Machine Learning — Statistical Methods for Machine Learning

The Nearest Neighbor algorithm

Instructor: *Nicolò Cesa-Bianchi* version of January 13, 2026

We now introduce a concrete learning algorithm: the nearest neighbor algorithm (NN) for binary classification tasks with numerical features ($\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{-1, 1\}$). NN is a **non-parametric** learning algorithm: Unlike ERM, searching for a predictor within a fixed class $\mathcal{F}$, NN allows the class to scale with the data size. Given a training set, the classifier generated by NN 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 $S \equiv \big\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_m, y_m)\big\}$, the **nearest neighbour** algorithm A_{NN} with input S 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 $\{-1, 1\}$ (for instance, the most frequent label in the training set).

Note that $h_{\text{NN}}(\boldsymbol{x}_t) = y_t$ for every training example $(\boldsymbol{x}_t, y_t)$. The distance between $\boldsymbol{x} = (x_1, \dots, x_d)$ and $\boldsymbol{x}_t = (x_{t,1}, \dots, x_{t,d})$, denoted by $\|\boldsymbol{x} - \boldsymbol{x}_t\|$, is computed using the Euclidean distance,[1]

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

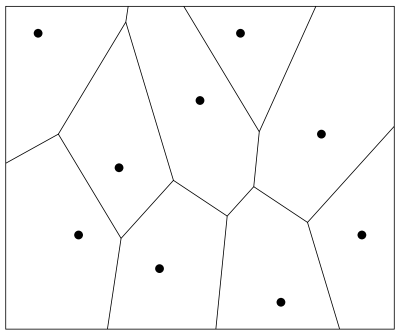


Figure 1: Voronoi diagram for a training set in $\mathbb{R}^2$.

[1] When the numerical features are non-homogeneous and have different scales, it is standard practice to normalize data (e.g., to zero mean and unit variance) before applying NN, ensuring all features contribute equally to the distance metric.

It is important to realize that any binary classifier $f : \mathbb{R}^d \to \{-1, 1\}$ induces a partition of $\mathbb{R}^d$ in two regions: $\{\boldsymbol{x} \in \mathbb{R}^d : f(\boldsymbol{x}) = 1\}$ and $\{\boldsymbol{x} \in \mathbb{R}^d : f(\boldsymbol{x}) = -1\}$. The classifier $h_{\mathrm{NN}} = A_{\mathrm{NN}}(S)$ induces a partition of $\mathbb{R}^d$ in two regions formed by unions of **Voronoi cells**. where each training instance $\boldsymbol{x}_t$ in S is the center of 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). All the points $\boldsymbol{x}$ in the interior of a cell with center $\boldsymbol{x}_t$ are such that $h_{\mathrm{NN}}(\boldsymbol{x}) = y_t$.

As NN typically stores the entire training set, the algorithm does not scale well with the number $|S| = m$ of training points. Moreover, given any test point $\boldsymbol{x}$, computing $h_{\mathrm{NN}}(\boldsymbol{x})$ is costly, as it requires computing the distance between $\boldsymbol{x}$ and every point of the training set, which in $\mathbb{R}^d$ takes time $\Theta(dm)$ (shorter running times are possible when distances are approximated rather than being computed exactly). Finally, note that NN always generates a classifier h_{NN} such that $\ell_S(h_{\mathrm{NN}}) = 0$. This is not surprising because, as we already said, NN stores the entire training set.

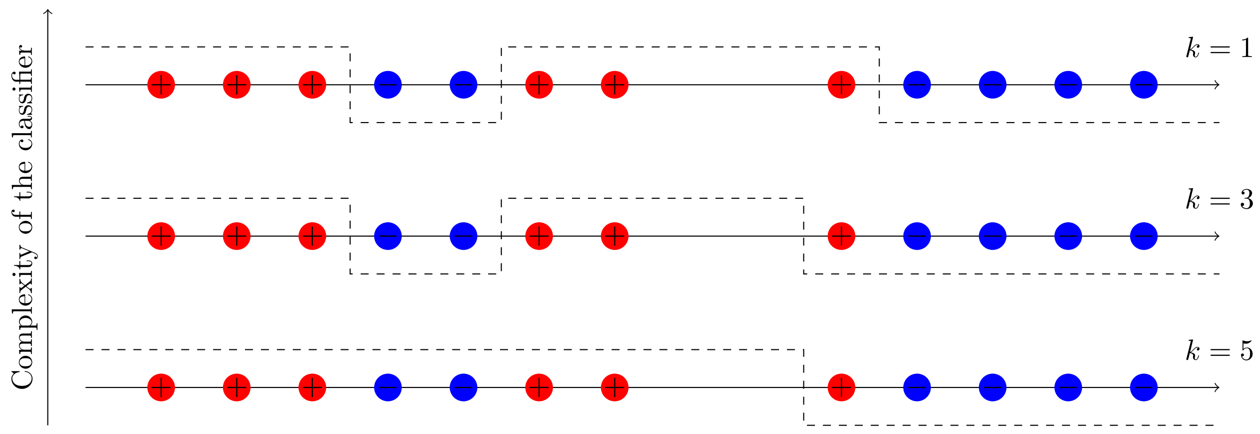


Figure 2: Plot of the $h_{k-\mathrm{NN}}$ classifier for $k = 1, 3, 5$ on a 1-dimensional training set. As k increases, the classifier becomes simpler and the number of mistaken points in the training set increases.

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 $h_{k-\mathrm{NN}}$ such that $h_{k-\mathrm{NN}}(\boldsymbol{x})$ 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 $\boldsymbol{x}$.[2] Hence, in order to compute $h_{k-\mathrm{NN}}(\boldsymbol{x})$, we perform the following operations:

1. Find the k training points $\boldsymbol{x}_{t_1}, \ldots, \boldsymbol{x}_{t_k}$ closest to $\boldsymbol{x}$. 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 $h_{k-\mathrm{NN}}(\boldsymbol{x}) = +1$; if the majority is -1, then $h_{k-\mathrm{NN}}(\boldsymbol{x}) = -1$.

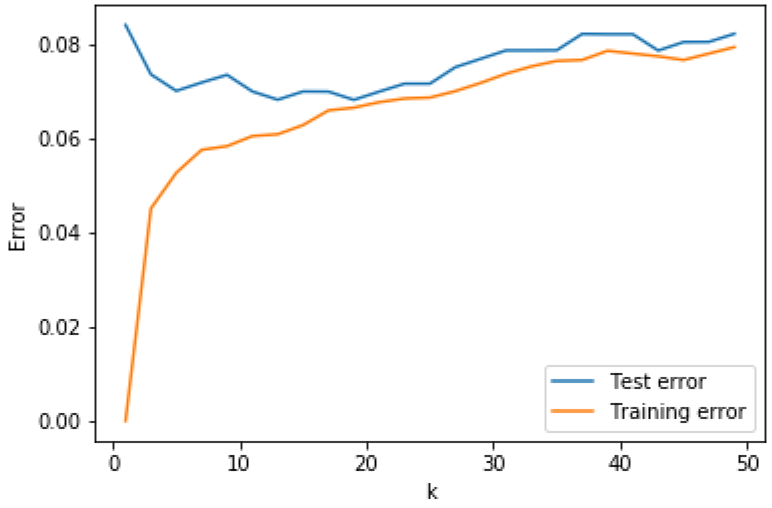
Note that, for each $k \geq 1$ and for each $\boldsymbol{x}_t$ in the training set, $\boldsymbol{x}_t$ is always included in the k points that are closest to $\boldsymbol{x}_t$.

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

[2] Ties in distance are resolved arbitrarily (e.g., by index).

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

The geometry of the k-NN classifiers is still a partition of $\mathbb{R}^d$ in Voronoi cells. However, whereas in 1-NN a Voronoi cell contains all the points which have the same closest point in the training set, in k-NN (for $k > 1$) the Voronoi cells contain all the points that have the same k closest neighbors in the training set. For any $k \geq 1$, the Voronoi cells of k-NN are defined by the intersection of a finite number of halfspaces.



The figure above shows the typical trend of training error (orange curve) and test error (blue curve) of the k-NN classifier for increasing values of the parameter k on a real dataset (Breast Cancer Wisconsin) for binary classification with zero-one loss. Note that the minimum of the test error is attained at a value corresponding to a $h_{k-\mathrm{NN}}$ classifier with training error generally bigger than zero. The learning algorithm suffers from high test error for small values of k (overfitting) and for large values of k (underfitting). Unlike ERM, where the trade-off between overfitting and underfitting is determined by the size of the training set relative to the size of $\mathcal{F}$, in this case the trade-off is determined by how the complexity of the k-NN predictor (controlled by k) scales with the training set size.

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, we operate like in the binary case and predict using the label 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.

For regression, using a simple average can be sensitive to outliers. A common variant is distance-weighted k-NN, where neighbors contribute to the prediction proportional to the reciprocal of their distance. This aligns with the statistical intuition that closer points provide more information than faraway points.