Advanced Data Analysis: Report 9

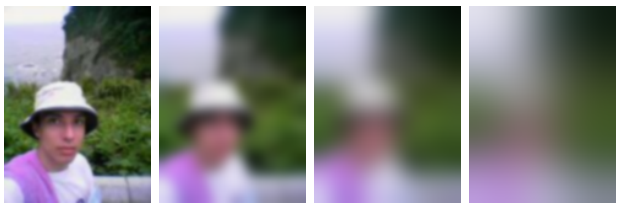
David GAVILAN (05D38070)
Computer Science Dept.
e-mail: david@img.cs.titech.ac.jp

June 21st, 2005

1 Data Visualization

1.1 Data Set

As in previous reports, 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. 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 $n = width \times height$. For this exercise, I reduced the image to 30×40 pixels, so the total number of samples will be $n = 1200$.



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

Figure 1: Scale Space of an image.

1.2 Kernels

For the "kernel trick" we need to define a reproducing kernel, used to compute the inner product in the feature space. For my tests, I will use the *Gaussian kernel*,

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(\frac{-\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{c}\right) \qquad c > 0. \tag{1}$$

In Matlab, this is given by the following code:

```
function K=Kgaussian(X,c);

[n d]=size(X);

for i=1:n
    D=X-ones(n,1)*X(i,:);
    K(:,i)=sum(D.^2,2);
end
K=exp((-K.^2)/c);
```

1.3 KPCA

To project our data using the Kernel PCA, first we have to solve the dual problem

$$\boldsymbol{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha} \tag{2}$$

where $\boldsymbol{K}_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Then, the embedding of the feature vectors $\{\boldsymbol{f}_i\}_{i=1}^n$ is given by

$$\boldsymbol{G} = (\boldsymbol{g}_1 \mid \boldsymbol{g}_2 \mid \cdots \mid \boldsymbol{g}_n) = \boldsymbol{\Lambda}^{1/2}\boldsymbol{A}; \tag{3}$$

where $\boldsymbol{A} = (\boldsymbol{\alpha}^{(1)} \mid \boldsymbol{\alpha}^{(2)} \mid \cdots \mid \boldsymbol{\alpha}^{(m)})^T$ and $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$.

This is the simple Matlab code that gives the solution:

```
function [G,K,Xc]=KPCA(X,m,kernel,c);
% [G,K,Xc]=KPCA(X,m,kernel,c);
%
% 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
% m: number of dimensions in which to project
% kernel: the type of kernel K
%        ('gaussian', 'polynomial')
% c: A parameter for the kernel
%
% G: The projected data into the feature space
%    (n x m)
%
% See: PCA, LPP, PPursuit, KLPP
%

[n, d]=size(X);

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

% Compute the kernel
K=sparse(eval(sprintf('K%s(Xc,c)',kernel)));

[A,La]=eigs(K,m);

G=(La.^0.5)*A';
G=G';
```

2 Data Mining

In order to have some reference, let us remember the result of the embedding using the linear PCA (Report 3). Figure 2 shows the 2D projection of the 1200 samples of the picture in Figure 1.

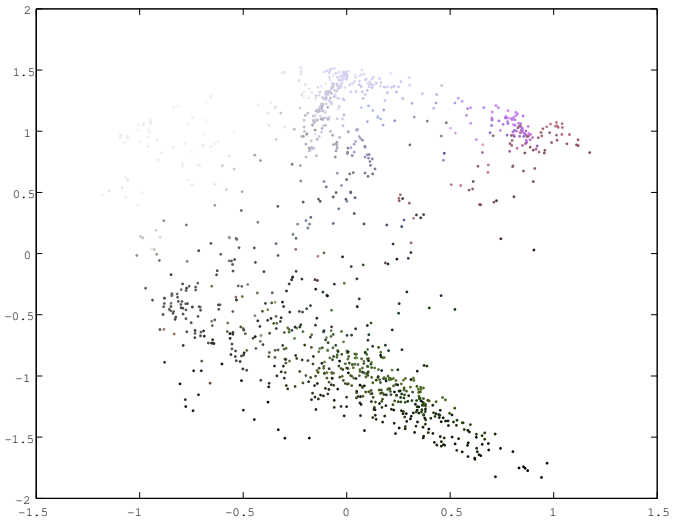


Figure 2: Linear PCA embedding.

For using the Gaussian Kernel PCA, we just need to set the deviation of the gaussian (the parameter c). Figure 3 is the kernel matrix for $c = 4$. Figures 4–7 show the embedding result for increasing values of c. Notice that the shape of the projection tends to a horseshoe, similar to that used to order color gamuts. The extremes of these horseshoe, also, are somehow close to red, green, and blue colors.

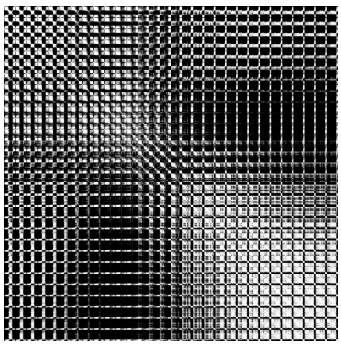


Figure 3: Gaussian kernel matrix, $c = 4$.

Therefore, by observing this, it is easy to think that we are just projecting the color, and the scale information is lost. However, we can see that the scale is also part of the embedding by clustering the embedded data. Figure 8 compares the result of clustering the image using K-means in the feature space and directly in the color space. Notice that in the feature space, the objects are better catpured.

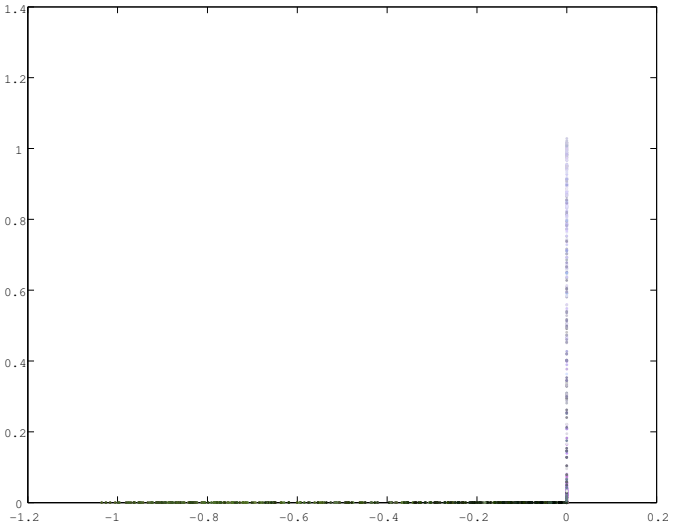


Figure 4: Gaussian KPCA embedding, $c = 0.1$.

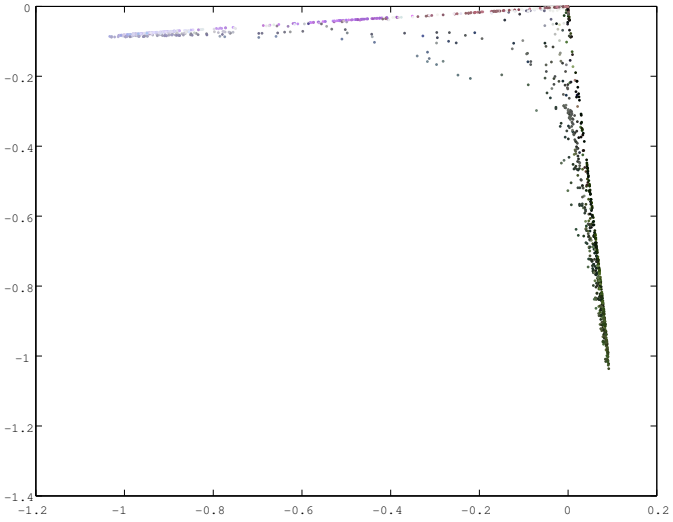


Figure 5: Gaussian KPCA embedding, $c = 1$.

Now, let us try to compare with the result of using the Locality Preserving Projection (LPP). Before (Report 3), I defined the weight matrix W as the spatial distance between pixels, so close pixels stay close. By defining a neighborhood of a few pixels, we can both forget very distant pixels and also have a sparse matrix. I added a parameter σ to change the shape of the gaussian of this distance. Figure 9 (a) is the image reflecting the distance to the origin; Figure 9 (b) is the whole matrix, with all the distances; and Figure 9 (c) is the matrix with a neighborhood limit of 4 pixels and $\sigma = 0.05$ (for an image of 30×40 pixels).

Then, the corrected[1] LPP embedding is performed can be solved using this code:

[1] In Report 3 I did not use the centered data.

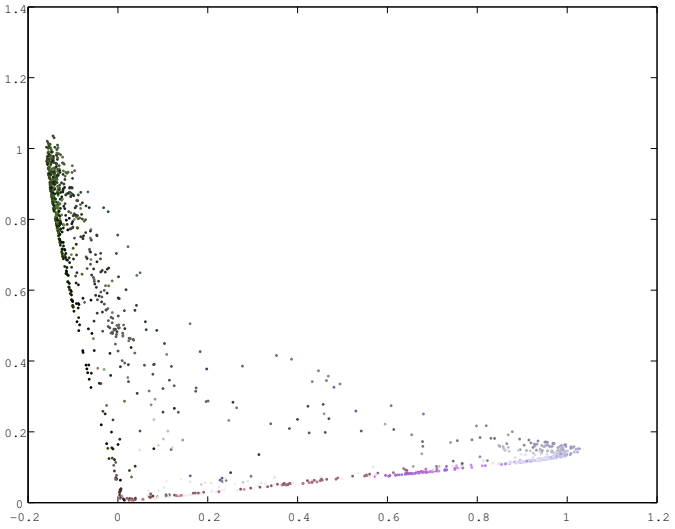


Figure 6: Gaussian KPCA embedding, $c = 2$.

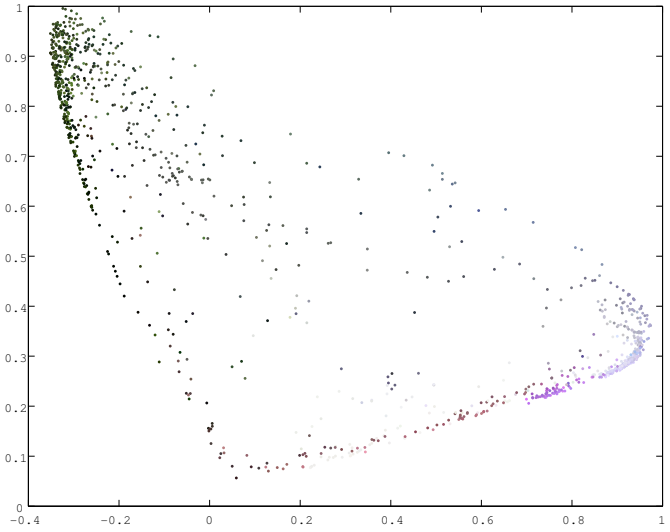


Figure 7: Gaussian KPCA embedding, $c = 4$.

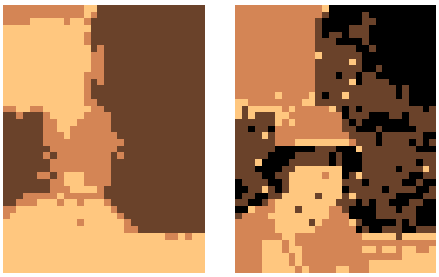
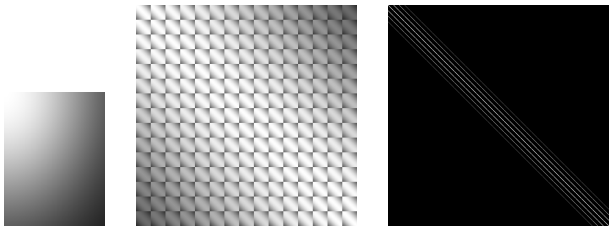


Figure 8: Left: clustering of the input data in the projected feature space using a Gaussian kernel of $c = 4$; right: direct clustering of the color data.



(a) (b) (c)

Figure 9: (a) Distance map to the origin; (b) weight matrix W with no neighborhood limits and $\sigma = 1$; (c) W with 4-pixels neighborhood and $\sigma = 0.05$.

```
DD=sparse(diag(sum(W)));
L=DD-W;
[veps,vaps]=eig(Xc'*L*Xc,Xc'*DD*Xc);
```

Figure 10 shows the result of the projection using the matrix of Figure 9 (c). It seems that using the spatial distance is not a good measure, since most of the points get packed together. Probably, the same problem will occur when applying the Kernel LPP. Therefore, we should define a new weighing function.

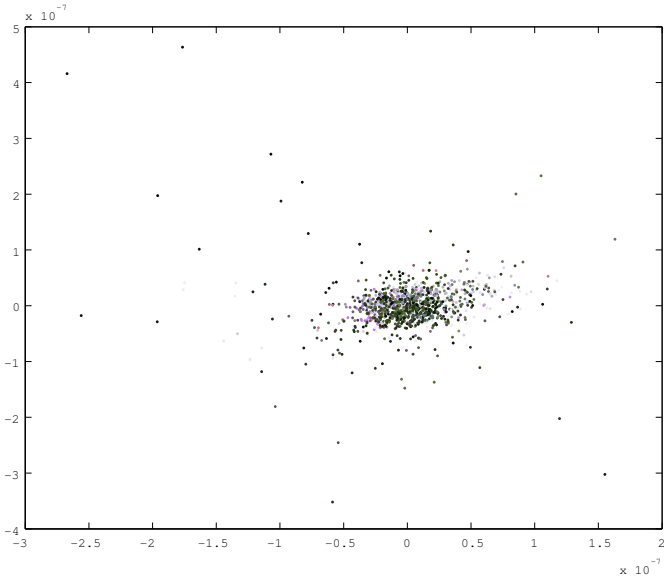


Figure 10: Linear LPP embedding.