STATS 235: Modern Data Analysis
Nearest Neighbor Methods

Babak Shahbaba

Department of Statistics, UCI

Background

- Given a training set (x, y) of size n, we want to predict the response value, $\tilde{y}$, of a test case (i.e., a future observation) with input values $\tilde{x}$
- So far, we have discussed models that build a map between the input and response variable using some parameters, θ, after which we can forget the training set and use the resulting map to estimate $\tilde{y}$ given $\tilde{x}$
- A possible issue with these method is that they rely on strong assumptions (e.g., linearity, normality), which could lead to better performance if the assumptions are close enough to reality, but they could also fail miserably if the assumptions are unrealistic
- Alternatively, we can avoid making strong assumptions and build memory-based models that remember the original training set and use it for predicting $\tilde{y}$ directly
- To this end, we can find training cases that are close to the test case in the input space and use their average y (or median, or mode) as our estimate of $\tilde{y}$

K-nearest neighbor

- To achieve this, we need a metric to measure closeness and specify k, the number of observations with the closest distance to the test case
- We commonly use Euclidean distance to measure closeness
- For each test case, after measuring its distance to all training cases and identifying the neighborhood N_k with the top k observations (with smallest distance to the test case), we estimate its response value as follows:

$$\hat{y}(\tilde{x}) = \frac{1}{k} \sum_{i \in N_k(x)} y_i$$

- We can use the same approach for regression, with a numerical response variable, and classification, where y is an indicator variable

A binary classification problem with $k = 15$

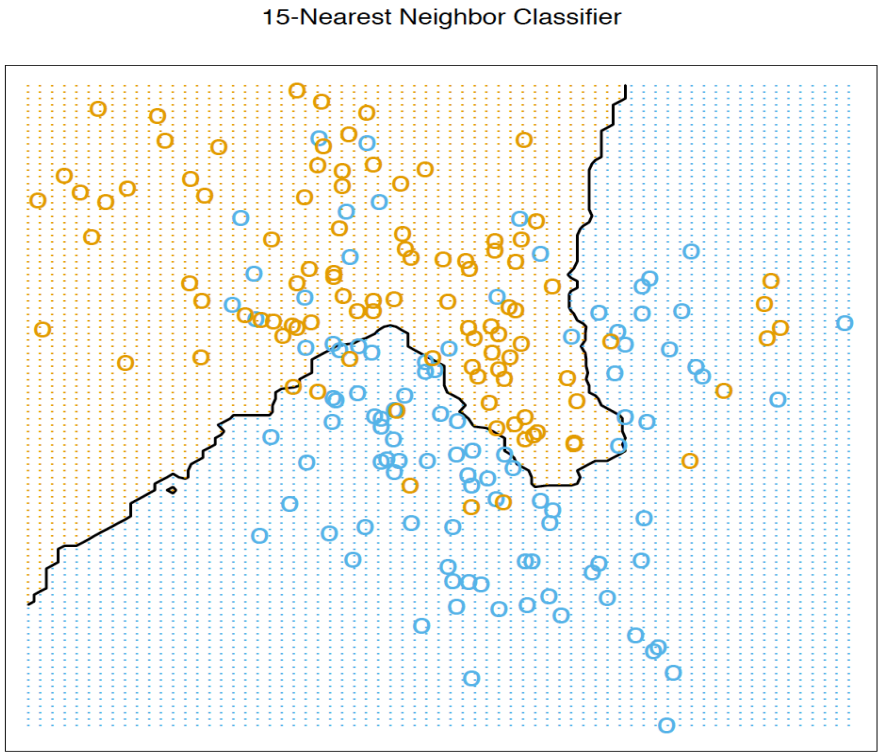

Figure 2.2 Hastie et. al. (2010).

A binary classification problem with $k = 1$

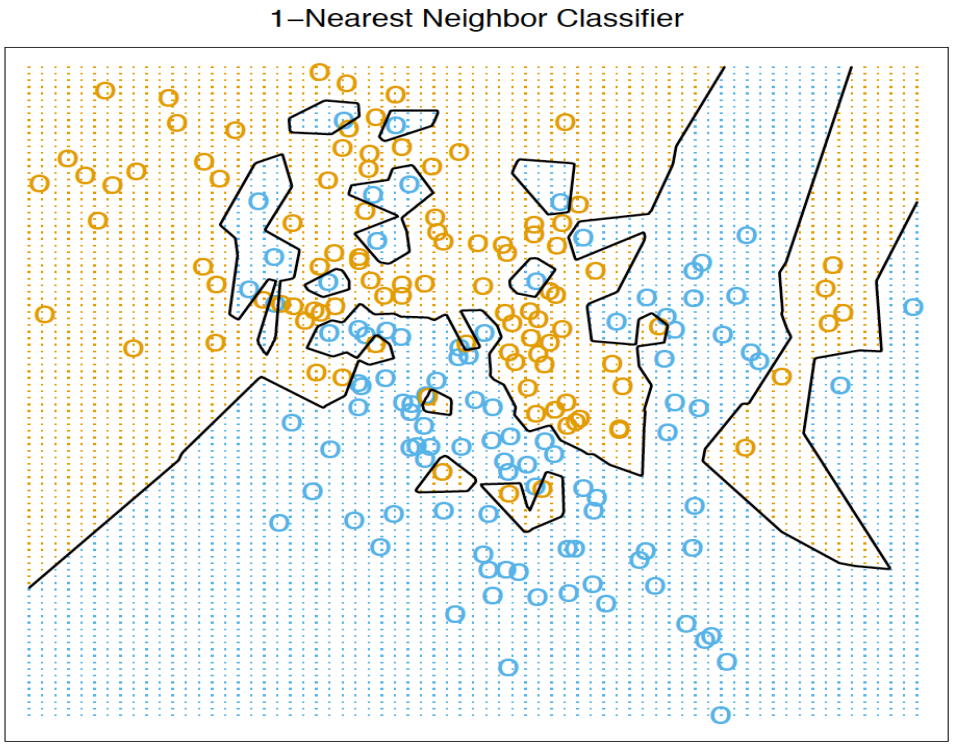

Figure 2.3 Hastie et. al. (2010).

Setting k

- As we can see in this example, the results could be very sensitive to the choice of the tuning parameter k
- With small values of k, we run the risk of overfitting; with large values of k, we average over cases far from the test case
- As usual, we can use cross-validation or data splitting strategy to set this tuning parameter
- Note that although there seems to be a single parameter k, the *effective* number of parameters is n/k, i.e., the number of means we have to estimate assuming the neighborhoods are not overlapping
- Finally, note that this approach assigns 0-1 weights to the training cases; in future, we will discuss *kernel* methods where the weights are a function of distance and go smoothly to zero as distance increases