**《多媒体数据处理》上机实验报告**

学号：10930130448 姓名：吕思勤

# 第一次实验：算术编码

## 1. 算法描述

算术编码的基本原理是将编码的消息表示成实数0和1之间的一个间隔（Interval），消息越长，编码表示它的间隔就越小，表示这一间隔所需的二进制位就越多。

算术编码用到两个基本的参数：符号的概率和它的编码间隔。信源符号的概率决定压缩编码的效率，也决定编码过程中信源符号的间隔，而这些间隔包含在0到1之间。编码过程中的间隔决定了符号压缩后的输出。

给定事件序列的算术编码步骤如下：

（1）编码器在开始时将“当前间隔” [ L， H) 设置为[0，1)。

（2）对每一事件，编码器按步骤（a）和（b）进行处理

（a）编码器将“当前间隔”分为子间隔，每一个事件一个。

（b）一个子间隔的大小与下一个将出现的事件的概率成比例，编码器选择子间隔对应于下一个确切发生的事件相对应，并使它成为新的“当前间隔”。

（3）最后输出的“当前间隔”的下边界就是该给定事件序列的算术编码。

主要迭代公式:

l o w = l o w + ( h i g h − l o w ) ∗ L

h i g h = l o w + ( h i g h − l o w ) ∗ H

解码就是编码过程的逆推。从编码得到的小数开始，不断地寻找小数落在了哪个概率区间，就能将原来的字符一个个地找出来。

## 核心代码

**def** encoder(singal, singal\_dict):#编码  
 Low = 0  
 High = 1  
 **for** s **in** singal:  
 CodeRange = High - Low  
 High = Low + CodeRange \* singal\_dict[s][1]  
 Low = Low + CodeRange \* singal\_dict[s][0]  
 **return** Low

**def** decoder(encoded\_number, singal\_dict, singal\_length):#解码  
 singal = []  
 **while** singal\_length:  
 **for** k, v **in** singal\_dict.items():  
 **if** v[0] <= encoded\_number < v[1]:  
 singal.append(k)  
 range = v[1] - v[0]  
 encoded\_number -= v[0]  
 encoded\_number /= range  
 **break** singal\_length -= 1  
 **return** singal

## 实验结果

一張含有 文字 的圖片

自動產生的描述

编码:

首先读入信号：C——因为C在最初始的间隔中是[0.5, 0.7)，所以读入C之后我们的编码间隔就变成[0.5, 0.7)了；

    紧接着，我们读入的是A，A在初始区间内是占整个区间的前10%，因此对应这个上来也是需要占这个编码间隔的前10%，因此编码区间变为：[0.5, 0.52)了；

    再然后是D，因为D占整个区间的70% ~ 100%，所以也是占用这个编码区间的70% ~ 100%，操作后的编码区间为[0.514, 0.52)

    ……

    直到最后将信号量全部读出。

一張含有 桌 的圖片

自動產生的描述

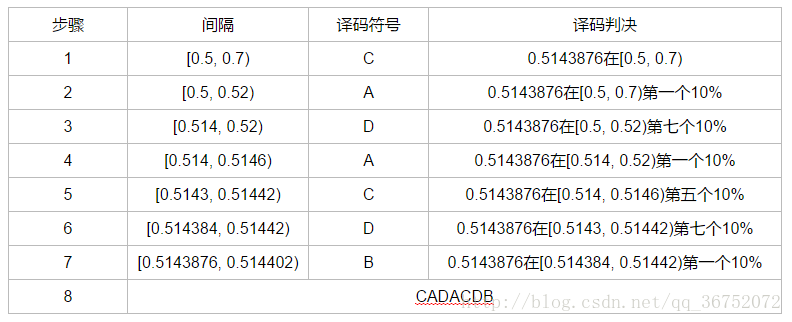
解码:

待解码的数据0.5143876在[0.5, 0.7)内，因此，我们解出的第一个码值为C

    同理，我们继续计算0.5143876处于[0.5, 0.7)的第一个10%内因此解出的第二个码值为A

    ……

    这个过程持续直到执行七次全部执行完为止。



## 分析与总结

各种媒体信息（特别是图像和动态视频）数据量非常之大，算术编码作为一种高效的数据编码方法在文本，图像，音频等压缩中有广泛的应用。所以，研究算术编码有非常好的前景与实用价值。

通过本次实验，我更清楚地了解了算术编码的原理，算术编码的压缩本质，就是在保留字符排列顺序的同时，对于更高频出现的字符，也就是概率更大的字符，赋予更大的小数区间。算术编码的目的，是要在最终的目标区间内，找一个二进制最短的小数作为最终编码。所以算术编码的实现途径就是：尽量使最终目标区间的范围更大。由于高频字符出现次数多，区间较大，而低频字符出现次数少，区间小。所以在遍历完所有字符之后，我们最终得到目标区间就更大，也就是小数精度更低。

**《多媒体数据处理》上机实验报告**

学号：10930130448 姓名：吕思勤

# 第二次实验：实现PCA主成分分析

## 1. 算法描述

PCA（Principal Component Analysis）是一种常用的数据分析方法。PCA通过线性变换将原始数据变换为一组各维度线性无关的表示，可用于提取数据的主要特征分量，常用于高维数据的降维。

PCA的算法步骤：

设有m条n维数据。

1）将原始数据按列组成n行m列矩阵X

2）将X的每一行（代表一个属性字段）进行零均值化，即减去这一行的均值

3）求出协方差矩阵 一張含有 文字 的圖片

自動產生的描述

4）求出协方差矩阵的特征值及对应的特征向量

5）将特征向量按对应特征值大小从上到下按行排列成矩阵，取前k行组成矩阵P

1. Y=PX即为降维到k维后的数据

## 2.核心代码

def pca(data,k):

X\_mean = np.mean(data, axis=0)

sdata=(data-X\_mean)#去均值

ew,ev=np.linalg.eig(sdata.T.dot(sdata))

ew\_order=np.argsort(ew)[::-1]# 特征向量按对应特征值从大到小排列

ew\_sort=ew[ew\_order]

ev\_sort=ev[:,ew\_order]

feature=ev\_sort[:,:k]#取前k行组成矩阵feature

new\_data=sdata.dot(feature)#和原矩阵相乘

return new\_data

## 3.实验结果

一張含有 桌 的圖片

自動產生的描述 一張含有 文字, 收據 的圖片

自動產生的描述

一張含有 文字, 收據 的圖片

自動產生的描述

一張含有 文字, 收據 的圖片

自動產生的描述

一張含有 文字, 收據 的圖片

自動產生的描述

一張含有 文字, 收據 的圖片

自動產生的描述

## 4.分析与总结

总结本次实验的主要步骤如下:

1.矩阵X的维度是(m,n)。表示m组数据，n维向量。

2.先对x去均值。（是否标准化看情况）

3.求协方差的特征值(ew)和特征向量（ev)。调用np.linalg.eig()函数即可。

4.将特征向量按对应特征值大小从上到下按行排列成矩阵，取前k行组成矩阵feature。

5.feature最后和原矩阵相乘即为降维后的矩阵。

实验时调用sklearn的pca和自己写的做比较

结果可能有负号差异，因为sklearn的奇异值分解函数有所不同。

能分析出降维后协方差是对角矩阵，除了对角线值其余接近0。

**《多媒体数据处理》上机实验报告**

学号：10930130448 姓名：吕思勤

# 第三次实验：图像特征提取与检索

## 1. 算法描述

Bag of Features (BOF)

feature为特征，即根据特征进行分类。

Bag of Feature 是一种图像特征提取方法，参考了Bag of Words的思路，把每幅图像描述为一个局部区域/关键点(Patches/Key Points)特征的无序集合。同时从图像抽象出很多具有代表性的「关键词」，形成一个字典，再统计每张图片中出现的「关键词」数量，得到图片的特征向量。

算法流程:

1.提取图像特征（常用SIFT）

2.训练字典，通过聚类算法对这些特征向量进行聚类（常用k-means），得到一部字典

3.量化图像特征，根据字典将图片表示成向量（直方图）；

4.训练分类器，根据数据库图片的向量以及图片的标签，训练分类器模型

5.再提取图像特征，根据字典量化直方图向量，用分类器模型对直方图向量进行分类

**实验环境**： python3**.7**  
配置好sift、PCV和vlfeat

## 2.核心代码

1. **对数据集做SIFT特征提取，采用K-means算法学习视觉词典**

用SIFT算法提取特征描述子，然后保存词汇。Vocabulary类包含了一个由单词聚类中心VOC与每个单词对应的逆向文档频率构成的向量，为了在某些图像集上训练词汇，train（）方法获取包含有.sift后缀的描述子文件列表和词汇单词数k。

*# list of downloaded filenames*imlist = get\_imlist(**"D:/corel/pic/0/"**)  
**for** i **in** range(1,10):  
 tmpimlist = get\_imlist(**"D:/corel/pic/{}/"**.format(i))  
 imlist.extend(tmpimlist)  
nbr\_images = len(imlist)  
*#获取特征列表*featlist = [imlist[i][:-3]+**'sift' for** i **in** range(nbr\_images)]  
*#提取文件夹下图像的sift特征***for** i **in** range(nbr\_images):  
 sift.process\_image(imlist[i], featlist[i])  
*#生成词汇*voc = vocabulary.Vocabulary(**'ukbenchtest'**)  
voc.train(featlist, 1000, 10)  
*#保存词汇  
# saving vocabulary***with** open(**"D:/corel/pic/vocabulary500-10.pkl"**, **'wb'**) **as** f:  
 pickle.dump(voc, f)  
print (**'vocabulary is:'**, voc.name, voc.nbr\_words)

2.创建索引，建立数据库testImage2.db。

*#创建索引*indx = imagesearch.Indexer(**'testImage2.db'**,voc)  
indx.create\_tables()  
*#遍历所有的图像，并将它们的特征投影到词汇上***for** i **in** range(nbr\_images)[:1000]:*#1000是因为有1000张图片* locs,descr = sift.read\_features\_from\_file(featlist[i])  
 indx.add\_to\_index(imlist[i],descr)  
*# commit to database  
#提交到数据库*indx.db\_commit()

3.在数据库中搜索图像

建立好图像索引，就可以在数据库中搜索相似图片了。

src = imagesearch.Searcher(**'testImage2.db'**,voc)  
*# index of query image and number of results to return  
#查询图像索引和查询返回的图像数*q\_ind = 777  
nbr\_results = 10  
*# regular query  
# 常规查询(按欧式距离对结果排序)*res\_reg = [w[1] **for** w **in** src.query(imlist[q\_ind])[:nbr\_results]]  
print (**'top matches (regular):'**, res\_reg)  
*# load image features for query image  
#载入查询图像特征*q\_locs,q\_descr = sift.read\_features\_from\_file(featlist[q\_ind])  
*# 显示查询结果  
#常规*imagesearch.plot\_results(src,res\_reg[:nbr\_results])

## 3.实验结果

q\_ind = 340

一張含有 文字 的圖片

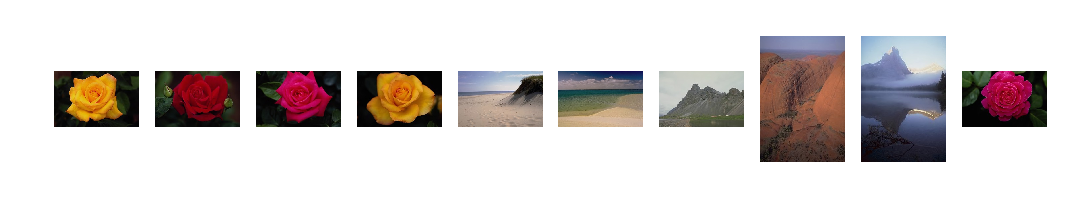
自動產生的描述

q\_ind = 518

一張含有 文字, 差異 的圖片

自動產生的描述

q\_ind = 600



q\_ind = 777

一張含有 文字 的圖片

自動產生的描述

对比结果可以看出此算法在一些检测程度上效果还不足，这说明可能在创建词汇上，特征描述相似，还不够准确独特，这就需要深入研究了。在图像特征比较明显，或者相同图像的数据集较多的情况之下，图像的匹配效果就会比较好。

## 4.分析与总结

实验中出现的错误及解决办法：

1. No module named 'pysqlite2’

问题来源：

from pysqlite2 import dbapi2 as sqlite

解决办法：

将pysqlite2 改成sqlite3

from sqlite3 import dbapi2 as sqlite

2. ‘cmp’ is an invalid keyword argument for sort()

参照解决：Python3-找回sort-中消失的cmp参数

将cmp改为key=cmp\_to\_key()

另外python3已经不支持cmp()函数。参照解决办法：link

cmp()函数是Python 2中的一个用于比较两个列表, 数字或字符串等的大小关系的函数

在本实验中将报错的代码改为：

tmp.sort(key=cmp\_to\_key(lambda x, y: operator.gt(x[1], y[1])))

3.TypeError: a bytes-like object is required, not 'str’

问题来源：

python3和Python2在套接字返回值解码上有区别

报的错误其实就是函数需要bytes类型数据，但输入的是str型

解决方式: 将imagesearch的返回值改为bytes类型数据

return pickle.loads(bytes(s[0]))

总结:

建立图像数据库，对图像进行索引就是从这些图像中提取描述子，利用词汇将描述子转换成视觉单词，并保存视觉单词及对应图像的单词直方图，从而可以利用图像对数据库进行查询，并返回相似的图像作为搜索结果。如果图像数据库很大，逐一比较整个数据库中的所有直方图往往是不可行的，我们需要找到一个大小合理的候选集，单词索引的作用便在于此：我们可以利用单词索引获得候选集，然后只需在候选集上进行逐一比较。利用一些考虑到特征几何关系的准则重排搜索到的靠前结果，可以提高准确率。最常用的方法是在查询图像与靠前图像的特征位置间拟合单应性。

为了提高效率，可以将特征位置存储在数据库中，并有特征的单词id决定它们之间的关联，只有在词汇足够大，使单词id包含很多准确匹配时，它才起作用。实验出现出现偏差的根本原因是数据库图像数目不够多，因而类间差距不够大，类内联系不够紧密，如果扩大图像库规模，会效果更好。

**《多媒体数据处理》上机实验报告**

学号：10930130448 姓名：吕思勤

# 第四次实验：LSH 索引实现最近邻搜索

## 1. 算法描述

查询corel数据集前1000点，查找前10个最近邻，使用原始汉明编码，计算汉明距离，使用 LSH 索引，实现近似最近邻搜索。输入：corel数据集，汉明码长度K，建表数量L，哈希桶最大容量B；

算法流程：

1对数据集预处理：乘100取整，获得最大值C与数据维数n；

2在n\*C的范围内生成k个随机数，存入表H，为要取到的汉明码位数。新建哈希索引表；

3将数据点的坐标转化为0与1的汉明编码，取2中生成的随机数位，组成字符串pi；

4若哈希索引表pi桶的数量大于B则不记录，否则在pi桶记录该数据点；

5重复操作2，3，4，生成L张H表与哈希索引表；

6对前1000行的点进行最邻近搜索，即对数据点求它的汉明编码，对每张H表和每张哈希索引表，根据H表的位数生成字符串，将哈希索引表的该字符串编码处的点作为候选项，将所有的候选项排序选择频次最高的前十个作为结果；

7计算召回率 Recall、准确率 Accuracy、时间 time cost

## 2.核心代码

# 生成哈希索引表  
def set\_table(H,C,B,data):  
 # H：生成的k个随机数，C：取到的最大值，B：桶的最大数量  
 # data：处理过的数据集  
 table = {}  
 f = {}  
 for i, line in enumerate(data):  
 pi = []  
 # var是汉明编码  
 var = ''.join(Embedding(C,line))  
 # 取汉明编码的H对应位  
 for j in H:  
 pi.append(var[j])  
 # f字典用于记录每个哈希桶包含的行数，小于B则+1  
 if (''.join(pi)) in f:  
 if f[''.join(pi)]<B:  
 f[''.join(pi)] += 1  
 table[i] = ''.join(pi)  
 else:  
 f[''.join(pi)]=1  
 table[i] = ''.join(pi)  
 return table

# 求line的LSH最邻近点  
def search(Hlist,tblist,var):  
 # var：一行的汉明编码  
 # 返回该行的所有匹配点（有重复项）  
 keys = []  
 # 用Hlist的每个H求对应位组合  
 for i,h in enumerate(Hlist):  
 p = []  
 for hi in h:  
 p.append(var[hi])  
 # 寻找所以哈希相同的点  
 for key, value in tblist[i].items():  
 if value== ''.join(p):  
 keys.append(key)  
 return keys

## 3.实验结果

一張含有 文字 的圖片

自動產生的描述

## 4.分析与总结

通过这次实验我了解并实现了LSH这种针对海量高维数据的快速最近邻查找算法。将高维数据降维到一维，然后归置桶中。多次用不同方式，结果良好，速度快。

问题一：高维数据降维到一维？ 点积方法。问题二：wx+b 降维到一维了，通过查阅资料是分桶既是//n，就是除以一个数字并且向下取整。我当时想着说可以确定一个半径（就是认为9.9相比较10.4 与10更近邻），找数据周围的小标，然后通过距离查找。最后结果也不错。问题三：考虑到降维到一维的准确性与[w1,w2,...wd]关系很大，所以在这里想到要使用多个随机w向量进行投影。只要有符合结果的就加入检索数组中。（这里为了方便将半径查找改成//n向下取整）最后去掉重复下标进行距离查找。