Machine Learning — Statistical Methods for Machine Learning

The Nearest Neighbour algorithm

Instructor: *Nicolò Cesa-Bianchi* version of March 5, 2019

We now introduce a concrete example of a learning algorithm for classification. For now, we restrict our attention to binary classification tasks with only numerical features, namely $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{-1, +1\}$. Given a training set, the classifier generated by this algorithm is based on the following simple rule: predict every point in the training set with its own label and predict any other point with the label of the point in the training set which is closest to it.

More formally, given a training set S of examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$, the *nearest neighbour* algorithm (NN) generates a classifier $h_{\text{NN}} : \mathbb{R}^d \to \{-1, +1\}$ defined by:

$$h_{\text{NN}}(\boldsymbol{x}) = \text{label } y_t \text{ of the point } \boldsymbol{x}_t \in S \text{ closest to } \boldsymbol{x}.$$

If there is more than one point in S with smallest distance to $\boldsymbol{x}$, then the algorithm predicts with the majority of the labels of these closest points. If there is an equal number of closest points with positive and negative labels, then the algorithm predicts a default value in $\{-1, +1\}$.

Note that $h_{\text{NN}}(\boldsymbol{x}_t) = y_t$ for every training example $(\boldsymbol{x}_t, y_t)$. The distance between $\boldsymbol{x}$ and $\boldsymbol{x}_t$, denoted by $\|\boldsymbol{x} - \boldsymbol{x}_t\|$, is computed with the well-known formula for the Euclidean distance,

$$\|\boldsymbol{x} - \boldsymbol{x}_t\| = \sqrt{\sum_{i=1}^{d} (x_i - x_{i,t})^2} \ .$$

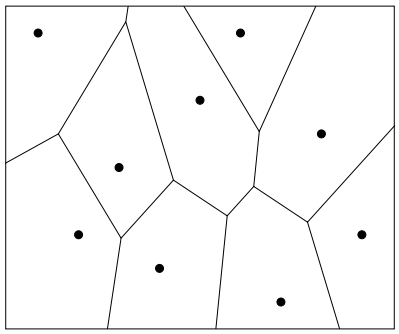

Figure 1: Voronoi diagram for a set of points in the plane.

The classifier generated by NN induces a partition of $\mathbb{R}^d$ in *Voronoi cells*, where each training instance $\boldsymbol{x}_t$ (here called a "center") is contained in a cell, and the border between two cells is the set of points in $\mathbb{R}^d$ that have equal distance from the two cell centers (see Figure 1).

As NN typically stores the entire training set, the algorithm does not scale well with the number of training points. Moreover, given any test point $\boldsymbol{x}$, computing $h_{\text{NN}}(\boldsymbol{x})$ is costly, as it generally

requires computing the distance between $\boldsymbol{x}$ and every point of the training set. Finally, note that NN always generates a classifier h_{NN} such that $\widehat{\ell}(h_{\text{NN}}) = 0$. This is not surprising because, as we already said, NN stores the entire training set.

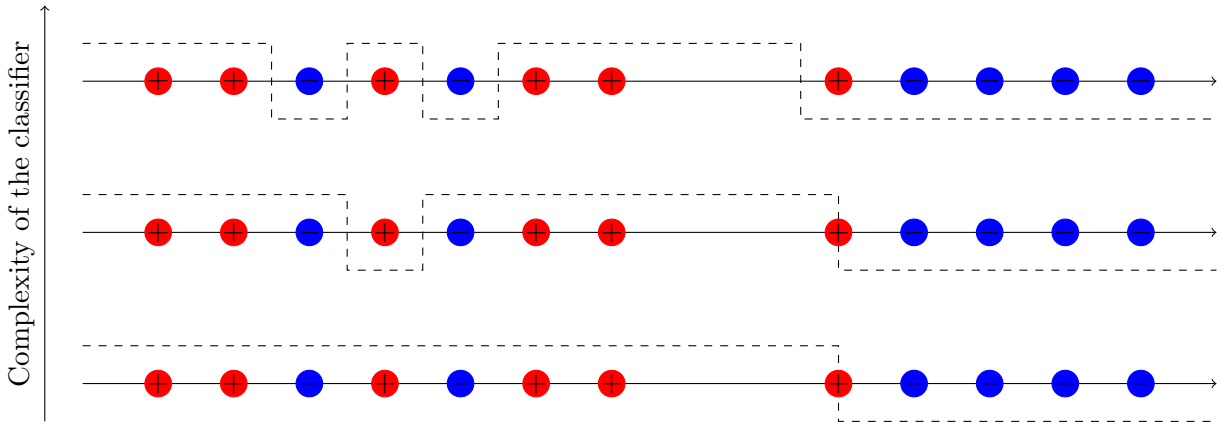

Figure 2: Plot of the $h_{k-\text{NN}}$ classifier for $k = 1, 3, 5$ on a 1-dimensional training set. As k increases, the classifier becomes simpler. What is the training error in each case?

Starting from NN, we can obtain a family of algorithms denoted by k-NN for $k = 1, 3, 5, \ldots$, where k cannot be taken larger than the size of the training set. These algorithms are defined as follows: given a training set $S = (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$, k-NN generates a classifier $h_{k-\text{NN}}$ such that $h_{k-\text{NN}}(\boldsymbol{x})$ is the label $y_t \in \{-1, +1\}$ appearing in the majority of the k points $\boldsymbol{x}_t \in S$ which are closest to $\boldsymbol{x}$.[1] Hence, in order to compute $h_{k-\text{NN}}(\boldsymbol{x})$, we perform the following operations:

1. Find the k training points $\boldsymbol{x}_{t_1}, \ldots, \boldsymbol{x}_{t_k}$ closest to $\boldsymbol{x}$.[1] Let $y_{t_1}, \ldots, y_{t_k}$ be their labels.
2. If the majority of the labels $y_{t_1}, \ldots, y_{t_k}$ is $+1$, then $h_{k-\text{NN}}(\boldsymbol{x}) = +1$; if the majority is -1, then $h_{k-\text{NN}}(\boldsymbol{x}) = -1$; if there is a tie, then $h_{k-\text{NN}}(\boldsymbol{x})$ is set to a default value in $\{-1, +1\}$.

Note that, for each $k \geq 1$, $\boldsymbol{x}_t$ is always included in the k points that are closest to $\boldsymbol{x}_t$.

In addition to binary classification, k-NN can be used to solve multiclass classification problems (where $\mathcal{Y}$ contains more than two symbols) and also regression problems (where $\mathcal{Y} = \mathbb{R}$). In the first case the prediction is —just like in the binary case— the labels corresponding to the majority of the labels of the k closest training points. In the second case, the prediction is the average of the labels of the k closest training points.

It is important to note that, unlike 1-NN, in general we have that $\widehat{\ell}(h_{k-\text{NN}}) > 0$. Moreover, in Figure 2 we see that, as k grows, the classifiers generated by k-NN become simpler. In particular,

[1] Just like in the case of 1-NN, there could be training points at the same distance from $\boldsymbol{x}$ such that more than k points are closest to $\boldsymbol{x}$. In this case we proceed by ranking the training points based on their distance from $\boldsymbol{x}$ and then taking the k' closest points where k' is the smallest integer bigger or equal to k such that the $(k'+1)$-th point in the ranking has distance from $\boldsymbol{x}$ strictly larger that the k'-th point. If no such k' exists, then we take all the points in the training set. If k' is strictly bigger than k, even, and there is an equal number of closest points with positive and negative labels, then the algorithm predicts a default value in $\{-1, +1\}$.

when k is equal to the size of the training set, $h_{k-\text{NN}}$ becomes a constant classifier that always predicts the most common label in the training set.

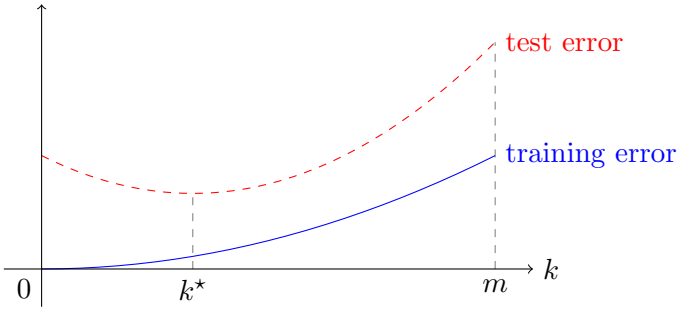

Figure 3: Typical trend of training error (blue solid curve) and test error (red dashed curve) of the k-NN classifier for increasing values of the parameter k. Note that the minimum of the test error is attained at a value $k^\star$ corresponding to a $h_{k-\text{NN}}$ classifier with training error generally bigger than zero. Values $k \ll k^\star$ cause overfitting (small training error and large test error), while values $k \ll k^\star$ cause underfitting (training and test error both large).

Figure 3 shows the typical trend of training error and test error of $h_{k-\text{NN}}$ classifiers for increasing values of k on a dataset with noise. In this case, overfitting takes place because the classifier is complex enough to match the noisy labels.