电类工程导论(C 类)实验 13 报告

贾萧松 516030910548

一、实验概述

在 Lab12，我们学习并使用了原始 LSH 算法完成了在在图片数据库中检索与目标图片相似图片的过程。

二、实验环境

Ubuntu14.04+Pythton2.7+Openv2.4.11+Numpy1.13.3

三、实验内容

1. 得到 12 维 P 向量并量化

1.1 原理

数据(图像、视频、音频等)都表示成一个 d 维的整数向量。

$$\boldsymbol{p} = (p_1, p_2, \cdots p_d)$$

在本实验中，每幅图像用一个 12 维的颜色直方图 p 表示，构成方式如下图所示。其中 $H_i, i = 1,2,3,4$ 是 3 维颜色直方图。

1.2 实现

如下（截取了计算 H1 部分）

```
#得到12维P向量
def get_vector(img):
    B, G, R = cv2.split(img)
    vec = []
    height, width = img.shape[0:2]
    #H1
    b = B[0:int(height/2), 0:int(width/2)].sum()
    g = G[0:int(height/2), 0:int(width/2)].sum()
    r = R[0:int(height/2), 0:int(width/2)].sum()
    total = b + g + r
    vec += [float(b)/total, float(g)/total, float(r)/total]
```

原理为计算在 H1 区域的颜色直方图，然后加到结果中。

2. 特征向量的量化

2.1 原理

上述得到的特征向量 $\boldsymbol{p}=(p_1,\cdots p_{12})$
每个分量满足 $0\le p_j\le 1$ 将其量化成
3个区间分别用0 1 2表示:

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

于是最终得到的特征向量的每个元素满足:

$$p_j\in\{0,1,2\}$$

2.2 实现

```
#量化P 到 0,1,2
def quantization(p, high, low):
    q = [0]*len(p)
    for i in range(len(p)):
        if p[i] > high:
            q[i] = 2
        elif p[i] > low:
            q[i] = 1
        else:
            q[i] = 0
    return q
```

3. 将 P 向量映射到 Hamming 空间

3.1 原理

*d*维整数向量***p***可用*d'=d*C*维的Hamming码表示：

$$v(\boldsymbol{p}) = \mathrm{Unary}_C(p_1)\cdots\mathrm{Unary}_C(p_d)$$

其中 $\mathrm{Unary}_C(p_1)$ 表示C个二进制数，前p1个为1，后C-p1个为0。如当C=10：

$$\mathrm{Unary}_C(5) = 1111100000 \qquad \mathrm{Unary}_C(3) = 1110000000$$

如***p***=(0,1,2,1,0,2)，这里*d*=6,*C*=2,于是

$$v(\mathbf{p}) = 001011100011$$

选取集合 $\{1, 2, \ldots, d'\}$ 的L个子集 $\{I_i\}_{i=1}^{L}$，定义v(***p***)在集合

$$I_i = \{i_1, i_2, \cdots i_m\} : 1 \le i_1 < i_2 < \cdots < i_m \le d'$$

上的投影为 $g_i(\boldsymbol{p}) = p_{i1}p_{i2}\cdots p_{im}$，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)

- 不必显式的将*d*维空间中的点***p***映射到*d'*维Hamming空间向量v(***p***)。
- I|i表示I中范围在(i-1)*C+1~i*C中的坐标：

$$I = \{1,3,7,8\}, I|1 = \{1\}, I|2 = \{3\},$$
$$I|3 = \phi, I|4 = \{7,8\}, I|5 = I|6 = \phi$$

- 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：

$$|\{I|i\} - C*(i-1) \le x_i|$$

- 比如 $\{I|1\}$ 中小于等于 $x_1 = 0$ 的个数为0，投影：0；
- $\{I|2\} - 2$中小于等于 $x_2 = 1$的个数为1，投影：1；
- $\{I|4\} - 3*2$ 中小于等于 $x_4 = 1$ 的个数为1，投影：10；
- 串联得到：(0,1,1,0)

3.2 实现

```python
#投影到Hamming空间
def cast_to_hamming(p, cast_set):
    res = ""
    for num in cast_set:
        #求出对应p集合第xi个元素
        xi = int(round(float(num)/2.0) - 1)
        #如果为0，则必然投影为0
        if(p[xi] == 0):
            res += "0"
        #如果为2，则必然投影为1
        elif(p[xi] == 2):
            res += "1"
        #如果为1
        else:
            res += str((num-2*xi)%2)
    return res
```

这里我使用了字符串形式的 HASH，便于在 Python 中操作。

4. 建立索引

```python
#参数：目标图片， 数据文件夹， 最多输出k个数据(在同一桶中前k近的)
def LSH(file_name, file_path, k=5):
    cast_set = [1,3,7,8,20]
    #储存HASH类（桶） 键值对： 投射到汉明空间的值-[[文件名,特征向量],...]
    search_set = {}

    file_set = os.listdir(file_path)
    for file in file_set:
        pic = cv2.imread(file_path+"/"+file, cv2.IMREAD_COLOR)
        vec = get_vector(pic)
        p = get_cast(vec, cast_set)
        if(search_set.has_key(p)):
            search_set[p].append([file, vec])
        else:
            search_set[p] = [[file,vec]]
```

首先，求图片数据库中所有图片的 HASH 值。

然后，建立索引时使用了 Python 中的字典，从原理上讲相当于作了二次 Hash，不过这一次不需要 LSH（局部敏感哈希），使用常规 Hash 即可，字典的键值对为：投影到 Hamming 空间的值 —— [[文件名 1，特征向量 1], [文件名 2，特征向量 2]..]，即将所有有相同 Hamming 空间投影值的图片，放到了一个 list（桶）中。

5. 检索

```
    #获得要检索图片的HASH
    target = cv2.imread(file_name, cv2.IMREAD_COLOR)
    target_vec = get_vector(target)
    target_p = get_cast(target_vec, cast_set)

    #检索并输出结果
    #没有找到
    if(not search_set.has_key(target_p)):
        print "Sorry, there are no similar pictures by LSH"
        return
    else:
        print "Find Result(by LSH):"
        print "Filename\tCosine Similarity"
        print "cost", time.time()-start
        #储存桶内图片与目标图片的余弦相似度
        allsimilarity = []
        for ele in search_set[target_p]:
            allsimilarity.append((ele[0], get_cos(target_vec, ele[1])))
        #排序
        allsimilarity.sort(key=lambda x:x[1], reverse=True)
        #只输出前k大
        for i in range(min(len(allsimilarity), k)):
            print i+1,"\t",allsimilarity[i][0],"\t",allsimilarity[i][1]
        print
```

首先，获得目标图片的 Hamming 空间投影值 P。

然后，用 P 作为 Key 在索引（字典）中找到对应的结果（相当于找到桶）。

最后，计算桶内图片与目标图片的余弦相似度，并按降序输出前 K（参数可调）相似的。

6. 结果

```
Find Result(by LSH):
Filename    Cosine Similarity
cost 0.00223302841187
1   38.jpg  1.0
2   12.jpg  0.999383807182
3   23.jpg  0.996570944786
4   26.jpg  0.996055245399
5   40.jpg  0.995867908001
6   7.jpg   0.995663881302
7   25.jpg  0.995350003242
8   8.jpg   0.995301246643
9   15.jpg  0.994932621717
10  28.jpg  0.994907259941
```

最终，结果中有与目标图片完全相同的 38.jpg，还找到了其他 9 幅图片。
目标图片：

相似度前五的
38.jpg（余弦相似度 1.0）：

12.jpg（余弦相似度 0.999384）：

23.jpg（余弦相似度 0.99657）：

26.jpg（余弦相似度 0.99606）：

40.jpg（余弦相似度 0.99587）：

可以看出，前两个比较准确，反映在余弦相似度上是大于 0.999，第三个就偏差比较大了，第四第五个从背景上看出是非常相似的。

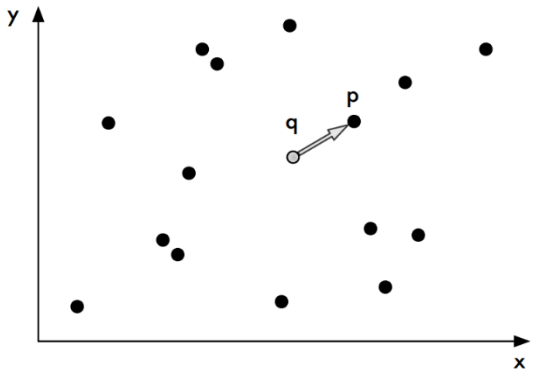
四、实验拓展

1. 与 KNN 比较

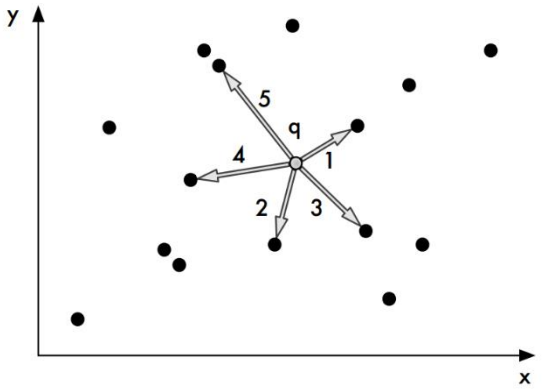
1.1 KNN 检索

原理：

在所有图片中，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。



NN 检索



KNN 检索

实现:

```
#计算两个特征向量的余弦相似度
def get_cos(x,y):
    x = np.array(x, dtype='float32')
    y = np.array(y, dtype='float32')
    res = (x.dot(y.T)) / (np.linalg.norm(x) * np.linalg.norm(y))
    #归一化
    res = 0.5 + res*0.5
    return res


#参数：目标图片，  数据文件夹，  最多输出k个数据（前k相似）
def NN(file_name, file_path, k=5):
    start = time.time()
    target = cv2.imread(file_name, cv2.IMREAD_COLOR)
    target_vec = get_vector(target)

    #储存所有图片与目标图片的余弦相似度
    allsimilarity = []
    file_set = os.listdir(file_path)
    for file in file_set:
        pic = cv2.imread(file_path+"/"+file, cv2.IMREAD_COLOR)
        vec = get_vector(pic)
        allsimilarity.append((file, get_cos(vec, target_vec)))
    #排序
    allsimilarity.sort(key=lambda x:x[1], reverse=True)
```

思路就是暴力检索，求出目标图片和数据库中每一张图片的余弦相似度，然后排序。

1.2 LSH 与 KNN 比较

```
Find Result(by LSH):
Filename    Cosine Similarity
cost 0.00337815284729 s
1   38.jpg  1.0
2   12.jpg  0.999383807182
3   23.jpg  0.996570944786
4   26.jpg  0.996055245399
5   40.jpg  0.995867908001
6   7.jpg   0.995663881302
7   25.jpg  0.995350003242
8   8.jpg   0.995301246643
9   15.jpg  0.994932621717
10  28.jpg  0.994907259941

Find Result(by NN):
Filename    Cosine Similarity
cost 0.168160915375 s
1   38.jpg  1.0
2   12.jpg  0.999383807182
3   23.jpg  0.996570944786
4   26.jpg  0.996055245399
5   40.jpg  0.995867908001
6   7.jpg   0.995663881302
7   25.jpg  0.995350003242
8   8.jpg   0.995301246643
9   15.jpg  0.994932621717
10  28.jpg  0.994907259941
```

这是 LSH 投影集[1,3,7,8]，LSH 和 NN 均输出前十相似的结果。
发现二者结果完全相同，但 LSH 用时：0.00337815284729s，
NN 用时：0.168160915375s，效率差距很大。

2. LSH 参数修改

2.1 量化高低门限修改

在 PPT 中提到：量化时要使 0，1，2 分布均匀，从而起到使 0，1，2 中的信息更多，减少信息损失的效果。

而原本的 0.3，0.6 的门限，使得量化后几乎没有 2，这一点，使得 2 失去作用，信息损失过多，于是我尝试来修改高低门限值。

首先，我统计数据库中，总共有 40 张图，共有 40*12=480 个数据。

在排序后，我找出了第 160 和 320 大的数据：0.319847、0.345166
设为高低门限。
再进行检索：

```
Find Result(by LSH):
Filename    Cosine Similarity
cost 0.00172996520996 s
1   38.jpg  1.0
2   26.jpg  0.996055245399
3   17.jpg  0.994861751795
4   37.jpg  0.954990237951
5   11.jpg  0.93280184269
```

可以看出，更均匀分布的 0，1，2 使得桶中数据量更少，在数据量较小时优势可能不太明显，当数据量比较大时，如果分配后桶内仍有很多数据，那么其实对效率影响还是比较大的。

不过，此时，在同一桶内内的数据只有第一个是比较精确的，这反映了在 LSH 过程中，当桶里的数据少了的时候，可能需要访问相邻的桶来获取更多更好的结果，这可能会影响正确率和效率。

2.2 投影集选择对结果的影响

当 I=[1,3,7,8]时

```
Find Result(by LSH):
Filename    Cosine Similarity
cost 0.00172996520996 s
1   38.jpg  1.0
2   26.jpg  0.996055245399
3   17.jpg  0.994861751795
4   37.jpg  0.954990237951
5   11.jpg  0.93280184269
```

当 I=[2,7,10,15]时

```
Find Result(by LSH):
Filename    Cosine Similarity
cost 0.00180697441101 s
1   38.jpg  1.0
2   12.jpg  0.999383807182
3   23.jpg  0.996570944786
4   26.jpg  0.996055245399
5   40.jpg  0.995867908001
6   17.jpg  0.994861751795
```

可以看出得到结果相对好一点，也就是说，选取合适的投影集，可以有效提高结果质量。

当 I=[1,3,7,8,10,20]时

```
Find Result(by LSH):
Filename    Cosine Similarity
cost 0.0128688812256 s
1   38.jpg  1.0
2   26.jpg  0.996055245399
```

可见相比[1,3,7,8]，桶内的数据更少了，说明提高投影集的维度可以减少桶内数据，提高效率；然而，相比[2,7,10,15]，它的结果质量并不是很高，说明提高投影集的维度，并不一定会提高召回率。

五、总结

在本实验中，我简单实现了一遍原始 LSH 算法，并用其进行了检索。除此之外，在网上的学习过程中，还了解到有优化原始 LSH 的 minHash 和 simHash，至于具体实现就留到今后的学习了。

最后，衷心感谢老师和助教们的精心准备和辛勤付出，你们整理完善的 ppt 和到位的答疑大大提高了我学习 LSH 的效率，感谢~

贾萧松 516030910548
2017.12.23

六、源代码

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-
import sys
reload(sys)
sys.setdefaultencoding('utf-8')

import cv2
import numpy as np
from math import *
import os
import time



#传入图片和投影集合,得到一张图片的投影到Hamming空间的向量
def get_cast(p, cast_set):
    #量化P 到 0,1,2
    p = quantization(p, 0.345166, 0.319847)
    #投影到Hamming空间
    cast_vec = cast_to_hamming(p, cast_set)
    return cast_vec
```

```python
#得到12维P向量
def get_vector(img):
    B, G, R = cv2.split(img)
    vec = []
    height, width = img.shape[0:2]
    #H1
    b = B[0:int(height/2), 0:int(width/2)].sum()
    g = G[0:int(height/2), 0:int(width/2)].sum()
    r = R[0:int(height/2), 0:int(width/2)].sum()
    total = b + g + r
    vec += [float(b)/total, float(g)/total, float(r)/total]
    #H2
    b = B[0:int(height/2), int(width/2)+1:width].sum()
    g = G[0:int(height/2), int(width/2)+1:width].sum()
    r = R[0:int(height/2), int(width/2)+1:width].sum()
    total = b + g + r
    vec += [float(b)/total, float(g)/total, float(r)/total]
    #H3
    b = B[int(height/2)+1:height, 0:int(width/2)].sum()
    g = G[int(height/2)+1:height, 0:int(width/2)].sum()
    r = R[int(height/2)+1:height, 0:int(width/2)].sum()
    total = b + g + r
    vec += [float(b)/total, float(g)/total, float(r)/total]
    #H4
    b = B[int(height/2)+1:height, int(width/2)+1:width].sum()
    g = G[int(height/2)+1:height, int(width/2)+1:width].sum()
    r = R[int(height/2)+1:height, int(width/2)+1:width].sum()
    total = b + g + r
    vec += [float(b)/total, float(g)/total, float(r)/total]
    return vec


#量化P 到 0,1,2
def quantization(p, high, low):
    q = [0]*len(p)
    for i in range(len(p)):
        if p[i] > high:
            q[i] = 2
        elif p[i] > low:
            q[i] = 1
        else:
            q[i] = 0
    return q
```

```
#投影到Hamming空间
def cast_to_hamming(p, cast_set):
    res = ""
    for num in cast_set:
        #求出对应p集合第xi个元素
        xi = int(round(float(num)/2.0) - 1)
        #如果为0，则必然投影为0
        if(p[xi] == 0):
            res += "0"
        #如果为2，则必然投影为1
        elif(p[xi] == 2):
            res += "1"
        #如果为1
        else:
            res += str((num-2*xi)%2)
    return res
```

```
#参数：目标图片，  数据文件夹，  最多输出k个数据(在同一桶中前k近的)
def LSH(file_name, file_path, k=5):
    cast_set = [2,7,10,15]
    #储存HASH类（桶） 键值对：投射到汉明空间的值-[[文件名,特征向量],...]
    search_set = {}

    file_set = os.listdir(file_path)
    for file in file_set:
        pic = cv2.imread(file_path+"/"+file, cv2.IMREAD_COLOR)
        vec = get_vector(pic)
        p = get_cast(vec, cast_set)
        if(search_set.has_key(p)):
            search_set[p].append([file, vec])
        else:
            search_set[p] = [[file,vec]]

    start = time.time()
    #获得要检索图片的HASH
    target = cv2.imread(file_name, cv2.IMREAD_COLOR)
    target_vec = get_vector(target)
    target_p = get_cast(target_vec, cast_set)
```

```
    #检索并输出结果
    #没有找到
    if(not search_set.has_key(target_p)):
        print "Sorry, there are no similar pictures by LSH"
        return
    else:
        print "Find Result(by LSH):"
        print "Filename\tCosine Similarity"
        print "cost", time.time()-start,"s"
        #储存桶内图片与目标图片的余弦相似度
        allsimilarity = []
        for ele in search_set[target_p]:
            allsimilarity.append((ele[0], get_cos(target_vec, ele[1])))
        #排序
        allsimilarity.sort(key=lambda x:x[1], reverse=True)
        #只输出前k大
        for i in range(min(len(allsimilarity), k)):
            print i+1,"\t",allsimilarity[i][0],"\t",allsimilarity[i][1]
        print
```

```
#计算两个特征向量的余弦相似度
def get_cos(x,y):
    x = np.array(x, dtype='float32')
    y = np.array(y, dtype='float32')
    res = (x.dot(y.T)) / (np.linalg.norm(x) * np.linalg.norm(y))
    #归一化
    res = 0.5 + res*0.5
    return res
```

```
#参数：目标图片，  数据文件夹，  最多输出k个数据（前k相似）
def NN(file_name, file_path, k=5):
    start = time.time()
    target = cv2.imread(file_name, cv2.IMREAD_COLOR)
    target_vec = get_vector(target)

    #储存所有图片与目标图片的余弦相似度
    allsimilarity = []
    file_set = os.listdir(file_path)
    for file in file_set:
        pic = cv2.imread(file_path+"/"+file, cv2.IMREAD_COLOR)
        vec = get_vector(pic)
        allsimilarity.append((file, get_cos(vec, target_vec)))
    #排序
    allsimilarity.sort(key=lambda x:x[1], reverse=True)
    #只输出前k大
    print "Find Result(by NN):"
    print "Filename\tCosine Similarity"
    print "cost", time.time()-start,"s"
    for i in range(min(len(allsimilarity), k)):
        print i+1,"\t",allsimilarity[i][0],"\t",allsimilarity[i][1]
```

```
def main():
    LSH("target.jpg", "dataset", 10)
    NN("target.jpg", "dataset", 10)


main()
```