4 Gaussian Mixture Models

Once you have a collection of feature vectors you will need to describe their distribution. You will do this using a Gaussian Mixture Model. The GMM comprises a collection of Gaussians, one for each class.

At the end of this tutorial you should be able to write code that inputs a collection of labelled feature vectors, and outputs a GMM.

4.1 GMMs

Gaussian mixture models are a common form of classifier. They are used to estimate the density of $K$ dimensional input data by

$$ p(x|\Theta) = \sum_{i=1}^{N} p(x|\theta_i)p(\theta_i). $$

The likelihood, $p(x|\theta_i)$ is specified to be a Gaussian:

$$ p(x|\theta_i) \triangleq \frac{1}{(2\pi)^{K/2}|C_i|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu_i)^T C_i^{-1} (x - \mu_i) \right) $$

the $\triangleq$ means “is defined to be equal to”. Each of the $N$ Gaussians in the model is a “component”, that is assumed to cover a region of feature space corresponding to a class. So that datum that “belong to” the component may be said to belong to the same class.

The prior, $p(\theta)$, counts the number of points in a data set that belong to a given class. More exactly, it is the fraction of the total number of points that belong to a class. This is the probability that a point picked at random belongs to a given class before (prior to) computing the likelihood. We have

$$ 1 = \sum_{i=1}^{N} p(\theta_i) $$

which means that a picked point must belong to some component.

In principle GMMs operate in any number of dimensions, but here we’ll consider versions in 1 and 2 dimensions. Let’s start with Gaussians in one dimension.

4.2 Gaussians in 1D

The Gaussian distribution, also called the Normal distribution is perhaps the best known of all distributions; commonly called the “bell curve”. It is defined by setting $K = 1$ in the above to give:

$$ p(x|\theta) \triangleq \frac{1}{(2\pi)^{1/2}|C|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu)^T C^{-1} (x - \mu) \right). $$
Here $\mu$ is the mean—the location of the peak of the Gaussian; and $C$ is its standard deviation—which defines its width. The area under the Gaussian is 1 i.e.

$$\int_{-\infty}^{\infty} \frac{1}{(2\pi)^{1/2}|C|^{1/2}} \exp \left(-\frac{1}{2}(x-\mu)^T C^{-1} (x-\mu) \right) dx = 1.$$

The Gaussian distribution in 1D is often expressed as follows:

$$p(x|\theta) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{(x-\mu)^2}{2\sigma^2} \right)$$

The Gaussian, $p(x|\theta)$, indicates the density of data points at a particular location, $x$. Suppose a data set contains $M$ points, then we would expect to see $Np(x|\theta)$ points close to (say plus of minus $dx/2$) the location $x$.

The 1D Gaussian is easy to make in MATLAB, Here is part of a script:

```matlab
dx = 0.1;
x = -10 : dx : 10;
mu = 5;
C = 2^2;
px_gauss = exp( -0.5*(x-mu) .* C^(-1) .* (x-mu) ) / (sqrt(2*pi)*sqrt(C));
s = sum( px_gauss * dx )
figure;
plot( x, px_gauss );
```

It is deliberately left uncommented, and the semi-colon is deliberately missing from one of the lines. Add comments, and figure out why $s$ is not 1.00. How can this be fixed?

Write code that creates more than one 1D Gaussian, each with a distinct mean and standard deviation. Visualise all the Gaussians in a single plot.

Develop you code to scale each Gaussian by a weight and then add the weighted Gaussians to make a single function. Ensure the weights are all positive and that they sum to make 1. Modify your visualisation to show the weighted components and the final result.

If you wish, you may compare your code with `Gauss_1D.m`.

### 4.3 Gaussians in 2D

In 2D, the Gaussian has a bell shape that has an elliptical cross section (contour lines are ellipses). It has a mean $\mu$, which is a point on the 2D plane that lies directly under the peak. The spread of the bell (shape and angle of the ellipse) is governed by the covariance matrix $C$, which is the analogue of the standard deviation in the 1D case. As in the 1D case, the Gaussian gives the density of data points at each location $x$ but now $x$ is a location on a 2D plane rather than on a (1D) line.

The covariance matrix $C$ takes a little work to understand. As already mentioned, all Gaussians are elliptical in cross-section; the matrix $C$ controls
the particular ellipse—whether it is long and narrow or short and round, and its orientation.

The covariance matrix is always a real symmetric matrix; real means each of its entries are real rather than complex numbers, and symmetric means it is the same as its transpose, so \( C = C^T \).

These turn out to be useful properties because the covariance can be written as the product of two matrices, \( U \) and \( L \), using a process called eigenvalue decomposition:

\[ C = ULU^T. \]

The \( U \) matrix is a rotation and it is responsible for turning the Gaussian, formally it is an orthonormal matrix, meaning \( UU^T = U^TU = I \), where \( I \) is the identity matrix. The \( L \) matrix is a diagonal matrix, meaning the only non-zero elements appear along the leading diagonal; it is responsible for the shape (eg long and thin) of the Gaussian and its size (wide or narrow spread).

There is more: the diagonal elements of \( L \), denoted here as \( l_{ii} \) are called the eigenvalues. Each eigenvalue is associated with an eigenvector; the \( i^{th} \) eigenvalue \( l_{ii} \) is associated with the \( i^{th} \) eigenvector, which is just the \( i^{th} \) column of \( U \). Each eigenvector has unit size and they are mutually orthogonal (which is another way to say that \( U \) is orthonormal); the size of each eigenvector is indicated by the (square root of the) corresponding eigenvalue.

In this case, the eigenvectors points along the axes of the ellipse that is the cross-section of the Gaussian. Each eigenvalue is the variance of the data along the direction of its corresponding eigenvector. This means the following: if all the data were to be perpendicularly projected onto a line defined by the eigenvector, the corresponding eigenvalue would be the variance of the projection. The square root of the variance is the standard deviation, and useful contours of the Gaussian are defined via the standard deviation.

### 4.4 A 2D Gaussian in MATLAB

Here is some code to define a 2D Gaussian distribution in MATLAB:

```matlab
%% Specify the parameters
% This defines the square around the origin (0, 0) in which the probability
% densities will be calculated
L = 10;

mu = [3;-2.4]; % the mean (a point in 2D)
theta = 30*pi/180; % rotation angle (which specifies the eigenvectors)

%% Calculate the covariance matrix
% Effectively we use a matrix C to rotate the point [x y], scale it, and then
```
% rotate it back
% R performs the rotation, remember R' = inv(R), and S the scaling
R = [cos(theta) -sin(theta); sin(theta) cos(theta)]; % rotation matrix

S = [scale1 0; 0 scale2]; % scale matrix, a diagonal matrix
C = R*S*R'; % the covariance

%% Calculate the probability distribution
X = -L:L; % vector of x locations
Y = -L:L; % vector of y locations
probG = zeros( 2*L+1, 2*L+1 ); % an array to hold the Gaussian distribution
%probability values
K = 1/(2*pi*sqrt(det(C))); % a constant

for j = 1:length(Y)
    for i = 1:length(X)
        x = X(i) - mu(1);
        y = Y(j) - mu(2);
        probG(j,i) = K*exp( -0.5* [x y] * inv(C) * [x; y] );
    end
end

%% Plot it in 3D
figure
surf( probG );

% All going well, this should sum to 1.0
sum(probG(:))

The above code is rather slow. Here more efficient code for the same thing:

%% Specify the parameters
% This defines the square around the origin (0, 0) in which the probability
% densities will be calculated
L = 10;

mu = [3;-2.4]; % the mean (a point in 2D)
theta = 30*pi/180; % rotation angle (which specifies the eigenvectors)

%% Do the calculations
[x,y] = meshgrid(-L:L);

R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
S = [3 0; 0 1]; C = R*S*R';
\% string all locations out into an array, with mean subtracted
\[ z = [x(:) - \mu(1), y(:) - \mu(2)]'; \]

\[ \text{probG} = \exp\left(-0.5 \cdot \text{sum}(z \cdot (C \cdot z), 1)\right) / (2 \cdot \pi \cdot \sqrt{\text{det}(C)}); \]

\text{figure;}
\text{surf(x, y, reshape(probG, 2*L+1, 2*L+1));}
\text{sum(probG(:));}

4.5 Exercises

Satisfy yourself that the 2D Gaussian can be moved, shaped, sized, and rotated by changing the parameters. Ensure that the volume under the 2D Gaussian is always close to 1, in accordance with the property

\[ 1 = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{1}{(2\pi)^{1/2} |C_i|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu_i)^T C_i^{-1} (x - \mu_i) \right) \, dy \, dx. \]

Modify your code to take the sum of more than one distinct 2D Gaussian. As in the 1D case use weighted sums, and ensure the weights sum to 1. Visualise the result.

In the next lab, you will model (the feature vectors of) each shape class (e.g. alien or arrow) using a 2D Gaussian distribution, by learning the mean vector and covariance matrix of each class. The probability distribution for all shapes will therefore be represented by a Gaussian Mixture Model. (Optionally, an individual shape class could be modelled using a Gaussian Mixture Model but this will require knowledge of the expectation-maximization (EM) algorithm.)