

20

An Example Inference Task: Clustering

Human brains are good at finding regularities in data. One way of expressing regularity is to put a set of objects into groups that are similar to each other. For example, biologists have found that most objects in the natural world fall into one of two categories: things that are brown and run away, and things that are green and don't run away. The first group they call animals, and the second, plants. We'll call this operation of grouping things together *clustering*. If the biologist further sub-divides the cluster of plants into sub-clusters, we would call this 'hierarchical clustering'; but we won't be talking about hierarchical clustering yet. In this chapter we'll just discuss ways to take a set of N objects and group them into K clusters.

There are several motivations for clustering. First, a good clustering has predictive power. When an early biologist encounters a new green thing he has not seen before, his internal model of plants and animals fills in predictions for attributes of the green thing: it's unlikely to jump on him and eat him; if he touches it, he might get grazed or stung; if he eats it, he might feel sick. All of these predictions, while uncertain, are useful, because they help the biologist invest his resources (for example, the time spent watching for predators) well. Thus, we perform clustering because we believe the underlying cluster labels are meaningful, will lead to a more efficient description of our data, and will help us choose better actions. This type of clustering is sometimes called 'mixture density modelling', and the objective function that measures how well the predictive model is working is the information content of the data, $\log 1/P(\{\mathbf{x}\})$.

Second, clusters can be a useful aid to communication because they allow lossy compression. The biologist can give directions to a friend such as 'go to the third *tree* on the right then take a right turn' (rather than 'go past the large green thing with red berries, then past the large green thing with thorns, then ...'). The brief category name 'tree' is helpful because it is sufficient to identify an object. Similarly, in lossy image compression, the aim is to convey in as few bits as possible a reasonable reproduction of a picture; one way to do this is to divide the image into N small patches, and find a close match to each patch in an alphabet of K image-templates; then we send a close fit to the image by sending the list of labels k_1, k_2, \dots, k_N of the matching templates. The task of creating a good library of image-templates is equivalent to finding a set of cluster centres. This type of clustering is sometimes called 'vector quantization'.

We can formalize a vector quantizer in terms of an *assignment rule* $\mathbf{x} \rightarrow k(\mathbf{x})$ for assigning datapoints \mathbf{x} to one of K codenames, and a *reconstruction rule* $k \rightarrow \mathbf{m}^{(k)}$, the aim being to choose the functions $k(\mathbf{x})$ and $\mathbf{m}^{(k)}$ so as to

minimize the *expected distortion*, which might be defined to be

$$D = \sum_{\mathbf{x}} P(\mathbf{x}) \frac{1}{2} \left[\mathbf{m}^{(k(\mathbf{x}))} - \mathbf{x} \right]^2. \quad (20.1)$$

[The ideal objective function would be to minimize the psychologically perceived distortion of the image. Since it is hard to quantify the distortion perceived by a human, vector quantization and lossy compression are not so crisply defined problems as data modelling and lossless compression.] In vector quantization, we don't necessarily believe that the templates $\{\mathbf{m}^{(k)}\}$ have any natural meaning; they are simply tools to do a job. We note in passing the similarity of the assignment rule (i.e., the encoder) of vector quantization to the *decoding* problem when decoding an error-correcting code.

A third reason for making a cluster model is that failures of the cluster model may highlight interesting objects that deserve special attention. If we have trained a vector quantizer to do a good job of compressing satellite pictures of ocean surfaces, then maybe patches of image that are not well compressed by the vector quantizer are the patches that contain ships! If the biologist encounters a green thing and sees it run (or slither) away, this misfit with his cluster model (which says green things don't run away) cues him to pay special attention. One can't spend all one's time being fascinated by things; the cluster model can help sift out from the multitude of objects in one's world the ones that really deserve attention.

A fourth reason for liking clustering algorithms is that they may serve as models of learning processes in neural systems. The clustering algorithm that we now discuss, the K-means algorithm, is an example of a *competitive learning* algorithm. The algorithm works by having the K clusters compete with each other for the right to own the data points.

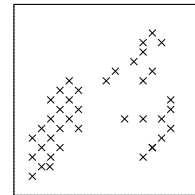


Figure 20.1. $N = 40$ data points.

► 20.1 K-means clustering

The K-means algorithm is an algorithm for putting N data points in an I -dimensional space into K clusters. Each cluster is parameterized by a vector $\mathbf{m}^{(k)}$ called its mean.

The data points will be denoted by $\{\mathbf{x}^{(n)}\}$ where the superscript n runs from 1 to the number of data points N . Each vector \mathbf{x} has I components x_i . We will assume that the space that \mathbf{x} lives in is a real space and that we have a metric that defines distances between points, for example,

$$d(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \sum_i (x_i - y_i)^2. \quad (20.2)$$

To start the K-means algorithm (algorithm 20.2), the K means $\{\mathbf{m}^{(k)}\}$ are initialized in some way, for example to random values. K-means is then an iterative two-step algorithm. In the *assignment step*, each data point n is assigned to the nearest mean. In the *update step*, the means are adjusted to match the sample means of the data points that they are responsible for.

The K-means algorithm is demonstrated for a toy two-dimensional data set in figure 20.3, where 2 means are used. The assignments of the points to the two clusters are indicated by two point styles, and the two means are shown by the circles. The algorithm converges after three iterations, at which point the assignments are unchanged so the means remain unmoved when updated. The K-means algorithm always converges to a fixed point.

About the name... As far as I know, the 'K' in K-means clustering simply refers to the chosen number of clusters. If Newton had followed the same naming policy, maybe we would learn at school about 'calculus for the variable x '. It's a silly name, but we are stuck with it.

Initialization. Set K means $\{\mathbf{m}^{(k)}\}$ to random values.

Assignment step. Each data point n is assigned to the nearest mean. We denote our guess for the cluster $k^{(n)}$ that the point $\mathbf{x}^{(n)}$ belongs to by $\hat{k}^{(n)}$.

$$\hat{k}^{(n)} = \underset{k}{\operatorname{argmin}} \{d(\mathbf{m}^{(k)}, \mathbf{x}^{(n)})\}. \quad (20.3)$$

An alternative, equivalent representation of this assignment of points to clusters is given by ‘responsibilities’, which are indicator variables $r_k^{(n)}$. In the assignment step, we set $r_k^{(n)}$ to one if mean k is the closest mean to datapoint $\mathbf{x}^{(n)}$; otherwise $r_k^{(n)}$ is zero.

$$r_k^{(n)} = \begin{cases} 1 & \text{if } \hat{k}^{(n)} = k \\ 0 & \text{if } \hat{k}^{(n)} \neq k. \end{cases} \quad (20.4)$$

What about ties? – We don’t expect two means to be exactly the same distance from a data point, but if a tie does happen, $\hat{k}^{(n)}$ is set to the smallest of the winning $\{k\}$.

Update step. The model parameters, the means, are adjusted to match the sample means of the data points that they are responsible for.

$$\mathbf{m}^{(k)} = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{R^{(k)}} \quad (20.5)$$

where $R^{(k)}$ is the total responsibility of mean k ,

$$R^{(k)} = \sum_n r_k^{(n)}. \quad (20.6)$$

What about means with no responsibilities? – If $R^{(k)} = 0$, then we leave the mean $\mathbf{m}^{(k)}$ where it is.

Repeat the assignment step and update step until the assignments do not change.

Algorithm 20.2. The K-means clustering algorithm.

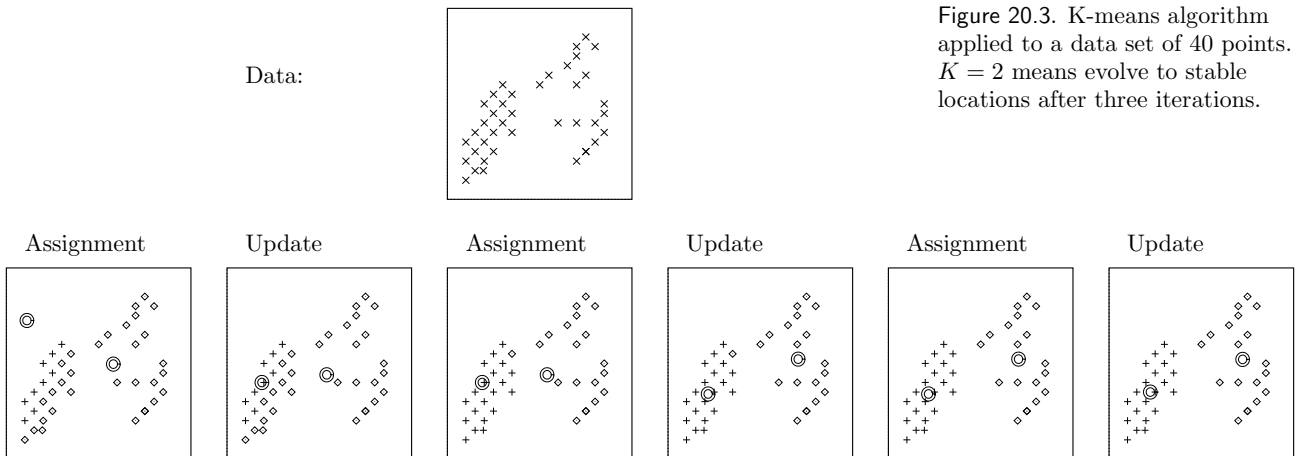


Figure 20.3. K-means algorithm applied to a data set of 40 points. $K = 2$ means evolve to stable locations after three iterations.

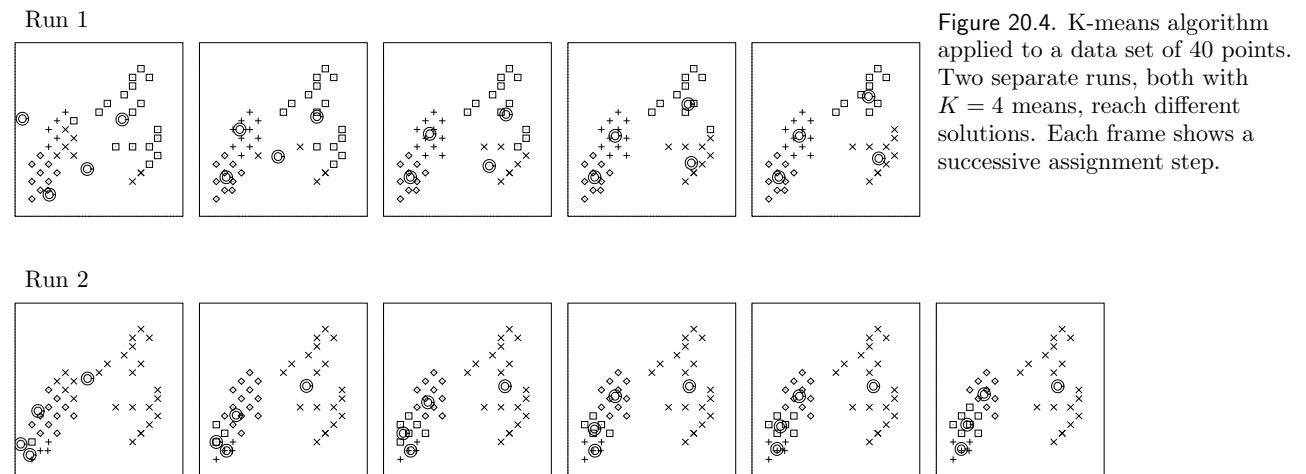


Figure 20.4. K-means algorithm applied to a data set of 40 points. Two separate runs, both with $K = 4$ means, reach different solutions. Each frame shows a successive assignment step.

Exercise 20.1. [4, p.291] See if you can prove that K-means always converges.

[Hint: find a physical analogy and an associated Lyapunov function.]

[A Lyapunov function is a function of the state of the algorithm that decreases whenever the state changes and that is bounded below. If a system has a Lyapunov function then its dynamics converge.]

The K-means algorithm with a larger number of means, 4, is demonstrated in figure 20.4. The outcome of the algorithm depends on the initial condition. In the first case, after five iterations, a steady state is found in which the data points are fairly evenly split between the four clusters. In the second case, after six iterations, half the data points are in one cluster, and the others are shared among the other three clusters.

Questions about this algorithm

The K-means algorithm has several *ad hoc* features. Why does the update step set the ‘mean’ to the mean of the assigned points? Where did the distance d come from? What if we used a different measure of distance between \mathbf{x} and \mathbf{m} ? How can we choose the ‘best’ distance? [In vector quantization, the distance

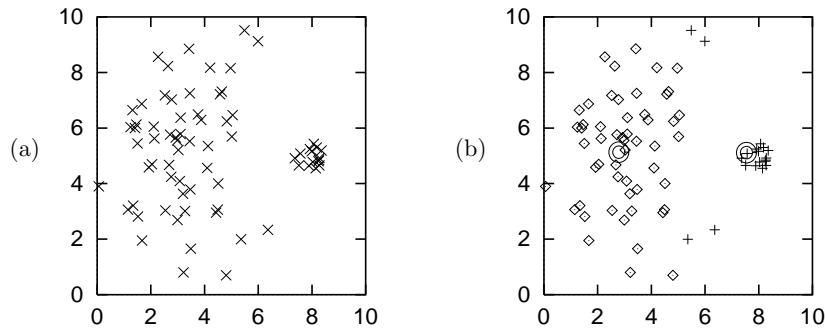


Figure 20.5. K-means algorithm for a case with two dissimilar clusters. (a) The “little ‘n’ large” data. (b) A stable set of assignments and means. Note that four points belonging to the broad cluster have been incorrectly assigned to the narrower cluster. (Points assigned to the right-hand cluster are shown by plus signs.)

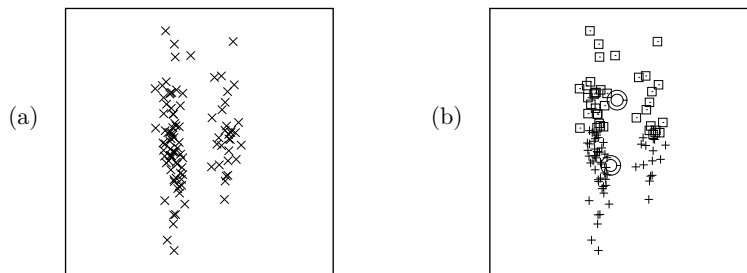


Figure 20.6. Two elongated clusters, and the stable solution found by the K-means algorithm.

function is provided as part of the problem definition; but I’m assuming we are interested in data-modelling rather than vector quantization.] How do we choose K ? Having found multiple alternative clusterings for a given K , how can we choose among them?

Cases where K-means might be viewed as failing.

Further questions arise when we look for cases where the algorithm behaves badly (compared with what the man in the street would call ‘clustering’). Figure 20.5a shows a set of 75 data points generated from a mixture of two Gaussians. The right-hand Gaussian has less weight (only one fifth of the data points), and it is a less broad cluster. Figure 20.5b shows the outcome of using K-means clustering with $K = 2$ means. Four of the big cluster’s data points have been assigned to the small cluster, and both means end up displaced to the left of the true centres of the clusters. The K-means algorithm takes account only of the distance between the means and the data points; it has no representation of the weight or breadth of each cluster. Consequently, data points that actually belong to the broad cluster are incorrectly assigned to the narrow cluster.

Figure 20.6 shows another case of K-means behaving badly. The data evidently fall into two elongated clusters. But the only stable state of the K-means algorithm is that shown in figure 20.6b: the two clusters have been sliced in half! These two examples show that there is something wrong with the distance d in the K-means algorithm. The K-means algorithm has no way of representing the size or shape of a cluster.

A final criticism of K-means is that it is a ‘hard’ rather than a ‘soft’ algorithm: points are assigned to exactly one cluster and all points assigned to a cluster are equals in that cluster. Points located near the border between two or more clusters should, arguably, play a *partial* role in determining the locations of all the clusters that they could plausibly be assigned to. But in the K-means algorithm, each borderline point is dumped in one cluster, and

has an equal vote with all the other points in that cluster, and no vote in any other clusters.

► 20.2 Soft K-means clustering

These criticisms of K-means motivate the ‘soft K-means algorithm’, algorithm 20.7. The algorithm has one parameter, β , which we could term the *stiffness*.

Assignment step. Each data point $\mathbf{x}^{(n)}$ is given a soft ‘degree of assignment’ to each of the means. We call the degree to which $\mathbf{x}^{(n)}$ is assigned to cluster k the *responsibility* $r_k^{(n)}$ (the responsibility of cluster k for point n).

$$r_k^{(n)} = \frac{\exp(-\beta d(\mathbf{m}^{(k)}, \mathbf{x}^{(n)}))}{\sum_{k'} \exp(-\beta d(\mathbf{m}^{(k')}, \mathbf{x}^{(n)}))}. \quad (20.7)$$

The sum of the K responsibilities for the n th point is 1.

Update step. The model parameters, the means, are adjusted to match the sample means of the data points that they are responsible for.

$$\mathbf{m}^{(k)} = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{R^{(k)}} \quad (20.8)$$

where $R^{(k)}$ is the total responsibility of mean k ,

$$R^{(k)} = \sum_n r_k^{(n)}. \quad (20.9)$$

Algorithm 20.7. Soft K-means algorithm, version 1.

Notice the similarity of this soft K-means algorithm to the hard K-means algorithm 20.2. The update step is identical; the only difference is that the responsibilities $r_k^{(n)}$ can take on values between 0 and 1. Whereas the assignment $\hat{k}^{(n)}$ in the K-means algorithm involved a ‘min’ over the distances, the rule for assigning the responsibilities is a ‘soft-min’ (20.7).

- Exercise 20.2.^[2] Show that as the stiffness β goes to ∞ , the soft K-means algorithm becomes identical to the original hard K-means algorithm, except for the way in which means with no assigned points behave. Describe what those means do instead of sitting still.

Dimensionally, the stiffness β is an inverse-length-squared, so we can associate a lengthscale, $\sigma \equiv 1/\sqrt{\beta}$, with it. The soft K-means algorithm is demonstrated in figure 20.8. The lengthscale is shown by the radius of the circles surrounding the four means. Each panel shows the final fixed point reached for a different value of the lengthscale σ .

► 20.3 Conclusion

At this point, we may have fixed some of the problems with the original K-means algorithm by introducing an extra complexity-control parameter β . But how should we set β ? And what about the problem of the elongated clusters,

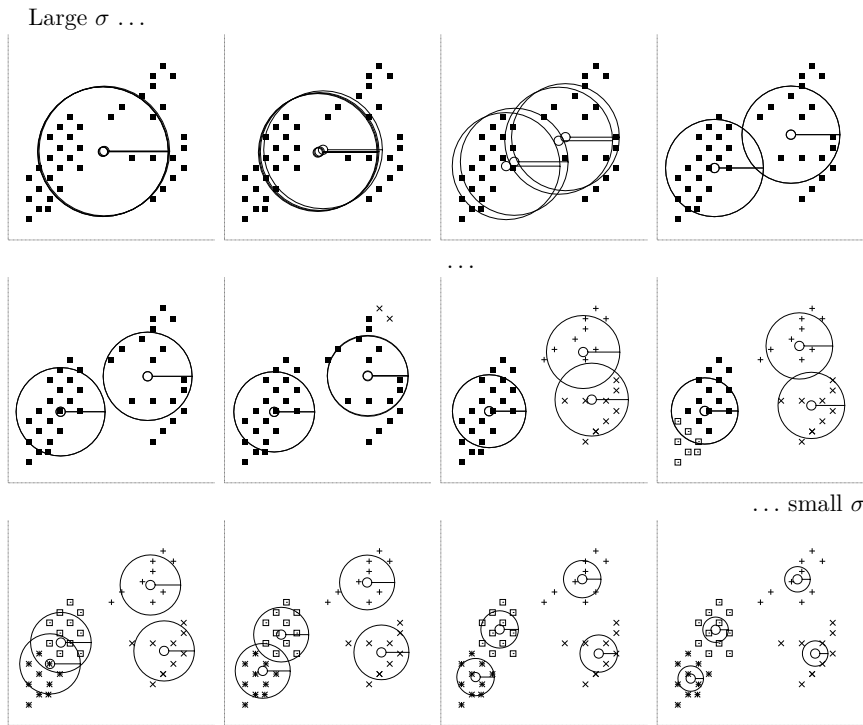


Figure 20.8. Soft K-means algorithm, version 1, applied to a data set of 40 points. $K = 4$. Implicit lengthscale parameter $\sigma = 1/\beta^{1/2}$ varied from a large to a small value. Each picture shows the state of all four means, with the implicit lengthscale shown by the radius of the four circles, after running the algorithm for several tens of iterations. At the largest lengthscale, all four means converge exactly to the data mean. Then the four means separate into two groups of two. At shorter lengthscales, each of these pairs itself bifurcates into subgroups.

and the clusters of unequal weight and width? Adding one stiffness parameter β is not going to make all these problems go away.

We'll come back to these questions in a later chapter, as we develop the mixture-density-modelling view of clustering.

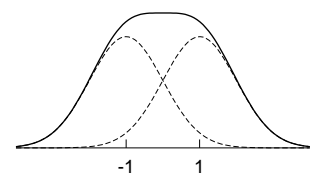
Further reading

For a vector-quantization approach to clustering see (Luttrell, 1989; Luttrell, 1990).

► 20.4 Exercises

- ▷ Exercise 20.3.^[3] Explore the properties of the soft K-means algorithm, version 1, assuming that the datapoints $\{\mathbf{x}\}$ come from a *single* separable two-dimensional Gaussian distribution with mean zero and variances $(\text{var}(x_1), \text{var}(x_2)) = (\sigma_1^2, \sigma_2^2)$, with $\sigma_1^2 > \sigma_2^2$. Set $K = 2$, assume N is large, and investigate the fixed points of the algorithm as β is varied. [Hint: assume that $\mathbf{m}^{(1)} = (m, 0)$ and $\mathbf{m}^{(2)} = (-m, 0)$.]

- ▷ Exercise 20.4.^[3] Consider the soft K-means algorithm applied to a large amount of one-dimensional data that comes from a mixture of two equal-weight Gaussians with true means $\mu = \pm 1$ and standard deviation σ_P , for example $\sigma_P = 1$. Show that the hard K-means algorithm with $K = 2$ leads to a solution in which the two means are further apart than the two true means. Discuss what happens for other values of β , and find the value of β such that the soft algorithm puts the two means in the correct places.



► 20.5 Solutions

Solution to exercise 20.1 (p.287). We can associate an ‘energy’ with the state of the K-means algorithm by connecting a spring between each point $\mathbf{x}^{(n)}$ and the mean that is responsible for it. The energy of one spring is proportional to its squared length, namely $\beta d(\mathbf{x}^{(n)}, \mathbf{m}^{(k)})$ where β is the stiffness of the spring. The total energy of all the springs is a *Lyapunov function* for the algorithm, because (a) the assignment step can only decrease the energy – a point only changes its allegiance if the length of its spring would be reduced; (b) the update step can only decrease the energy – moving $\mathbf{m}^{(k)}$ to the mean is the way to minimize the energy of its springs; and (c) the energy is bounded below – which is the second condition for a Lyapunov function. Since the algorithm has a Lyapunov function, it converges.

Solution to exercise 20.3 (p.290). If the means are initialized to $\mathbf{m}^{(1)} = (m, 0)$ and $\mathbf{m}^{(2)} = (-m, 0)$, the assignment step for a point at location x_1, x_2 gives

$$r_1(\mathbf{x}) = \frac{\exp(-\beta(x_1 - m)^2/2)}{\exp(-\beta(x_1 - m)^2/2) + \exp(-\beta(x_1 + m)^2/2)} \quad (20.10)$$

$$= \frac{1}{1 + \exp(-2\beta mx_1)}, \quad (20.11)$$

and the updated m is

$$m' = \frac{\int dx_1 P(x_1) x_1 r_1(\mathbf{x})}{\int dx_1 P(x_1) r_1(\mathbf{x})} \quad (20.12)$$

$$= 2 \int dx_1 P(x_1) x_1 \frac{1}{1 + \exp(-2\beta mx_1)}. \quad (20.13)$$

Now, $m = 0$ is a fixed point, but the question is, is it stable or unstable? For tiny m (that is, $\beta\sigma_1 m \ll 1$), we can Taylor-expand

$$\frac{1}{1 + \exp(-2\beta mx_1)} \simeq \frac{1}{2} (1 + \beta mx_1) + \dots \quad (20.14)$$

so

$$m' \simeq \int dx_1 P(x_1) x_1 (1 + \beta mx_1) \quad (20.15)$$

$$= \sigma_1^2 \beta m. \quad (20.16)$$

For small m , m either grows or decays exponentially under this mapping, depending on whether $\sigma_1^2 \beta$ is greater than or less than 1. The fixed point $m = 0$ is *stable* if

$$\sigma_1^2 \leq 1/\beta \quad (20.17)$$

and *unstable* otherwise. [Incidentally, this derivation shows that this result is general, holding for any true probability distribution $P(x_1)$ having variance σ_1^2 , not just the Gaussian.]

If $\sigma_1^2 > 1/\beta$ then there is a bifurcation and there are two stable fixed points surrounding the unstable fixed point at $m = 0$. To illustrate this bifurcation, figure 20.10 shows the outcome of running the soft K-means algorithm with $\beta = 1$ on one-dimensional data with standard deviation σ_1 for various values of σ_1 . Figure 20.11 shows this pitchfork bifurcation from the other point of view, where the data’s standard deviation σ_1 is fixed and the algorithm’s lengthscale $\sigma = 1/\beta^{1/2}$ is varied on the horizontal axis.

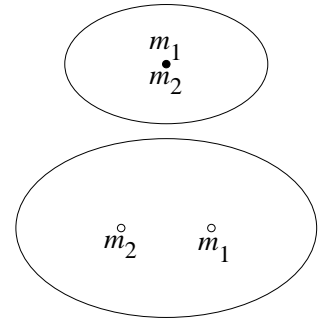


Figure 20.9. Schematic diagram of the bifurcation as the largest data variance σ_1 increases from below $1/\beta^{1/2}$ to above $1/\beta^{1/2}$. The data variance is indicated by the ellipse.

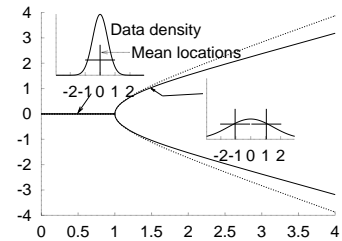


Figure 20.10. The stable mean locations as a function of σ_1 , for constant β , found numerically (thick lines), and the approximation (20.22) (thin lines).

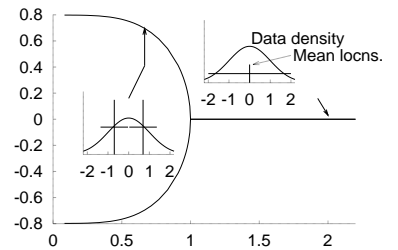


Figure 20.11. The stable mean locations as a function of $1/\beta^{1/2}$, for constant σ_1 .

Here is a cheap theory to model how the fitted parameters $\pm m$ behave beyond the bifurcation, based on continuing the series expansion. This continuation of the series is rather suspect, since the series isn't necessarily expected to converge beyond the bifurcation point, but the theory fits well anyway.

We take our analytic approach one term further in the expansion

$$\frac{1}{1 + \exp(-2\beta m x_1)} \simeq \frac{1}{2} (1 + \beta m x_1 - \frac{1}{3} (\beta m x_1)^3) + \dots \quad (20.18)$$

then we can solve for the shape of the bifurcation to leading order, which depends on the fourth moment of the distribution:

$$m' \simeq \int dx_1 P(x_1) x_1 (1 + \beta m x_1 - \frac{1}{3} (\beta m x_1)^3) \quad (20.19)$$

$$= \sigma_1^2 \beta m - \frac{1}{3} (\beta m)^3 3\sigma_1^4. \quad (20.20)$$

[At (20.20) we use the fact that $P(x_1)$ is Gaussian to find the fourth moment.] This map has a fixed point at m such that

$$\sigma_1^2 \beta (1 - (\beta m)^2 \sigma_1^2) = 1, \quad (20.21)$$

i.e.,

$$m = \pm \beta^{-1/2} \frac{(\sigma_1^2 \beta - 1)^{1/2}}{\sigma_1^2 \beta}. \quad (20.22)$$

The thin line in figure 20.10 shows this theoretical approximation. Figure 20.10 shows the bifurcation as a function of σ_1 for fixed β ; figure 20.11 shows the bifurcation as a function of $1/\beta^{1/2}$ for fixed σ_1 .

- ▷ Exercise 20.5. [2, p.292] Why does the pitchfork in figure 20.11 tend to the values $\sim \pm 0.8$ as $1/\beta^{1/2} \rightarrow 0$? Give an analytic expression for this asymptote.

Solution to exercise 20.5 (p.292). The asymptote is the mean of the rectified Gaussian,

$$\frac{\int_0^\infty \text{Normal}(x, 1) x dx}{1/2} = \sqrt{2/\pi} \simeq 0.798. \quad (20.23)$$