Chapter 18

SEGMENTATION AND FITTING USING PROBABILISTIC METHODS

All the segmentation algorithms we described in the previous chapter involve essentially local models of similarity. Even though some algorithms attempt to build clusters that are good globally, the underlying *model* of similarity compares individual pixels. Furthermore, none of these algorithms involved an explicit probabilistic model of how measurements differed from the underlying abstraction that we are seeking.

We shall now look at explicitly probabilistic methods for segmentation. These methods attempt to explain data using models that are global. These models will attempt to explain a large collection of data with a small number of parameters. For example, we might take a set of tokens and fit a line to them; or take a pair of images and attempt to fit a parametric set of motion vectors that explain how pixels move from one to the other.

The first important concept is that of a missing data problem. In many segmentation problems, there are several possible sources of data (for example, a token might come from a line, or from noise); if we knew from which source the data had come (i.e. whether it came from the line, or from noise), the segmentation problem would be easy. In section 18.1, we deal with a number of segmentation problems by phrasing them in this form, and then using a general algorithm for missing data problems.

We then set up the problem of fitting lines to a set of tokens as a quite general example of a probabilistic fitting problem (section 17.2), and discuss methods for attacking this problem. This leads us to situations where occasional data items are hugely misleading, and we discuss methods for making fitting algorithms robust (section 18.2). We sketch how to generalize our line fitting techniques to handle curve fitting problems in section 17.3. Finally, we discuss methods for determining how many elements (lines, curves, segments, etc.) to fit to a particular data set (section 18.3).

18.1 Missing Data Problems, Fitting and Segmentation

Segmentation can rather naturally be phrased as a missing data problem. In this approach, there are a series of different possible sources of pixel values, and the missing data is the source from which the pixel was drawn. A number of other interesting problems can be phrased in this way, too, and there is a standard, quite simple, algorithm for this problem.

18.1.1 Missing Data Problems

It is quite common to encounter problems which contain missing data. There are two contexts: in the first, some terms in a data vector are missing for some instances and present for other (perhaps someone responding to a survey was embarrassed by a question); in the second, which is far more common in our applications, an inference problem can be made very much simpler by rewriting it using some variables whose values are unknown. We will demonstrate this method and appropriate algorithms with two examples.

Example: Owls and Rats

In a study area, there are g different species of rat. A rat of the l'th species appears in an owl's diet with probability π_l It is hard to observe owls eating rats. Instead, owl pellets — the indigestible bits of rat, regurgitated by owls — are found and rat skulls in the pellets are measured. These measurements form observations of a random vector $\boldsymbol{W}$. The j'th observation is $\boldsymbol{W}_j$. The conditional probability of observing measurements of a rat skull in a pellet given it comes from species l is known to be $f_l(\boldsymbol{W})$.

All this means that the probability density for a set of measurements of a rat skull is:

$$p(\boldsymbol{W}) = \sum_l \pi_l f_l(\boldsymbol{W})$$

This is a form of probability model that we shall encounter quite often. It is often referred to as a **mixture model**, because it consists of a finite mixture of distributions (the coefficients of the distributions are often referred to as **mixing weights** — clearly, they must sum to one). A sample can be drawn from this model by selecting the l'th component with probability π_l, and then drawing a sample from $f_l(\boldsymbol{W})$.

Under this model, the likelihood of a set of observations is:

$$\prod_{j \in \text{observations}} \left(\sum_{l=1}^{g} \pi_l f_l(\boldsymbol{W}_j) \right)$$

We would like to infer π_l. If we knew which species of rat was represented by each skull, the problem would be easy; we could estimate π_l by counting the number of rat skulls of the l'th species, and dividing by the total number of skulls. The

difficulty is that we have only the measurements of the skulls, not the species of the rat associated with the skull.

Example: Image Segmentation

(This is a formulation due to Malik and his students []) At each pixel in an image, we compute a d-dimensional feature vector $\boldsymbol{x}$, which encapsulates position, colour and texture information. This feature vector could contain various colour representations, and the output of a series of filters centered at a particular pixel. Our image model is that each pixel is produced by a density associated by an image segment. This means that the image is produced by a mixture model, which consists of a weighted sum of g Gaussian densities. This model is a density in feature vector space that consists of a set of "blobs", each of which is associated with an image segment. We should like to determine: (1) the parameters of each of these blobs; (2) the mixing weights and (3) from which component each pixel came (thereby segmenting the image).

We encapsulate these parameters into a parameter vector, writing the mixing weights as α_l and the parameters of each blob as $\theta_l = (\boldsymbol{\mu}_l, \Sigma_l)$, to get $\Theta = (\alpha_1, \ldots, \alpha_g, \theta_1, \ldots, \theta_g)$. The mixture model then has the form

$$p(\boldsymbol{x}|\Theta) = \sum_{l=1}^{g} \alpha_l p_l(\boldsymbol{x}|\theta_l)$$

Again, this model is sampled by choosing a component and then sampling that component.

Each component density is the usual Gaussian:

$$p_l(\boldsymbol{x}|\theta_l) = \frac{1}{(2\pi)^{d/2}\det(\Sigma_i)^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\Sigma_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\}$$

The likelihood function for an image is:

$$\prod_{j \in \text{observations}} \left(\sum_{l=1}^{g} \alpha_l p_l(\boldsymbol{x}_j|\theta_l)\right)$$

Each component is associated with a segment, and Θ is unknown.

If we knew the component from which each pixel came, then it would be simple to determine Θ. We could use maximum likelihood estimates for each θ_l, and then the fraction of the image in each component would give the α_l. Similarly, if we knew Θ, then for each pixel, we could determine the component that is most likely to have produced that pixel — this yields an image segmentation. The difficulty is that we know neither.

Strategy

For each of these examples, if we know the missing data then we can estimate the parameters effectively. Similarly, if we know the parameters, the missing data will

follow. This suggests an iterative algorithm:

1. Obtain some estimate of the missing data, using a guess at the parameters;
2. now form a maximum likelihood estimate of the free parameters using the estimate of the missing data.

and we would iterate this procedure until (hopefully!) it converged. In the case of the owls and the rats, this would look like:

1. Obtain some estimate of the number of rats of each species in each owl pellet, using a guess at π_l;
2. now form a revised estimate of π_l using the number of rats of each species in each pellet.

For image segmentation, this would look like:

1. Obtain some estimate of the component from which each pixel's feature vector came, using an estimate of the θ_l.
2. Now update the θ_l, using this estimate.

18.1.2 The EM Algorithm

Although it would be nice if the procedures given above converged, there is no particular reason to believe that they do. In fact, given appropriate choices in each stage, they do. This is most easily shown by showing that they are examples of a general algorithm, the **expectation-maximization** algorithm.

A Formal Statement of Missing Data Problems

Assume we have two spaces, the **complete data space** $\mathcal{X}$ and the **incomplete data space** $\mathcal{Y}$. There is a map f, which takes $\mathcal{X}$ to $\mathcal{Y}$. This map "loses" the missing data; for example, it could be a projection. For the example of the owls and rats, the complete data space consists of the measurements of the skulls *and* a set of variables indicating from which type of rat the skull came; the incomplete data is obtained by dropping this second set of variables. For the example of image segmentation, the complete data consists of the measurements at each pixel *and* a set of variables indicating from which component of the mixture the measurements came; the incomplete data is obtained by dropping this second set of variables.

There is a parameter space $\mathcal{U}$. For the owls and rats, the parameter space consists of the mixing weights; for the image segmentation example, the parameter space consists of the mixing weights and of the parameters of each mixture component. We wish to obtain a maximum likelihood estimate for these parameters, given only incomplete data. If we had complete data, we could use the probability

density function for the complete data space, written $p_c(\boldsymbol{x}; \boldsymbol{u})$. The complete data log-likelihood is

$$\begin{aligned} L_c(\boldsymbol{x}; \boldsymbol{u}) &= \log\{\prod_j p_c(\boldsymbol{x}_j; \boldsymbol{u})\} \\ &= \sum_j \log\left(p_c(\boldsymbol{x}_j; \boldsymbol{u})\right) \end{aligned}$$

In either of our examples, this log-likelihood would be relatively easy to work with. In the case of the owls and rats, the problem would be to estimate the mixing weights, given the type of rat from which each skull came. In the case of image segmentation, the problem would be to estimate the parameters for each image segment, given the segment from which each pixel came.

The problem is that we don't have the complete data. The probability density function for the *incomplete* data space is $p_i(\boldsymbol{y}; \boldsymbol{u})$. Now

$$p_i(\boldsymbol{y}; \boldsymbol{u}) = \int_{\boldsymbol{x}|f(\boldsymbol{x})=\boldsymbol{y}} p_c(\boldsymbol{x}; \boldsymbol{u}) d\boldsymbol{x}$$

that is, the probability density function for the incomplete data space is obtained by integrating the probability density function for the complete data space over all values that give the same $\boldsymbol{y}$. The incomplete data likelihood is

$$\prod_{j \in \text{observations}} p_i(\boldsymbol{y}_j; \boldsymbol{u})$$

We could form a maximum likelihood estimate for $\boldsymbol{u}$, given $\boldsymbol{y}$ by writing out the likelihood and maximising it. This isn't easy, because both the integral and the maximisation can be quite difficult to do. The usual strategy of taking logs doesn't make things easier, because of the integral *inside* the log. We have:

$$\begin{aligned} L_i(\boldsymbol{y}; \boldsymbol{u}) &= \log\left\{\prod_j p_i(\boldsymbol{y}_j; \boldsymbol{u})\right\} \\ &= \sum_j \log\left(p_i(\boldsymbol{y}_j; \boldsymbol{u})\right) \\ &= \sum_j \log\left(\int_{\boldsymbol{x}|f(\boldsymbol{x})=\boldsymbol{y}_j} p_c(\boldsymbol{x}; \boldsymbol{u}) d\boldsymbol{x}\right) \end{aligned}$$

This form of expression is difficult to deal with. The reason that we are stuck with the incomplete data likelihood is that we don't know which of the many possible $\boldsymbol{x}$'s that could correspond to the $\boldsymbol{y}$'s that we observe actually does correspond. Forming the incomplete data likelihood involves averaging over all such $\boldsymbol{x}$'s.

The key idea in E-M is to obtain a working value for this $\boldsymbol{x}$ by computing an expectation. In particular, we will fix the parameters at some value, and then

compute the expected value of $\boldsymbol{x}$, given the value of $\boldsymbol{y}$ and the parameter values. We then plug the expected value of $\boldsymbol{x}$ into the complete data log-likelihood, which is much easier to work with, and obtain a value of the parameters by maximising that. Now at this point, the expected value of $\boldsymbol{x}$ may have changed. We obtain an algorithm by alternating the expectation step with the maximisation step, and iterate until convergence.

More formally, given $\boldsymbol{u}^s$, we form $\boldsymbol{u}^{s+1}$ by:

1. Computing an expected value for the *complete* data using the incomplete data and the current value of the parameters. This estimate is given by:

$$\overline{\boldsymbol{x}}_j^s = \int_{\boldsymbol{x}|f(\boldsymbol{x})=\boldsymbol{y}_j} \boldsymbol{x} p_c(\boldsymbol{x}; \boldsymbol{u}^s) d\boldsymbol{x}$$

   This is referred to as **the E-step**.

2. Maximizing the *complete* data log likelihood with respect to $\boldsymbol{u}$, using the expected value of the complete data computed in the E-step. That is, we compute

$$\boldsymbol{u}^{s+1} = \arg\max_{\boldsymbol{u}} L_c(\overline{\boldsymbol{x}}^s; \boldsymbol{u})$$

   This is known as **the M-step**.

It can be shown that the incomplete data log-likelihood is increased at each step, meaning that the sequence $\boldsymbol{u}^s$ converges to a (local) maximum of the incomplete data log-likelihood (e.g. []). Of course, there is no guarantee that this algorithm converges to the *right* local maximum, and in some of the examples below we will show that finding the right local maximum can be a nuisance.

Example: Owls and Rats, Revisited:

The complete data vector here is the number of rats of each species in each owl pellet. The incomplete data log-likelihood is:

$$\sum_{k \in \text{observations}} \log \left(\sum_i \pi_i f_i(\boldsymbol{W}_k) \right)$$

which is difficult to deal with because of the sum inside the logarithm. We represent the type of rat a skull came from with a $g \times n$ array of binary random variables $\mathcal{Z}$, where the l, j'th element of $\mathcal{Z}$ is z_{lj}. This element is one if the j'th skull is of type l, and zero otherwise.

The M-step: If we know $\mathcal{Z}$, then an estimate of the π_l is easy. A maximum likelihood estimate of the probability a rat of a particular type is eaten is given by the frequency with which those rats are eaten, i.e. by

$$\pi_l = \frac{\sum_{j=1}^n z_{lj}}{n}$$

(if you're not confident about this, you should check it, perhaps with chapter **??** in hand). Now we will never have $\mathcal{Z}$, but will have the expected value of $\mathcal{Z}$. The elements of this expected value will *not necessarily* be only 0 or 1 — they could take any value between zero or one. The correct intepretation of a value of, say, 0.2 is that this represents an observation that occured 0.2 times (i.e. rather less than once). Thus, in the likelihood, we raise the term corresponding to this data item to the 0.2'th power (in the same way that, if an observation occured three times, the term corresponding to the data item would appear as third power because three copies of the term would be multiplied together). This means that our maximum likelihood estimate obtained from $\overline{\mathcal{Z}}$ would be

$$\pi_l = \frac{\sum_{j=1}^{n} \overline{z}_{lj}}{n}$$

The E-step: Now for the E-step we want an expectation for z_{lj} given the $\boldsymbol{W}_k$'s. This expectation is:

$$\begin{aligned}\overline{z}_{lj} &= 1 \times P\{z_{lj} = 1 | \boldsymbol{W}_1 \dots \boldsymbol{W}_n\} + 0 \times P\{z_{lj} = 0 | \boldsymbol{W}_1 \dots \boldsymbol{W}_n\} \\ &= P\{z_{lj} = 1 | \boldsymbol{W}_j\} \\ &= \frac{P\{\boldsymbol{W}_j | z_{lj} = 1\} P\{z_{lj}\}}{P\{\boldsymbol{W}_j\}}\end{aligned}$$

Now we assume a uniform prior on z_{ij}, and deal with the question of $P\{\boldsymbol{W}_j\}$ by getting the probability to normalize to one, yielding

$$\overline{z}_{lj} = \frac{\pi_l f_l(\boldsymbol{W}_j)}{\sum_{i=1}^{n} \pi_i f_i(\boldsymbol{W}_j)}$$

Example: Image Segmentation, Revisited

This example works rather like the previous example. Assume there are a total of n pixels. The missing data forms an n by g array of indicator variables $\mathcal{I}$. In each row there is a single one, and all other values are zero — this indicates the blob from which each pixel's feature vector came.

The E-step: Now the expected value of the l, m'th entry of $\mathcal{I}$ is given by the probability that the l'th pixel comes from the m'th blob, using the same reasoning as for the owls and the rats. Assuming that the parameters are for the s'th iteration are $\Theta^{(s)}$, we have:

$$\overline{I}_{lm} = \frac{\alpha_m^{(s)} p_m(\boldsymbol{x}_l | \theta_l^{(s)})}{\sum_{k=1}^{K} \alpha_k^{(s)} p_k(\boldsymbol{x}_l | \theta_l^{(s)})}$$

(keeping in mind that $\alpha_m^{(s)}$ means the value of α_m on the s'th iteration!).

M-step: Once we have an expected value of $\mathcal{I}$, the rest is easy. We are essentially forming maximum likelihood estimates of Θ^{s+1}. Again, the expected value of the indicator variables is not in general going to be zero or one; instead, they

will take some value in that range. This should be interpreted as an observation of that particular case that occurs with that frequency, meaning that the term in the likelihood corresponding to a particular indicator variable is raised to the power of the expected value. The calculation yields expressions for a weighted mean and weighted standard deviation that should be familiar:

$$\begin{aligned}
\alpha_m^{(s+1)} &= \frac{1}{r}\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)}) \\
\boldsymbol{\mu}_m^{(s+1)} &= \frac{\sum_{l=1}^{r} \boldsymbol{x}_l p(m|\boldsymbol{x}_l, \Theta^{(s)})}{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})} \\
\Sigma_m^{s+1} &= \frac{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)}) \left\{(\boldsymbol{x}_l - \boldsymbol{\mu}_m^{(s)})(\boldsymbol{x}_l - \boldsymbol{\mu}_m^{(s)})^T\right\}}{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})}
\end{aligned}$$

(again, keeping in mind that $\alpha_m^{(s)}$ means the value of α_m on the s'th iteration!).

Example: Line Fitting with EM

An EM line fitting algorithm follows the lines of the owls and rats example above; the missing data is an array of indicator variables $\mathcal{L}$ whose l, j'th element l_{lj} is one if point l is drawn from line j, zero otherwise. As in that example, the expected value is given by determining $P(l_{lj} = 1|\text{point } l, \text{line } j\text{'s parameters})$, and this probability is proportional to

$$\exp\left(-\frac{\text{distance from point } l \text{ to line } j^2}{2\sigma^2}\right)$$

for σ as above. The constant of proportionality is most easily determined from the fact that

$$\sum_i P(l_{ij} = 1|\text{point } i, \text{line } j\text{'s parameters}) = \sum_j P(l_{ij} = 1|\text{point } i, \text{line } j\text{'s parameters}) = 1$$

The maximisation follows the form of that for fitting a single line to a set of points, only now it must be done k times and the point coordinates are weighted by the value of $\overline{l_{lj}}$. Convergence can be tested by looking at the size of the change in the lines, or by looking at the sum of perpendicular distances of points from their lines (which operates as a log likelihood, see question **??**).

Both algorithms are inclined to get stuck in local minima. In one sense, these local minima are unavoidable — they follow from the assumption that the points are interchangeable (see figure 18.1). This can be dodged by noticing that the final configuration of either fitter is a deterministic function of its start point, and using carefully chosen start points. One strategy is to start in many different (randomly chosen) configurations, and sift through the results looking for the best fit. Another

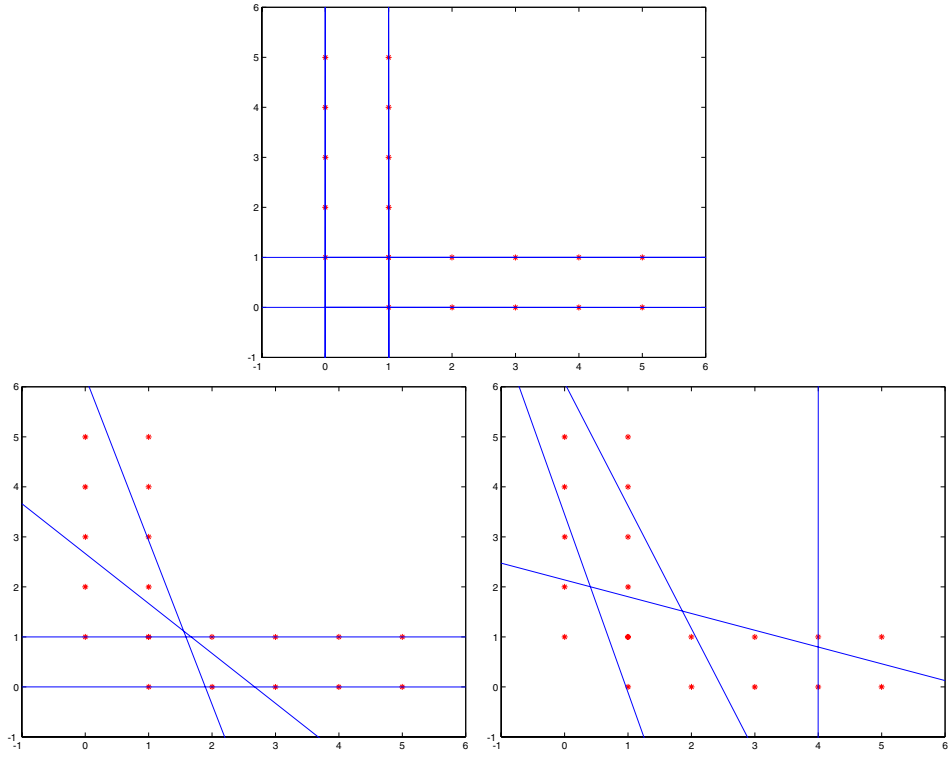

Figure 18.1. The top figure shows a good fit obtained using EM line fitting. The two bad examples in the bottom row were run with the right number of lines, but have converged to poor fits — which can be fairly good interpretations of the data, and are definitely local minima. This implementation adds a term to the mixture model that models the data point as arising uniformly and at random on the domain; a point that has a high probability of coming from this component has been identified as noise. Further examples of poor fits appear in figure 18.2.

is to preprocess the data using something like a Hough transform to guess good initial line fits. Neither is guaranteed. A cleaner approach is to notice that we are seldom, if ever, faced with a cloud of indistinguishable points and required to infer some structure on that cloud; usually, this is the result of posing a problem poorly. If points are not indistinguishable and have some form of linking structure, then a good start point should be much easier to choose.

A second difficulty to be aware of is that some points will have extremely small expected weights. This presents us with a numerical problem; it isn't clear what will happen if we regard small weights as being equivalent to zero (this isn't usually a wise thing to do). In turn, we may need to adopt a numerical representation which allows us to add many very small numbers and come up with a non-zero result.

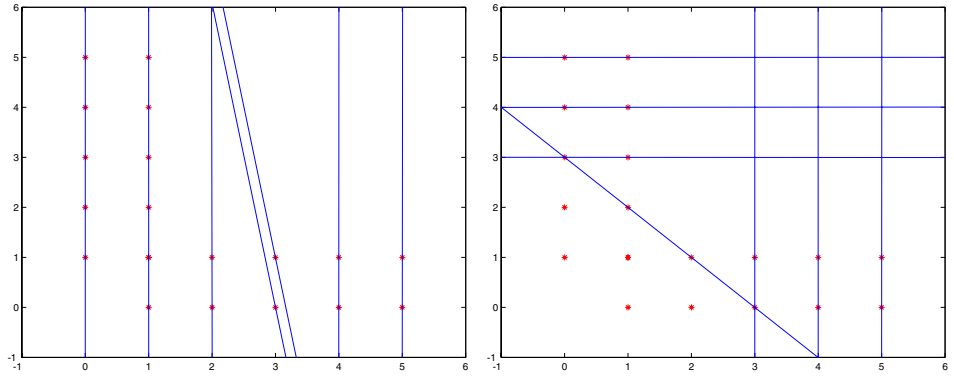

Figure 18.2. More poor fits to the data shown in figure 18.1; for these examples, we have tried to fit seven lines to this data set. Notice that these fits are fairly good interpretations of the data; they are local extrema of the likelihood. This implementation adds a term to the mixture model that models the data point as arising uniformly and at random on the domain; a point that has a high probability of coming from this component has been identified as noise. The fit on the bottom left has allocated some points to noise, and fits the others very well.

```
Choose k lines (perhaps uniformly at random)
or choose L
Until convergence
  e-step:
    recompute L, from perpendicular distances
  m-step:
    refit lines using weights in L
```

Algorithm 18.1: *EM line fitting by weighting the allocation of points to each line, with the closest line getting the highest weight*

This issue is rather outside the scope of this book; you should not underestimate its nuisance value because we don't treat it in detail.

18.1.3 Colour and Texture Segmentation with EM

We have already done most of the work for this example in section 18.1. It remains to specify appropriate feature vectors, and discuss such matters as starting the EM algorithm. The results shown in figures 18.3-18.4 use three colour features — the coordinates of the pixel in L*a*b*, after the image has been smoothed — and three texture features — which use filter outputs to estimate local scale, anisotropy and contrast (figure 18.3); other features may well be more effective — and the position

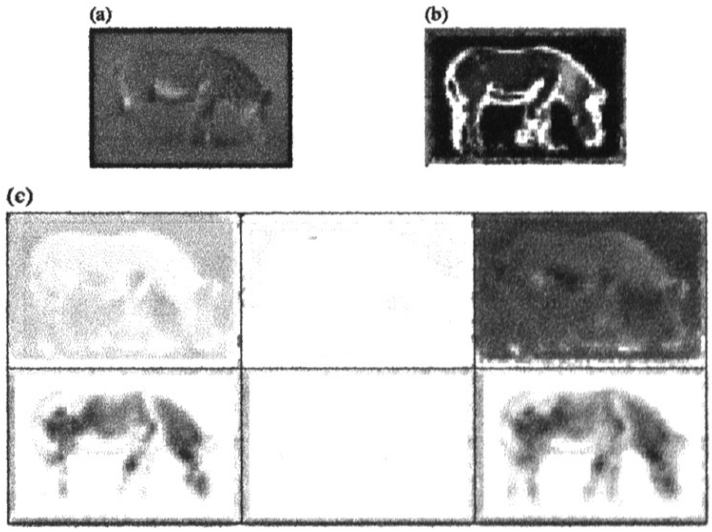

Figure 18.3. The image of the zebra in (a) gives local scale measurements shown in (b). These scale measurements essentially measure the scale of the change around a pixel; at edges, the scale is narrow, and in stripey regions it is broad, for example. The features that result are shown in (c); the top three images show the smoothed colour coordinates and the bottom three show the texture features (*ac*, *pc* and *c* — the scale and anisotropy features are weighted by contrast). *figure from Belongie et al, Color and Texture Based Image Segmentation Using EM and Its Application to Content Based Image Retrieval, ICCV98, p 5, in the fervent hope that permission will be granted*

of the pixel.

What should the segmenter report? One option is to choose for each pixel the value of m for which $p(m|\boldsymbol{x}_l, \Theta^s)$ is a maximum. Another is to report these probabilities, and build an inference process on top of them.

18.1.4 Motion Segmentation and EM

Missing data problems turn up all over computer vision. For example, motion sequences quite often consist of large regions which have quite similar motion internally. Let us assume for the moment that we have a very short sequence — two frames — and wish to determine the motion field at each point on the first frame. We will assume that the motion field comes from a mixture model. Recall that a general mixture model is a weighted sum of densities — the components do not have to have the Gaussian form used in section 18.1.1 (missing data, the EM algorithm and general mixture models turn up rather naturally together in vision applications).

A generative model for a pair of images would have the following form:

```
Choose a number of segments

Construct a set of support maps, one per segment,
containing one element per pixel.  These support maps
will contain the weight associating a pixel with a segment

Initialize the support maps by either:

    Estimating segment parameters from small
    blocks of pixels, and then computing weights
    using the E-step;

    Or randomly allocating values to the support maps.

Until convergence

    Update the support maps with an E-Step

    Update the segment parameters with an M-Step

end
```

Algorithm 18.2: *Colour and texture segmentation with EM*

- At each pixel in the first image, there is a motion vector connecting it to a pixel in the second image;
- there are a set of different parametric motion fields, each of which is given by a different probabilistic model;
- the overall motion is given by a mixture model, meaning that to determine the image motion at a pixel, we firstly determine which component the motion comes from, and then secondly draw a sample from this component.

This model encapsulates a a set of distinct, internally consistent motion fields — which might come from, say, a set of rigid objects at different depths and a moving camera (figure 18.5) — rather well.

Now assume that the motion fields have a parametric form, and that there are g different motion fields. Given a pair of images, we wish to determine (1) which motion field a pixel belongs to and (2) the parameter values for each field. All this should look a great deal like the first two examples, in that if we knew the first, the second would be easy, and if we knew the second, the first would be easy. This

For each pixel location l

    For each segment m

      Insert $\alpha_m^{(s)} p_m(\boldsymbol{x}_l|\theta_l^{(s)})$
      in pixel location l in the support map m

end
    Add the support map values to obtain
    $\sum_{k=1}^{K} \alpha_k^{(s)} p_k(\boldsymbol{x}_l|\theta_l^{(s)})$
    and divide the value in location l in each support map by this term

end

Algorithm 18.3: *Colour and texture segmentation with EM: - the E-step*

For each segment m

    Form new values of the segment parameters
    using the expressions:

$$\alpha_m^{(s+1)} = \frac{1}{r}\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})$$

$$\boldsymbol{\mu}_m^{(s+1)} = \frac{\sum_{l=1}^{r} \boldsymbol{x}_l p(m|\boldsymbol{x}_l, \Theta^{(s)})}{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})}$$

$$\Sigma_m^{s+1} = \frac{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})\left\{(\boldsymbol{x}_l - \boldsymbol{\mu}_m^{(s)})(\boldsymbol{x}_l - \boldsymbol{\mu}_m^{(s)})^T\right\}}{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})}$$

    Where $p(m|\boldsymbol{x}_l, \Theta_{(s)})$ is the value
    in the m'th support map for pixel location l

end

Algorithm 18.4: *Colour and texture segmentation with EM: - the M-step*

is again a missing data problem: the missing data is the motion field to which a pixel belongs, and the parameters are the parameters of each field and the mixing

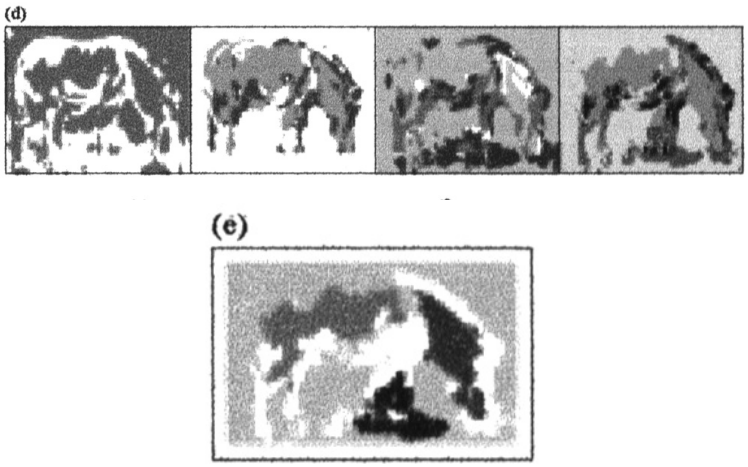

Figure 18.4. Each pixel of the zebra image of figure 18.3 is labelled with the value of m for which $p(m|\boldsymbol{x}_l, \Theta^s)$ is a maximum, to yield a segmentation. The images in (d) show the result of this process for $K = 2, 3, 4, 5$. Each image has K grey-level values corresponding to the segment indexes. *figure from Belongie et al, Color and Texture Based Image Segmentation Using EM and Its Application to Content Based Image Retrieval, ICCV98, p 5, in the fervent hope that permission will be granted*

Missing Figure

Figure 18.5. A pair of frames from a sequence of a garden, with motion segments overlaid

weights.

Assume that the pixel at (u, v) in the first image belongs to the l'th motion field, with parameters θ_l. This means that this pixel has moved to $(u, v) + \boldsymbol{m}(u, v; \theta_l)$ in the second frame, and so that the intensity at these two pixels is the same up to measurement noise. We will write $I_1(u, v)$ for the image intensity of the first image at the u, v'th pixel, and so on. The missing data is the motion field to which the

pixel belongs. We can represent this by an indicator variable $V_{uv,l}$ where

$$V_{uv,l} = \left\{ \begin{array}{c} 1, \text{ if the } u, v\text{'th pixel belongs to the } l\text{'th motion field} \\ 0, \text{ otherwise} \end{array} \right\}$$

We assume Gaussian noise with standard deviation σ in the image intensity values, so the complete data log-likelihood is

$$L(V, \Theta) = -\sum_{ij,l} V_{uv,l} \frac{(I_1(u,v) - I_2(u + m_1(u,v;\theta_l), v + m_2(u,v;\theta_l)))^2}{2\sigma^2} + C$$

where $\Theta = (\theta_1, \ldots, \theta_g)$. Setting up the EM algorithm from here on is straightforward. As above, the crucial issue is determining

$$P\{V_{uv,l} = 1 | I_1, I_2, \Theta\}$$

The more interesting question is the appropriate choice of parametric motion model. A common choice is an **affine motion model**, where

$$\left\{ \begin{array}{c} m_1 \\ m_2 \end{array} \right\} (i, j; \theta_l) = \left\{ \begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right\} \left\{ \begin{array}{c} i \\ j \end{array} \right\} + \left\{ \begin{array}{c} a_{13} \\ a_{23} \end{array} \right\}$$

and $\theta_l = (a_{11}, \ldots, a_{23})$.

18.1.5 The Number of Components

At each stage of this section, we have assumed that the number of components in the mixture model is known. This is generally not the case in practice. Finding the number of components is, in essence, a model selection problem — we will search through the collection of models (where different models have different numbers of components) to determine which fits the data best. Of course, if we simply look at the likelihood, the more components in the model, the better the fit, so we need to apply a term of that penalizes the model for more components. This issue arises quite widely, and we deal with it in detail in section 18.3.

18.1.6 How Many Lines are There?

It isn't possible to do anything sensible about this problem without a model. After all, there could be one line for every pair of points, or no lines at all and a hyperactive noise process.

Incremental line fitters appear to solve this problem without a model — after all, one applies the incremental line fitter, and some collection of lines emerges. In fact, the series of tests that are applied to determine whether a line is present constitute a model. These tests are typically a test on the residual to determine whether edge points are associated with a particular line, and some test to ensure that lines have sufficient support. Because the model is not explicit, it can be difficult to understand its implications.

When there is an explicit model, this problem is a model selection problem (as in section **??**): does the evidence support a model with $r + 1$ lines better than a model with r lines? There are a variety of model selection methods that apply to this problem as well as to telling how many segments are in an image, etc. We review these methods in section 18.3.

18.2 Robustness

In this section, we discuss a very general difficulty through the lens of our model problem. In particular, all of the line-fitting methods we have described involve squared error terms. This can lead to very poor fits in practice, because a single wildly inappropriate data point can give errors that are dominate those due to many good data points; these errors can result in a substantial bias in the fitting process (figure 18.6). It appears to be very difficult to avoid such data points — usually called **outliers** — in practice. Errors in collecting or transcribing data points is one important source of outliers. Another common source is a problem with the model — perhaps some rare but important effect has been ignored, or the magnitude of an effect has been badly underestimated[1]. Vision problems are usually full of outliers.

One approach to this problem puts the model at fault: the model predicts these outliers occuring perhaps once in the lifetime in the universe, and they clearly occur much more often than that. The natural response is to improve the model, either by allowing an explicit outlier model (section 18.2.1) or by giving the noise "heavier tails" (section 18.2.2). Finally, we could search for points that appear to be good (section **??**).

18.2.1 Explicit Outliers

The line fitters we have described have difficulty with outliers because they encounter outliers with a frequency that is wildly underpredicted by the model. Outliers are often referred to as being "in the **tails**" of a probability distribution. In probability distributions like the normal distribution, there is a large collection of values with very small probability; these values are the tails of the distribution (probably because these values are where the distribution *tails off*). A natural mechanism for dealing with outliers is to modify the model so that the distribution has **heavier tails** (i.e. that there is more probability in the tails).

One way to do this is to construct an explicit model of outliers, which is usually quite easy to do. We form a weighted sum of the likelihood $P(\text{measurements}|\text{model})$ and a term for outliers $P(\text{outliers})$, to obtain:

$$(1 - \lambda)P(\text{measurements}|\text{model}) + \lambda P(\text{outliers})$$

here $\lambda \in [0, 1]$ models the frequency with which outliers occur, and $P(\text{outliers})$ is some probability model for outliers; failing anything better, it could be uniform over the possible range of the data.

[1] At least one author of a textbook on statistics claims to have two patents due to outliers [], p.

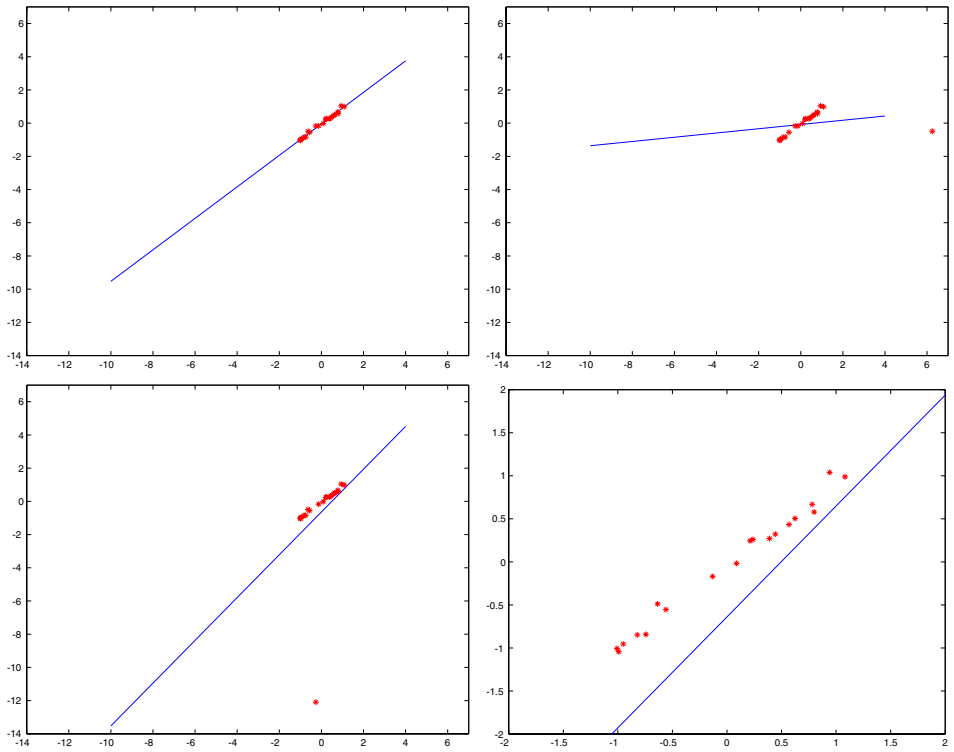

Figure 18.6. Least squares line fitting is hideously sensitive to outliers, both in x and y coordinates. At the top left, a good least-squares fit of a line to a set of points. Top-right shows the same set of points, but with the x-coordinate of one point corrupted. In this case, the slope of the fitted line has swung wildly. Bottom-left shows the same set of points, but with the y-coordinate of one point corrupted. In this particular case, the x-intercept has changed. These three figures are on the same set of axes for comparison, but this choice of axes does not clearly show how bad the fit is for the third case; Bottom-right shows a detail of this case — the line is clearly a very bad fit.

The natural way to deal with this model is to construct a variable that indicates which component generated each point. With this variable, we have a complete data likelihood function with an easy form. Of course, we don't *know* this variable, but this is a missing data problem, and we know how to proceed here using EM (you provide the details in the exercises!). The usual difficulties with EM occur here, too. In particular, it is easy to get trapped in local minima, and we may need to be careful about the numerical representation adopted for very small probabilities.

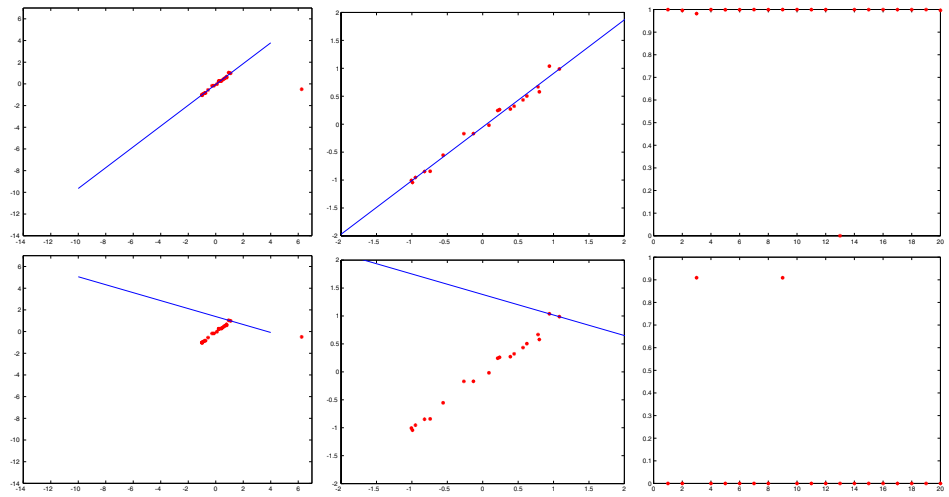

Figure 18.7. EM can be used to reject outliers; here we demonstrate a line fit to the second data set of figure 18.6. The top row shows the correct local minimum, and the bottom row shows another local minimum. The first column shows the line superimposed on the data points using the same axes as figure 18.6; the second column shows a detailed view of the line, indicating the region around the data points; and the third column shows a plot of the probability that a point comes from the line, rather than from the noise model, plotted against the index of the point. Notice that at the correct local minimum, all but one point is associated with the line, whereas at the incorrect local minimum, there are two points associated with the line and the others are allocated to noise.

18.2.2 M-estimators

The difficulty with modelling the source of outliers is that the model might be wrong. Generally, the best we can hope for from a probabilistic model of a process is that it is quite close to the right model. Assume that we are guaranteed that our model of a process is close to the right model — say, the distance between the density functions in some appropriate sense is less than ϵ. We can use this guarantee to reason about the design of estimation procedures for the parameters of the model. In particular we can choose an estimation procedure by assuming that nature is malicious and well-informed about statistics[2]. In this line of reasoning, we assess the goodness of an estimator by assuming that somewhere in the collection of processes close to our model is the real process, and it just happens to be the one that makes the estimator produce the worst possible estimates. The best estimator is the one that behaves best on the worst distribution close to the parametric model. This is a criterion which can be used to produce a wide variety of estimators.

[2]Generally, sound assumptions for any enterprise; the world is full of opportunities for painful and expensive lessons in practical statistics

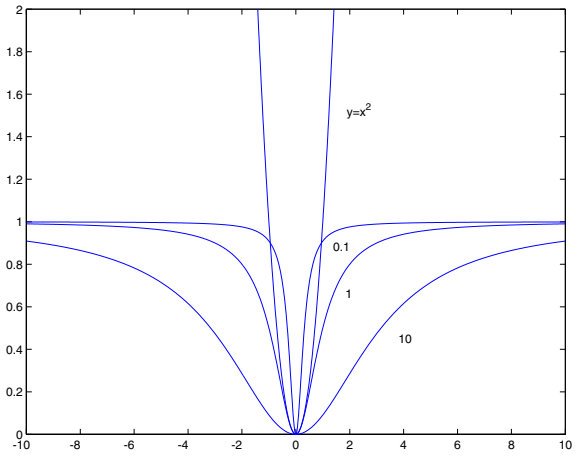

Figure 18.8. The function $\rho(x;\sigma) = x^2/(\sigma^2 + x^2)$, plotted for $\sigma^2 = 0.1$, 1 and 10, with a plot of $y = x^2$ for comparison. Replacing quadratic terms with ρ reduces the influence of outliers on a fit — a point which is several multiples of σ away from the fitted curve is going to have almost no effect on the coefficients of the fitted curve, because the value of ρ will be close to 1 and will change extremely slowly with the distance from the fitted curve.

An **M-estimator** estimates parameters by minimizing an expression of the form

$$\sum_i \rho(r_i(\boldsymbol{x}_i, \theta); \sigma)$$

where θ are the parameters of the model being fitted and $r_i(\boldsymbol{x}_i, \theta)$ is the residual error of the model on the i'th data point. Generally, $\rho(u;\sigma)$ looks like u^2 for part of its range, and then flattens out. A common choice is

$$\rho(u;\sigma) = \frac{u^2}{\sigma^2 + u^2}$$

The parameter σ controls the point at which the function flattens out; we have plotted a variety of examples in figure 18.8. There are many other M-estimators available. Typically, they are discussed in terms of their **influence function**, which is defined as

$$\frac{\partial \rho}{\partial \theta}$$

This is natural, because our criterion is

$$\sum_i \rho(r_i(\boldsymbol{x}_i, \theta); \sigma) \frac{\partial \rho}{\partial \theta} = 0$$

For the kind of problems we consider, we would expect a good influence function to be antisymmetric — there is no difference between a slight over prediction and a slight under prediction — and to tail off with large values — because we want to limit the influence of the outliers.

There are two tricky issues with using M-estimators. Firstly, the extremisation problem is non-linear and must be solved iteratively. The standard difficulties apply: there may be more than one local minimum; the method may diverge; and the behaviour of the method is likely to be quite dependent on the start point. A common strategy for dealing with this problem is to draw a subsample of the data set, fit to that subsample using least squares, and use this as a start point for the fitting process. We do this for a large number of different subsamples, enough to ensure that there is a very high probability that in that set there is at least one that consists entirely of good data points.

Secondly, as figures 18.9 and 18.10 indicate, the estimators require a sensible estimate of σ, which is often referred to as scale. Typically, the scale estimate is supplied at each iteration of the solution method; a popular estimate of scale is

$$\sigma^{(n)} = 1.4826 median_i |r_i^{(n)}(x_i; \theta^{(n-1)})|$$

For $s = 1$ to $s = k$
  draw a subset of r distinct points, chosen uniformly at random

  Fit to this set of points using maximum likelihood
  (usually least squares) to obtain θ_s^0

  estimate σ_s^0 using θ_s^0

  Until convergence (usually $|\theta_s^n - \theta_s^{n-1}|$ is small):
    take a minimising step using θ_s^{n-1}, σ_s^{n-1}
    to get θ_s^n
    now compute σ_s^n

report the best fit of this set, using the median of the
residuals as a criterion

Algorithm 18.5: *Using an M-estimator to fit a probabilistic model*

An M-estimator can be thought of as a trick for ensuring that there is more probability in the tails than would otherwise occur with a quadratic error. The function that is minimised looks like distance for small values of $\boldsymbol{x}$ — thus, for valid data points the behaviour of the M-estimator should be rather like maximum

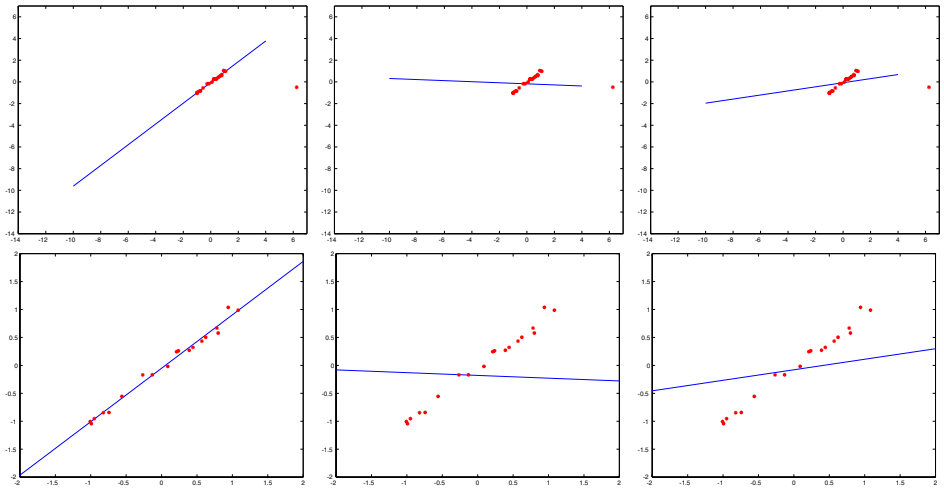

Figure 18.9. The top row shows lines fitted to the second dataset of figure 18.6 using a weighting function that de-emphasizes the contribution of distant points (the function ϕ of figure 18.8). On the left, μ has about the right value; the contribution of the outlier has been down-weighted, and the fit is good. In the center, the value of μ is too small, so that the fit is insensitive to the position of all the data points, meaning that its relationship to the data is obscure. On the right, the value of μ is too large, meaning that the outlier makes about the same contribution that it does in least-squares. The bottom row shows close-ups of the fitted line and the non-outlying data points, for the same cases.

likelihood — and like a constant for large values of $\boldsymbol{x}$ — meaning that a component of probability is given to the tails of the distribution. The strategy of the previous section can be seen as an M-estimator, but with the difficulty that the influence function is discontinuous, meaning that obtaining a minimum is tricky.

18.2.3 RANSAC

An alternative to modifying the generative model to have heavier tails is to search the collection of data points for good points. This is quite easily done by an iterative process: first, we choose a small subset of points and fit to that subset; then we see how many other points fit to the resulting object. We continue this process until we have a high probability of finding the structure we are looking for.

For example, assume that we have a data set that consists of about 50% outliers. If we draw pairs of points uniformly and at random, then about 1/4 of these pairs will consist entirely of good data points. We can identify these good pairs, by noticing that a large collection of other points will lie close to the line fitted to such a pair. Of course, a better estimate of the line could then be obtained by fitting a line to the points that lie close to our current line.

This approach leads to an algorithm — search for a random sample that leads

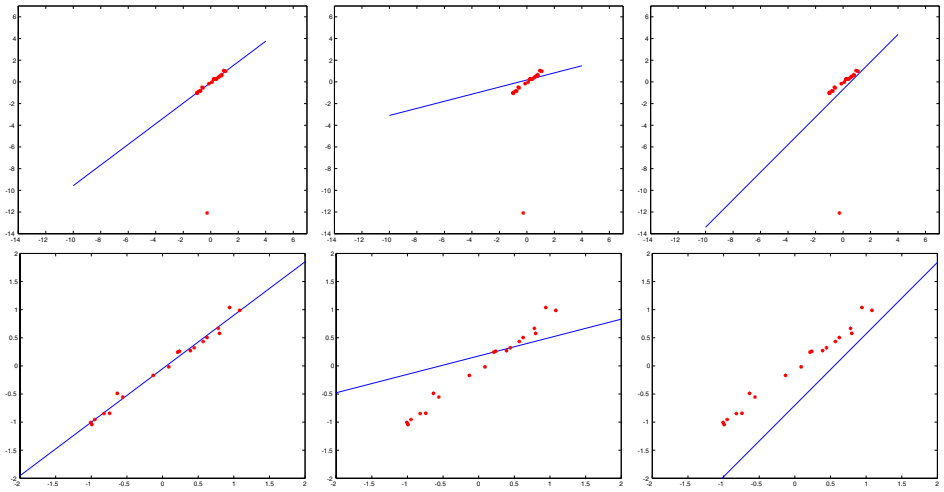

Figure 18.10. The top row shows lines fitted to the third dataset of figure 18.6 using a weighting function that de-emphasizes the contribution of distant points (the function ϕ of figure **??**). On the left, μ has about the right value; the contribution of the outlier has been down-weighted, and the fit is good. In the center, the value of μ is too small, so that the fit is insensitive to the position of all the data points, meaning that its relationship to the data is obscure. On the right, the value of μ is too large, meaning that the outlier makes about the same contribution that it does in least-squares. The bottom row shows close-ups of the fitted line and the non-outlying data points, for the same cases.

to a fit on which many of the data points agree. The algorithm is usually called `RANSAC`, for RANdom SAmple Consensus. To make this algorithm practical, we need to be able to choose three parameters.

How Many Samples are Necessary?

Our samples will consist of sets of points drawn uniformly and at random from the data set. Each sample will contain the minimum number of points required to fit the abstraction we wish to fit; for example, if we wish to fit lines, we will draw pairs of points; if we wish to fit circles, we will draw triples of points, etc. We assume that we need to draw n data points, and that w is the fraction of these points that are good (we will need only a reasonable estimate of this number). Now the expected value of the number of draws k required to get one point is given by

$$\begin{aligned}\mathrm{E}[k] &= 1P(\text{one good sample in one draw}) + 2P(\text{one good sample in two draws}) + \dots \\ &= w^n + 2(1-w^n)w^n + 3(1-w^n)^2 w^n + \dots \\ &= w^{-n}\end{aligned}$$

(where the last step takes a little manipulation of algebraic series). Now we would like to be fairly confident that we have seen a good sample, so we would wish to

draw rather more than w^{-n} samples; a natural thing to do is to add a few standard deviations to this number (see section **??** for an inequality that suggests why this is the case). The standard deviation of k can be obtained as

$$SD(k) = \frac{\sqrt{1 - w^n}}{w^n}$$

An alternative approach to this problem is to choose to look at a number of samples that guarantees a low probability z of seeing only bad samples. In this case, we have

$$(1 - w^n)^k = (1 - z)$$

which means that

$$k = \frac{\log(1 - z)}{\log(1 - w^n)}$$

How Far Away is Agreement?

We need to determine whether a point lies close to a line fitted to a sample. We will do this by determining the distance between the point and the fitted line, and testing that distance against a threshold d; if the distance is below the threshold, then the point lies close. In general, specifying this parameter is part of the modelling process. For example, when we fitted lines using maximum likelihood, there was a term σ in the model (which disappeared in the manipulations to find an maximum). This term gives the average size of deviations from the model being fitted.

In general, obtaining a value for this parameter is relatively simple. We generally need only an order of magnitude estimate, and the same value will apply to many different experiments. The parameter is often determined by trying a few values and seeing what happens; another approach is to look at a few characteristic data sets, fitting a line by eye, and estimating the average size of the deviations.

How Many Points Need to Agree?

Assume that we have fitted a line to some random sample of two data points. We need to know whether that line is good or not. We will do this by counting the number of points that lie within some distance of the line (the distance was determined in the previous section). In particular, assume that we know the probability that an outlier lies in this collection of points; write this probability as y. We should like to choose some number of points t such that y^t is small (say, less than 0.05).

There are two ways to proceed. One is to notice that $y \leq (1 - w)$, and to choose t such that $(1 - w)^t$ is small. Another is to get an estimate of y from some model of outliers — for example, if the points lie in a unit square, the outliers are uniform, and the distance threshold is d, then $y \leq 2\sqrt{2}d$.

These observations lead to algorithm 6, originally due to [**?**].

```
Determine n, t, d and k as above
Until there is a good fit or k iterations have occurred
    draw a sample of n points from the data
    uniformly and at random

    fit to that set of n points

    for each data point outside the sample

      test the distance from the point to the line
      against t; if the distance from the point to the line
      is less than t, the point is close

    end
    if there are d or more points close to the line
    then there is a good fit.  Refit the line using all
    these points, and terminate
end
```

Algorithm 18.6: *RANSAC: fitting using random sample consensus*

18.3 How Many are There?

In all the discussion above, we have assumed that the number of components (lines, etc.) was known. This is hardly ever the case in practice. Instead, we need to refer to some statistical procedure to infer the number of components from the data and the model. This is another instance of model selection (which we discussed in section 21). We adopt a strategy of searching over the number of components — perhaps restricting the search to a reasonable range for reasons of computational complexity — and evaluating a score for each case. The best score gives the number of components.

Generally, the value of the negative log-likelihood is a poor guide to the number of components because, in general, a model with more parameters will fit a dataset better than a model with fewer parameters. This means that simply minimizing the negative log-likelihood as a function of the number of components will tend to lead to too many components. For example, we can fit a set of lines extremely accurately by passing a line through each pair of points — there may be a lot of lines, but the fitting error is zero. We resolve this difficulty by adding a term that *increases* with the number of components — this penalty compensates for the decrease in negative log-likelihood caused by the increasing number of parameters.

It is important to understand that there is no *canonical* model selection process. Instead, we can choose from a variety of techniques, each of which uses a

different discount corresponding to a different extremality principle (and different approximations to these criteria!).

18.3.1 Basic Ideas

Model selection is a general problem in fitting parametric models. The problem can be set up as follows: there is a data set, which is a sample from a parametric model which is itself a member of a family of models. We wish to determine (1) which model the data set was drawn from and (2) what the parameters of that model were. A proper choice of the parameters will predict future samples from the model — a **test set** — *as well as* the data set (which is often called the **training set**); unfortunately, these future samples are not available. Furthermore, the estimate of the model's parameters obtained using the data set is likely to be biased, because the parameters chosen ensure that the model fits the *training set* — rather than the entire set of possible data — optimally. The effect is known as **selection bias**. The training set is a subset of the entire set of data that could have been drawn from the model, and represents the model exactly only if it is infinitely large. This is why the negative log-likelihood is a poor guide to the choice of model: the fit looks better, because it is increasingly biased.

The correct penalty to use comes from the **deviance**, given by

> twice (log-likelihood of the best model minus log-likelihood of the current model).

(from Ripley, [], p. 348); the best model should be the true model. Ideally, the deviance would be zero; the argument above suggests that the deviance on a training set will be larger than the deviance on a test set. A natural penalty to use is the difference between these deviances averaged over both test and training sets. This penalty is applied to twice the log-likelihood of the fit — the factor of two appears for reasons we cannot explain, but has no effect in practice. Let us write the best choice of parameters as Θ^* and the log-likelihood of the fit to the data set as $L(\boldsymbol{x}; \Theta^*)$.

18.3.2 AIC — An Information Criterion

Akaike proposed a penalty, widely called **AIC**[3] which leads to minimizing

$$-2L(\boldsymbol{x}; \Theta^*) + 2p$$

where p is the number of free parameters. There is a collection of statistical debate about the AIC (which could be followed up in [Torr, 1997; Ripley, 1996; **?**]). The first main point is that it lacks a term in the *number* of data points. This is suspicious, because the deviance between a fitted model and the real model should go down as the number of data points goes up. Secondly, there is a body of experience that the AIC tends to **overfit** — that is, to choose a model with too many parameters which fits the training set well but doesn't perform as well on test sets (e.g. []).

[3] For "An information criterion," *not* "Akaike information criterion", []

18.3.3 Bayesian methods and Schwartz' BIC

We discussed Bayesian model selection in section **??**, coming up with the expression:

$$\begin{aligned} P(\text{model}|\text{data}) &= \frac{P(\text{data}|\text{model})}{P(\text{data})} \\ &= \frac{\int P(\text{data}|\text{model}, \text{parameters})P(\text{parameters})d\{\text{parameters}\}}{P(\text{data})} \\ &\propto \int P(\text{data}|\text{model}, \text{parameters})P(\text{parameters})d\{\text{parameters}\} \end{aligned}$$

A number of possibilities now present themselves. We could give the posterior over the models, or choose the MAP model. The first difficulty is that we need to specify a prior over the models. For example, in the case of EM based segmentation, we would need to specify a prior on the number of segments. The second difficulty is that we need to compute the integral over the parameters. This could be done using a sampling method ([], or section **??**). An alternative is to construct some form of approximation to the integral, which yields another form of the penalty term.

For simplicity, let us write $\mathcal{D}$ for the data, $\mathcal{M}$ for the model, and θ for the parameters. The extent to which data support model i over model j is proportional to the **Bayes factor**

$$\frac{P(\mathcal{D}|\mathcal{M}_i)}{P(\mathcal{D}|\mathcal{M}_j)} = \frac{\int P(\mathcal{D}|\mathcal{M}_i, \theta)P(\theta)d\theta}{\int P(\mathcal{D}|\mathcal{M}_j, \theta)P(\theta)d\theta} = \frac{P(\mathcal{M}_i|\mathcal{D})}{P(\mathcal{M}_j|\mathcal{D})}\frac{P(\mathcal{M}_j)}{P(\mathcal{M}_i)} \propto \frac{P(\mathcal{M}_i|\mathcal{D})}{P(\mathcal{M}_j|\mathcal{D})}$$

Assume that $P(\mathcal{D}, \theta|\mathcal{M})$ looks roughly normal — i.e. has a single mode, roughly elliptical level curves, and doesn't die off too fast. Now write the value of θ at the mode as θ^*. The **Hessian** at the mode is a matrix $\mathcal{H}$ whose i, j'th element is

$$\frac{\partial^2 \log p(\boldsymbol{x}_i; \Theta)}{\partial\theta_i \partial\theta_j}$$

evaluated at $\theta = \theta^*$.

If we take a Taylor series approximation to the log of $P(\mathcal{D}, \theta|\mathcal{M})$ as a function of θ at the mode ($\theta = \theta^*$), we get

$$\begin{aligned} \log P(\mathcal{D}, \theta|\mathcal{M}) &= L(\mathcal{D}; \theta) + \log p(\theta) + \text{constant} \\ &= \Phi(\theta) \\ &= \Phi(\theta^*) + (\theta - \theta^*)^T\mathcal{H}(\theta - \theta^*) + O((\theta - \theta^*)^3) \\ &\approx \Phi(\theta^*) + (\theta - \theta^*)^T\mathcal{H}(\theta - \theta^*) \end{aligned}$$

(the linear term is missing because θ^* is the mode). Now

$$P(\mathcal{D}|\mathcal{M}) = \int P(\mathcal{D}, \theta|\mathcal{M})d\theta$$

$$\begin{aligned} &= \int \exp \Phi(\theta) d\theta \\ &\approx \exp \Phi(\theta^*) \int \exp(\theta - \theta^*)^T \mathcal{H}(\theta - \theta^*) d\theta \\ &= \{\exp \Phi(\theta^*)\} (2\pi)^{\frac{p}{2}} |\mathcal{H}^{-1}|^{\frac{1}{2}} \end{aligned}$$

All this implies that

$$\log p(\mathcal{D}|\mathcal{M}) \approx L(\mathcal{D}; \theta^*) + \log p(\theta^*) + \frac{p}{2} \log(2\pi) + \frac{1}{2} \log |\mathcal{H}^{-1}|$$

Given a reasonable optimisation process and a bit of luck, we could find θ^* and $\mathcal{H}$, and evaluate this expression and so the Bayes factor. This analysis offers a criterion

$$-L(\mathcal{D}; \theta^*) - \left\{ \log p(\theta^*) + \frac{p}{2} \log(2\pi) + \frac{1}{2} \log |\mathcal{H}^{-1}| \right\}$$

(the signs are to allow comparison with the AIC in section 18.3.2). This might be difficult to evaluate. A series of approximations starting here leads to a criterion

$$-L(\mathcal{D}; \theta^*) + \frac{p}{2} \log N$$

called the **Bayes information criterion** or BIC.

18.3.4 Description Length

Models can be selected by criteria that are not intrinsically statistical; after all, we are selecting the model and we can say why we want to select it. A criterion that is somewhat natural is to choose the model that encodes the data set most crisply. This **minimum description length** criterion chooses the model that allows the most efficient transmission of the data set. To transmit the data set, one codes and transmits the model parameters, and then codes and transmits the data given the model parameters. If the data fits the model very poorly, then this latter term is large, because one has to code a noise-like signal.

A derivation of the criterion used in practice is rather beyond our needs. The details appear in [**?**]; similar ideas appear in [**?**; **?**]. Surprisingly, the BIC emerges from this analysis, yielding

$$-L(\mathcal{D}; \theta^*) + \frac{p}{2} \log N$$

18.3.5 Other Methods for Estimating Deviance

The key difficulty in model selection is that we should be using a quantity we can't measure — the model's ability to predict data not in the training set. Given a sufficiently large training set, we could split the training set into two components,

use one to fit the model and the other the test the fit. This is an approach known as **cross-validation**.

We can use cross-validation to determine the number of components in a model by splitting the data set, fitting a variety of different models to one side of the split, and then choosing the model that performs best on the other side. We expect this process to estimate the number of components, because a model that has too many parameters will fit the one data set well, but fit the other badly.

Using a single choice of a split into two components introduces a different form of selection bias, and the safest thing to do is to average the estimate over all such splits. This becomes unwieldy if the test set is large, because the number of splits is huge. The most usual version is **leave-one-out cross-validation**. In this approach we fit a model to each set of $N-1$ of the training set, compute the error on the remaining data point, and sum these errors to obtain an estimate of the model error. The model that minimizes this estimate is then chosen.

To our knowledge, this approach — which is standard for model selection in other kinds of problems — has not been used in fitting applications. It is certainly appropriate for estimating the number of components. First, assume that we compute the model error for a model with too few components to describe the image accurately. In this case, the model error will be large, because for many pixels the model will be insufficiently flexible to describe the pixel that was left out. Similarly, if we use too many components, the model will predict the left out pixel rather poorly.

18.4 Discussion

The topic of priors over models attracts considerable rhetoric. Objectors complain that this prior is arbitrary, and can be important. Supporters retort that working from such criteria as AIC or likelihood-ratios, etc., itself involves an implicit choice of prior, which might be a perverse choice for some cases.

Assignments

Exercises

- Write out the EM line fitting algorithm in detail, using the sketch in the text, the description of the owls and mice problem, and the model of section **??**. In particular, why does the E-step have the form it does? and the M-step?

- Derive the expressions of section 18.1.3 for segmentation. One possible modification is to use the new mean in the estimate of the covariance matrices. Perform an experiment to determine whether this makes any difference in practice.

- Derive the expressions for EM for motion.

- Describe using leave-one-out cross-validation for selecting the number of segments

Programming Assignments

- Build an EM segmenter that uses colour, position (ideally, use texture too) to segment images; use a model selection term to determine how many segments there should be. How significant a phenomenon is the effect of local minima?
- Build an EM line fitter that works for a fixed number of lines. Investigate the effects of local minima. One way to avoid being distracted by local minima is to start from many different start points, and then look at the best fit obtained from that set. How successful is this? how many local minima do you have to search to obtain a good fit for a typical data set? Can you improve things using a Hough transform?
- Expand your EM line fitter to incorporate a model selection term, so that the fitter can determine how many lines fit a dataset. Compare the choice of AIC and BIC.
- Insert a noise term in your EM line fitter, so that it is able to perform robust fits. What is the effect on the number of local minima? Notice that, if there is a low probability of a point arising from noise, most points will be allocated to lines, but the fits will often be quite poor; if there is a high probability of a point arising from noise, points will be allocated to lines only if they fit well. What is the effect of this parameter on the number of local minima?
- Construct a RANSAC fitter that can fit an arbitrary (but known) number of lines to a given data set. What is involved in extending your fitter to determine the best number of lines?