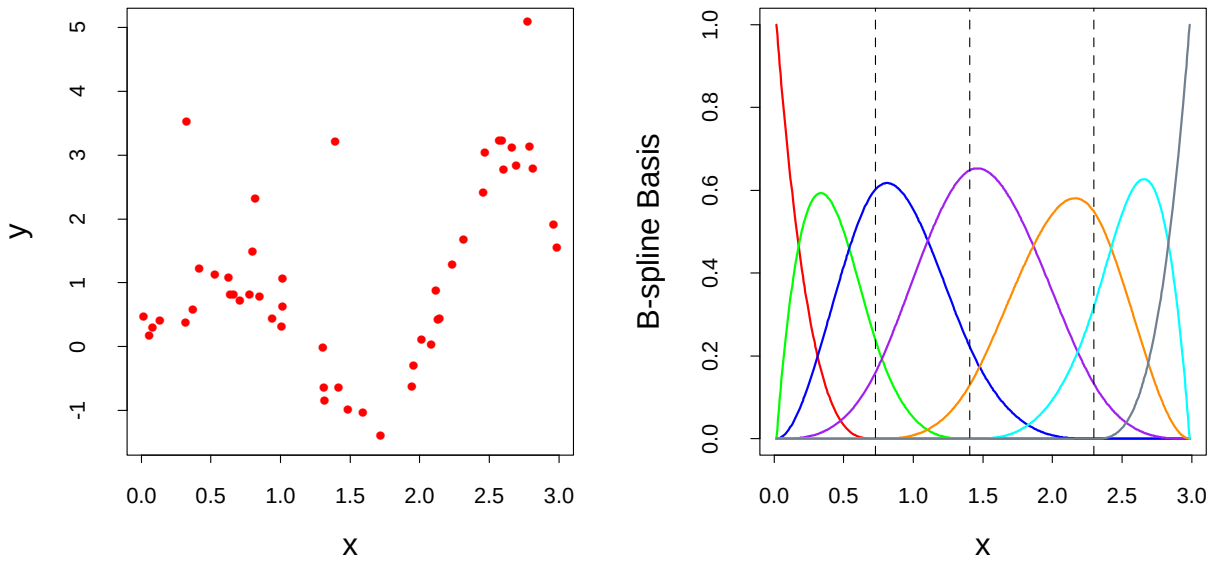

FIGURE 8.1. *(Left panel): Data for smoothing example. (Right panel:) Set of seven B-spline basis functions. The broken vertical lines indicate the placement of the three knots.*

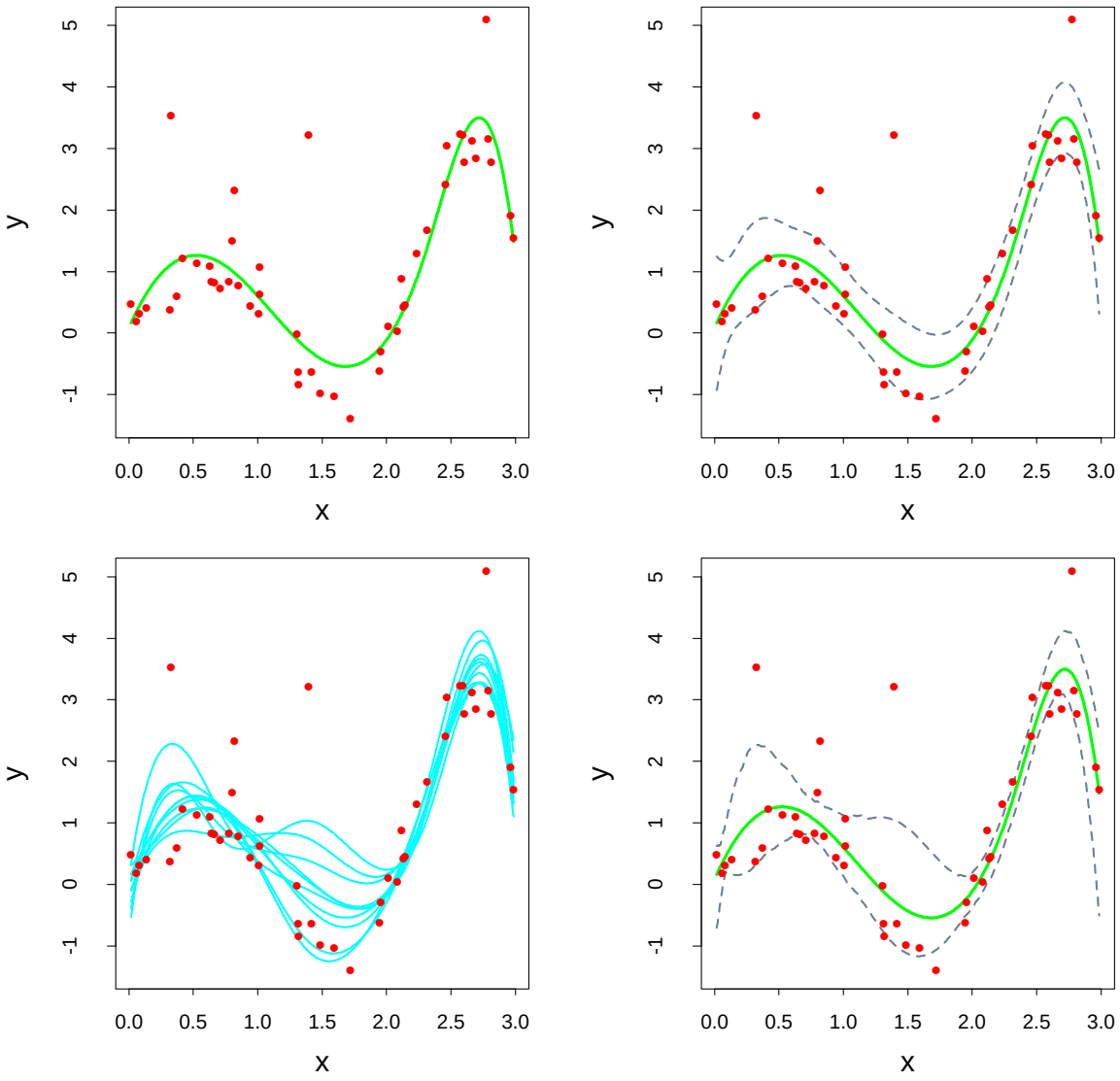

FIGURE 8.2. *(Top left:) B-spline smooth of data. (Top right:) B-spline smooth plus and minus* $1.96\times$ *standard error bands. (Bottom left:) Ten bootstrap replicates of the B-spline smooth. (Bottom right:) B-spline smooth with* 95% *standard error bands computed from the bootstrap distribution.*

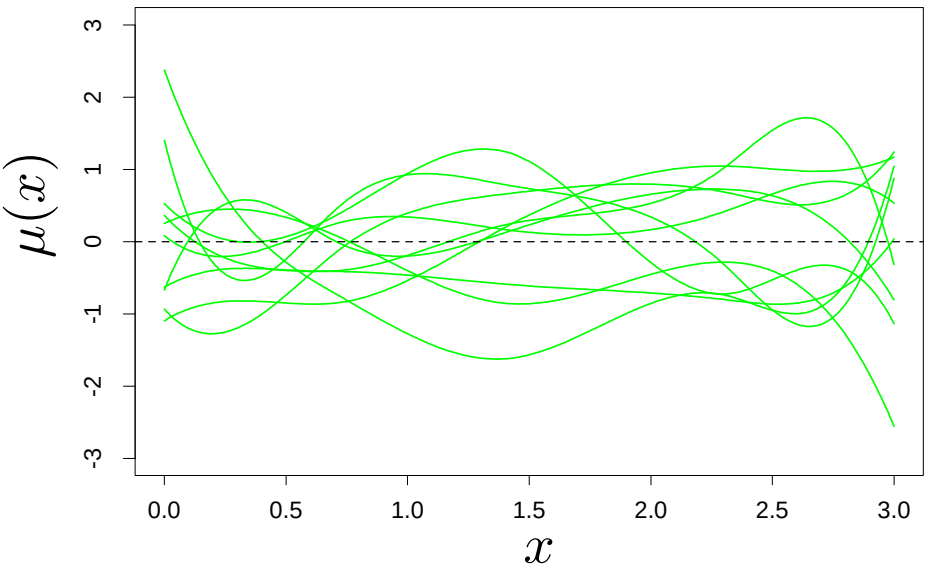

FIGURE 8.3. *Smoothing example: Ten draws from the Gaussian prior distribution for the function* $\mu(x)$.

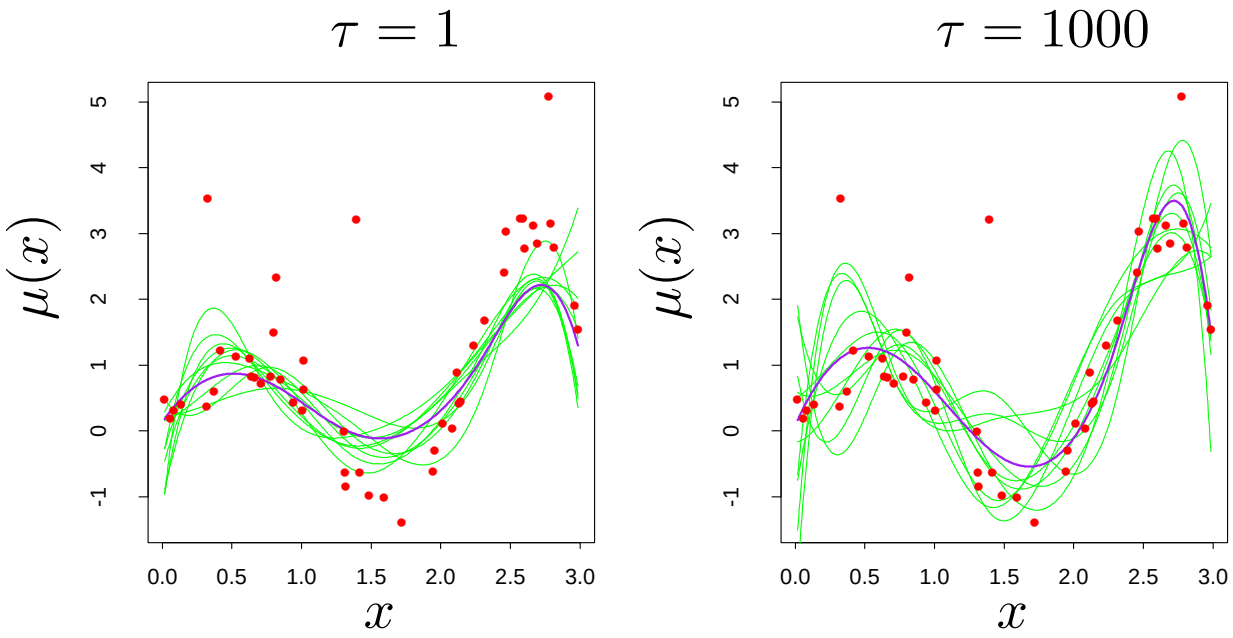

FIGURE 8.4. *Smoothing example: Ten draws from the posterior distribution for the function $\mu(x)$, for two different values of the prior variance τ. The purple curves are the posterior means.*

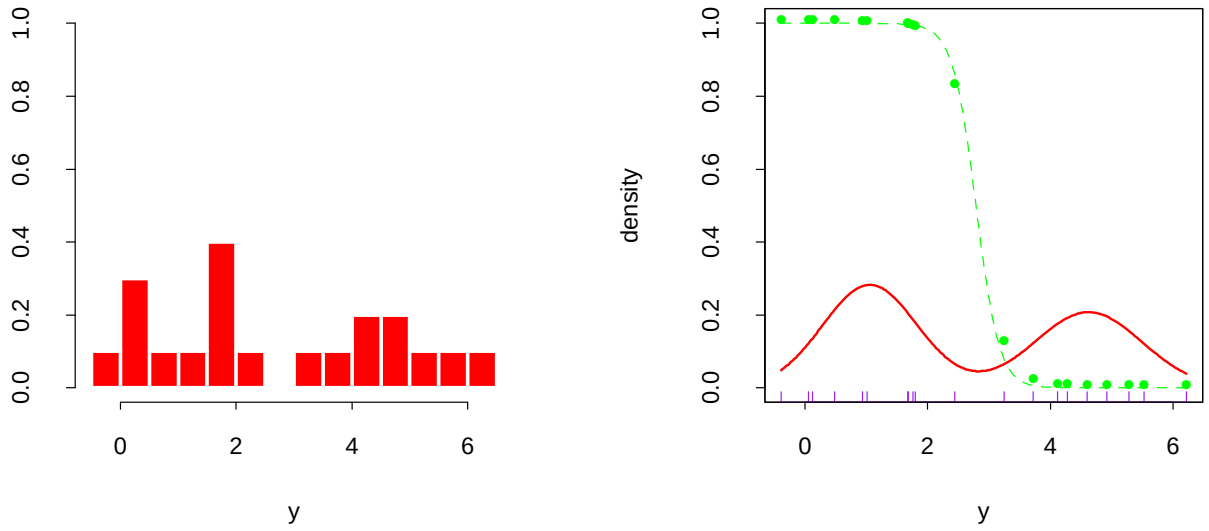

FIGURE 8.5. *Mixture example. (Left panel:) Histogram of data. (Right panel:) Maximum likelihood fit of Gaussian densities (solid red) and responsibility (dotted green) of the left component density for observation y, as a function of y.*

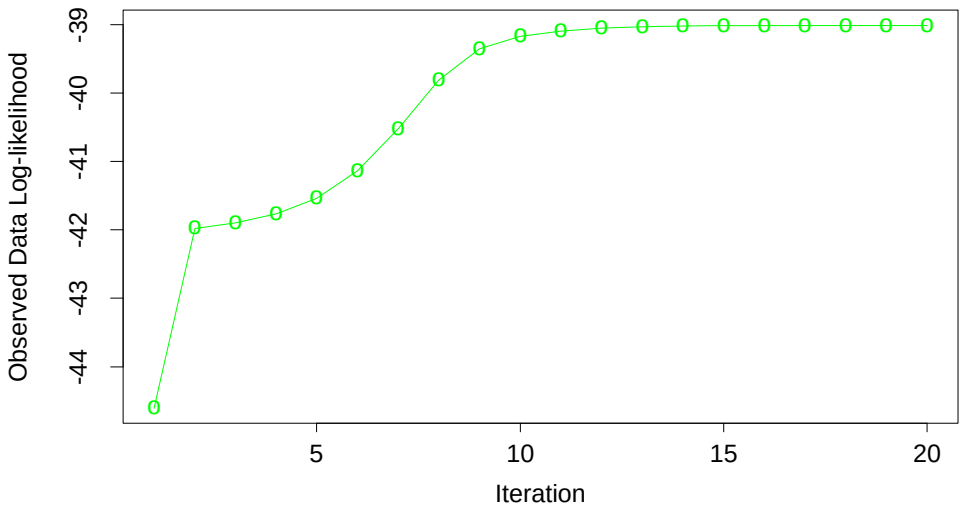

FIGURE 8.6. *EM algorithm: observed data log-likelihood as a function of the iteration number.*

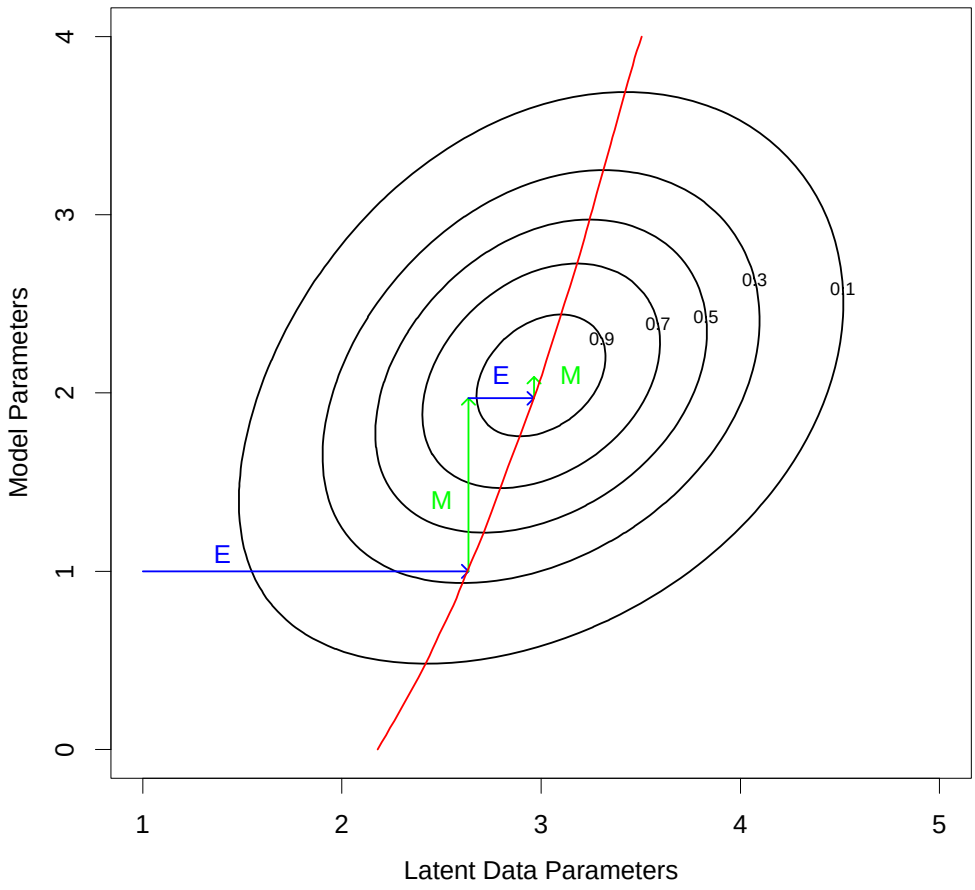

FIGURE 8.7. *Maximization–maximization view of the EM algorithm. Shown are the contours of the (augmented) observed data log-likelihood* $F(\theta', \tilde{P})$. *The E step is equivalent to maximizing the log-likelihood over the parameters of the latent data distribution. The M step maximizes it over the parameters of the log-likelihood. The red curve corresponds to the observed data log-likelihood, a* profile *obtained by maximizing* $F(\theta', \tilde{P})$ *for each value of* θ'.

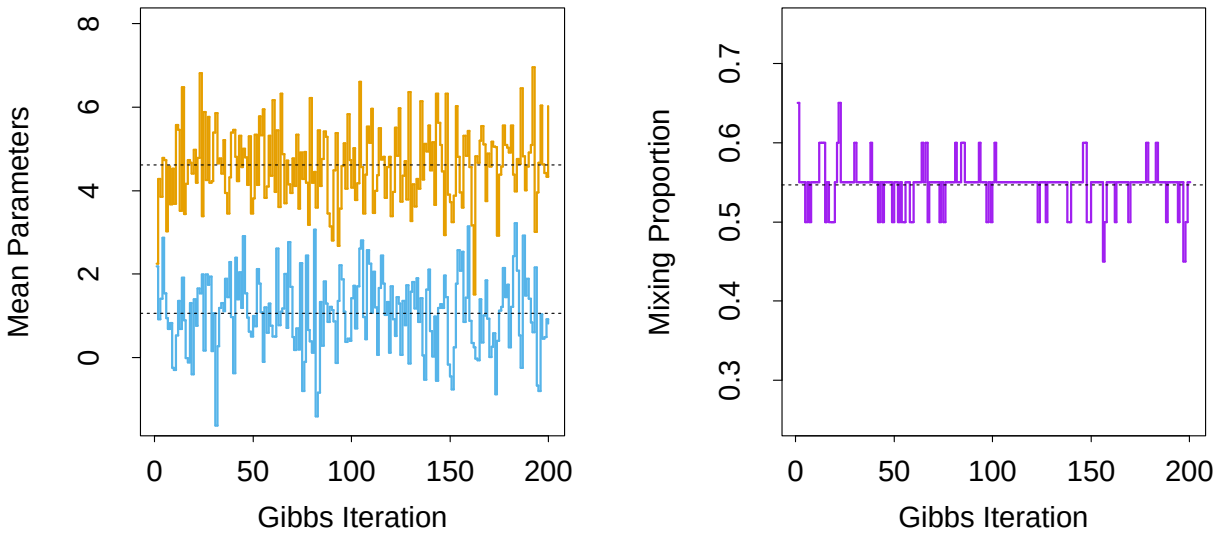

FIGURE 8.8. *Mixture example. (Left panel:)* 200 *values of the two mean parameters from Gibbs sampling; horizontal lines are drawn at the maximum likelihood estimates* $\hat{\mu}_1$, $\hat{\mu}_2$. *(Right panel:) Proportion of values with* $\Delta_i = 1$, *for each of the* 200 *Gibbs sampling iterations; a horizontal line is drawn at* $\sum_i \hat{\gamma}_i / N$.

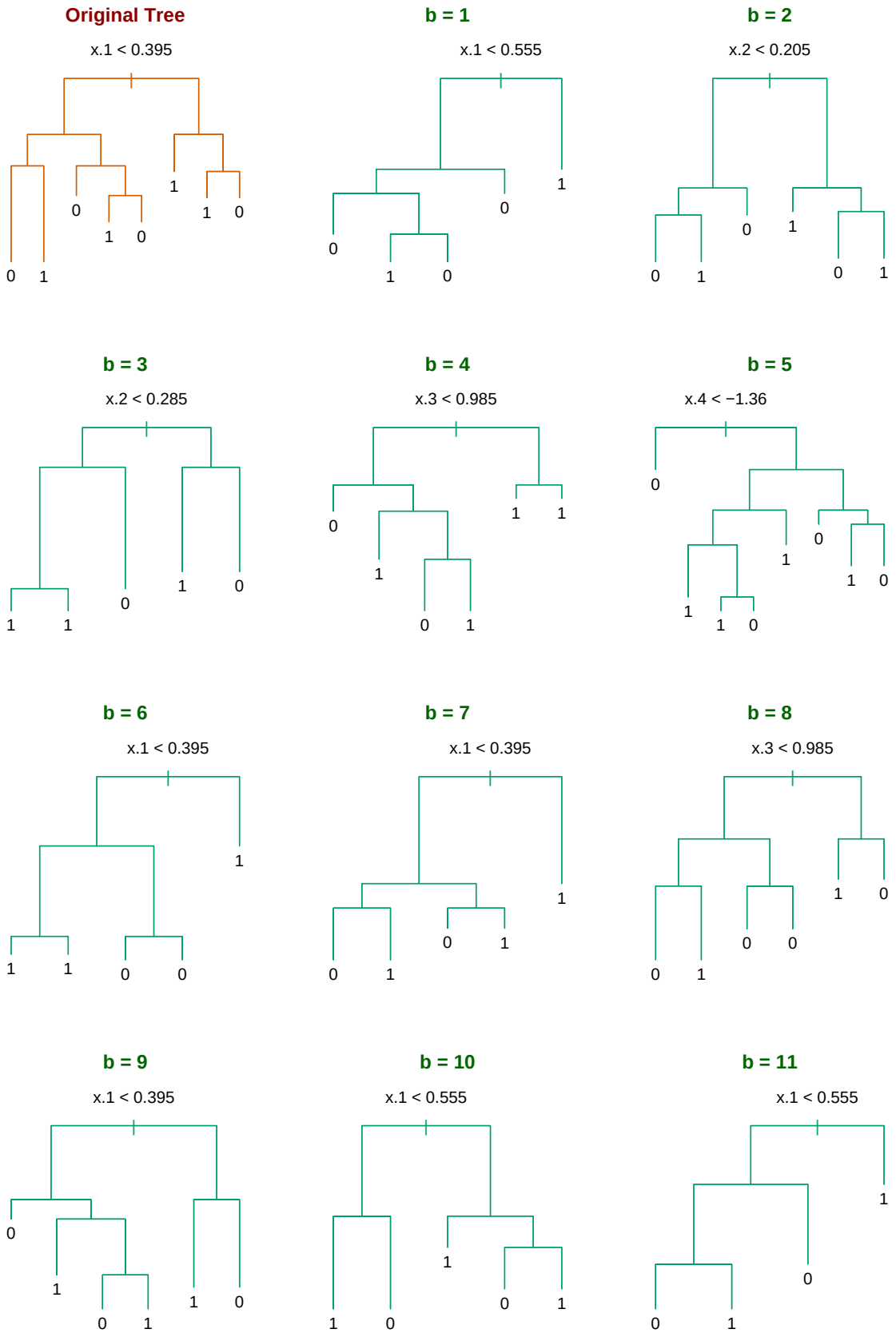

FIGURE 8.9. *Bagging trees on simulated dataset.*

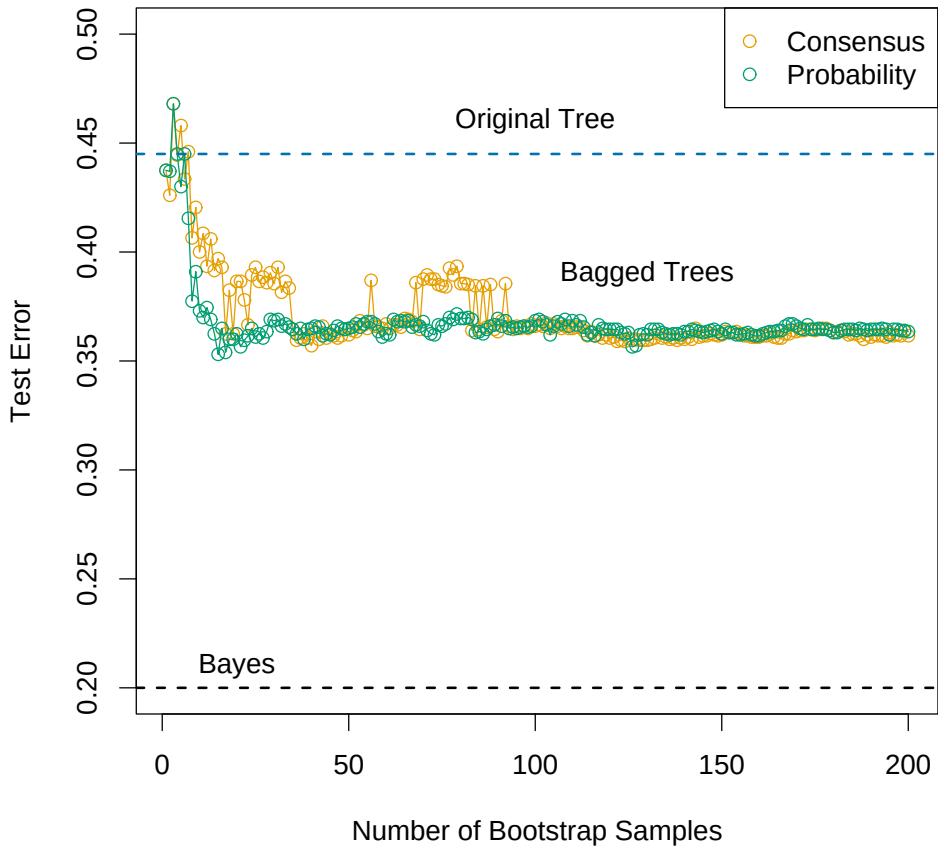

FIGURE 8.10. *Error curves for the bagging example of Figure 8.9. Shown is the test error of the original tree and bagged trees as a function of the number of bootstrap samples. The orange points correspond to the consensus vote, while the green points average the probabilities.*

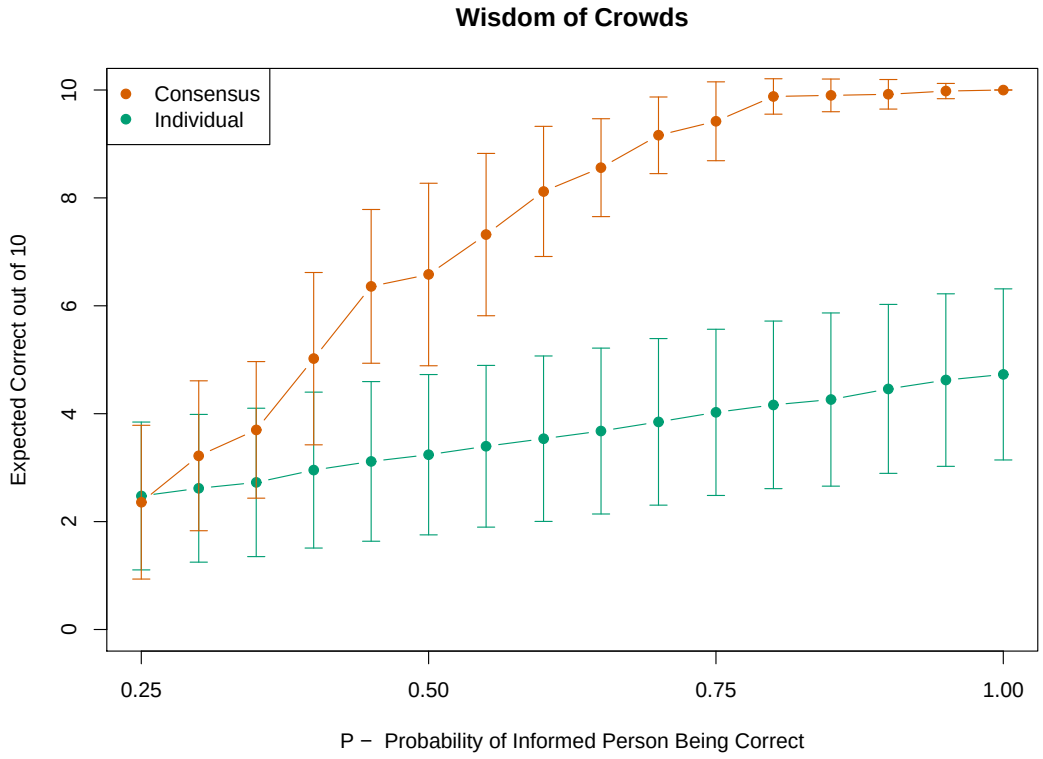

FIGURE 8.11. *Simulated academy awards voting. 50 members vote in 10 categories, each with 4 nominations. For any category, only 15 voters have some knowledge, represented by their probability of selecting the "correct" candidate in that category (so $P = 0.25$ means they have no knowledge). For each category, the 15 experts are chosen at random from the 50. Results show the expected correct (based on 50 simulations) for the consensus, as well as for the individuals. The error bars indicate one standard deviation. We see, for example, that if the 15 informed for a category have a 50% chance of selecting the correct candidate, the consensus doubles the expected performance of an individual.*

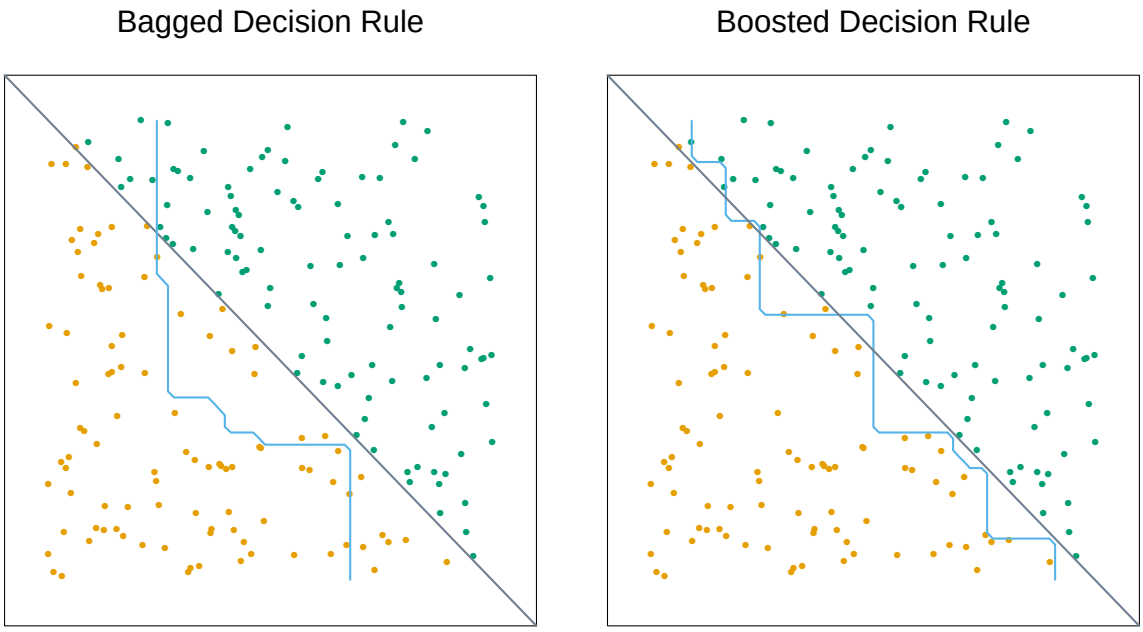

FIGURE 8.12. *Data with two features and two classes, separated by a linear boundary. (Left panel:) Decision boundary estimated from bagging the decision rule from a single split, axis-oriented classifier. (Right panel:) Decision boundary from boosting the decision rule of the same classifier. The test error rates are* 0.166, *and* 0.065, *respectively. Boosting is described in Chapter 10.*

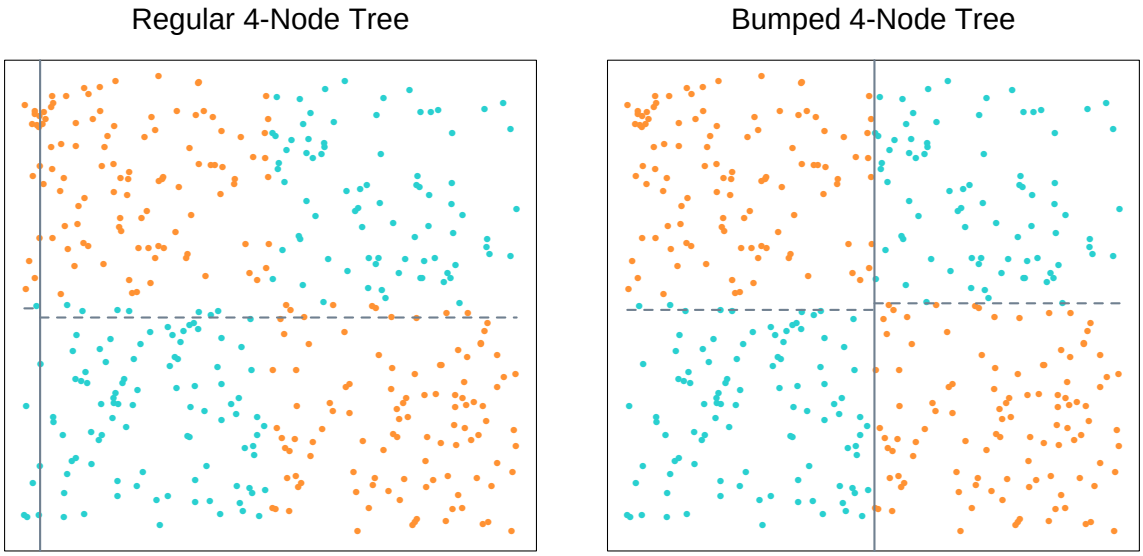

FIGURE 8.13. *Data with two features and two classes (blue and orange), displaying a pure interaction. The left panel shows the partition found by three splits of a standard, greedy, tree-growing algorithm. The vertical grey line near the left edge is the first split, and the broken lines are the two subsequent splits. The algorithm has no idea where to make a good initial split, and makes a poor choice. The right panel shows the near-optimal splits found by bumping the tree-growing algorithm* 20 *times.*