Advanced Data Analysis: Report 3

David GAVILAN (05D38070)
Computer Science Dept.
e-mail: david@img.cs.titech.ac.jp

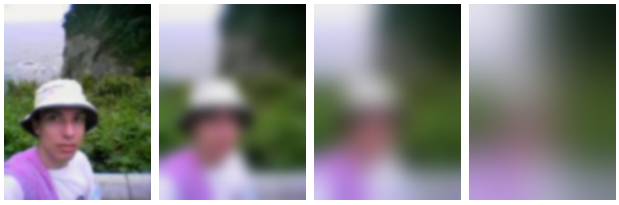
April 19th, 2005

1 Data Visualization

1.1 Data Set

I want to explore the *scale-space* of an image in the color domain and see if it can give relevant information for image segmentation, and particularly, for finding the most relevant objects in the image.

Let us try to cluster pixels independently. For doing so, first we will successively convolve the image with a gaussian kernel of increasing scale σ, as shown in Figure 1. Then, we will transform the $sRGB$ color space to the HSV space. I fixed 9 different scales, so the dimension of every pixel will be $d = 9 \times 3 = 27$. The number of samples is the size of the image $n = width \times height$.



$\sigma = 1$ $\sigma = 5$ $\sigma = 10$ $\sigma = 20$

Figure 1: Scale Space of an image.

Here is the Matlab code to prepare the data.

```
function ssv=scaleSpaceVector(l);
% ssv=scaleSpaceVector(l);
% l: an sRGB image of type double.
%    It will be converted to hsv
% ssv: a matrix where every row is a vector
%         of hsv values in the S-S

[h,w,c]=size(l);

kernelSize=3;
ssv=reshape(rgb2hsv(l),w*h,c);
figure, imshow(l);
for s=[0.1 1 3 5 8 10 15 20]
    kernelSize=2*ceil(3*s)+1;
    hG = fspecial('gaussian',kernelSize,s);
    g=imfilter(l,hG,'conv','symmetric');

    imshow(maxcon(g));
    ssv=[ssv,reshape(rgb2hsv(g),w*h,c)];
end
```

1.2 PCA

The solution of the problem

$$\mathbf{B}_{min} = \underset{\mathbf{B}\in\mathbb{R}^{m\times d}}{\arg\min} \left[tr(\mathbf{B}\mathbf{B}^T) \right] \qquad \text{s. t. } \mathbf{B}\mathbf{B}^T = \mathbf{I}_m \tag{1}$$

is given by

$$\mathbf{B}_{PCA} = (\psi_1 \mid \psi_2 \mid \cdots \mid \psi_m)^T; \tag{2}$$

where $\{\psi_i\}$ are the eigenvectors of the covariance matrix C of our data. The data will need to be centered first. This is the simple Matlab code that gives the solution:

```
function [veps,vaps,Xc]=PCA(X);
% [veps,vaps,m]=PCA(X);
%
% Principal Component Analysis
% X: the data in an array of the form n x d,
%    where d is the dimension of each vector,
%    and n is the total number of samples
%
% veps: Eigenvectors, ordered by their eigenvalue
% vaps: Eigenvalues, in ascent order
% Xc:   The centered data
[n, d]=size(X);

% center the data
m=mean(X);
Xc = X - ones(n, 1) * m;

% covariance matrix
C = Xc'*Xc;

% eigendecomposition
[veps,vaps]=eig(C);
```

For instance, for a given image l we just need to call these two methods:

```
ssv=scaleSpaceVector(double(l)/255);
[ve,va,dc]=PCA(ssv);
```

To project the data into three dimensions, we pick the three most important eigenvectors, that is, with the higher eigenvalue, and apply the transformation to the centered data. Figure 2 shows using false color the result of this projection, for both the most important and less important eigenvectors. This can be computed easily with:

```
imshow(reshape(dc*ve(:,25:27),160,120,3))
imshow(reshape(dc*ve(:,1:3),160,120,3))
```

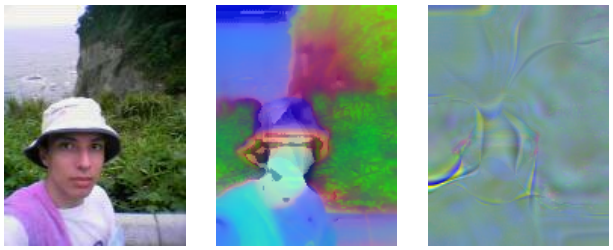


Figure 2: Original image and the false color of the transformation using the three most important eigenvectors, and the three less important ones, respectively.

2 Data Mining

We need not only good ways to cluster our data, but also good ways of visualizing it. For instance, by taking a look at Figure 2 we can easily observe that most of the energy of the image is conserved with the first transformation, which, of course, is congruent with the nature of the PCA. Numerically, we can compute the fraction of energy by:

```
v=diag(vaps);
d=size(v,1);
sum(v(d-m+1:d))/sum(v)
```

where m is the number of dimension where we are projecting. For the above example, where $m = 3$, the contribution of energy of these 3 eigenvectors is 87.93%.

Figure 3 shows the 3D space obtained after the transformation. Each point is rendered with the color it would have in the original picture.

If we try to cluster the space with a K-means algorithm, in just a few clusters, and visualize the original data replacing each vector by its label, we get the segmentation result of Figure 4 (a). Notice the difference of directly clustering the HSV color space in Figure 4 (b).

After observing all these figures, we have to guess what is the meaning of each axis of our eigen-decomposition. From Figure 3 we observe that pixels of similar color are most probably grouped together, but not always. If we project into a 2D space (Figure 5) we can observe more easily that the pixels the skin follows a line that crosses all the space.

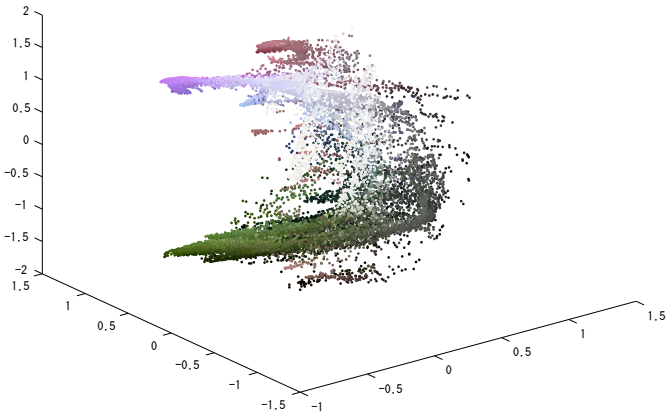


Figure 3: The space formed by our eigenvectors. Notice that these data is the same as the one in Figure 2 (middle).

(a) (b)

Figure 4: (a) The labeling result of clustering our projected data, setting 5 clusters for K-means; (b) clustering of the HSV color space setting K-means to 5 clusters.

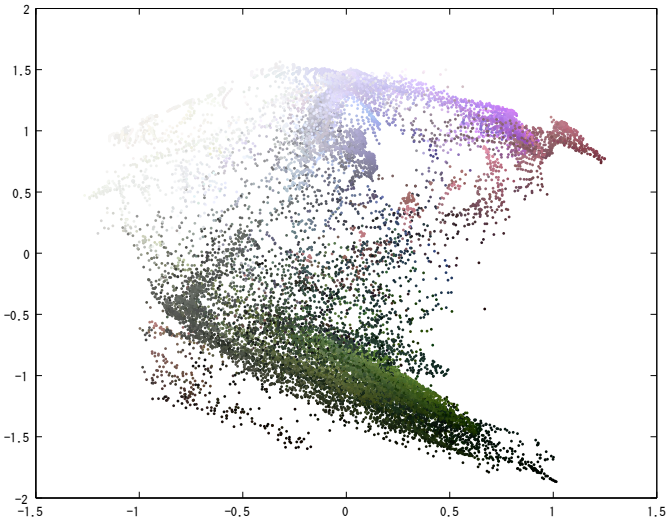


Figure 5: The space formed by two eigenvectors.

What about the scale? Probably the density of the cloud of points reflects the scale, since for bigger regions the color will remain more or less unaltered. This fact also points out that traditional clustering methods may not be appropriate for these data. Nevertheless, the k-means algorithm gives an idea of the nature of this clustering. Figure 6 shows another segmentation example.

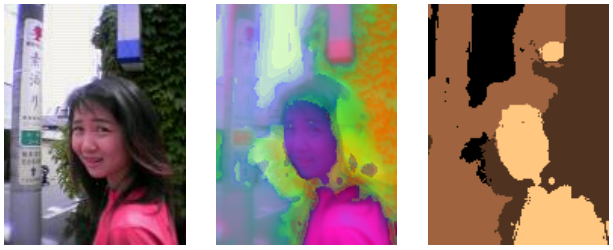


Figure 6: Another clustering example.

3 Problems

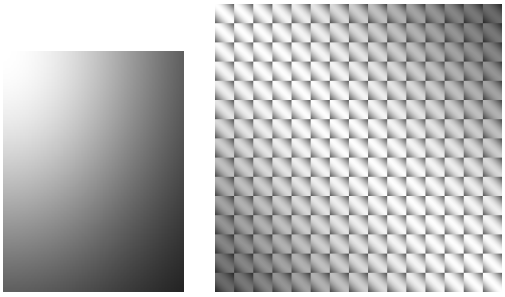
To reflect better the locality, I tried also using the LPP method with a weighing function that just gives the spatial distance between pixels. The weight matrix W is built using this code:

```
function W=euclideanMatrix(w,h,cx,cy,k);
% k: neighboring distance. If 0,
%    computes them all.
W=zeros(h,w);
for i=1:w
    for j=1:h
        if (k==0) | ((i-cx < k) & (j-cy<k))
           W(j,i)=exp(-((i-cx)*(i-cx)/(w*w)
   +(j-cy)*(j-cy)/(h*h)));
        end
    end
end

function W=euclideanComb(w,h,k);
W=sparse(0,0);
for i=1:w
    for j=1:h
        W=[W,sparse(
reshape(
euclideanMatrix(w,h,i,j,k),w*h,1))];
    end
end
```

Figure 7 (a) is the image reflecting the distance to the origin, and Figure 7 (b) is the whole matrix, with all the distances.

Then, the LPP can be solved using this function:

```
function [veps,vaps,Xc]=LPP(X,W);

[n, d]=size(X);
```



(a) (b)

Figure 7: (a) Distance map to the origin; (b) weight matrix W.

```
% center the data
m=mean(X);
Xc = X - ones(n, 1) * m;

DD=sparse(eye(n,n).*(ones(n,1)*sum(W)));
L=DD-W;

% generalized eigendecomposition
[veps,vaps]=eigs(X'*L*X,X'*DD*X);
```

However, when working with images of size 120×160, as in the example, W is a matrix of 19200×19200, that may require more than 1GB of memory in the computer! Using k-neighboring, we can make the matrix spare, so we can handle it. But still, I encountered a precision problem. The determinant of W is a very small value. When operated to compute D, D may become not invertible because of the lack of precision. Trying to reduce the image, of course, helps to overcome this, although we face the same problem in the last matrix $X \cdot D \cdot X^T$.

In fact, I had to reduce the size of the image up to 12×16, which, of course, is not desirable at all. I will try to find a better definition of the distance function.