实验 4 图像聚类实验

编写代码完成以下实验内容，请将代码和运行效果截图插入到问题下方。

本次实验主要练习几种聚类方法, 并利用它们对图像进行聚类, 从而寻找相似的图像组。聚类可以用于识别、划分图像数据集，组织与导航。

第一部分 K-means 聚类

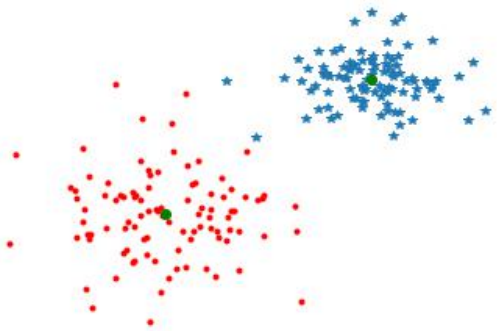
K-means 是一种将输入数据划分成 k 个簇的简单的聚类算法。K-means 算法的最大缺陷是必须预先设定聚类数 k，如果选择不恰当则会导致聚类出来的结果很差。算法的优点是容易实现，可以并行计算，并且对于很多别的问题不需要任何调整就能直接使用。

SciPy 聚类包

1、 K-means 算法比较简单，可以自己编写，也可以直接调用 SciPy 矢量量化包 scipy.cluster.vq 中 K-means 的实现。下面给出代码范例，请自行输入代码并观察运行结果，解释说明代码的执行过程，并学会调用 K-means 方法。

```
In [4]: from scipy.cluster.vq import *

In [5]: class1 = 1.5*randn(100,2)
        class2 = randn(100,2)+array([5,5])
        features = vstack((class1,class2))

In [6]: features
array([[ 9.95286252e-01,  1.10134535e-01],
       [-6.82388301e-01,  1.92806250e-01],
       [ 4.43783281e-01,  4.66199768e+00],
       [ 2.38080792e+00, -5.32196319e-01],
       [-1.31108248e+00, -2.07910117e+00],
       [-3.16740870e-01,  4.10585719e-01],
       [-3.58027561e-01, -1.43943116e+00],
       [-1.01971327e+00, -5.33419401e-02],
       [-8.47977278e-01, -2.07726071e-01],
       [-2.29629280e+00,  8.18786048e-01],
       [-1.03670052e+00, -1.08355780e+00],
       [ 2.40483420e-01, -1.06180376e+00],
       [ 1.05577704e+00,  9.68713744e-01],
       [ 6.35300545e-01, -3.75340395e-01],
       [-2.15346429e+00,  2.63813129e+00],
       [-4.52495775e-01, -3.68348448e+00],
       [ 1.20835887e-01,  2.54726088e+00],
       [ 1.17515607e+00, -2.06043927e+00],
       [-4.52104090e-01,  1.65065585e+00],
       [ 1.04108041e+00, -2.70522759e-01],

In [7]: centroids,variance = kmeans(features,2)

In [8]: centroids
Out[8]: array([[ 5.03549508,  5.14001595],
               [-0.09760249,  0.22798767]])

In [9]: variance
Out[9]: 1.5339524580840274

In [10]: code,distance = vq(features,centroids)
```

```
In [11]: figure()
         ndx = where(code==0)[0]
         plot(features[ndx,0],features[ndx,1],'*')
         ndx = where(code==1)[0]
         plot(features[ndx,0],features[ndx,1],'r.')
         plot(centroids[:,0],centroids[:,1],'go')
         axis('off')
         show()
```



解释说明：1.生成两类二维正态分布数据；2.用 k=2 对这些数据进行聚类；3.利用 SciPy 中自带的 kmeans 函数得到方差最小的聚类中心(centroids)及其方差(variance), 由于 SciPy 中实现的 K-means 会计算若干次（默认为 20 次），并为我们选择方差最小的结果，所以这里返回的方差并不是我们真正需要的，因此用 SciPy 包中的矢量量化函数对每个数据点进行归类，得到其每个观察值的代码簿索引(code)及其失真情况(distance)；4.通过上面得到的 code，我们可以检查是否有归类错误。为了将其可视化，画出这些数据点及最终的聚类中心。

图像的主成分分析 PCA

PCA 是一个非常有用的降维技巧，它可以在使用尽可能少维数的前提下，尽量多地保持训练数据的信息。由于图像具有很高的维数，在许多计算机视觉应用中，经常使用降维操作。PCA 产生的投影矩阵可以被视为将原始坐标变换到现有的坐标系，坐标系中的各个坐标按照重要性递减排列。

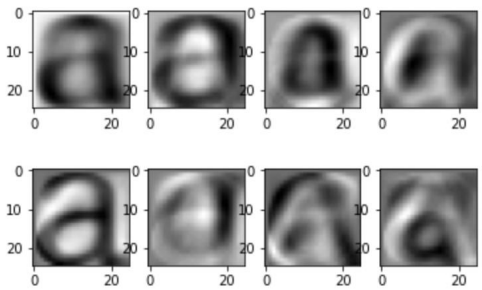
2、 如下代码所示为 pca 函数的实现，请解释以下代码的执行过程。

```
from PIL import Image
from numpy import *


def pca(X):
    """    Principal Component Analysis
        input: X, matrix with training data stored as flattened arrays in rows
        return: projection matrix (with important dimensions first), variance and mean.
    """

    # get dimensions
    num_data,dim = X.shape

    # center data
    mean_X = X.mean(axis=0)
    X = X - mean_X

    if dim>num_data:
        # PCA - compact trick used
        M = dot(X,X.T) # covariance matrix
        e,EV = linalg.eigh(M) # eigenvalues and eigenvectors
        tmp = dot(X.T,EV).T # this is the compact trick
        V = tmp[::-1] # reverse since last eigenvectors are the ones we want
        S = sqrt(e)[::-1] # reverse since eigenvalues are in increasing order
        for i in range(V.shape[1]):
            V[:,i] /= S
    else:
        # PCA - SVD used
        U,S,V = linalg.svd(X)
        V = V[:num_data] # only makes sense to return the first num_data

    # return the projection matrix, the variance and the mean
    return V,S,mean_X
```

解释说明：1.该函数首先获取训练矩阵的维度；2.用 numpy.mean 函数获得每一维的算术平均数，通过减去每一维的均值将数据中心化；3.计算协方差矩阵对应最大特征值的特征向量，返回投影矩阵、方差和均值。

3、 fontimages.zip 文件包含采用不同字体的字符 a 的缩略图。编写代码实现：利用 pca 函数来对字体图像进行 PCA 变换，并绘制均值图像和前 7 个具有最大方差的方向模式的图像。执行完 PCA 操作后的图像需要从一维表示重新转换为二维图像。给出代码和运行效果。

In [17]:
```
def get_imlist(path):
    """ 返回目录中所有 JPG 图像的文件名列表 """
    return [os.path.join(path, tmp) for tmp in os.listdir(path) if tmp.endswith('.jpg')]

imlist = get_imlist('../pics4/fontimages')
im = array(Image.open(imlist[0]))
m,n = im.shape[0:2]
imnbr = len(imlist)
immatrix = array([array(Image.open(im)).flatten()
    for im in imlist],'f')
V,S,immean = pca(immatrix)
figure()
gray()
subplot(2,4,1)
imshow(immean.reshape(m,n))
for i in range(7):
 subplot(2,4,i+2)
 imshow(V[i].reshape(m,n))
show()
```



pickle 模块

Python 中的 pickle 模块可以用来保存计算结果和数据。pickle 模块可以寄售大部分的 Pyhon 对象，并且将其转换成字符串表示，该过程叫封装。从字符串表示中重构该对象，称为拆封。

4、保存上一题中字体图像的平均图像和主成分，可以通过下面的语句完成。

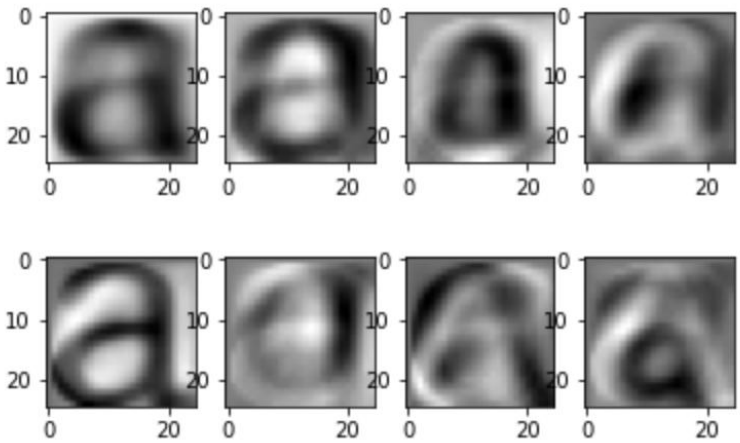
```
f = open('font_pca_modes.pkl','wb')
```

```
pickle.dump(immean,f)
pickle.dump(V,f)
f.close()
```

5、再打开保存的文件，并重新加载到平均图像和主成分变量中。绘制平均图像和前 7 个主成分图像，并和第 3 题中的结果进行比较。编码实现上述要求。

In [19]:
```
import pickle
f = open('../pics4/font_pca_modes.pkl', 'wb')
pickle.dump(immean,f)
pickle.dump(V,f)
f.close()
```

In [22]:
```
f = open('../pics4/font_pca_modes.pkl', 'rb')
immean = pickle.load(f)
V = pickle.load(f)
f.close()
figure()
gray()
subplot(2,4,1)
imshow(immean.reshape(m,n))
for i in range(7):
    subplot(2,4,i+2)
    imshow(V[i].reshape(m,n))
show()
```



图像聚类

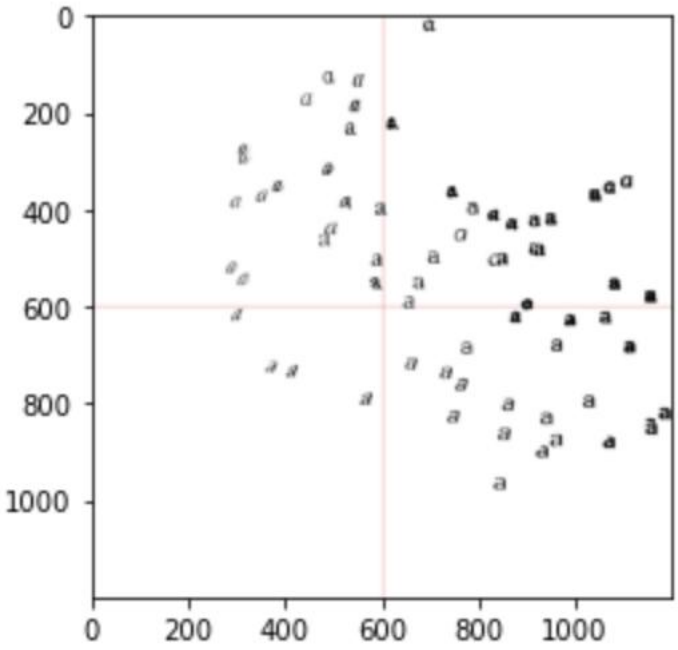
6、对于字体图像，采用 K-means 算法对其进行图像聚类。文件 selectedfontimages.zip 中包含了 66 幅来自 fontimages 中的图像。利用前面 PCA 算法中计算过的前 40 个主成分进行投影，用投影系数作为每幅图像的向量描述符。用 pickle 模块载入模型文件，在主成分上对图像进行投影，然后再使用 K-means 进行图像聚类。编写代码实现上述要求，并对聚类的结果进行可视化显示。

In [34]:
```
import pickle
from scipy.cluster.vq import *
# 获取 selected-fontimages 文件下图像文件名，并保存在列表中
imlist = get_imlist('../pics4/selectedfontimages')
imnbr = len(imlist)
# 载入模型文件
with open('../pics4/font_pca_modes.pkl','rb') as f:
    immean = pickle.load(f)
    V = pickle.load(f)
# 创建矩阵，存储所有拉成一组形式后的图像
immatrix = array([array(Image.open(im)).flatten()
 for im in imlist],'f')
# 投影到前 40 个主成分上
immean = immean.flatten()
projected = array([dot(V[:40],immatrix[i]-immean) for i in range(imnbr)])
# 进行 k-means 聚类
projected = whiten(projected)
centroids,distortion = kmeans(projected,4)
code,distance = vq(projected,centroids)

for k in range(4):
    ind = where(code==k)[0]
    figure()
    gray()
    for i in range(minimum(len(ind),40)):
        subplot(4,10,i+1)
        imshow(immatrix[ind[i]].reshape((25,25)))
        axis('off')
show()
```

在主成分上可视化图像

7、 在一对主成分方向的坐标上可视化上述图像，观察主成分算法是如何进行聚类的。编码实现: 将图像投影到两个主成分上, 并用 PIL 中的 ImageDraw 模块进行可视化。给出代码和可视化图像。

In [35]:
```
projected = array([dot(V[[0,2]],immatrix[i]-immean) for i in range(imnbr)])

from PIL import Image, ImageDraw
# 高和宽
h,w = 1200,1200
# 创建一幅白色背景图
img = Image.new('RGB',(w,h),(255,255,255))
draw = ImageDraw.Draw(img)
# 绘制坐标轴
draw.line((0,h/2,w,h/2),fill=(255,0,0))
draw.line((w/2,0,w/2,h),fill=(255,0,0))
# 缩放以适应坐标系
scale = abs(projected).max(0)
scaled = floor(array([ (p / scale) * (w/2-20,h/2-20) + (w/2,h/2) for p in projected]))
# 粘贴每幅图像的缩略图到白色背景图片

for i in range(imnbr):
    nodeim = Image.open(imlist[i])
    nodeim.thumbnail((25,25))
    ns = nodeim.size
    img.paste(nodeim,(int(scaled[i][0]-ns[0]//2),int(scaled[i][1]-
        ns[1]//2),int(scaled[i][0]+ns[0]//2+1),int(scaled[i][1]+ns[1]//2+1)))
img.save('pca_font.jpg')
figure()
imshow(img)
show()
```



像素聚类

前面已经进行了多幅图像的聚类实验，下面进行单幅图像的聚类。这里，我们在RGB 三通道的像素值上运行 K-means 进行聚类

8、 载入图像“empire.jpg”，用一个步长为 steps 的方形网格在图像中滑动，每滑一次对网格中图像区域像素求平均值，将其作为新生成的低分辨率图像对应位置处的像素值，并用 K-means 进行聚类。参考代码如下，解释说明程序的执行过程，并复现代码和绘制结果图像。

```
from scipy.cluster.vq import *
from skimage.transform import resize
```

```
steps = 50
im =array(Image.open('empire.jpg'))
dx = im.shape[0]//steps
dy = im.shape[1]//steps
```

```
features = []
for x in range(steps):
    for y in range(steps):
        R = mean(im[x*dx:(x+1)*dx,y*dy:(y+1)*dy,0])
        G = mean(im[x*dx:(x+1)*dx,y*dy:(y+1)*dy,1])
        B = mean(im[x*dx:(x+1)*dx,y*dy:(y+1)*dy,2])
        features.append([R,G,B])
features = array(features,'f')
```

```
centroids,variance = kmeans(features,3)
code,distance = vq(features,centroids)
```

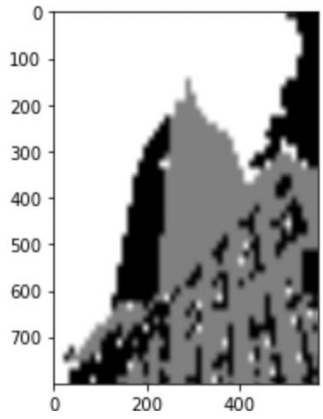
```
codeim = code.reshape(steps,steps)
```

```
codeim = resize(codeim,im.shape[:2])
```

```
figure()
imshow(codeim)
show()
```

解释说明:用一个步长为 steps 的方形网格在图像中滑动,每滑一次通过 mean 函数对网格中图像区域 R，G，B 三个颜色通道求像素平均值，最后得到的 features 矩阵是一个 steps*steps 行 3 列的矩阵，每列代表 R，G，B 三个颜色通道的像素平均值，将其作为新生成的低分辨率图像对应位置处的像素值，并用 K-means 进行聚类，得到中心聚类(centroids)和方差(variance)，进一步用 SciPy 包中的矢量量化函数对每个数据点进行归类，得到其每个观察值的代码簿索引(code)及其失真情况(distance)，用聚类标记创建图像。

In [37]:
```
from scipy.cluster.vq import *
from skimage.transform import resize
steps = 50 # 图像被划分成 steps×steps 的区域
im = array(Image.open('../pics4/empire.jpg'))
dx = im.shape[0] // steps
dy = im.shape[1] // steps
# 计算每个区域的颜色特征
features = []
for x in range(steps):
    for y in range(steps):
        R = mean(im[x*dx:(x+1)*dx,y*dy:(y+1)*dy,0])
        G = mean(im[x*dx:(x+1)*dx,y*dy:(y+1)*dy,1])
        B = mean(im[x*dx:(x+1)*dx,y*dy:(y+1)*dy,2])
        features.append([R,G,B])
features = array(features,'f') # 变为数组
# 聚类
centroids,variance = kmeans(features,3)
code,distance = vq(features,centroids)
# 用聚类标记创建图像
codeim = code.reshape(steps,steps)
codeim = resize(codeim,im.shape[:2])
figure()
imshow(codeim)
show()
```



第二部分 层次聚类

层次聚类是另一种简单而有效的聚类算法，它是基于样本间成对距离建立一个简相似性树。该算法首先将特征向量距离最近的两个样本归并为一组，并在树中创建一个平均节点，将这两个距离最近的样本作为该平均节点下的子节点。然后在剩下的包含任意平均节点的样本中寻找下一个最近的对，重复进行前面的操作。在每一个节点处保存了两个子节点间的距离。遍历整个树，通过设定的阈值，遍历过程可以在比阈值大的节点位置终止，从而提取出聚类簇。

1、 层次聚类算法实现

下面给出层次聚类算法的实现，请说明该聚类算法执行的基本过程。

```
from numpy import *
from itertools import combinations
```

```
class ClusterNode(object):
    def __init__(self,vec,left,right,distance=0.0,count=1):
        self.left = left
        self.right = right
        self.vec = vec
        self.distance = distance
        self.count = count # only used for weighted average

    def extract_clusters(self,dist):
        """ Extract list of sub-tree clusters from
            hcluster tree with distance<dist. """
        if self.distance < dist:
            return [self]
        return self.left.extract_clusters(dist) + self.right.extract_clusters(dist)

    def get_cluster_elements(self):
        """        Return ids for elements in a cluster sub-tree. """
        return self.left.get_cluster_elements() + self.right.get_cluster_elements()

    def get_height(self):
        """        Return the height of a node,
            height is sum of each branch. """
        return self.left.get_height() + self.right.get_height()

    def get_depth(self):
        """        Return the depth of a node, depth is
            max of each child plus own distance. """
        return max(self.left.get_depth(), self.right.get_depth()) + self.distance

    def draw(self,draw,x,y,s,imlist,im):
        """        Draw nodes recursively with image
            thumbnails for leaf nodes. """

        h1 = int(self.left.get_height()*20 / 2)
        h2 = int(self.right.get_height()*20 /2)
        top = y-(h1+h2)
        bottom = y+(h1+h2)

        # vertical line to children
        draw.line((x,top+h1,x,bottom-h2),fill=(0,0,0))

        # horizontal lines
        ll = self.distance*s
        draw.line((x,top+h1,x+ll,top+h1),fill=(0,0,0))
```

```
            draw.line((x,bottom-h2,x+ll,bottom-h2),fill=(0,0,0))

            # draw left and right child nodes recursively
            self.left.draw(draw,x+ll,top+h1,s,imlist,im)
            self.right.draw(draw,x+ll,bottom-h2,s,imlist,im)


class ClusterLeafNode(object):
    def __init__(self,vec,id):
        self.vec = vec
        self.id = id

    def extract_clusters(self,dist):
        return [self]

    def get_cluster_elements(self):
        return [self.id]

    def get_height(self):
        return 1

    def get_depth(self):
        return 0

    def draw(self,draw,x,y,s,imlist,im):
        nodeim = Image.open(imlist[self.id])
        nodeim.thumbnail([20,20])
        ns = nodeim.size
        im.paste(nodeim,[int(x),int(y-ns[1]//2),int(x+ns[0]),int(y+ns[1]-ns[1]//2)])


def L2dist(v1,v2):
    return sqrt(sum((v1-v2)**2))


def L1dist(v1,v2):
    return sum(abs(v1-v2))


def hcluster(features,distfcn=L2dist):
    """ Cluster the rows of features using
        hierarchical clustering. """

    # cache of distance calculations
```

```
    distances = {}

    # initialize with each row as a cluster
    node = [ClusterLeafNode(array(f),id=i) for i,f in enumerate(features)]

    while len(node)>1:
        closest = float('Inf')

        # loop through every pair looking for the smallest distance
        for ni,nj in combinations(node,2):
            if (ni,nj) not in distances:
                distances[ni,nj] = distfcn(ni.vec,nj.vec)

            d = distances[ni,nj]
            if d<closest:
                closest = d
                lowestpair = (ni,nj)
        ni,nj = lowestpair

        # average the two clusters
        new_vec = (ni.vec + nj.vec) / 2.0

        # create new node
        new_node = ClusterNode(new_vec,left=ni,right=nj,distance=closest)
        node.remove(ni)
        node.remove(nj)
        node.append(new_node)

    return node[0]


from PIL import Image,ImageDraw

def draw_dendrogram(node,imlist,filename='clusters.jpg'):
    """        Draw a cluster dendrogram and save to a file. """

    # height and width
    rows = node.get_height()*20
    cols = 1200

    # scale factor for distances to fit image width
    s = float(cols-150)/node.get_depth()

    # create image and draw object
```

```
im = Image.new('RGB',(cols,rows),(255,255,255))
draw = ImageDraw.Draw(im)

# initial line for start of tree
draw.line((0,rows/2,20,rows/2),fill=(0,0,0))

# draw the nodes recursively
node.draw(draw,20,(rows/2),s,imlist,im)
im.save(filename)
im.show()
```

解释说明：树节点有 ClusterNode 和 ClusterLeafNode 两个类，这两个类将用于创建聚类树，其中函数 hcluster() 用于创建树。

首先创建一个包含叶节点的列表，然后根据选择的距离度量方式将距离最近的对归并到一起，返回的终节点即为树的根。对于一个行为特征向量的矩阵，运行 hcluster() 会创建和返回聚类树。距离度量的选择依赖于实际的特征向量，这里我们利用欧式距离 L2（同时提供了 L1 距离度量函数）。对于每个子树，计算其所有节点特征向量的平均值，作为新的特征向量来表示该子树，并将每个子树视为一个对象。

通过 ClusterNode 的 extract_clusters() 方法从顶部遍历树直至一个距离小于设定阈值的节点终止获取聚类簇，如果节点间距离小于阈值，则用一个列表返回节点，否则调用子节点（叶节点通常返回它们自身）。调用该函数会返回一个包含聚类簇的子树列表。对于每一个子聚类簇，为了得到包含对象 id 的叶节点，需要遍历每个子树，并用 get_cluster_elements() 方法返回一个包含叶节点的列表，从而提取出聚类簇。

2、 通过随机函数创建一些二维数据点，对这些数据点进行聚类，设定阈值，从列表中提取这些聚类簇，并将其打印出来。编码实现上述过程。

In [39]:
```
class1 = 1.5 * randn(100,2)
class2 = randn(100,2) + array([5,5])
features = vstack((class1,class2))
tree = hcluster(features)
clusters = tree.extract_clusters(5)
print('number of clusters', len(clusters))
for c in clusters:
    print(c.get_cluster_elements())
```

```
number of clusters 2
[21, 100, 169, 123, 188, 197, 120, 121, 181, 192, 124, 138, 194, 147, 162, 110, 153, 145, 171, 131, 175, 182, 196, 16
1, 164, 176, 198, 112, 166, 193, 134, 190, 117, 108, 116, 135, 179, 109, 119, 101, 103, 136, 102, 154, 187, 113, 199,
118, 191, 183, 125, 143, 137, 160, 155, 178, 106, 107, 141, 128, 146, 115, 156, 177, 184, 151, 129, 149, 104, 195, 18
0, 148, 157, 174, 105, 165, 150, 152, 170, 186, 142, 163, 167, 168, 159, 133, 185, 139, 122, 144, 172, 130, 127, 158,
173, 140, 189, 114, 132, 111, 126]
[14, 94, 87, 16, 76, 2, 61, 48, 75, 50, 55, 60, 18, 51, 69, 3, 26, 73, 90, 0, 4, 8, 22, 43, 15, 49, 46, 80, 11, 53, 2
5, 65, 58, 54, 59, 81, 56, 83, 9, 68, 78, 64, 84, 7, 71, 37, 99, 20, 32, 39, 52, 17, 34, 10, 47, 89, 24, 95, 96, 5, 8
8, 62, 31, 44, 91, 27, 41, 1, 13, 72, 30, 82, 86, 45, 12, 92, 77, 6, 79, 35, 85, 19, 42, 57, 23, 40, 63, 66, 67, 97,
70, 33, 93, 98, 74, 28, 29, 36, 38]
```

3、 图像聚类

下面进行一个基于图像颜色信息对图像进行聚类的实验。

（1）文件 sunsets.zip 中包含了 100 张图像。我们用颜色直方图作为每幅图像的特征向量，即：将 RGB 三个颜色通道作为特征向量，将其传递到 NumPy 的 histogramdd()中，通过该函数来计算多维直方图。输入并运行如下代码，结合代码和运行结果说明分层聚类的过程。

```
import os
path = 'sunsets/'
imlist = [os.path.join(path,f) for f in os.listdir(path)
          if f.endswith('.jpg')]
```

```
features = zeros([len(imlist),512])
for i,f in enumerate(imlist):
    im = array(Image.open(f))
    h,edges = histogramdd(im.reshape(-1,3),8,normed=True,
                          range=[(0,255),(0,255),(0,255)])
    features[i] = h.flatten()
```

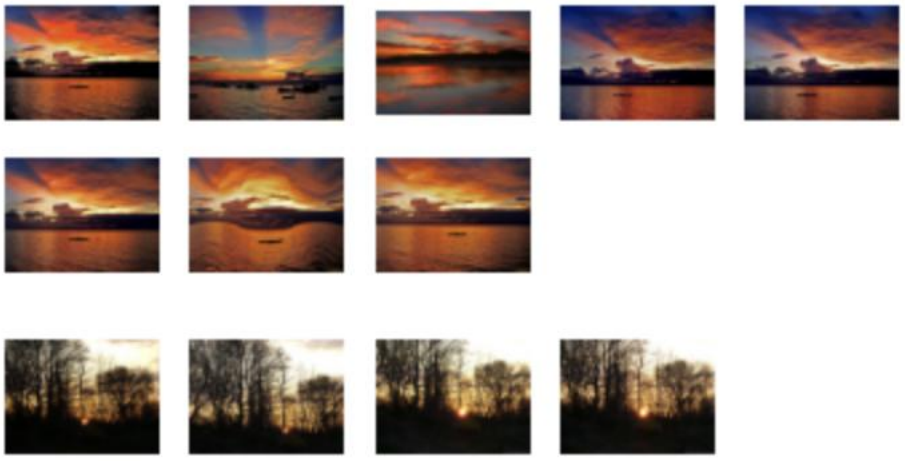
```
tree = hcluster(features)
```

```
from PIL import Image,ImageDraw

def draw_dendrogram(node,imlist,filename='clusters.jpg'):
    """    Draw a cluster dendrogram and save to a file. """

    # height and width
    rows = node.get_height()*20
    cols = 1200

    # scale factor for distances to fit image width
    s = float(cols-150)/node.get_depth()

    # create image and draw object
    im = Image.new('RGB',(cols,rows),(255,255,255))
    draw = ImageDraw.Draw(im)

    # initial line for start of tree
    draw.line((0,rows/2,20,rows/2),fill=(0,0,0))

    # draw the nodes recursively
    node.draw(draw,20,(rows/2),s,imlist,im)
    im.save(filename)
    im.show()
```

```
draw_dendrogram(tree,imlist,filename='sunset.pdf')
```

```
clusters = tree.extract_clusters(0.23*tree.distance)
```

```
for c in clusters:
    elements = c.get_cluster_elements()
    nbr_elements = len(elements)
    if nbr_elements>3:
        figure()
        for p in range(minimum(nbr_elements,20)):
            subplot(4,5,p+1)
            im = array(Image.open(imlist[elements[p]]))
            imshow(im)
            axis('off')
show()
```



解释说明：将 R、G、B 三个颜色通道作为特征向量，将其传递到 NumPy 的 histogramdd() 中计算其三维直方图。在每个颜色通道中使用 8 个小区间进行量化，将三个通道量化后的小区间拉成一行后便可用 512（8×8×8）维的特征向量描述每幅图像。为避免图像尺寸不一致，用“normed=True”归一化直方图，并将每个颜色通道范围设置为 0...255。将 reshape() 第一个参数设置为 -1 会自动确定正确的尺寸，故可以创建一个输入数组来计算以 RGB 颜色值为行向量的直方图。

（2）为前面实验中的字体图像进行层次聚类，并创建一个树状图，将该树状图绘制出来。给出代码和树状图（截屏）。

In [75]:
```
tree = hcluster(projected)
path = '../pics4/selectedfontimages'
imlist = [os.path.join(path,f) for f in os.listdir(path) if f.endswith('.jpg')]
draw_dendrogram(tree,imlist,filename='fonts.jpg')
```