Learning from Data 1
Density Estimation

David Barber
dbarber@anc.ed.ac.uk
course page : http://anc.ed.ac.uk/~dbarber/lfd1/lfd1.html

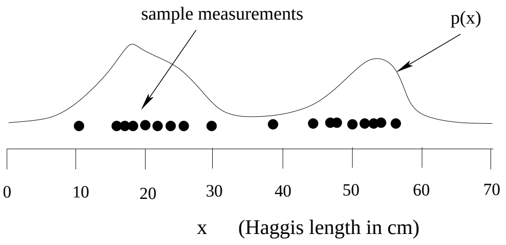

Figure 1: The measured lengths of haggis. This suggests a distribution $p(x)$, where x is haggis length, as shown.

1 Why Density Estimation?

Consider our example of handwritten digits, where each digit is a $28 \times 28 = 784$ dimensional vector $\mathbf{x}$. It is intuitively clear that, once we have seen only a few examples of someones handwriting style, we have a pretty good idea of how they write the number seven. It maybe that they do not always write *exactly* in the same way, but we would probably have little difficulty in recognising another seven, even though *every* single seven that they write is different. What this means is that we have in some sense, captured the way that a person writes a seven – sometimes there may be minor modifications, but we have a pretty good idea of what effect these would have. Since *every* seven that someone writes is different from the others, it makes more sense to use probabilities to describe how probable certain types of seven are than others. That is, we can describe how Mavis writes sevens by using a distribution $p(\mathbf{x})$. Thus, perhaps sevens that are very curly have a high probability density value, whilst very straight sevens have a low probability. The usefulness of probability is that it enables us to define a likelihood value for *every* one of the *infinite* number of different sevens that Mavis could write.

Haggis example

Haggis are a commonly occurring wild animal in the highlands of Scotland. Being relatively easy to catch (due to their unfortunate evolutionary quirk of having one leg longer than another to make contouring around hills easier) many measurements have been made over the years of the size of adult haggis. These are plotted in fig(1) (Actually, most of the larger measurements are from female haggis, whilst the smaller measurements are typically from the males). A good summary of this data is given by using the probability density function (p.d.f) $p(x)$ as drawn. This gives a probability density value $p(x)$ for any length of haggis. The distribution must satisfy the normal rules of probability, $\int p(x)dx = 1, p(x) \geq 0$.

1.1 How do we find $p(x)$?

Density estimation is the "learning" of $p(\mathbf{x})$ given examples of $\mathbf{x}$. Usually, in order to compress the information, we paramterise the probability density function is some way. A very important class of probability density functions is the Gaussians which in one dimension are:

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{1.1}$$

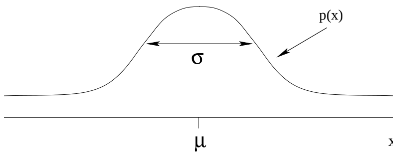

Figure 2: The Gaussian (or normal distribution) $p(x|\mu, \sigma^2)$

This distribution is the classic "bell shaped" distribution as plotted in fig(2). There are two parameters to this distribution

$$\mu = \int xp(x)dx, \text{ the mean, and} \qquad \sigma^2 = \int (x-\mu)^2 p(x)dx, \text{ the variance.} \tag{1.2}$$

If we had a set of datapoints $X = \{x^1, \dots x^P\}$, how could we find the best setting of μ and σ^2 to give the best fit of $p(x|\mu, \sigma^2)$ to the data?

1.2 Maximum Likelihood fitting

If we make the standard assumptions that each observed datapoint has been drawn independently from the same (identical) distribution, $p(x|\theta)$, where θ are some parameters describing the distribution (such as the mean and variance), the likelihood of generating the data X given our p.d.f is

$$p(X|\theta) = \prod_{i=1}^{P} p(x^i|\theta) \tag{1.3}$$

We can thus calculate the log-likelihood of the data as

$$L = \sum_{i=1}^{P} \log p(x^i|\theta) \tag{1.4}$$

In the case of using a Gaussian this is

$$L(\mu, \sigma^2) = -\frac{1}{2\sigma^2}\sum_{i=1}^{P}(x^i-\mu)^2 - \frac{P}{2}\log(2\pi\sigma^2) \tag{1.5}$$

We wish to find the paramters $\theta = (\mu, \sigma^2)$ that maximise the likelihood (since the logarithm is a monotonically increasing function, this is the same as maximising the log-likelihood). Differentiating L with respect to μ gives, and equating to zero gives

$$\mu = \frac{1}{P}\sum_{i=1}^{P} x^i \tag{1.6}$$

Similarly, differentiating with respect to σ^2 and equating to zero gives

$$\sigma^2 = \frac{1}{P}\sum_{i=1}^{P}(x^i-\mu)^2 \tag{1.7}$$

These are known as the maximum likelihood (ML) sample estimates of the parameters. There are other ways fit distributions to data, and these can give different parameter settings to those obtained by ML. The ML estimator of σ^2 is biased in the sense that if we were to repeat the experiment many times by drawing samples from a distribution with known mean and variance, the ML estimated variance, averaged over many trials, would differ from the correct variance slightly. The unbiased estimator for the variance is $\sigma^2 = \frac{1}{P-1}\sum_{i=1}^{P}(x^i-\mu)^2$. This difference is typically negligible in practice.

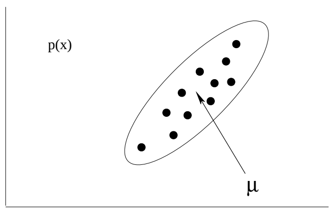

Figure 3: Fitting a Gaussian to 2 dimensional data. We need to find an estimate for the mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.

1.3 Fitting Gaussians – the multi-dimensional case

Given a dataset $X = \{\mathbf{x}^1, \ldots, \mathbf{x}^P\}$, we wish to fit a Gaussian to this data. For example, how can we fit the data in fig(3)? The multi-dimensional Gaussian is given by

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{1.8}$$

It is straightforward to show that the maximum likelihood estimates are given by

$$\boldsymbol{\mu} = \frac{1}{P}\sum_{i=1}^{P}\mathbf{x}^i \tag{1.9}$$

and

$$\boldsymbol{\Sigma} = \frac{1}{P}\sum_{i=1}^{P}(\mathbf{x}^i-\boldsymbol{\mu})(\mathbf{x}^i-\boldsymbol{\mu})^T \tag{1.10}$$

Note that, for an n-dimensional vector $\mathbf{x}$, the covariance matrix will have $n(n-1)/2$ elements. If we are to accurately estimate these parameters, we will need a lot of data to do so. It is therefore often a good idea to reduce the dimensionality of the data first (using PCA say), so that the number of parameters in the covariance matrix is reduced significantly.

1.4 Fitting a Gaussian to the Handwritten Digits

In some experiments later, we shall be looking at classifying the handwritten digits one and seven. In order to reduce the dimensionality of the data, I first use PCA on 600 training examples of our usual 28×28 dimensional data. This training data contains 300 ones and 300 sevens for training. Using PCA I reduced the dimensionality down to 20. However, fitting a single Gaussian to the PCA representation of both ones and sevens does not make much sense since we believe that there will be two high probability regions in the 20 dimensional space, separated by a region of low probability (that is, there are two clusters, one corresponding to the ones, and one corresponding to the sevens). If we do this, fitting a single Gaussian, and then sample 100 pca vectors from this distribution, the reconstructions are given in fig(4). Instead, we can fit a *class conditional* distribution to the PCA data. That is, we fit one Gaussian to the PCA representation of the sevens, and another to the PCA representation of the ones. We again sample 100 PCA representations from each of the two Gaussians, and these are plotted in figure fig(5). Note how the sample reconstructions are much more like ones and sevens than the single global model.

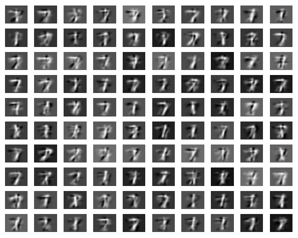

Figure 4: Reconstructions using 100 samples from a 20 dimensional Gaussian fitted to 600 training examples of ones and sevens. Note that the reconstructions are not particularly good examples of either ones or sevens, with a lot of mixed types.

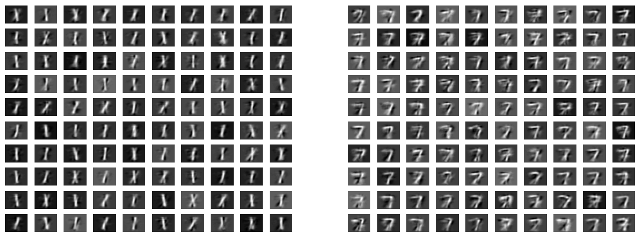

Figure 5: Reconstructions using 100 samples from a 20 dimensional Gaussian fitted to 300 training examples of ones (left) and sevens (right).

1.5 Making Classifications using Class Conditional Densities

If we have data with two (or more classes), we can fit a separate Gaussian to each class, $p(\mathbf{x}|class1), \ldots, p(\mathbf{x}|classK)$ where each Gaussian is characterised by its mean $\boldsymbol{\mu}^(k)$ and covariance matrix $\boldsymbol{\Sigma}^(k)$, $k = 1, \ldots, K$. (Note that here the upper index k does not refer to taking powers – it just indexes the class).

What we are really interested in for classification is the following: given a new datapoint $\mathbf{x}^*$, what is $p(class = k|\mathbf{x}^*)$?

To evaluate this probability, we use *Bayes rule*

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \tag{1.11}$$

In this case this gives

$$p(class = k|\mathbf{x}^*) = \frac{p(\mathbf{x}^*|class = 1)p(class = 1)}{p(\mathbf{x}^*)} \tag{1.12}$$

Decision Boundary

Since the denominator on the right hand sides of the above equation is independent of the class, we will classify $\mathbf{x}^*$ as class k provided

$$p(\mathbf{x}^*|class = k)p(class = k) > p(\mathbf{x}^*|class = l)p(class = l) \quad \text{for all } \ 1 \neq \text{k} \tag{1.13}$$

It is straightforward to show that the maximum likelihood estimate of $p(class = k)$ is simply given by the relative frequency of occurrence of each class in the training set. For example, if there are 100 class 1 training points and 50 class 2 training points, then $p(class = 1) = 2/3$ and $p(class = 2) = 1/3$. Normally we take logarithms of the above (this is allowed since it is a monotonic function and will not affect the decision process). The reason for this is that this is a numerically more stable procedure. That is, we will classify $\mathbf{x}^*$ as class k provided

$$\begin{aligned}\log p(\mathbf{x}^*|class = k) + \log p(class = k) \\ > \log p(\mathbf{x}^*|class = l) + \log p(class = l) \quad \text{for all } \ 1 \neq \text{k}\end{aligned} \tag{1.14}$$

In our case, using the fact that each density is Gaussian, we classify $\mathbf{x}^*$ as class 1 if

$$-\frac{1}{2}\left(\mathbf{x}^*-\boldsymbol{\mu}^{(1)}\right)^T(\boldsymbol{\Sigma}^{(1)})^{-1}\left(\mathbf{x}^*-\boldsymbol{\mu}^{(1)}\right)-\frac{1}{2}\log\det\boldsymbol{\Sigma}^{(1)}+\log p(class=1)$$
$$>-\frac{1}{2}\left(\mathbf{x}^*-\boldsymbol{\mu}^{(2)}\right)^T(\boldsymbol{\Sigma}^{(2)})^{-1}\left(\mathbf{x}^*-\boldsymbol{\mu}^{(2)}\right)-\frac{1}{2}\log\det\boldsymbol{\Sigma}^{(2)}+\log p(class=2) \quad (1.15)$$

An example is given in fig(6) where two dimensional data of two different classes is plotted. The classification was produced using the code below.

```
% Demo for fitting Gaussians and using Bayes to classify : 2 classes

% generate some fake training data for class 1 :
X1 = randn(2,10);

% generate some fake training data for class 2 :
X2 = randn(2,15) + repmat(3*ones(2,1),1,15);

% fit a Gaussian to data X1 :
m1 = mean(X1')'; S1=cov(X1')'; invS1 = inv(S1);
logdetS1=trace(logm(S1));
p1 = size(X1,2)/(size(X1,2)+size(X2,2)); % prior

% fit a Gaussian to data X2 :
m2 = mean(X2')'; S2=cov(X2')'; invS2 = inv(S2);
logdetS2=trace(logm(S2));
p2 = 1-p1; % prior

Xnew =2*randn(2,50)+ 2*repmat(ones(2,1),1,50);; % some test points
% calculate the decisions :
d1 = (Xnew-repmat(m1,1,size(Xnew,2))); d2 =
(Xnew-repmat(m2,1,size(Xnew,2)))

for i = 1 : size(Xnew,2)
  if  d2(:,i)'*invS2*d2(:,i)+logdetS2 -2*log(p1) > d1(:,i)'*invS1*d1(:,i)+logdetS1-2*log(p2)
    class(1,i)=1;
  else
    class(1,i)=2;
  end
end

% plot a few things :
plot(X1(1,:),X1(2,:),'bx','markersize',10,'linewidth',2); hold on; % class 1;
plot(X2(1,:),X2(2,:),'ro','markersize',10,'linewidth',2);  % class 2;
plot(Xnew(1,find(class==1)),Xnew(2,find(class==1)),'bx');  % class 1;
plot(Xnew(1,find(class==2)),Xnew(2,find(class==2)),'ro'); hold off % class 2;
```

The decision boundary is therefore quadratic – in a later chapter we shall encounter simpler, linear decision boundaries. The decision boundary becomes linear (a straight line or hyperplane) in the case that the two class covariances are equal.

An important point to stress though is that we no longer need to store the dataset to make our decisions – we just can use the rule above, resulting in a much faster decision rule than that used, for example, in nearest neighbour classification. It is usually a good idea to calculate and store the inverse matrices offline, as in the code above, so that, during classification we do not need to keep inverting the matrices. (However, if the data is very high

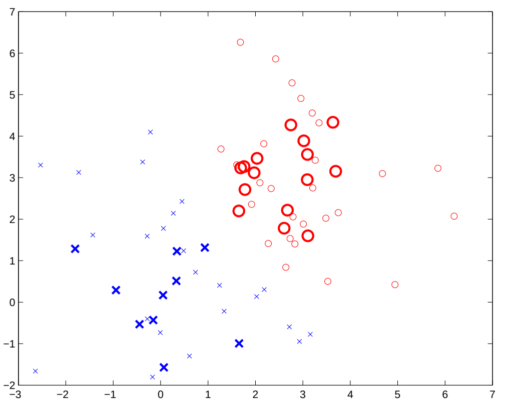

Figure 6: Classification using two classes. The large symbols are the training data. The small symbols are the positions of the test data and they are given the symbol according to the Bayes decision rule.

dimensional, there may are better ways to calculate the decision boundary without computing the inverse of the covariance matrices.)

High dimensional Data

If the data is very high dimensional (say above 1000) then calculating $\log\det\Sigma$ is numerically very difficult (since det is the product of the eigenvalues so that under/overflow problems can occur). A numerically more stable way to do this is to use the identity $\text{trace}\,(\log)\, M \equiv \log\det M$ where $\log M$ is the *matrix* logarithm (not the element-wise logarithm). Thus, in MATLAB, one can replace `log(det(Sigma))` with `trace(logm(Sigma))`.

1.6 Making Classifications using Class Conditional Densities

In the above sections, we fitted two probability distributions to data, $p(\mathbf{y}|digit = 1)$ and $p(\mathbf{y}|digit = 7)$ where $\mathbf{y}$ are the PCA representations of the data.

What we are really interested in for classification is the following: given a new $\mathbf{y}^*$ (the PCA representation of a new test point $\mathbf{x}^*$), what is $p(digit = 1|\mathbf{y}^*)$?

To evaluate this probability, we use *Bayes rule*

$$p(digit = 1|\mathbf{y}^*) = \frac{p(\mathbf{y}^*|digit = 1)p(digit = 1)}{p(\mathbf{y}^*)} \tag{1.16}$$

Similarly,

$$p(digit = 7|\mathbf{y}^*) = \frac{p(\mathbf{y}^*|digit = 7)p(digit = 7)}{p(\mathbf{y}^*)} \tag{1.17}$$

Since the denominators on the right hand sides of the above two equations are equal, we will classify $\mathbf{y}^*$ as a one provided

$$p(\mathbf{y}^*|digit = 1)p(digit = 1) > p(\mathbf{y}^*|digit = 7)p(digit = 7) \tag{1.18}$$

In our case, we have Gaussians modelling the distributions $p(\mathbf{y}^*|digit = 1)$ and $p(\mathbf{y}^*|digit = 7)$. The values $p(digit = 1)$ and $p(digit = 7)$ are both 0.5 since there are an equal number of training examples of ones and sevens. Thus, in our case, the decision is that $\mathbf{y}^*$ is a one provided

$$\log p(\mathbf{y}^*|digit = 1) > \log p(\mathbf{y}^*|digit = 7) \tag{1.19}$$

That is,

$$-\frac{1}{2}\left(\mathbf{y}^* - \boldsymbol{\mu}^{(1)}\right)^T (\boldsymbol{\Sigma}^{(1)})^{-1}\left(\mathbf{y}^* - \boldsymbol{\mu}^{(1)}\right) - \frac{1}{2}\log\det\boldsymbol{\Sigma}^{(1)} > -\frac{1}{2}\left(\mathbf{y}^* - \boldsymbol{\mu}^{(7)}\right)^T (\boldsymbol{\Sigma}^{(7)})^{-1}\left(\mathbf{y}^* - \boldsymbol{\mu}^{(7)}\right) - \frac{1}{2}\log\det\boldsymbol{\Sigma}^{(7)} \tag{1.20}$$

Using the above decision method on the 600 test points gives an error on only 10 examples. Note that this is therefore about the same as the nearest neighbour method (there we used 50 components and had an error of 18), if not a little better.

As opposed to the nearest neighbour method, we no longer need to store the dataset to make our decisions – we just can use the rule above where we only need to store the various means and covariances and prior parameters, resulting in a much more efficient and faster decision rule.

1.7 Why might this density approach work better than KNN?

Generative Models

One of the reassuring aspects of this density approach is that we can used our fitted models to generate fake data from each class – such models are called *generative models*. We can relatively easily see if the fake generated data 'looks like' the training data and gain some confidence that our underlying model is a good model of the data if the generated data are representative of data from that class. In general, according to our main arguments about machine learning, we expect certain directions in the space to be more important than others. Fitting Gaussians to each class enables us to find the important directions, and weight them accordingly – that is, in general, the covariance matrix in each dimension will be non-diagonal. Thus, this Gaussian density approach is a simple, yet useful way to model dependencies in high dimensional input spaces – something that could not be done in any principled way using KNN.