6 Generative Models

6.1 Modelling classes

In the previous sections we have introduced the idea that understanding the world should be based on a model of the world in a probabilistic sense. That is, building a good recognition system means estimating a large density function about labels in of objects from sensory data. What we have done so far is to used classification models as a discriminative recognition model that take feature values $\mathbf{x}$ and make a prediction of an output (label) y. In the probabilistic formulation, the models where formulated as parameterized functions that represent the conditional probability $p(y|\mathbf{x};\theta)$. A model that discriminates between classes based on the feature values is called a **discriminative models**. Building a discriminative model directly from example data can be a daunting task as we have to learn how each item is distinguished from every other possible item. We have mainly used simple models in low dimensions to illustrate the ideas, and many real world problems have much larger dimensions.

A different strategy, which seems much more resembling human learning, is to learn first about the nature of specific classes and then use this knowledge when faced with a classification task. Learning about classes and representing them in neural code is a form of **representational learning**. This is an important part of modern learning theory and we will encounter this concept several times. Thus representational models can be even be used to 'generate' examples of the class objects, and this models are therefore also called **generative models** of individual classes,

$$p(\mathbf{x}|y;\theta) \tag{6.1}$$

With generative models we can use an inference engine to use this knowledge in diverse tasks such as classification. For example, we might first learn about chairs, and independently about tables, and when we are shown pictures with different furnitures we can draw on this knowledge to classify them.

In order to use probabilistic generative model as in eq. 6.1 for classification, we need to ask how we can combine the knowledge about the different classes to do classification. Of course, the answer is provided by Bayes' theorem, so that in this case we can use the rules of probability theory as inference engine. In order to make a discriminative model from the generative models, we need to the **class priors**, e.g. what the relative frequencies of the classes is. We can then calculate the probability that an item with features $\mathbf{x}$ belong to a class y as

$$p(y|\mathbf{x};\theta) = \frac{p(\mathbf{x}|y;\theta)p(y)}{p(x)}. \tag{6.2}$$

A decision can be made directly based on this conditional probability. The Bayesian decision criterion of predicting the class with the largest posterior probability is

$$\arg\max_y p(y|\mathbf{x};\theta) = \arg\max_y \frac{p(\mathbf{x}|y;\theta)p(y)}{p(x)} \tag{6.3}$$

$$= \arg\max_y p(\mathbf{x}|y;\theta)p(y), \tag{6.4}$$

where we have used the fact that the denominator does not depend on y and can hence be ignores. In the case of binary classification, this reads:

$$\arg\max_y p(y|\mathbf{x};\theta) = \arg\max_y (p(\mathbf{x}|y=0;\theta)p(y=0), p(\mathbf{x}|y=1;\theta)p(y=1). \tag{6.5}$$

While using generative models for classification seem to be much more elaborate, we will see later that there are several arguments which make generative models attractive for machine learning. To start with, it seems much easier and efficient to learn to generalize from similar objects than to learn from possibly difficult discrimination examples.

6.2 Supervised Gaussian model: Discriminant analysis

Classification with generative models have been used for some time. We will be discussing an example here which is related to a method called **linear discriminant analysis** and goes back to a paper by R. Fisher in 1936. In the following example we consider that there are k classes, and we first assume that each class has members which are Gaussian distribution over the n feature value. An example for $n = 2$ is shown in Fig.6.1A.

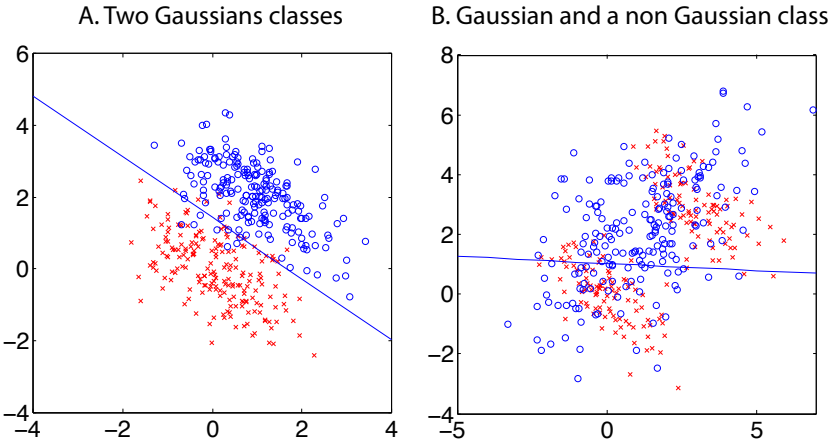

Fig. 6.1 Linear Discriminant analysis on a two class problem with different class distributions.

Each of the classes have a certain class prior

$$p(y=k) = \phi_k, \tag{6.6}$$

and each class itself is multivariate Gaussian distributed, generally with different means, μ_k and variances, Σ_k,

$$p(\mathbf{x}|y=k) = \frac{1}{\sqrt{2\pi}^n \sqrt{|\Sigma_0|}} e^{-\frac{1}{2}(\mathbf{x}-\mu_k)^T \Sigma_k^{-1}(\mathbf{x}-\mu_k)} \tag{6.7}$$

$$\tag{6.8}$$

Since we have supervised data with examples for each class, we can use maximum likelihood estimation to estimate the most likely values for the parameters $\theta = (\phi_k, \mu_k, \Sigma_k)$. For the class priors, this is simply the relative frequency of the training data,

$$\phi_k = \frac{|K|}{m} \tag{6.9}$$

where K is the set of examples of class k and $|K|$ is the number of examples is this set. Thus we estimated the parameter ϕ_k with the maximum likelihood for this Bernoulli random variable, and we omitted the "hat" to indicate that it is an estimate since this should now be clear from the context. The estimates of the means and variances within each class are given by the corresponding maximum likelihood estimates for the Gaussian parameters,

$$\mu_k = \frac{1}{|K|} \sum_{i \in K} \mathbf{x}^{(i)} \tag{6.10}$$

$$\Sigma_k = \frac{1}{|K|} \sum_{i \in K} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T. \tag{6.11}$$

With these estimates, we can calculate the optimal (in a Bayesian sense) decision rule, $G(x;\theta)$, as a function of $\mathbf{x}$ with parameters θ, namely

$$G(x) = arg\max_k p(y=k|\mathbf{x}) \tag{6.12}$$

$$= arg\max_k [p(\mathbf{x}|y=k;\theta)p(y=k)] \tag{6.13}$$

$$= arg\max_k [log(p(\mathbf{x}|y=k;\theta)p(y=k))] \tag{6.14}$$

$$= arg\max_k [-log(\sqrt{2\pi}^n \sqrt{|\Sigma_0|}) - \frac{1}{2}(\mathbf{x}-\mu_k)^T \Sigma_k^{-1}(\mathbf{x}-\mu_k) + log(\phi_k)] \tag{6.15}$$

$$= arg\max_k [-\frac{1}{2}\mathbf{x}^T \Sigma_k^{-1}\mathbf{x} - \frac{1}{2}\mu_k^T \Sigma_k^{-1}\mu_k + \mathbf{x}^T \Sigma_k^{-1}\mu_k + log(\phi_k)], \tag{6.16}$$

since the first term in equation 6.15 does not depend on k and we can multiply out the other terms. With the maximum likelihood estimates of the parameters, we have all we need to make this decision.

In order to calculate the decision boundary between classes l and k, we make the common additional assumption that the covariance matrices of the classes are the same,

$$\Sigma_k =: \Sigma. \tag{6.17}$$

The decision point between the two classes with equal class priors is then given by the point where the probabilities for the two classes (eq.6.16) is the same. This gives

$$log(\frac{\phi_k}{\phi_l}) - \frac{1}{2}(\mu_k - \mu_l)^T \Sigma^{-1}(\mu_k + \mu_l) + \mathbf{x}\Sigma^{-1}(\mu_k - \mu_l) = 0. \tag{6.18}$$

The first two terms do not depend on x and can be summarized as constant $\mathbf{a}$. We can also introduce the vector

$$\mathbf{w} = \Sigma^{-1}(\mu_k - \mu_l). \tag{6.19}$$

With these simplifying notations is it easy to see that this decision boundary is a linear,

$$\mathbf{a} + \mathbf{w}\mathbf{x} = 0, \tag{6.20}$$

and this method with the Gaussian class distributions with equal variances is called **Linear Discriminant Analysis (LDA)**. The vector $\mathbf{w}$ is perpendicular to the decision surface. Examples are shown in Figure 6.1. If we do not make the assumption of equal variances of the classes, then we have a quadratic equation for the decision boundary, and the method is then called **Quadratic Discriminant Analysis (QDA)**. With the assumptions of LDA, we can calculate the contrastive model directly using Bayes rule.

$$p(y = k|\mathbf{x};\theta) = \frac{\phi_k \frac{1}{\sqrt{2\pi}^n\sqrt{|\Sigma|}} e^{-\frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}-\mu_k)}}{\phi_k \frac{1}{\sqrt{2\pi}^n\sqrt{|\Sigma|}} e^{-\frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}-\mu_k)} + \phi_l \frac{1}{\sqrt{2\pi}^n\sqrt{|\Sigma|}} e^{-\frac{1}{2}(\mathbf{x}-\mu_l)^T\Sigma_l^{-1}(\mathbf{x}-\mu_l)}} \tag{6.21}$$

$$= \frac{1}{1 + \frac{\phi_l}{\phi_k} exp^{-\theta^T x}}, \tag{6.22}$$

where θ is an appropriate function of the parameters μ_k, μ_l, and Σ. Thus, the contrastive model is equivalent to logistic regression discussed in the previous chapter, although we use different parametrizations and the two methods will therefore usual give different results on specific data sets. So which method should be used? In LDA we made the assumption that each class is Gaussian distributed. If this is the case, then LDA is the best method we can use. Discriminant analysis is also popular since it is easy to apply and often works still well even when the classes are not strictly Gaussian. However, as can be seen in Figure 6.1B, it can produce quite bad results if the data are multimodal distributed. Logistic regression is somewhat more general since it does not make the assumption that the class distributions are Gaussian. However, as ;long as we consider only linear models, logistic regression would have also problems with the data shown in Figure 6.1B.

Finally, we should note that Fisher's original method was slightly more general than the examples discussed here since he did not assume Gaussian distributions. Instead considered within-class variances compared to between-class variances, something which resembles a signal-to-noise ratio. In **Fisher discriminant analysis (FDA)**, the separating hyperplane is defined as

$$\mathbf{w} = (\Sigma_k + \Sigma_l)^{-1}(\mu_k - \mu_l). \tag{6.23}$$

which is the same as in LDA in the case of equal covariance matrices.

6.3 Unsupervised example: K-means clustering

In the previous learning problems we had training examples with feature vectors $\mathbf{x}$ and labels $\mathbf{y}$. In now discuss a form of unsupervised learning problems in which no

Table 6.1 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

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.

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 6.1. An example is shown in Figure 6.2 and the corresponding program is shown below.

```
from pylab import *
n=100 # number of training data in each class
x = randn(2*n,2);  x[:n,:]+=1; x[n:,:]+=5  # with mean (1,1) and (5,5)
plot(x[:,0],x[:,1],'ko') # plot points
mu1=[5,1]; mu2=[1,5] # initial centers (arbitrary)


plot(mu1[0],mu1[1],'rx',markersize=12)
plot(mu2[0],mu2[1],'bx',markersize=12)
draw()

# repeat this block
y = ((x-mu1)**2).sum(1) < ((x-mu2)**2).sum(1) # expectation
x1=x[y>0.5];   x2=x[y<0.5];
plot(x1[:,0],x1[:,1],'rs');
plot(x2[:,0],x2[:,1],'b*');
```

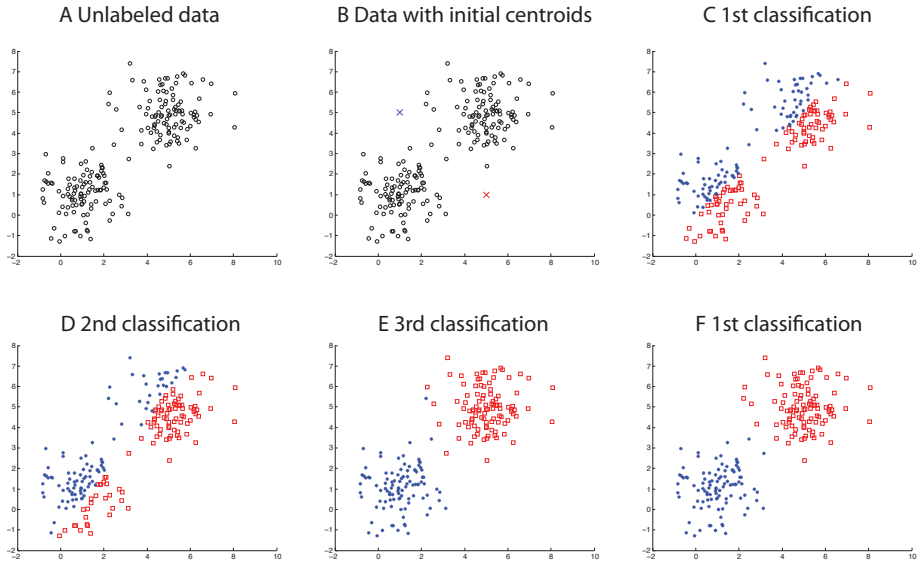

Fig. 6.2 Example of k-means clustering with two clusters.

```
mu1=x1.mean(0);    mu2=x2.mean(0); # maximization
plot(mu1[0],mu1[1],'kx',markersize=12)
plot(mu2[0],mu2[1],'kx',markersize=12)
draw()
```

6.4 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,

$$p(z^{(i)} = j) \propto \text{multinomial}(\Phi_j) \tag{6.24}$$

$$p(x^{(i)}|z^{(i)} = j) \propto N(\mu_j, \Sigma_j) \tag{6.25}$$

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

$$l(\Phi, \mu, \sigma) = \sum_{i=1}^{m} \log \sum_{z^{(i)}=1}^{k} p(x^{(i)}|z^{(i)}; \mu, \Sigma) p(z^{(i)}; \Phi). \tag{6.26}$$

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

$$l(\Phi, \mu, \sigma) = \sum_{i=1}^{m} \log p(x^{(i)}; z^{(i)}, \mu, \Sigma), \tag{6.27}$$

which we could use to calculate the maximum likelihood estimates of the parameter (see equations 6.9-6.11),

$$\phi_k = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}(z^{(i)} = j) \tag{6.28}$$

$$\mu_k = \frac{\sum_{i=1}^{m} \mathbb{1}(z^{(i)} = j)\mathbf{x}^{(i)}}{\sum_{i=1}^{m} \mathbb{1}(z^{(i)} = j)} \tag{6.29}$$

$$\Sigma_k = \frac{\sum_{i=1}^{m} \mathbb{1}(z^{(i)} = j)(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} \mathbb{1}(y^{(i)} = k)}. \tag{6.30}$$

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.6.3. 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 ϕ, μ, Σ 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

$\phi_j = \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}$

$\mu_j = \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}$

$\Sigma_k = \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}$.

$\}$ convergence

Fig. 6.3 EM algorithm

An example is shown in Fig. 6.4. 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. 6.4A 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.

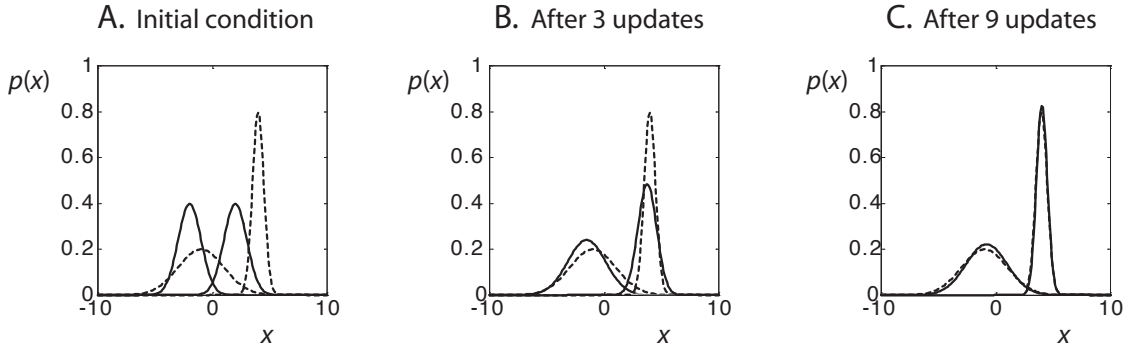

Fig. 6.4 Example of the expectation maximization (EM) algorithm for a world model with two Gaussian distributions. The Gaussian distributions of the world data (input data) are shown with dashed lines. (A) The generative model, shown with solid lines, is initialized with arbitrary parameters. In the 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 .

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. 6.4A 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 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. The distributions after three and nine such iterations, where we have chosen new data points in each iteration, are shown in Figs 6.4B and C.