Learning from Data
Mixture Models

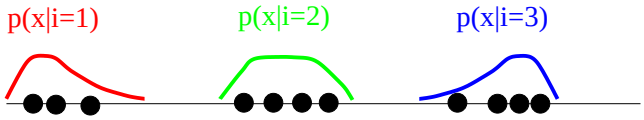
Course lecturer: Amos Storkey
a.storkey@ed.ac.uk
Course page : http://www.anc.ed.ac.uk/∼amos/lfd/

It is not uncommon for data to come in "clusters". For example, consider the problem of trying to learn the characteristics of handwritten digits, 0-9. Intuitively, examples of 2's look quite similar, as do examples of 3's. It is rare that a 2 will look like a 3. This suggests that all the examples of 2's form a cluster of points in the high dimensional space of the data, somewhat separated from clusters of representing other digits. Of course, if there were never any overlap of these clusters, life would be easy since classification would be straightforward.

Consider the case where we do not know any class labels and just have a dataset $\{\mathbf{x}^\mu, \mu = 1, \ldots, P\}$. Initially, we will think of using mixture models to cluster data in an unsupervised way – that is, simply to make a model of data which reflects our belief that the data has clusters. Why is this useful? Compression of data is one obvious example. Once we have a good model of data, we can always use it for compression.

1 Data Clusters

Consider the one dimensional data distribution $p(x)$ depicted below: It is



clear that the black dots, which represent the one dimensional data values are naturally clustered into three groups. Hence, a reasonable model of this data would be

$$p(x) = p(x|1)p(1) + p(x|2)p(2) + p(x|3)p(3) = \sum_{i=1}^{3} p(x|i)p(i) \tag{1.1}$$

where $p(x|i)$ is the model for the data in cluster i, and $\sum_i p(i) = 1$.

Mixture Models

If we believe that data lies around a number of clusters, we can model each cluster i by a distribution $p(\mathbf{x}|i)$. To complete the model for the whole dataset, we need to describe how much weight/probability mass to attach to each cluster, $p(i)$. The complete data set is described by the distribution

$$p(\mathbf{x}) = \sum_i p(\mathbf{x}|i)p(i) \tag{1.2}$$

where $p(i)$ is the probability that component/cluster i contributes to modelling the data.

2 Gaussian Mixture Models

In Gaussian mixture models we use the probability density functions,

$$p(\mathbf{x}|i) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma}_i)}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right) \tag{2.1}$$

The idea is to place 'blobs' of probability mass in the space to cover the data well, see fig(1).

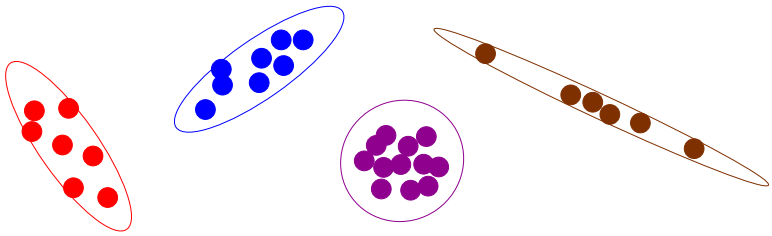


Figure 1: Gaussian Mixture Models place blobs of probability mass in the space. Here we have 4 mixture components in a 3 dimensional space. Each mixture has a different covariance matrix and mean.

2.1 Training Algorithm

Maximum likelihood training using the EM algorithm is one way to train mixture models. The derivation is straightforward but beyond the scope of these lectures. However, the resulting algorithm for optimising this model is straightforward and intuitive. The following algorithm is used to fit a mixture of M Gaussians (that is, to find their means $\boldsymbol{\mu}_i$ and covariances $\boldsymbol{\Sigma}_i$, $i = 1, \ldots, M$) to data $\mathbf{x}^n, n = 1, \ldots P$

1. Initialise the means and covariance matrices to some values. One good choice is to set the means to a random subset of the training points, and to set the covariance matrices to be large multiples of the identity matrix. Set the mixing coefficients $p(i)$ to be $1/M$.

2. Update the means and covariance matrices according to

$$\boldsymbol{\mu}_i^{new} = \frac{\sum_{n=1}^{P} p(i|\mathbf{x}^n)\mathbf{x}^n}{\sum_{n=1}^{P} p(i|\mathbf{x}^n)} \tag{2.2}$$

$$\boldsymbol{\Sigma}_i^{new} = \frac{\sum_{n=1}^{P} p(i|\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_i)(\mathbf{x}^n - \boldsymbol{\mu}_i)^T}{\sum_{n=1}^{P} p(i|\mathbf{x}^n)} \tag{2.3}$$

$$p(i)^{new} = \frac{1}{P}\sum_{n=1}^{P} p(i|\mathbf{x}^n) \tag{2.4}$$

where

$$p(i|\mathbf{x}^n) = \frac{p(\mathbf{x}^n|i)p(i)}{\sum_{i=1}^{M} p(\mathbf{x}^n|i)p(i)} \tag{2.5}$$

In all the expressions above, we use equation (2.1) to compute the probabilities $p(\mathbf{x}^n|i)$. The probabilities $p(i|\mathbf{x}^n)$ are called the 'responsibilities', since they effectively measure how responsible the i^{th} mixture is in representing the data point $\mathbf{x}^n$. Step 2 above is repeated until convergence.

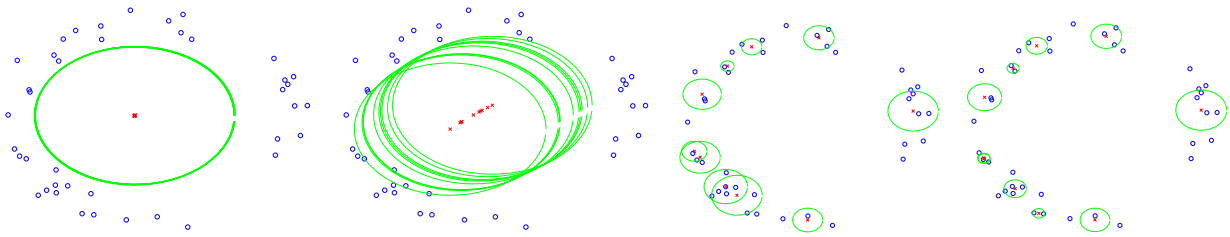
These equations are intuitive since, for example, equation (2.2) says essentially the following: go through the dataset, and find which datapoints are close to mixture component i (that is, those data points for which the responsibilities $p(i|\mathbf{x}^n)$ are high) – the mean of cluster i is then the mean of those data points for which cluster i is responsible.

If we wish that each Gaussian blob is isotropic, (covariance matrix is a multiple of the identity), then we only need to estimate the variance of the Gaussian. In this case, we simply replace equation (2.3) with

$$(\sigma_i^2)^{new} = \frac{\sum_{n=1}^{P} p(i|\mathbf{x}^n)(\mathbf{x}^n - \boldsymbol{\mu}_i)^T(\mathbf{x}^n - \boldsymbol{\mu}_i)}{\sum_{n=1}^{M} p(i|\mathbf{x}^n)} \tag{2.6}$$

Warning! There is a problem with the above algorithm – it has a tendency to make very narrow width Gaussians around single data points. It is customary to prevent this by disallowing very small values of σ_i^2. For the case of anisotropic Gaussians, a reasonable criterion would be to stop updating the covariance $\boldsymbol{\Sigma}_i$ if any of its eigenvalues go below a set threshold.

An example An example of this algorithm in action is shown in fig(2), which fits a mixture of 10 isotropic Gaussians to a set of two dimensional data. The data was fitted using the code below. Note that the code is not heavily vectorised for readability.



(a) 1 iteration (b) 20 iterations (c) 50 iterations (d) 100 iterations

Figure 2: Training a mixture of 10 Gaussians (a) If we start with large variances for the Gaussians, even after one iteration, the Gaussians are centred close to the mean of the data. (b) The Gaussians begin to separate (c) One by one, the Gaussians move towards appropriate parts of the data (d) The final converged solution. Note that here, the Gaussians were constrained so that the variances could not go below 0.01.

```
% demo for fitting mixture of isotropic Gaussians

%make an annulus of data :
l = 0.2; r1 = 0.5; for r = 1:50
  rad = r1 + rand*l;  theta = rand*2*pi;  X(1,r) = rad*cos(theta); X(2,r) = rad*sin(theta);
end

h = 5;          % number of mixtures
d = size(X,1); % dimension of the space
n = size(X,2); % number of training patterns
Smin = 0.001;  % minimum variance of Gaussians

r = randperm(n); M = X(:,r(1:h)); % initialise the centres to random datapoints
S = 100*ones(1,h); % initialise the variances to be large
P = ones(1,h)./h;  % intialise the component probilities to be uniform

for its = 1:150 % number of iterations

  for i = 1:h
    for k = 1:n
      v = X(:,k) - M(:,i);
      Q(k,i) = exp(-0.5*(v'*v)/S(i)).*P(i)./sqrt((S(i))^d);
    end
  end
  su = sum(Q,2);
  for k =1:n
    Q(k,:) = Q(k,:)./su(k); % responsibilities p(i|x^n)
  end

  for i = 1:h % now get the new parameters for each component
    N(i) = sum(Q(:,i));
    Mnew(:,i) = X*Q(:,i)./N(i);
    Snew(i) = (1/d)*sum( (X - repmat(Mnew(:,i),1,n)).^2 )*Q(:,i)./N(i);
    if Snew(i) < Smin % don't decrease the variance below Smin
      Snew(i) = Smin;
    end
  end

  Pnew = N; Pnew = Pnew./sum(Pnew);
  S = Snew; M = Mnew; P = Pnew; % update the parameters
end
```

3 K Means

A non-probabilistic limit of fitting Gaussian mixtures to data is given by the K means algorithm, in which we simply represent an original set of P datapoints by K points.

3.1 The algorithm

1. Initialise the centres $\boldsymbol{\mu}_i$ to K randomly chosen datapoints.
2. For each cluster mean, j, find all the $\mathbf{x}$ for which cluster j is the nearest cluster. Call this set of points S_j. Let N_j be the number of datapoints

in set S_j.

3. Assign

$$\boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in S_j} \mathbf{x} \tag{3.1}$$

We then iterate steps 2 and 3 above until some convergence criterion.

The code below implements this algorithm.

```
% demo for K Means
x = [randn(2,50) 5+randn(2,50) (repmat([-4 4]',1,50)+randn(2,50))]; % 150 2-dim datapoints

K = 3; % number of clusters
r = randperm(size(x,2));
m(:,1:K) = x(:,r(1:K)); % initialise the clusters to K randomly chosen datapoints
mold =m;

for its = 1: 100 % maximum number of iterations

  for p = 1:size(x,2)   % calculate the distances (this could be vectorised)
    for k = 1:K
      v = x(:,p) - m(:,k); d(k,p) = v'*v;
    end
  end

  [a,b]=min(d);   % find the nearest centres

  for k = 1:K
    if length(find(b==k))>0
      m(:,k) = mean(x(:,find(b==k))')';
    end
  end

  if mean(sum( (m-mold).^2)) < 0.001; break; end; mold =m; % termination criterion
end

cla; plot(x(1,:),x(2,:),'.'); hold on;
plot(m(1,:),m(2,:),'rx','markersize',15);
```

An example is given in fig(3) in which we represented 150 datapoints using 3 clusters.

Note that the Kmeans algorithm can be dervived as the limit $\sigma \to 0$ for fitting isotropic Gaussian mixture components.

3.2 Uses of K Means

The K means algorithm, despite its simplicity is very useful. Firstly, it converges extremely quickly and often gives a reasonable clustering of the data, provided that the centres are initialised reasonably (using the above procedure for example). We can use the centres we found as positions in which to place basis function centres in the linear parametric models chapter.

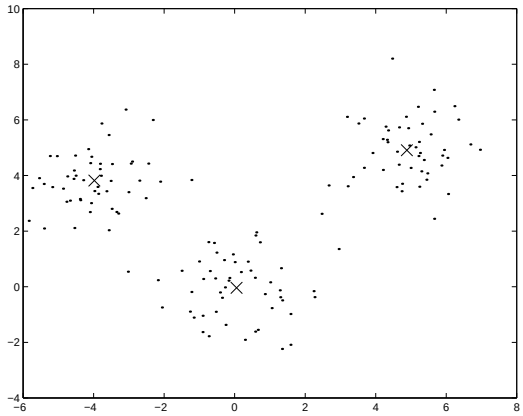


Figure 3: Result of fitting $K = 3$ means to 150 two dimensional datapoints. The means are plotted as crosses.

4 Classification using Mixture Models

One popular use of mixture models is in classification. Consider the case in which we have two classes, 1 and 2. We can fit a Gaussian Mixture model to each class. That is, we could fit a mixture model to the data from class 1 :

$$p(\mathbf{x}|c=1) = \sum_{k=1}^{K} p(\mathbf{x}|k, c=1)p(k|c=1) \tag{4.1}$$

and a mixture model to the data from class 2. (One could use a different number of mixture components for the different classes, although in practice, one might need to avoid overfitting one class more than the other. Using the same number of mixture components for both classes avoids this problem.)

$$p(\mathbf{x}|c=2) = \sum_{k=1}^{K} p(\mathbf{x}|k, c=2)p(k|c=2) \tag{4.2}$$

So that each class has its own set of mixture model parameters. We can then form a classifier by using Bayes rule :

$$p(c=i|\mathbf{x}) = \frac{p(\mathbf{x}|c=i)p(c=i)}{p(\mathbf{x})} \tag{4.3}$$

Only the numerator is important in determining the classification since the denominator is the same for the case of $p(c=2|\mathbf{x})$. This is a more powerful approach than our original approach in which we fitted a single Gaussian to each digit class. Using more Gaussians enables us to get a better model for how the data in each class is distributed and this will usually result in a better classifier.