Nearest Neighbor Classification

NEAREST NEIGHBOR ALGORITHM

Given: Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \dots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.

m-nearest neighbor rule

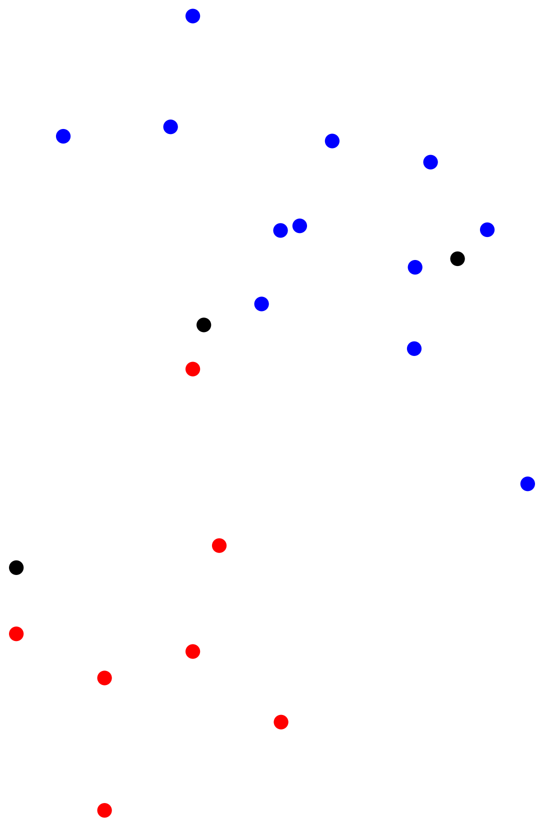
Fix $m \in \mathbb{N}$.

Classify data point $\mathbf{x}$ as:

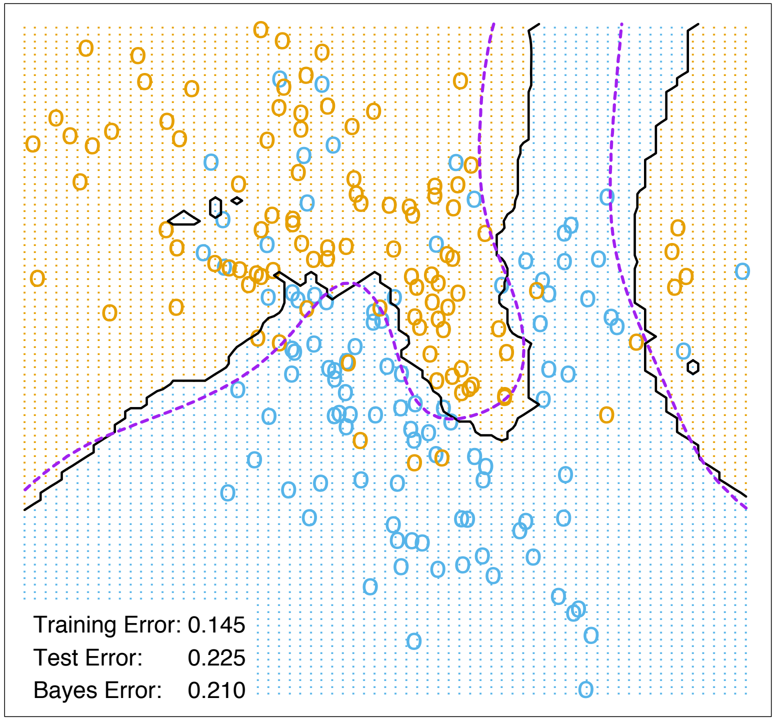
1. Find m training data points that are closest to $\mathbf{x}$ in $\mathbb{R}^d$.
2. Assign $\mathbf{x}$ to the class the majority of these m points belong to.

Remarks

- Works for any number of classes.
- For two classes, m is usually chosen as an odd number to avoid ties. For more than two classes, one has to decide on a tie-breaking strategy in case no single class produces a majority (e.g. choose one of the classes that are in majority at random).
- There is no training algorithm. The training data is used directly to compute the prediction.



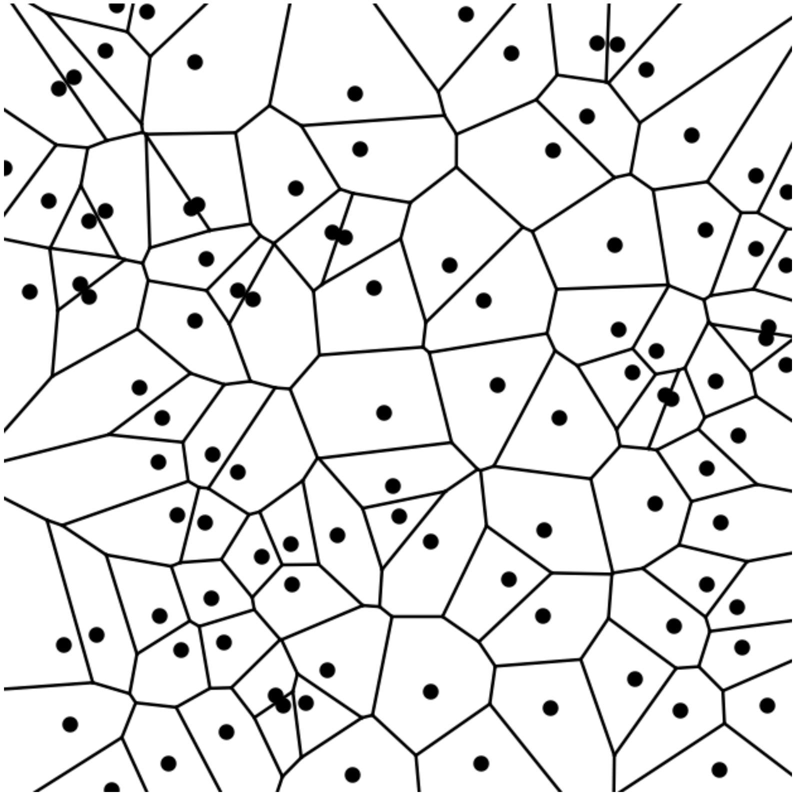
Example: Two classes



7-NN solution

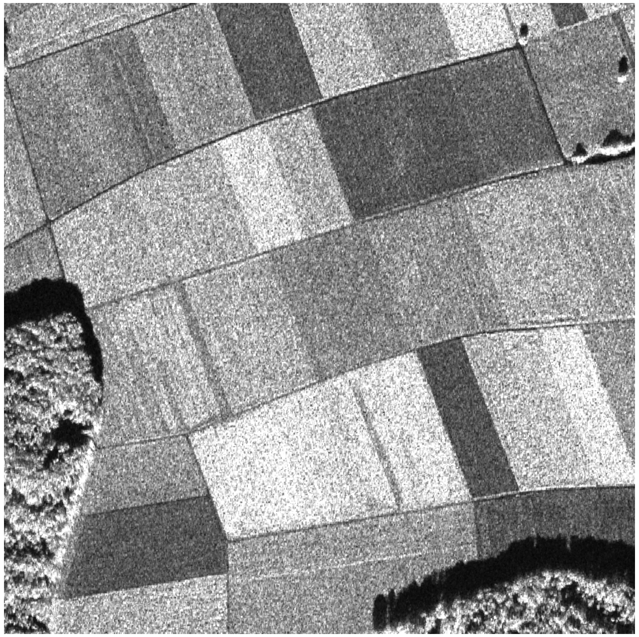
Source: Hastie/Tibshirani/Freedman, “The elements of statistical learning”

For $m = 1$, one can plot subdivide $\mathbb{R}^d$ into regions closest to each training point:

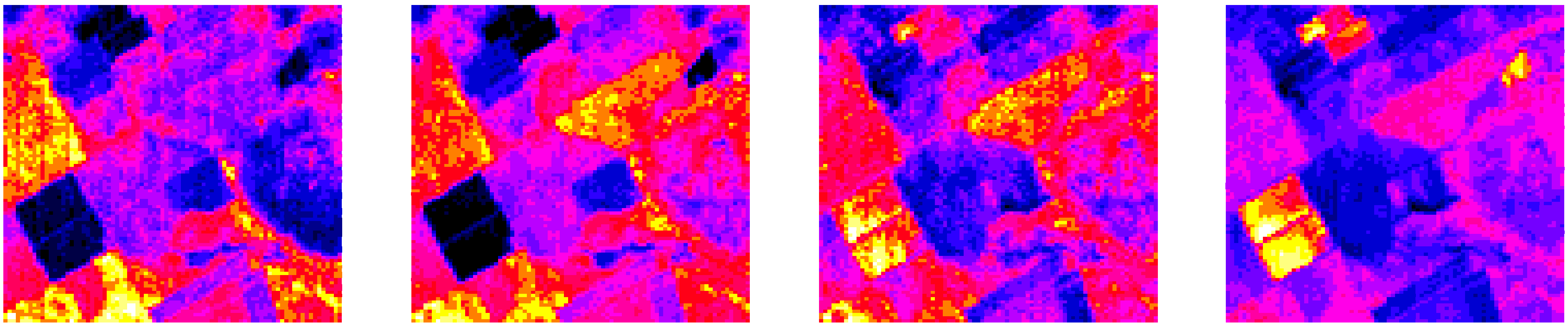


If a data point $\mathbf{x}$ falls into one of the cells, the 1-NN rule will assign it to the class label of the point defining the cell.

For $m > 1$, this becomes harder to plot.



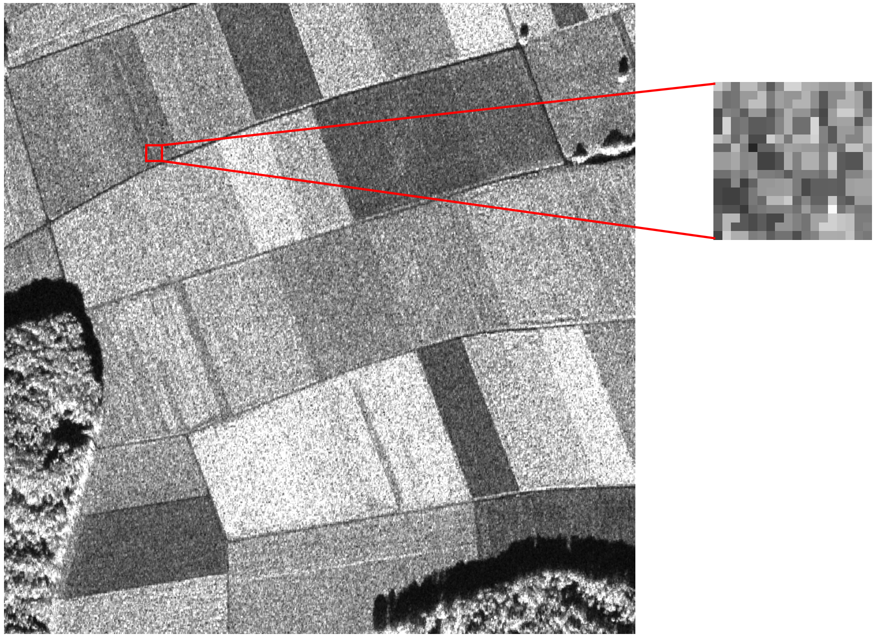
Example: Land Usage Classification



- These are the four “channels” (spectral bands) of a LANDSAT image.
- The land it shows is used for agriculture.
- There are 7 types of land usage (red soil, cotton, . . .).
- For some images, training data is available.
- The goal is to build a classifier that can classify land use in new images.

Source: Hastie/Tibshirani/Freedman, “The elements of statistical learning”

Extracting local image statistics



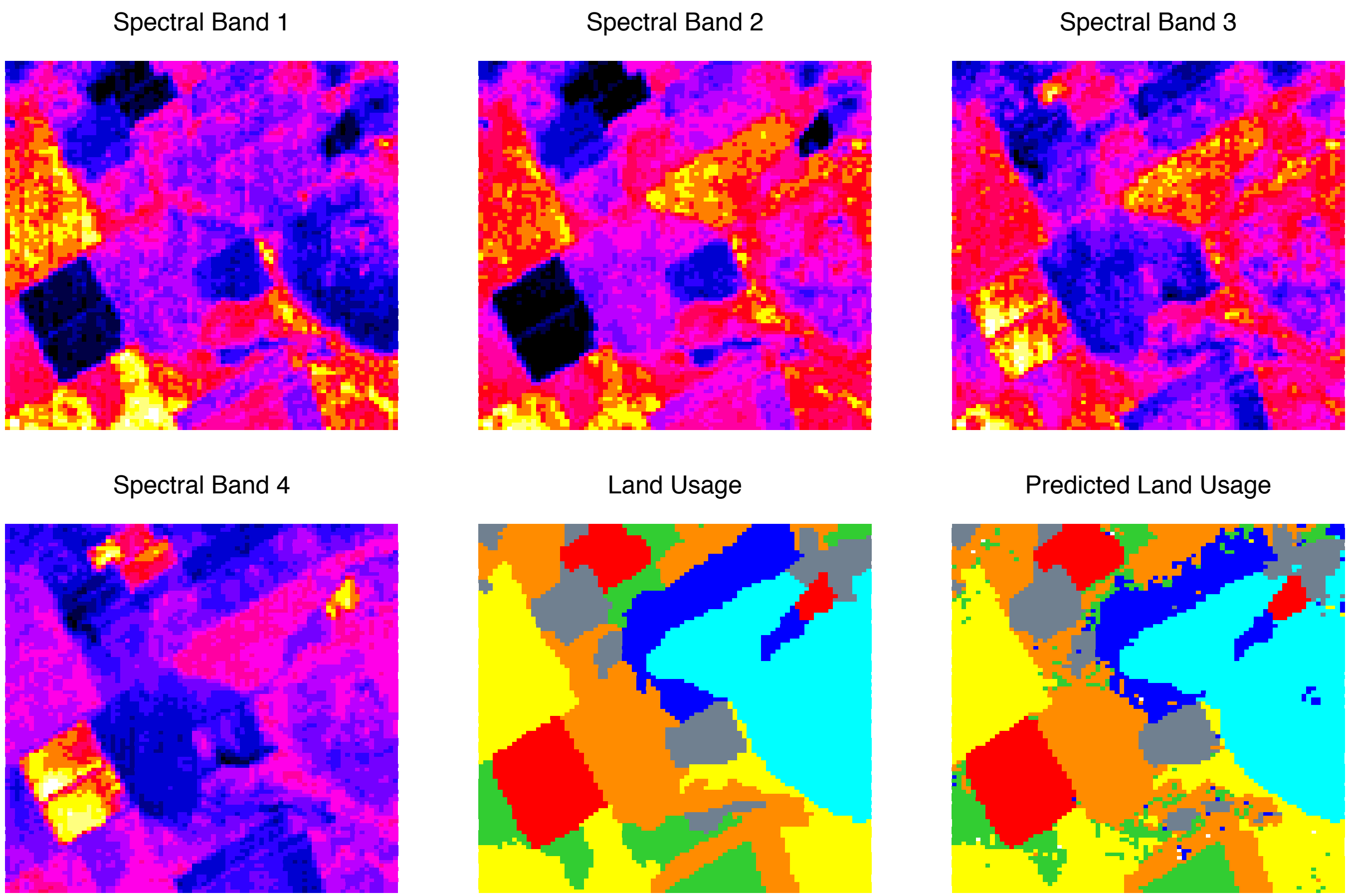
1. Place a small window (size $l \times l$) at the location.
2. Extract the pixel values inside the window. Write them into a vector ($\rightarrow$ dimension l^2).

Resulting data

- We use $l = 3$, so each window contains $3 \times 3 = 9$ pixels.
- Since there are four channels we obtain $9 \times 4 = 36$ scalars characterizing each location.
- We use a nearest neighbor classifier on $\mathbb{R}^{36}$.
- To classify locations in a new image: Again extract a vector using a window, and feed that vector into the NN classifier.

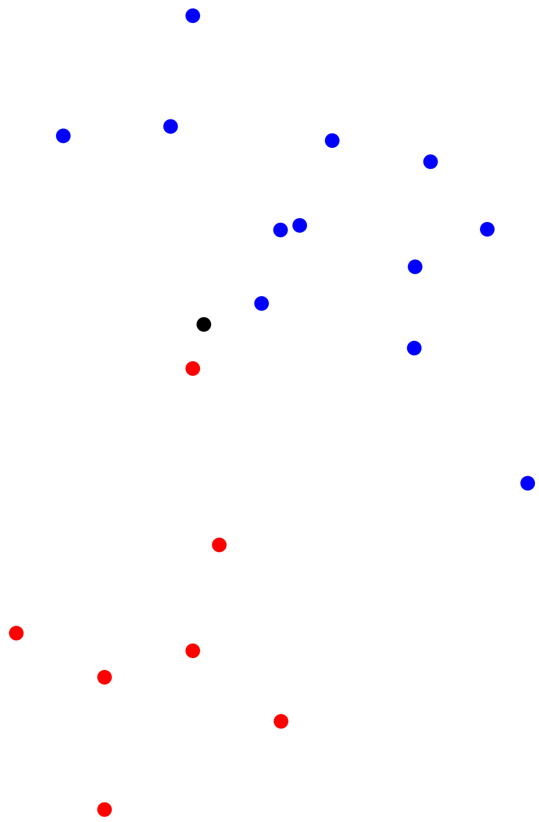
Source: Hastie/Tibshirani/Freedman, "The elements of statistical learning"

Land Usage Prediction



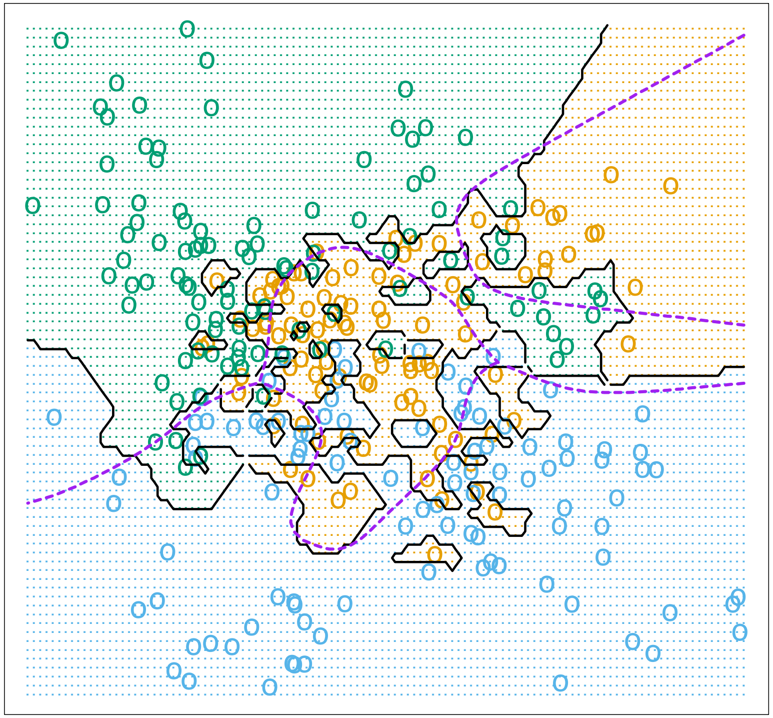
Source: Hastie/Tibshirani/Freedman, “The elements of statistical learning”

EXAMPLE

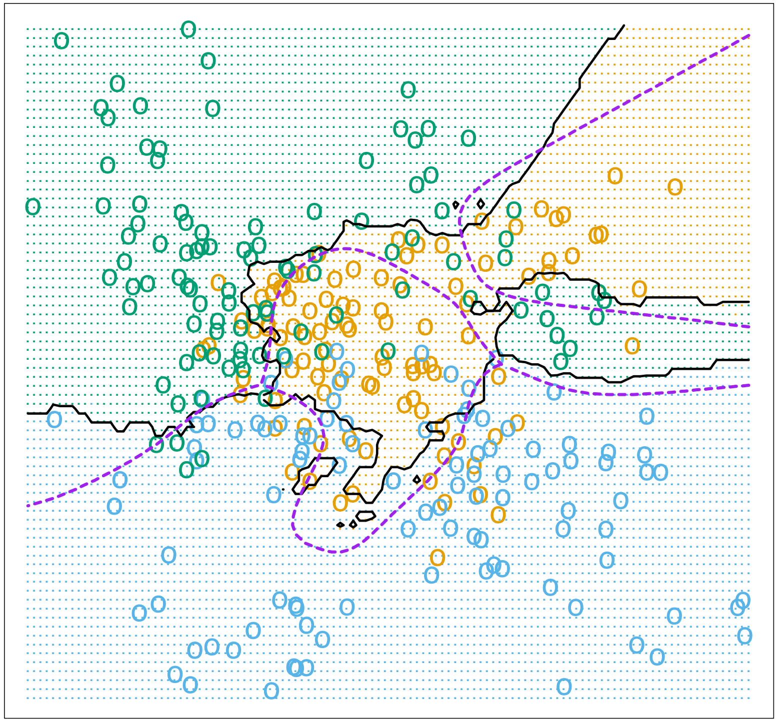


- 1-NN: Classified as “red”.
- 2-NN: Tie.
- m-NN with $m > 2$: Classified as “blue”.

INFLUENCE OF m



1-NN solution



15-NN solution

Source: Hastie/Tibshirani/Freedman, "The elements of statistical learning"

Advantages

- Simple.
- Can be applied to any number of classes.
- Often works very well.

Disadvantages

For large training data sets:

- Requires a lot of memory.
- The entire training set has to be searched for each decision.

Also:

- We are not "learning" anything, even though we can predict.

Evaluating and Tuning a Classifier

What Next

We will consider two problems:

1. Once we have trained a classifier, how do we decide whether it is "good"?
2. We have already seen classifiers with a tuning parameter (e.g. the number of neighbors m in m-NN.) How do we choose the parameter?

Suppose we have trained a classifier f on training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \dots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.
(We use 0-1 loss, so we simply count mistakes.)

Measuring performance

We can measure performance as an "error rate":

$$\text{error rate of } f \;=\; \text{percentage of data points produced by the data source that } f \text{ misclassifies}$$

Note: For 0-1 loss and i.i.d. data, this coincides with the risk of f.

Interpreting the error rate

Consider a two class problem, where each class is equally probable.

- The "baseline" error rate is 50%. Classifiers that do worse than that are irrelevant.

Explanation:

- If we predict by flipping a fair coin (and completely ignore the data), we will achieve 50% error rate.
- If f has error rate $> 50\%$, we can turn it into a classifier with error rate $< 50\%$ by swapping the classes.

How do we measure the error rate in practice?

- We don't have access to the data source itself.
- We only have access to data from the source.
- We can estimate the error rate by measuring it on data: If $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$ are data points, we can compute

$$\text{error rate of } f \approx \frac{\sum_{i=1}^{m} \mathbb{I}\{f(\mathbf{x}_i) \neq y_i\}}{m} = \frac{\text{number of misclassified data point}}{\text{number of data points}}$$

- To do so, we need *labeled* data points.

Can we use the training data?