Advanced Data Analysis: Report 5

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

May 24th, 2005

1 General non-Gaussianity method

Projection Pursuit: we want to iteratively find non-Gaussian directions. For the *general non-Gaussianity measure*, the solution to our projection problem is given by ψ_{min} or ψ_{max}, being

$$\psi_{min} = \underset{b \perp \{\psi_i\}_{i=1}^{k-1}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} G(\langle b, x_i \rangle) \quad \text{s.t. } \|b\|^2 - 1 = 0; \quad (1)$$

that can be solved with the Lagrangian:

$$L(b, \lambda) = \frac{1}{n} \sum_{i=1}^{n} G(\langle b, x_i \rangle) + \lambda(\|b\|^2 - 1). \quad (2)$$

To iteratively try to find a solution to

$$f(b) = \frac{\partial L}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} x_i G'(\langle b, x_i \rangle) + 2\lambda b = 0, \quad (3)$$

we can use Newton's method:

$$b_{k+1} \leftarrow b_k - \left(\left. \frac{\partial f}{\partial b} \right|_{b=b_k} \right)^{-1} f(b_k). \quad (4)$$

The derivate of equation (3) is given by:

$$\frac{\partial f}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T G''(\langle b, x_i \rangle) + 2\lambda I. \quad (5)$$

Let us assume that $x_i x_i^T$ and G'' are linearly independent[1]. Then, we can rewrite the first term in equation (5) as

$$\frac{1}{n} \sum_{i=1}^{n} x_i x_i^T \cdot \frac{1}{n} \sum_{i=1}^{n} G''(\langle b, x_i \rangle). \quad (6)$$

The first part is the covariance matrix C_s of our data. For centered and sphered data, we proved last week that C_s is the identity matrix. Therefore, we can approximate ∂f as

$$\frac{\partial f}{\partial b} = \left(\frac{1}{n} \sum_{i=1}^{n} G''(\langle b, x_i \rangle) + 2\lambda \right) I. \quad (7)$$

Since the above equation is a scalar multiplied by the identity matrix, its inverse is simply:

$$\left(\frac{\partial f}{\partial b} \right)^{-1} \approx \frac{1}{\left(\frac{1}{n} \sum_{i=1}^{n} G''(\langle b, x_i \rangle) + 2\lambda \right)} I.. \quad (8)$$

Thus, the Newton method can be approximated by:

$$b - \left(\frac{\partial f}{\partial b} \right)^{-1} f(b) \approx \frac{\frac{b}{n} \sum G''(\langle b, x_i \rangle) + 2\lambda b - 2\lambda b - \frac{1}{n} \sum x_i G'(\langle b, x_i \rangle)}{\frac{1}{n} \sum G''(\langle b, x_i \rangle) + 2\lambda}. \quad (9)$$

Since the denominator is a scalar, and in the final algorithm we normalize b, we can just ignore it. Then, defining $g(s) = G'(s)$, we can rewrite the approximate Newton algorithm for the Projection Pursuit as:

$$b \leftarrow b \frac{1}{n} \sum_{i=1}^{n} g'(\langle b, x_i \rangle) - \frac{1}{n} \sum_{i=1}^{n} x_i g(\langle b, x_i \rangle) \quad (10)$$

$$b \leftarrow b - \sum_{i=1}^{k-1} \langle b, \psi_i \rangle \psi_i \quad (11)$$

$$b \leftarrow \frac{b}{\|b\|}. \quad (12)$$

2 Implementation

If the covariance matrix is very big, we will have problems inverting it. For that reason, I will use the pseudo-inverse. Also, for my data, when computing the squared root of the matrix, I get some imaginary numbers that are almost zero. In such case, I will work only with the real part.

Another consideration is the condition for convergence. Let us denote d_k as

$$d_k = \frac{1}{n} \sum_{i=1}^{n} G(\langle b_k, x_i \rangle), \quad (13)$$

then we will say our algorithm has converged when $d_{k+1} \geq d_k$ (we are moving outside the minimum), or when $\|b_{k+1} - b_k\| < \epsilon$ (the direction barely changes).

This is the Matlab code:

[1] This assumption is difficult to prove.

```
function [psi,Xs]=PPursuit(X,m,measure);
% [psi,Xs]=PPursuit(X,m,measure);
%
% Projection Pursuit
% 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: #Dimensions in which we want to project the data
% measure: the name of the non-Gaussianity function
%
% Xs:   The centered and sphered data
% psi:  The vectors that span our projection
%

[n, d]=size(X);

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

% covariance matrix
C = Xc'*Xc;

% sphere the data
C2 = pinv(C)^(1/2);
% the imaginary part is almost zero: real(C2)
%  C2=real(C2);
Xs=Xc * C2;

% Generalized Non-Gaussian Method

psi=zeros(d,1); % in the end, it will be d x m
for k=1:m
 oldng=1e30;
 ng=1e20; % value of ng-measure at b
 b=rand(1,d); % start with random direction
 db=1e20; % distance between b_{k+1} and b_k
 e=1e-6; % stop if it doesn't move much
 while (db>e & oldng>ng)
    old = b;
    % non-gaussian
    bX = Xs * b'; % n x 1
    g = eval(sprintf('%sD(bX)',measure));
    g = g * ones(1,d); % n x d
    dg= eval(sprintf('%sDD(bX)',measure));
    b = b * sum(dg)/n - sum(Xs.*g)/n;

    % projection onto the span({\psi})
    bP = b * psi; % 1 x k
    b = b - sum((bP'*ones(1,d)).*psi');

    % normalization to hypersphere of r=1
    b = b/(sum(b.^2).^0.5);

    % has b moved?
    db = sum((old - b).^2);
    % value
    oldng = ng;
    bX = Xs * b'; % n x 1
    ng = eval(sprintf('%s(bX)',measure));
    ng = sum(ng)/n; % minimize this
 end
 psi(:,k)=b';
end
```

Some possible non-gaussianity measures are given by these functions and their derivatives:

```
function S=ng_exp(S)
S = -exp(-0.5*S.^2);

function S=ng_expD(S)
S = S.*exp(-0.5*S.^2);

function S=ng_expDD(S)
S = (1-S.^2).*exp(-0.5*S.^2);

function S=ng_kurtosis(S)
S = S.^4;

function S=ng_kurtosisD(S)
S = 4*S.^3;

function S=ng_kurtosisDD(S)
S = 12*S.^2;

function S=ng_logcosh(S)
S = log(cosh(S));

function S=ng_logcoshD(S)
S = tanh(S);

function S=ng_logcoshDD(S)
S = 1-tanh(S).^2;
```

For example, given some multidimensional data, we can find the m principal components by:

```
[psi, Xs] = PPursuit(data, m, 'ng_kurtosis');
```

Figure 1 is an example of randomly generated data, following a Gaussian distribution in one dimension. Starting from the worst case, that is, the Gaussian direction $(0, 1)$, with $\epsilon = 10^{-6}$, the algorithm converges in 60 iterations using the Kurtosis function $G(s) = s^4$, in 59 iterations using $G(s) = \log\cosh(s)$, and in 72 iterations using $G(s) = -\exp(-s^2/2)$. In this example the data has been centered, but not sphered.

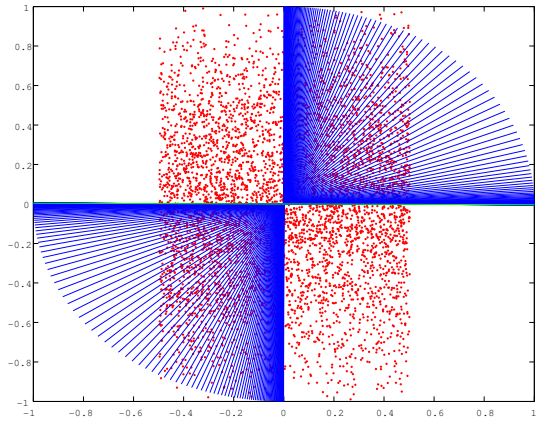

Figure 1: Newton algorithm to iteratively find the non-gaussian direction. Each blue line represents the direction found in each iteration, and the green horizontal line is the found non-gaussian direction.

3 Data Mining

As in the third report, I use the pixels of an image in the scale-space as my input data, being $d = 27$. The result of the 2D projection using PCA is shown again in Figure 2, where every point has been painted with the color it would have in the original picture.

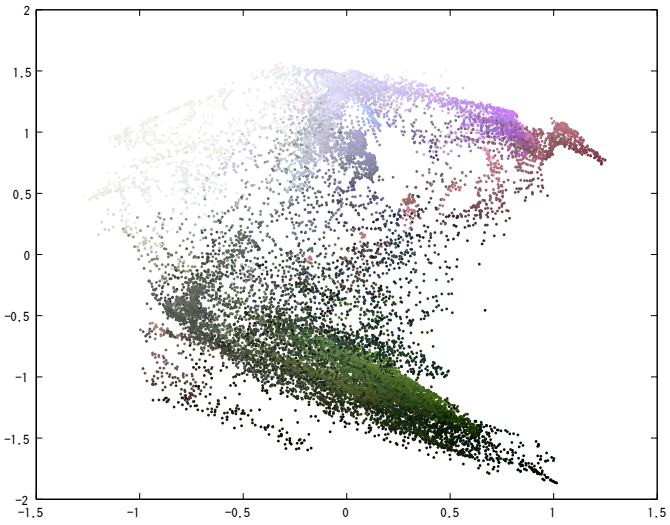

Figure 2: The space formed by two eigenvectors.

Figure 3 shows the projection of the data in two dimensions using the Kurtosis criteria, while Figure 4 is the result of the same algorithm but without applying sphering.

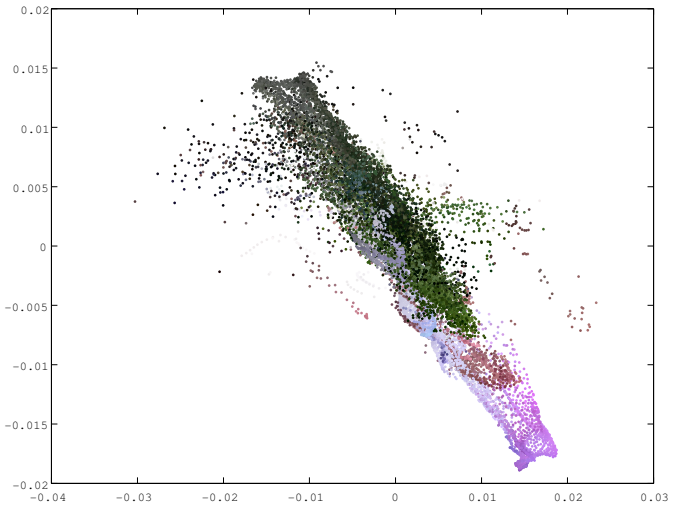

Figure 3: Projection of the centered and sphered data using PP and the Kurtosis measure.

The purpose of applying sphering is when variables have different scale, or "different units". But for these data, the "units" are always colors. Since I work in the HSV color space, after centering and sphering the Hue seems to lose relevance. Figure

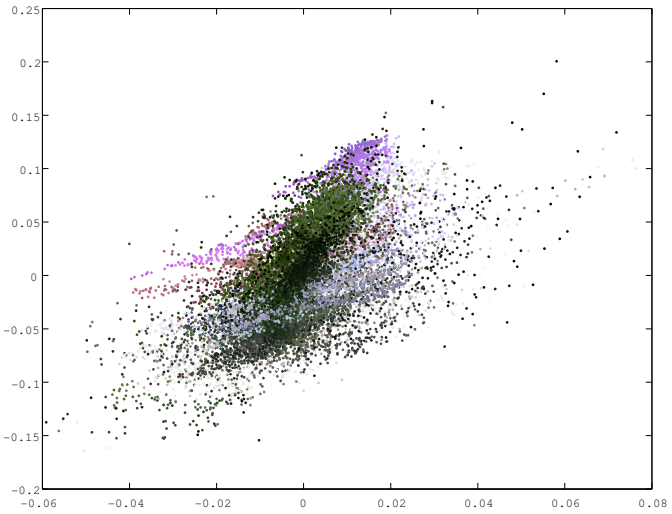

Figure 4: Projection of the centered data using PP and the Kurtosis measure.

5 shows the first scale of the original data, the first 3 dimensions of the centered and sphered data in the HSV color space, and the first 3 dimensions of the centered and sphered data in the RGB color space. Notice that after centering and sphering in the HSV color space, the image loses its hue.

Figure 5: The first 3 dimensions of the original data, and the centered and sphered data in the HSV and RGB color spaces, respectively.

In Figure 6 the data has been projected into the first three non-gaussian directions, and rebuild into an image. It is difficult to understand what they have encoded, but it seems the first component contains blob structures, that is, scale information; the second one, some color or saturation; and the third one, some contour notion.

Figure 6: The data projected with the first three non-gaussian directions.