友一起讨论出了解决方案。这次实验让我更加熟悉了 LSH 算法的实现，也让我对图像检索和匹配有了更深的理解。

7 源代码

7.1 以颜色分布直方图作为特征信息的 LSH 算法实现

```
import cv2
from matplotlib import pyplot as plt
import time
# 首先获取图像对应的特征向量
# 通过颜色直方图计算相应的特征向量
def color(filename):
    # 读入图（目标的特征向量）
    in_img = cv2.imread(filename+'.jpg')
    image = cv2.cvtColor(in_img, cv2.COLOR_BGR2RGB)
    # 需要注意是4块区域，分为BGR
    blue, green, red = [0,0,0,0], [0,0,0,0], [0,0,0,0]
    # 宽len(image),长len(image[0]), 读入的图片按照r,g,b的顺序
    # 分成四块来进行计算
    # 第一块
    for i in range(0,len(image)//2):
        for j in range(0,len(image[0])//2):
            blue[0] += image[i][j][2]
            green[0]+= image[i][j][1]
            red[0]  += image[i][j][0]
    # 归一化处理
    tot = blue[0] + green[0] + red[0]
    blue[0], green[0], red[0] = blue[0]*1.0/tot, green[0]*1.0/tot, red[0]*1.0/tot

    # 第二块
    for i in range(0,len(image)//2):
        for j in range(len(image[0])//2,len(image[0])):
            blue[1] += image[i][j][2]
            green[1]+= image[i][j][1]
            red[1]  += image[i][j][0]
    # 归一化处理
    tot = blue[1] + green[1] + red[1]
    blue[1], green[1], red[1] = blue[1]*1.0/tot, green[1]*1.0/tot, red[1]*1.0/tot

    # 第三块
    for i in range(len(image)//2,len(image)):
        for j in range(0,len(image[0])//2):
            blue[2] += image[i][j][2]
            green[2]+= image[i][j][1]
            red[2]  += image[i][j][0]
    # 归一化处理
    tot = blue[2] + green[2] + red[2]
    blue[2], green[2], red[2] = blue[2]*1.0/tot, green[2]*1.0/tot, red[2]*1.0/tot

    # 第四块
    for i in range(len(image)//2,len(image)):
        for j in range(len(image[0])//2,len(image[0])):
            blue[3] += image[i][j][2]
            green[3]+= image[i][j][1]
            red[3]  += image[i][j][0]
    # 归一化处理
    tot = blue[3] + green[3] + red[3]
    blue[3], green[3], red[3] = blue[3]*1.0/tot, green[3]*1.0/tot, red[3]*1.0/tot

    p = []
    for i in range(4):
```

```
            p.append(blue[i])
            p.append(green[i])
            p.append(red[i])
        # 计算出0、1、2类型的特征向量
        for i in range(len(p)):
            if p[i] < 0.3:
                p[i] = 0
            elif p[i] < 0.6:
                p[i] = 1
            else:
                p[i] = 2
        return p


    # 计算Hamming码来获取投影，其中g是需要被投影的方向
    def Hamming(p,g):
        # 由于p是12维的向量，构造v(p)需要24维的数组
        lsh = []
        v = [0] * 24
        for i in range(12):
            # 当前位置只有00,10,11三种情况，第一二位分别对应2*i,2*i+1
            if p[i] == 1:
                v[2*i] = 1
            if p[i] == 2:
                v[2*i] = 1
                v[2*i+1] = 1
        # eg.v(p)=001011100011
        # 对于上述 p，它在{1,3,7,8}上的投影为(0,1,1,0)
        for i in g:
            lsh.append(v[i-1])
        return lsh


    # 哈希函数计算lsh
    def hash(p,g):
        # I是Hash桶
        I  = [[] for i in range(12)]
        # print(I)
        lsh = []
        # I|i表示I中范围在(i-1)*C+1~i*C中的坐标:
        for i in g:
            # print((i-1)//2)
            I[(i-1)//2].append(i)
        for i in range(12):
            if I[i] != []:
                for Ii in I[i]:
                    if Ii-2*i <= p[i]:
                        lsh.append(1)
                    else:
                        lsh.append(0)
        return lsh


    # 对搜索库中的图像（共40张，均以i.jpg为文件名）建立索引
    def pre(g):
        Hash, Hash_id, character = [], [], []
        for i in range(1,41):
            filename ='dataset/'+str(i)
            p = color(filename)
```

```
            character.append(p)
            # 可用I的方式获取投影
            # 也可以用汉明法获取投影即lsh = Hamming(p,g)
            # lsh = Hamming(p,g)
            lsh = hash(p,g)
            # 需要注意的是，由于循环的时候文件用的是1.jpg->40.jpg
            # 然而数组append只能是从0开始，因而在具体实现的时候需要-1，同样回答了输出的+1原因
            try:
                Hash_id[Hash.index(lsh)].append(i - 1)
            except:
                Hash.append(lsh)
                Hash_id.append([i - 1])
        return Hash, Hash_id, character


    # 利用lsh检索图片，进行图片索引的查找
    def lsh_search(g, Hash, Hash_id, character):
        result = []
        p = color('target')
        # 也可以用汉明法获取投影即lsh = Hamming(p,g)
        # lsh = Hamming(p,g)
        lsh = hash(p,g)
        # 判断该图片的特征向量是否合理
        if lsh not in Hash:
            return result
        ID = Hash.index(lsh)
        # 这里的NN法可以直接判断向量是否相等，这样直接且高效
        # print(Hash_id[ID])
        for i in Hash_id[ID]:
            if character[i] == p:
                result.append('dataset/'+str(i+1)+'.jpg')
        return result

    # 不使用LSH，直接利用NN法进行搜索
    def nn_search(character):
        result = []
        p = color('target')
        for i in range(0,40):
            # 直接枚举
            if character[i] == p:
                result.append('dataset/'+str(i+1)+'.jpg')
        return result


    # g = [1,7,8,10,20]
    # g = [1,7,10,15,20]
    # g = [1,7,8,10,15,20]
    g = [1,3,7,8,17,19,21]


    # 首先进行预处理，为所有待查图像建立相应索引
    Hash, Hash_id, character = pre(g)
    # print(character)

    start = time.time()
    print("When using lsh_search......")
    result = lsh_search(g, Hash, Hash_id, character)
    end = time.time()
    print("The efficiency of lsh_search is {}ms".format((end-start)*1000))
```

```
    if result == []:
        print("No matching picture!")
    else:
        print("There is(are) {} result(s), and they are(it is):".format(len(result)))
        for i in result:
            print('\t' + i)









    start = time.time()
    print("When using nn_search......")
    result = nn_search(character)
    end = time.time()
    print("The efficiency of nn_search is {}ms".format((end-start)*1000))
    if result == []:
        print("No matching picture!")
    else:
        print("There is(are) {} result(s), and they are(it is):".format(len(result)))
        for i in result:
            print('\t' + i)
```

7.2 以灰度直方图作为特征信息的 LSH 算法实现

这里只需要将 color 函数中的代码改为以下代码片段即可。

```
    def color(filename):
    # 读入图（目标的特征向量）
    in_img = cv2.imread(filename+'.jpg')
    image = cv2.cvtColor(in_img, cv2.COLOR_BGR2GRAY)
    # 创建0~255的灰度值像素点数
    # 为了与12配套，故此处设为264 = 12 * 22
    gray = [0]*264
    tot  = 0
    for i in range(0,len(image)):
        for j in range(0,len(image[0])):
            # 直接将image[i][j]点的灰度像素值加到我们的gray上
            gray[image[i][j]] += 1
            tot += 1
    # 建立具体比例，存入p中
    p = []
    for i in range(12):
        cnt = 0
        for j in range(22):
            cnt += gray[i*22+j]
        p.append(cnt*1.0/tot)
    # 计算出0、1、2类型的特征向量
    for i in range(len(p)):
        if p[i] < 0.08:
            p[i] = 0
        elif p[i] < 0.16:
            p[i] = 1
        else:
            p[i] = 2
    return p
```

7.3 以梯度直方图作为特征信息的 LSH 算法实现

这里只需要将 color 函数中的代码改为以下代码片段即可。

```
    def color(filename):
    # 读入图（目标的特征向量）
    in_img = cv2.imread(filename+'.jpg')
    image = cv2.cvtColor(in_img, cv2.COLOR_BGR2GRAY)
    Ix, Iy = np.zeros((len(image),len(image[0]))), np.zeros((len(image),len(image[0])))
    M, B = np.zeros((len(image),len(image[0]))), np.zeros((len(image),len(image[0])))
    # 创建0~255的灰度值像素点数
    # 为了与12配套，故此处设为264 = 12 * 22
    N = [0]*361
    tot = 0
    for i in range(1,len(image)-1):
        for j in range(1,len(image[0])-1):
            # Ix是x方向的梯度，Iy即y方向的
            Ix[i][j] = int(image[i+1][j])-int(image[i-1][j])
            Iy[i][j] = int(image[i][j+1])-int(image[i][j-1])
            # M为梯度强度
            M[i][j]=(Ix[i][j]**2+Iy[i][j]**2)**0.5
            # N记录实际的i区间内的值
            N[math.floor(M[i][j])] += 1
            tot += 1
    # 建立具体比例，存入p中
    p = []
    for i in range(12):
        cnt = 0
        for j in range(30):
            cnt += N[i*30+j]
        p.append(cnt*1.0/tot)
    # 计算出0、1、2类型的特征向量
    for i in range(len(p)):
        if p[i] < 0.08:
            p[i] = 0
        elif p[i] < 0.16:
            p[i] = 1
        else:
            p[i] = 2
    return p
```