1 Unsupervised learning A

In the previous learning problems we had training examples with feature vectors $\mathbf{x}$ and labels $\mathbf{y}$. In this chapter we discuss unsupervised learning problems in which no labels are given. Training on unlabeled examples restricts the type of learning that can be done, but unsupervised learning has important applications and even can be an important part in aiding supervised learning. Unsupervised does not mean that the learning is not guided at all; the learning follows specific principles that are used to organize the system based on the characteristics provided by the data. We will discuss several examples in this chapter.

1.1 K-means clustering

The first example is **data clustering**. In this problem domain we are given unlabelled data described by a set of features and asked to put them into k categories. In the first example of such clustering we categorize the data by proximity to a mean value. That is, we assume a model that specifies a mean feature value of the data and classifies the data based on the proximity to the mean value. Of course, we do not know this mean value for each class. The idea of the following algorithm is that we start with a guess for this mean value and label the data accordingly. We then use the labeled data from this hypothesis to improve the model by calculating a new mean value, and repeat these steps until convergence is reached. Such an algorithm usually converges quickly to a stable solution. More formally, given a training set of data points $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ and a hypothesis of the number of clusters, k, the k-means clustering algorithm is shown in Table 1.1. An example is shown in Figure 1.1 and the corresponding program is shown is Table 1.2.

Table 1.1 k-means clustering algorithm

```
1. Initialize the means $\mu_1, ...\mu_k$ randomly.
2. Repeat until convergence: {
        Model prediction:
            For each data point $i$, classify data to class with closest mean
                $c^{(i)} = arg\min_j ||x^{(i)} - \mu_j||$
        Model refinement:
            Calculate new means for each class
                $\mu_j = \frac{1 \;\; 1(c^{(i)}=j)x^{(i)}}{1 \;\; 1(c^{(i)}=j)}$
} convergence
```

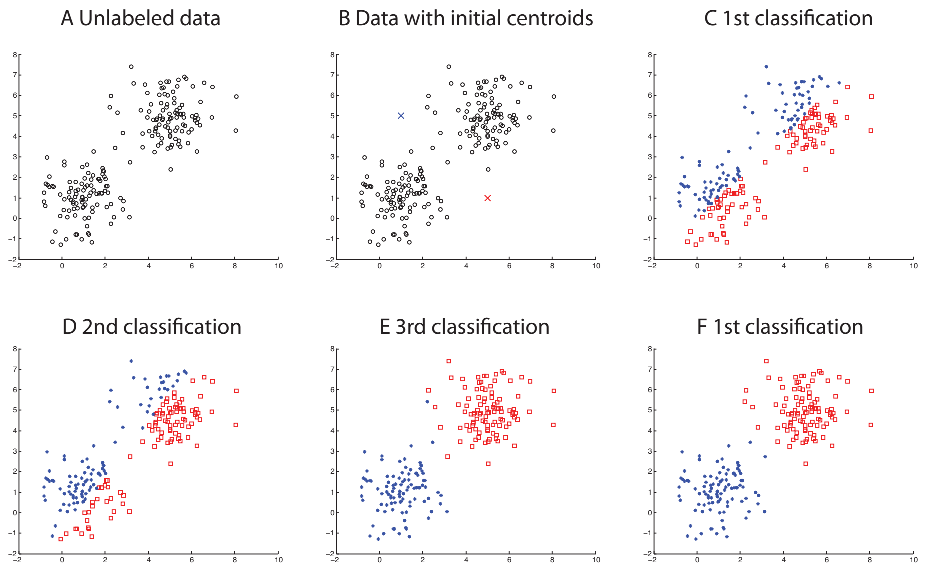

Fig. 1.1 Example of k-means clustering with two clusters.

1.2 Mixture of Gaussian and the EM algorithm

We have previously discussed generative models where we assumed specific models for the in-class distributions. In particular, we have discussed linear discriminant analysis where we had labelled data and assumed that each class is Gaussian distributed. Here we assume that we have k Gaussian classes, where each class is chosen randomly from a multinominal distribution,

$$p(z^{(i)} = j) \propto \text{multinomial}(\Phi_j) \tag{1.1}$$

$$p(x^{(i)}|z^{(i)} = j) \propto N(\mu_j, \Sigma_j) \tag{1.2}$$

This is called a **Gaussian Mixture Model**. The corresponding log-likelihood function is

$$l(\Phi, \mu, \sigma) = \sum_{i=1}^{m} \log \sum_{z^{(i)}=1}^{k} p(x^{(i)}|z^{(i)}; \mu, \Sigma) p(z^{(i)}; \Phi). \tag{1.3}$$

Since we consider here unsupervised learning in which we are given data without labels, the random variables $z^{(i)}$ are latent variables. This makes the problem hard. If we would be give the class membership, than the log-likelihood would be

$$l(\Phi, \mu, \sigma) = \sum_{i=1}^{m} \log p(x^{(i)}; z^{(i)}, \mu, \Sigma), \tag{1.4}$$

which we could use to calculate the maximum likelihood estimates of the parameter (see equations **??**-**??**),

$$\phi_k = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}(z^{(i)} = j) \tag{1.5}$$

Table 1.2 Program to demonstrate k-mean clustering on Gaussian Data.

```
clear; clf; hold on;

%% training data generation; 2 classes, each gaussian with mean (1,1) and (2,2) and diagonal
n0=100; %number of points in class 0
n1=100; %number of points in class 1

x=[1+randn(n0,1), 1+randn(n0,1); ...
   5+randn(n1,1), 5+randn(n1,1)];

plot(x(:,1),x(:,2),'ko'); % plotting points
mu1=[5 1]; mu2=[1 5]; % initial two centers
while(true) waitforbuttonpress;

plot(mu1(1),mu1(2),'rx','MarkerSize',12)
plot(mu2(1),mu2(2),'bx','MarkerSize',12)

for i=1:n0+n1;
    d1=(x(i,1)-mu1(1))^2+(x(i,2)-mu1(2))^2;
    d2=(x(i,1)-mu2(1))^2+(x(i,2)-mu2(2))^2;
    y(i)=(d1<d2)*1;
end

waitforbuttonpress;
x1=x(y>0.5,:);
x2=x(y<0.5,:);

clf; hold on;
plot(x1(:,1),x1(:,2),'rs');
plot(x2(:,1),x2(:,2),'b*');
mu1=mean(x1);
mu2=mean(x2);

end
```

$$\mu_k = \frac{\sum_{i=1}^m \mathbb{1}(z^{(i)} = j)\mathbf{x}^{(i)}}{\sum_{i=1}^m \mathbb{1}(z^{(i)} = j)} \tag{1.6}$$

$$\Sigma_k = \frac{\sum_{i=1}^m \mathbb{1}(z^{(i)} = j)(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^m \mathbb{1}(y^{(i)} = k)}. \tag{1.7}$$

While we do not know the class labels, we can follow a similar strategy to the k-means clustering algorithm and just propose some labels and use them to estimate the parameters. We can then use the new estimate of the distributions to find better labels for the data, and repeat this procedure until a stable configuration is reached. In general, this strategy is called the **EM algorithm** for expectation-maximization. The algorithm is outlined in Fig.1.2. In this version we do not hard classify the data into one or another class, but we take a more soft classification approach that considers the probability estimate of a data point belonging to each class.

```
1. Initialize parameters $\phi, \mu, \Sigma$ randomly.
2. Repeat until convergence: {
        E step:
            For each data point $i$ and class $j$ (soft-)classify data as
                $w_j^{(i)} = p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$
        M step:
            Update the parameters according to
                $\phi_j = \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}$
                $\mu_j = \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}$
                $\Sigma_k = \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} \mathbb{1} w_j^{(i)}}$.
} convergence
```

Fig. 1.2 EM algorithm

An example is shown in Fig. 1.3. In this simple world, data are generated with equal likelihood from two Gaussian distributions, one with mean $\mu_1 = -1$ and standard deviation $\sigma_1 = 2$, the other with mean $\mu_2 = 4$ and standard deviation $\sigma_2 = 0.5$. These two distributions are illustrated in Fig. 1.3A with dashed lines. Let us assume that we know that the world consists only of data from two Gaussian distributions with equal likelihood, but that we do not know the specific realizations (parameters) of these distributions. The pre-knowledge of two Gaussian distributions encodes a specific **hypothesis** which makes up this **heuristic model**. In this simple example, we have chosen the heuristics to match the actual data-generating system (world), that is, we have explicitly used some knowledge of the world.

Learning the parameters of the two Gaussians would be easy if we had access to the information about which data point was produced by which Gaussian, that is, which cause produced the specific examples. Unfortunately, we can only observe the data without a teacher label that could supervise the learning. We choose therefore a self-supervised strategy, which repeats the following two steps until convergence:

E-step: We make assumptions of training labels from the current model (expectation step)

M-step: use this hypothesis to update the parameters of the model to maximize the probability of the observations (maximization step).

Since we do not know appropriate parameters yet, we just choose some arbitrary values as the starting point. In the example shown in Fig. 1.3A we used $\mu_1 = 2$, $\mu_2 = -2$, $\sigma_1 = \sigma_2 = 1$. These distributions are shown with solid lines. Comparing the generated data with the environmental data corresponds to hypothesis testing.

The results are not yet very satisfactory, but we can use the generative model to express our **expectation** of the data. Specifically, we can assign each data point to the class which produces the larger probability within the current world model. Thus, we are using our specific hypothesis here as a **recognition model**. In the example we can

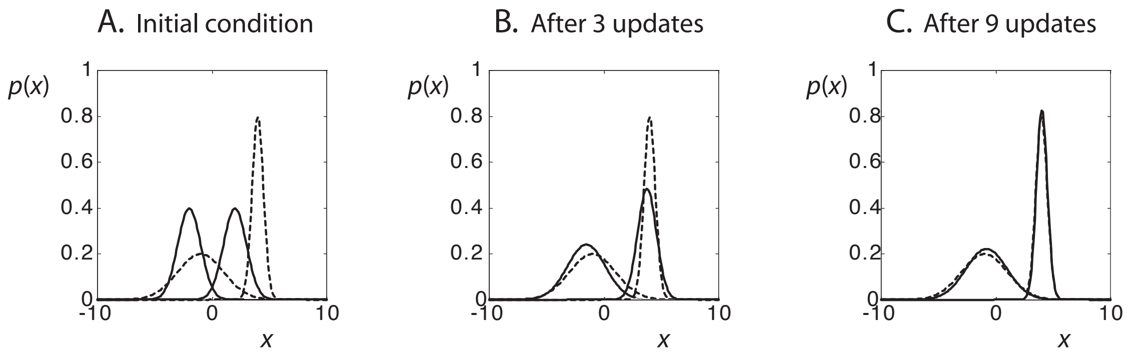

Fig. 1.3 Example of the expectation maximization (EM) algorithm for a world model with two Gaussian distributions. The Gaussian distributions of the world data (input data) are shown with dashed lines. (A) The generative model, shown with solid lines, is initialized with arbitrary parameters. In the EM algorithm, the unlabelled input data are labelled with a recognition model, which is, in this example, the inverse of the generative model. These labelled data are then used for parameter estimation of the generative model. The results of learning are shown in (B) after three iterations, and in (C) after nine iterations .

use Bayes' rule to invert the generative model into a recognition model as detailed in the simulation section below. If this inversion is not possible, then we can introduce a separate recognition model, Q, to approximate the inverse of the generative model. Such a recognition model can be learned with similar methods and interleaved with the generative model.

Of course, the recognition with the recognition model early in learning is not expected to be exact, but estimation of new parameters from the recognized data in the M-step to maximize the expectation can be expected to be better than the model with the initial arbitrary values. The new model can then be compared to the data again and, when necessary, be used to generate new expectations from which the model is refined. This procedure is known as the **expectation maximization** (EM) algorithm. The distributions after three and nine such iterations, where we have chosen new data points in each iteration, are shown in Figs 1.3B and C.

Simulation

The program used to produce Fig. 1.3 is shown in Table 1.3. The vector x_0, defined in Line 2, is used to plot the distributions later in the program. The arbitrary random initial conditions of the distribution parameters are set in Line 3. Line 4 defines an **inline function** of a properly normalized Gaussian since this function is used several times in the program. An inline function is an alternative to writing a separate function file. It defines the name of the functions, followed by a list of parameters and an expression, as shown in Line 4. The rest of the program consist of an infinite loop produced with the statement `while 1`, which is always true. The program has thus to be interrupted by closing the figure window or with the interruption command `Ctrl C`. In Lines 7–12, we produce plots of the real-world models (dotted lines) and the model distributions (plotted with a red and a blue curve when running the program).

Table 1.3 Program ExpectationMaximization.m

```
 %% 1d example EM algorithm
clear; hold on; x0=?10:0.1:10;
var1=1; var2=1; mu1=?2; mu2=2;
normal= @(x,mu,var) exp(?(x?mu).^2/(2?var))/sqrt(2?pi?var);
while 1
    %%plot distribution
    clf; hold on; ylim([0 1]);
    plot(x0, normal(x0,?1,4),k: );
    plot(x0,normal(x0,4 ,.25) , k: );
    plot(x0, normal(x0,mu1,var1),r);
    plot(x0, normal(x0,mu2,var2),b);
    waitforbuttonpress ;
    %% data
    x=[2?randn(50,1)?1;0.5?randn(50 ,1)+4;];
    %% recogintion
    p1=normal(x,mu1,var1);
    p2=normal(x,mu2,var2);
    nrm=p1+p2 ; p1=p1./nrm; p1=p1./sum(p1); p2=p2./nrm; p2=p2./sum(p2);
    %% maximization
    mu1 =x?p1; var1=(x?mu1).^2 ? p1;
    mu2 =x?p2; var2=(x?mu2).^2 ? p2;
    %equivalently :
    %p1=p1./nrm; p2=p2./nrm;
    %mu1=sum(x.?p1)/sum(p1); var1=sum(p1.?(x?mu1).^2)./sum(p1);
    %mu2=sum(x.?p2)/sum(p2); var2=sum(p2.?(x?mu2).^2) ./sum(p2);
end
```

The command `waitforbuttonpress` is used in Line 12 so that we can see the results after each iteration.

In Line 14 we produce new random data in each iteration. Recognition of this data is done in Line 16 by inverting the generative model using Bayes' formula,

$$P(c|\mathbf{x};G) = \frac{P(\mathbf{x}|c;G)P(c;G)}{P(\mathbf{x};G)}. \tag{1.8}$$

In this specific example, we know that the data are equally distributed from each Gaussian so that the **prior distribution over causes**, $P(c;G)$ is $1/2$ for each cause. Also, the **marginal distribution of data** is equally distributed, so that we can ignore this normalizing factor. The recognition model in Line 16 uses the Bayesian decision criterion, in which the data point is assigned to the cause with a larger **recognition distribution**, $P(c|\mathbf{x};G)$. Using the labels of the data generated by the recognition model, we can then use the data to obtain new estimates of the parameters for each Gaussian in Lines 17–21.

Note that when testing the system for a long time, it can happen that one of the distributions is dominating the recognition model so that only data from one distribution are generated. The model of one Gaussian would then be **explaining away** data from the other cause. More practical solutions must take such factors into account.