9

Supervised learning: Practice and theory of classification with the k-NN rule

A concise summary is provided at the end of this chapter, in §9.8.

9.1 Supervised learning

In supervised learning, we are given a labeled *training set* $Z = \{(x_i, y_i)\}_i$ with $y_i \in \pm 1$ (the ground truth labeled data) and the task is to learn a *classifier* so that we can classify new unlabeled observations of a *testing set* $Q = \{x'_i\}_i$. We shall see in this chapter a very simple algorithm that is nonetheless provably good to classify data: The *k-Nearest Neighbor rule*, or *k-NN rule* for short. When the training set has only two classes, we deal with *binary classification*, otherwise it is a multi-class classification problem. Statistical learning assumes that both the training set and the testing set are independently and identically sampled for an arbitrary but fixed unknown distribution.

9.2 Nearest neighbor classification: NN-rule

The *nearest neighbor classification rule* assigns a label to an element x as the class $l(x) \in \{-1, +1\}$ where $l(x)$ is the label of the closest labeled point

$\mathrm{NN}(x)$ in the training set: $l(x) = l(\mathrm{NN}_Z(x))$. That is, we have $l(x) = y_e$ for $e = \arg\min_{i=1}^{n} D(x, x_i)$ where $D(\cdot, \cdot)$ is an appropriate distance function (usually taken as the Euclidean distance). Let us notice that in case there exist several points yielding the same minimal distance, we choose arbitrarily one of those points. For example, we may use the *lexicographic order* on Z, and report the lowest index point of Z in case of nearest neighbor ties. Thus the notion of "nearest neighbor" is defined according to an appropriate distance function $D(\cdot, \cdot)$ between any two elements. For example, we have already surveyed in the previous chapters, the Euclidean distance $D(p, q) = \sqrt{\sum_{j=1}^{d} (p^j - q^j)^2}$ or a generalization called the Minkowski distances $D_l(p, q) = \left(\sum_{j=1}^{d} |p^j - q^j|^l \right)^{\frac{1}{l}}$ (metrics when $l \geq 1$) for numerical attributes, and the Hamming distance $D_H(p, q) = \sum_{j=1}^{d} (1 - \delta_{p^j}(q^j)) = \sum_{j=1}^{d} 1_{[p^j \neq q^j]}$ for categorical attributes (agreement distance). We denote by $\delta_x(y) = 1$ the *Dirac function* that is equal to 1 if and only if $y = x$, and 0 otherwise.

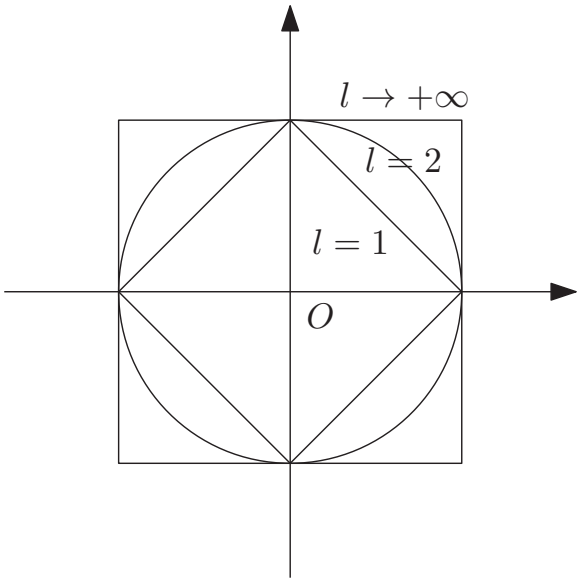

Figure 9.1 Minkowski balls $\{x \in \mathbb{R}^d \mid D_l(O, x) \leq 1\}$ with $D_l(p, q) = \left(\sum_{j=1}^{d} |p^j - q^j|^l \right)^{\frac{1}{l}} = \|p - q\|_l$ for different values of $l \geq 1$. For $l = 2$, we get the ordinary Euclidean ball. For $l = 1$, we obtain the Manhattan ball (with a "square" shape), and when $l \to +\infty$ we tend to a square shape, oriented 45 degrees apart the Manhattan ball.

We classify $t = |Q|$ new unlabeled observations of the testing set by answering t nearest neighbor (NN) queries in X. It is crucial to answer these queries as fast as possible, at least in sub-linear time so that we can beat the naive algorithm that scans all the points of X. There exists many data-structures to answer such NN queries but as the dimension d increases, it becomes (provably) difficult to beat significantly the naive algorithm. This is the phenomenon that bears the name of the *curse of dimensionality*! Historically, this curse of dimensionality was introduced by Bellman, the founder of the dynamic programming paradigm.

9.2.1 Optimizing Euclidean distance computation for nearest neighbor queries

We often choose the Euclidean distance as the underlying distance, and the NN queries can be optimized in practice. Indeed, let us first notice that a distance or a monotonically increasing function of this distance, like the square function, does not change the *relative ordering* of points according to a query point q. That is, we have $l = \arg\min_{i=1}^{n} D(q, x_i) = \arg\min_{i=1}^{n} D^2(q, x_i)$. This is useful observation as the squared Euclidean distance is easier to handle mathematically. Indeed, computing the squared Euclidean distance between two vector attributes of d dimensions amounts to compute d subtractions, d square operations, and $d-1$ sums. That is $3d-1$ elementary arithmetic operations. We can also interpret the squared Euclidean distance (or any other norm-based induced distance) as $D^2(q, x_i) = \langle q - x_i, q - x_i \rangle$, where $\langle x, y \rangle = x^\top y$ is the *scalar product* (technically, the Euclidean space can also be interpreted as a Hilbert space equipped with the dot product). Computing a scalar product between two d-dimensional vectors requires $2d-1$ operations. Now, if we preprocess the computation of the squared norms $\mathrm{norm}^2(p) = \langle p, p \rangle = \sum_{j=1}^{d} (p^j)^2$ in $(2d-1)n$ time for the points of the training set Z, with $|Z| = n$, then we can compute $D^2(p, q)$ as $D^2(p, q) = \mathrm{norm}^2(p) + \mathrm{norm}^2(q) - 2\langle p, q \rangle$: That is, it amounts to perform t scalar products for each query of the test set Q with $|Q| = t$. To classify the t unlabeled data, the naive method requires $(3d-1)nt$ while the method that preprocess by computing the $n+t$ norms in $(2d-1)(n+t)$ time requires an overall time $(2d-1)(n+t)+t(2+2d-1)$. Therefore for $t \ll n \ll d$, the obtained speed-up is $\frac{3dnt}{4dt} = 3n$. In practice, one can use the *Graphical Processing Units* (*GPUs*) of modern PCs that allows to quickly compute scalar product internally.

9.2.2 Nearest Neighbor (NN) rules and Voronoi diagrams

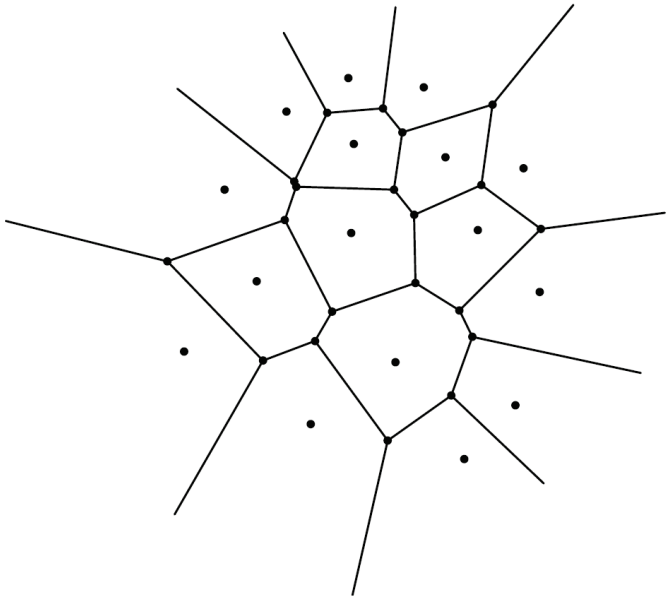

Figure 9.2 Example of a planar Voronoi diagram that partitions the space into proximity cells.

For a given training set Z with $n = |Z|$ d-dimensional numerical data elements, the space $\mathbb{R}^d$ is partitioned into n equivalence classes for the nearest neighbor labeling function (where the arg min is constant). These are precisely the *Voronoi cells* that decompose the space into proximity cells. We have already introduced the Voronoi diagrams in the k-means clustering chapter. We quickly recall that Voronoi diagram of a finite set of points $X = \{p_1, ..., p_n\}$ of $\mathbb{R}^d$ (called the Voronoi generators) partitions the space into Voronoi cells that are proximity cells. A Voronoi cell $V(x_i)$ is defined as the set of points of $\mathbb{R}^d$ that is closed to x_i than to any other generator x_j (with $j \neq i$). That is, $V(x_i) = \{x \in \mathbb{R}^d \mid \|x - x_i\| \leq \|x - x_j\| \ \forall j \neq i\}$. Figure 9.2 depicts a planar Voronoi diagram (observe that there are unbounded cells).

Here, we consider only bi-chromatic generators (two classes '-1' and '+1', or red/blue sites) for the Voronoi diagram. Thus the bichromatic Voronoi diagram decomposes the space into two types of colored cells, and the boundary between these color changes indicates the *decision boundary* of the *NN classifier*. Figure 9.3 illustrates these geometric aspects. Note that in practice one cannot

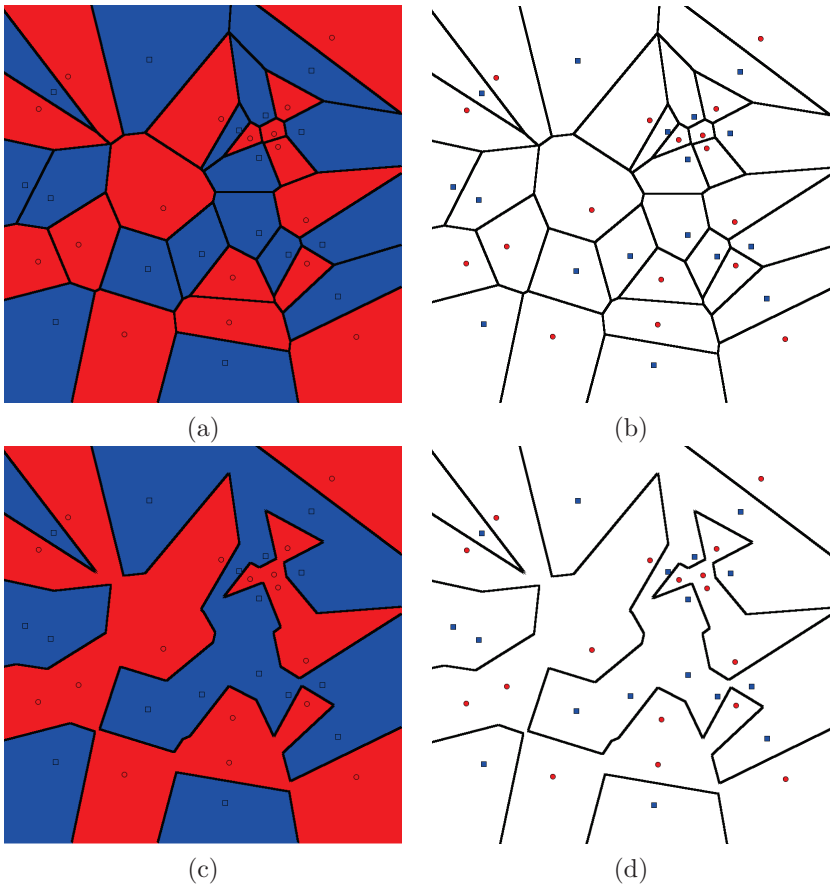

Figure 9.3 k-NN classification rules and bi-chromatic Voronoi diagrams: (a) bichromatic Voronoi diagram, (b) Voronoi bi-chromatic bisectors, classifier using the 1-NN rule (classes are monochromatic union of Voronoi cells), and (d) boundary decision defined as the interface of these two classes.

compute Voronoi diagrams in high dimension because of their exponential combinatorial complexity. Nevertheless, the facets of the bi-chromatic Voronoi diagrams that support cells of different colors define precisely the decision boundary. Thus the NN-rule has a piecewise linear decision boundary since bisectors (*i.e.*, the locii of points at equidistance to two generators) are hyperplanes.

9.2.3 Enhancing the NN-rule with the k-NN rule by voting!

In order for the classifier to be resilient to noisy data-sets (say, imprecise input and outliers), we may class a new observation q by choosing among the first k nearest neighbors of X, the dominant class. For binary classification, it is useful to choose an odd value for k in order to avoid vote ties. In practice, increasing k allows one to be tolerant to outliers in the data-set, but the drawback is that the decision boundary becomes more fuzzy as k increases. There exit many techniques or rules of thumbs to choose the most appropriate value of k for this k-NN rule. For example, the *cross-validation* method that uses part of the training set to train, and the remaining part to test. Note that the k-NN voting rule generalizes the NN-rule (by choosing $k = 1$, NN=1-NN). When dealing with multi-classes, the rule consists in choosing the dominant class inside the k-NNs. Technically speaking, the space $\mathbb{R}^d$ can also be decomposed into elementary cells, the k-order Voronoi cells (see exercise 9.9), where inside a cell, the k closest neighbor sites does not change. The k-order Voronoi diagram is an affine diagram and the k-NN decision boundary is also piecewise linear.

9.3 Evaluating the performance of classifiers

We described a family of piecewise linear classifier based on the k-NN rule to classify observations using the labeling function $l_k(x)$ that returns the majority class of the k nearest neighbor in the training set Z of point to classify, x. In order to choose the best classifier in that family, one needs to be able to assess the performance of classifiers.

9.3.1 Misclassification error rate

The *misclassification rate* or *error rate* on a testing set Q with t unlabeled observations to classify is simply defined by:

$$\boxed{\tau_{\text{Error}} = \frac{\#\text{misclassified}}{t} = 1 - \frac{\#\text{correctly classified}}{t} = \tau_{\text{misclassification}}}$$

This indicator is not discriminative when the class label proportions are unbalanced (in the testing set or even in the training set). For example, when we classify email messages into C_{spam} for spam and C_{ham} for non-spam (good

emails), we notice that we often have far less spams than regular emails. Thus if we seek to minimize the misclassification error rate, then it would suffice to classify non-spam all emails, and thus achieve a good error rate! This highlights the problem of taking into consideration the relative proportion of classes.

9.3.2 Confusion matrices and true/false positive/negative

The *confusion matrix* $M = [m_{i,j}]_{i,j}$ stores its coefficients $m_{i,j}$ of well-classified rates when x is classified as C_i (estimated class) with the ground-truth class being C_j:

$$\boxed{M = [m_{i,j}]_{i,j}, \quad m_{i,j} = \tau_{(x \text{ predicted as } C_i | x \in C_j)}}$$

For binary classification (*i.e.*, two classes), consider the following 2×2 array that indicates the four cases with

True	prediction is correct
False	prediction is wrong
Positive	predicted label is class C_{+1}
Negative	predicted label is class C_{-1}

		Predicted label	
		C_{+1}	C_{-1}
true label	C_{+1}	True Positive (TP)	False Negative (FN)
	C_{-1}	False Positive (FP)	True Negative (TN)

The diagonal of the confusion matrix M indicates the successful rate for all classes. This misclassified data can either by *false positive* (FP) or *false negative* (FN):

- A false positive (FP) is an observation x misclassified as C_1 (positive class) albeit it is C_{-1} (negative class). "Positive" means '+1' in this context.
- Similarly, a false negative (FN), is an observation x misclassified as C_{-1} (negative class) albeit it is C_{+1} (positive class). In this context, "negative" means '-1'.

The false positive are also called *type I error*, and the false negative are called *type II error*. Similarly, we define the *true negative* (TN) and the *false negative* (TP). Thus, the error rate can be rewritten as:

$$\tau_{\text{error}} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = 1 - \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}},$$

since $\text{TP} + \text{TN} + \text{FP} + \text{FN} = t = |Q|$, the number of queries to classify.

9.4 Precision, recall and F-score

We define the *precision* as the proportion of true positive in the true class (the TP and FP data):

$$\tau_{\text{Precision}} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

We can easily check that $0 \leq \tau_{\text{Precision}} \leq 1$. The precision is the percentage of correctly classified elements in the positive class.

The *recall rate* is the proportion of true $+1$ (TP) in the data classified $+1$ (TP and FN):

$$\tau_{\text{recall}} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

The *F-score* is a rate that is constructed in order to give as much weight to the false positive as to the false negative. It is defined as the harmonic mean[1], and is often used in practice:

$$\tau_{\text{F-score}} = \frac{2 \times \tau_{\text{Precision}} \times \tau_{\text{Recall}}}{\tau_{\text{Precision}} + \tau_{\text{Recall}}}$$

In practice, we choose classifiers that yield the best F-scores. For example, for several odd values of k, one can evaluates the k-NN classification rule using the F-score, and finally choose the best value of k that gave the best F-score.

9.5 Statistical machine learning and Bayes' minimal error bound

Nowadays, in the era of big data, it is reasonable to assume that both the training set Z and the test set Q can be modeled by statistical distributions (from generative models having probability densities). Classifier performances can then be studied mathematically. Let us assume that X (from $Z = (X, Y)$) and Q are two data sets, called observations, that are identically and independently distributed (*iid*) samples from random variables X, Y and Z. We write $X \sim_{\text{iid}} \mathcal{D}$ to state that X has been sampled iid. from a probability law $\mathcal{D}$ (say, a Gaussian distribution). An *univariate distribution* has its support in $\mathbb{R}$, the real line. Otherwise, we have *multivariate distributions* (say, with support in $\mathbb{R}^d$). We can interpret X as a random vector of dimension $n \times d$. Let us recall

[1] The harmonic mean is defined by $h(x, y) = \frac{1}{\frac{1}{2}\frac{1}{x} + \frac{1}{2}\frac{1}{y}} = \frac{2xy}{x+y}$. It is often used to average ratio quantities.

the probability fact that two random variables X_1 and X_2 are *independent* iff. $\Pr(X_1 = x_1, X_2 = x_2) = \Pr(X_1 = x_1) \times \Pr(X_2 = x_2)$. Statistical modeling allows to consider X as a statistical mixture. The density of a statistical mixture can mathematically be written as: $m(x) = w_1 p_1(x) + w_2 p_2(x)$ with w_1 and w_2 *a priori probabilities* of belonging to classes C_1 and C_2 ($w_1 = 1 - w_2$), and $p_1(x) = \Pr(X_1|Y_1 = C_1)$ and $p_2(x) = \Pr(X_2|Y_2 = C_2)$ the *conditional probabilities*. We seek for a classifier that yields a good performance in the large sample limit: That is, asymptotically when $n \to +\infty$.

9.5.1 Non-parametric probability density estimation

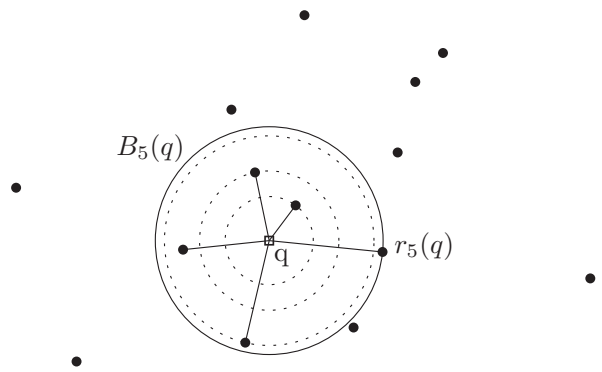

Figure 9.4 Illustration of a k-NN query for $k = 5$. The ball covering the k-NN has radius $r_k(q)$. The radius allows to estimate locally the underlying distribution of X by $p(x) \approx \frac{k}{nV(B_k(x))} \propto \frac{k}{nr_k(x)^d}$.

Given an iid observation set $X = \{x_1, ..., x_n\}$ that we assume sampled from a fixed but unknown density $p(x)$, we seek to model the underlying distribution. For a parametric law $p(x|\theta)$ (that belongs to a family of distributions indexed by a parameter vector θ), this amounts to estimate the parameter θ of this distribution. For example, for a Gaussian distribution $p(x|\theta = (\mu, \sigma^2))$, we estimate with the *maximum likelihood estimator* (MLE) the mean as $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and the (unbiased) variance as $v = \sigma^2$ with $\widehat{\sigma^2} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$. When the distributions is not indexed by a fixed-dimensional parameter, we say that the distribution is non-parametric. The parametric distributions are often (but not necessarily) unimodal[2] and these models lack flexibility to model

[2] The modes of a density function are its local maxima.

complex *multimodal density*. The *non-parametric density modeling* method is far more flexible since it allows to model any smooth density, including all multimodal smooth distributions. We state a key theorem in non-parametric statistical modeling:

Theorem 10

The balloon estimator allows to approximate a smooth density $p(x)$ with support in $\mathbb{R}^d$ by $p(x) \approx \frac{k}{nV(B)}$, where k is the number of samples of X that is contained in the ball B, and $V(B)$ its volume.

Proof

Let P_R denote the probability that a sample x falls inside a region R: $P_R = \int_{x \in R} p(x) \mathrm{d}x$. The probability that k of n samples fall inside R is thus given by the binomial law:

$$P_R^{(k)} = \binom{n}{k} P_R^k (1 - P_R)^{n-k},$$

and the expectation of k is $\mathbb{E}[k] = nP_R$. Thus the maximum likelihood estimator $\widehat{P_R}$ for P_R is $\frac{k}{n}$. Assume the density is continuous and the that region R is small enough so that $p(x)$ can be assumed to be constant in R. Then, we have:

$$\int_{x \in R} p(x) \mathrm{d}x \approx p(x) V_R,$$

with $V_R = \int_{x \in R} \mathrm{d}x$ the region volume. Thus the estimator of the density is $p(x) \approx \frac{k}{nV}$. □

We can apply this *ballon estimator* theorem in two ways:

- First, we fix the ball B radius (and henceforth its volume $V(B)$), and we count the number of points falling inside B for a given position x (this generalizes the 1D histogram method for approximating smooth univariate densities), or
- Second, we fix the value of k, and we seek the smallest ball centered at X that exactly contains k points. This approach is called the non-parametric estimation by k-NNs. Notice that for each different value of k, we have a different ballon estimator.

Let $r_k(x)$ denote the radius of the covering ball $B_k(x)$. The volume $V_k(x)$ is proportional to $r_k(x)^d$ up to a multiplicative constant that only depends on the dimension: $V_k(x) = c_d r_k(x)^d \propto r_k(x)^d$.

9.5.2 Probability of error and Bayes' error

First, let us observe that *any* classifier will necessarily have a non-zero misclassification rate since the distributions $X_{\pm 1}$ of the two classes share the same support: Thus we can never be 100% sure that we have correctly labeled a sample: A misclassification error always exist! In Bayesian decision theory (*i.e.*, assuming class *a priori* probabilities and class conditional probabilities), the *probability of error* is the minimal error of a classifier:

$$P_e = \Pr(\text{error}) = \int p(x)\Pr(\text{error}|x)\mathrm{d}x,$$

with

$$\Pr(\text{error}|x) = \begin{cases} \Pr(C_{+1}|x) & \text{rule decided } C_{-1}, \\ \Pr(C_{-1}|x) & \text{rule decided } C_{+1} \end{cases}$$

The *Bayesian error* generalize the error probability by taking into account a *cost matrix* $[c_{i,j}]_{i,j}$ for each potential classification scenario: Matrix coefficient $c_{i,j}$ denotes the cost of classifying a new observation x in class C_j knowing that x belongs to class C_i. The Bayesian error minimizes the expected risk, and coincides with the probability or error P_e when one chooses $c_{i,i} = 0$ (no penalty when correctly classified) and $c_{i,j} = 1$ (unit penalty cost for misclassification) for all $j \neq i$.

Recall that Bayes' fundamental identity (as known as *Bayes's rule* or *Bayes's theorem*) is:

$$\boxed{\Pr(C_i|x) = \frac{\Pr(x|C_i)\Pr(C_i)}{\Pr(x)}}$$

This can be easily shown using the *chain rule property* of probabilities:

$$\Pr(A \wedge B) = \Pr(A)\Pr(B|A) = \Pr(B)\Pr(B|A) \Rightarrow \Pr(B|A) = \frac{\Pr(B)\Pr(B|A)}{\Pr(A)}.$$

The optimal rule for Bayesian classification that minimizes the probability of error is the *maximum a posteriori* (*MAP* for short) rule: We classify x to class C_i if and only if:

$$\Pr(C_i|x) \geq \Pr(C_j|x).$$

In other words, we choose the class that maximizes the *a posteriori* probability. By using Bayes' identity and by canceling the common denominator term $\Pr(x)$, this amounts to choose class C_i such that:

$$w_i\Pr(x|C_i) \geq w_j\Pr(x|C_j),\ \forall j \neq i.$$

Since we neither know the conditional probability laws $\Pr(x|C_i)$ nor the *a priori* laws, we need to estimate them from observations in practice. We can estimate non-parametrically these distributions by using the balloon estimator that uses the nearest neighbor structures as follows: First, let us consider without loss of generality, the case of two classes $C_{\pm 1}$. We calculate the prior probabilities from the class frequencies of the observations:

$$\Pr(C_{\pm 1}) = w_{\pm 1} = \frac{n_{\pm 1}}{n}.$$

Then we compute the class-conditional probabilities as follows:

$$\Pr(x|C_{\pm 1}) = \frac{k_{\pm 1}}{n_{\pm 1} V_k},$$

with V_k the volume of the ball that cover the k-NNs of x.

Similarly, the non-conditional density (mixture of two distributions) can be estimated using the k-NNs by:

$$m(x) \approx \frac{k}{nV_k(x)}$$

We deduce the *a posteriori* probabilities using the MAP Bayesian's rule:

$$\Pr(C_{\pm 1}|x) = \frac{\Pr(x|C_{\pm 1})\Pr(C_{\pm 1})}{\Pr(x)} = \frac{\frac{k_{\pm 1}}{n_{\pm 1}V_k}\frac{n_{\pm 1}}{n}}{\frac{k}{nV_k}} = \frac{k_{\pm 1}}{k}.$$

Hence, we have proved that the voting rule of the k-NN classification rule is sound! We shall now quantify the relative performance of the k-NN classifier compared to the minimum error probability P_e.

9.5.3 Probability of error for the k-NN rule

When both the size of the training set and the size of the testing set become large enough, asymptotically tending to infinity ($t, n \to +\infty$), the probability of error $P_e(k-\mathrm{NN})$ of the k-NN rule is a worst twice the minimum probability of error P_e (induced by the MAP rule if one truly knew the class *a priori* probabilities $w_{\pm 1}$ and class-conditional probabilities $p_{\pm 1}(x)$):

$$\boxed{P_e \leq \tau_{\mathrm{error}}(\mathrm{NN}) \leq 2P_e}$$

For the the multi-class case ($m \geq 2$ classes) and the NN-rule, one can further prove that we have the following guaranteed upper-bound:

$$P_e \leq \tau_{\mathrm{error}}(\mathrm{NN}) \leq P_e\left(2 - \frac{m}{m-1}P_e\right).$$

Theorem 11

The optimal Bayesian MAP rule can be approximated by the k-NN voting rule within a multiplicative error factor of 2 when we estimate non-parametrically the class probabilities using the k-NN balloon estimator.

Let us notice that when the dimension is large, we need in practice many samples to get this theoretical bound. Once again, this is the phenomenon of the curse of dimensionality that explains that in high-dimensional spaces, problems become exponentially more difficult to solve!

9.6 Implementing nearest neighbor queries on a computer cluster

Let us consider P units of computation (UCs, or Processing Elements, PEs) with distributed memory. To classify a new query q, we shall use the *decomposable property* of the k-NN query: That is, we can partition arbitrarily $X = \biguplus_{l=1}^{p} X_l$ into pairwise disjoint groups, and we always have:

$$\mathrm{NN}_k(x, X) = \mathrm{NN}_k(x, \cup_{l=1}^{p} \mathrm{NN}_k(x, X_l)).$$

On P processors, we partition X into P groups of size $\frac{n}{P}$ (horizontal partitioning[3]), and answer locally the queries $\mathrm{NN}_k(x, X_i)$ on each processor. Finally, a *master processor* receives the kP elements from the slave processes, and perform a k-NN query on that aggregated set. Thus we speed-up the $O(dnk)$-time naive sequential algorithm ($P = 1$), and obtain a parallel query algorithm in time $O(dk\frac{n}{P}) + O(dk(kP))$. When $kP \leq \frac{n}{P}$ (that is, $P \leq \sqrt{\frac{n}{k}}$), we obtain an optimal linear speed-up in $O(P)$.

9.7 Notes and references

For statistical machine learning and more details concerning the k-NN rule, we recommend the textbook [43]. The performance of the k-NN rule has first been studied in 1967 [21]. The k-NN queries are well-studied but a difficult problem of *computational geometry* in practice, specially in high-dimensions [4].

[3] For very large dimensions, we may consider *vertical partitioning* that splits blockwise the dimension of data among the distributed memories.

In practice, graphics processing units (GPUs) are very well-suited for fast k-NN queries [34] using the built-in inner product facilities. An algorithm is said output-sensitive when its complexity can be analyzed using both the input size and the output size. An output-sensitive algorithm has been proposed for computing the decision region between two classes of points in the plane, see [15]. One can relax the exact k-NN queries to the problem of finding within a constant multiplicative factor $1+\epsilon$ the ϵ-NNs. The main advantage of classifying with the k-NN rule is that it is easy to program, can be straightforwardly parallelized, and that it guarantees asymptotically a performance bound with respect to Bayes' minimal misclassification error. In practice, one has to choose the right value of k for the k-NN rule: For large values of k, the decision boundary gets smoother and it yields a more robust non-parametric estimation of conditional probabilities, but it costs more time to answer queries and the non-parametric becomes less local!

In practice, classification using big data-sets exhibits experimentally an empirical *law of diminishing returns*: That is, the larger the size of the training set, the smaller the relative improvement of performance. This observed phenomenon is depicted schematically in Figure 9.5. This is due to the fact that the identically and independently sampled labeled observations assumption does not hold in practice. Bayes' error provides a lower bound on the performance of any classifier in statistical machine learning. It is often hard to calculate explicitly Bayes' error or the probability of error for statistical models in closed-form formula. Thus one rather seeks to upper bound Bayes' error using closed-form formula [72].

There exists numerous extensions of the k-NN rule. One can adjust for example the voting rule among the k neighbors by taking a weighted average vote [75]. We can prove that in high dimensions the k-NN boundary is piecewise linear by studying bichromatic Voronoi diagrams in high dimensions. However, computing such high-dimensional Voronoi diagrams are intractable in practice as they can have a combinatorial complexity in $O(n^{\lceil \frac{d}{2} \rceil})$ time (already quadratic for $d = 3$), where $\lceil x \rceil$ is the *ceil function* (that returns the smallest integer greater or equal to x).

Let us recall that the k-NN classification rule guarantees asymptotically an error factor of 2 compared to the optimal MAP Bayesian rule but that this classifier needs to store in memory *all* the training set. That can be prohibitive for large data-sets. Another renown classification technique are the *Support Vector Machines* (*SVMs*) that stores only $d+1$ points in dimension d for linearly separable bichromatic point sets. When classes are not linearly separable, one can use the so-called kernel trick to embed features in a higher-dimensional space so that it becomes separable [43] in that space. It is always possible to find such a kernel to separate classes.

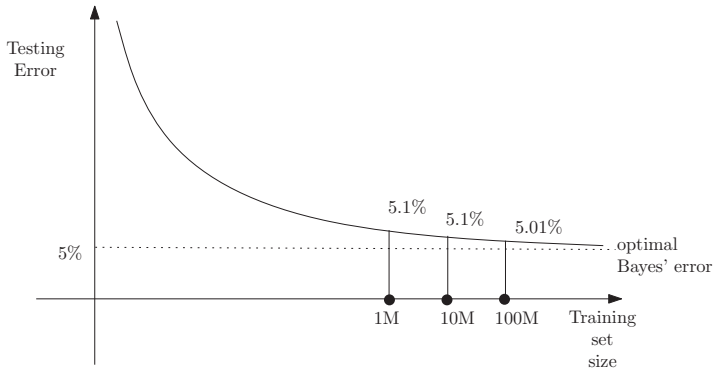

Figure 9.5 In practice, classification using big data exhibits experimentally a law of diminishing returns: The larger the size of the training set, the smaller the relative improvement of performance. This is due to the fact that the identically and independently sampled labeled observations assumption does not hold. Bayes' error provides a lower bound on the performance of any classifier in statistical machine learning.

To conclude, let us discuss about model complexity, bias and variance of learning machines, and prediction error. In Chapter 5, we quickly describe the linear regression to motivate the use of linear algebra in data science. Let us compare classification by regression with classification by k-NN as follows:

– regression model: The model complexity of a linear regression model is $d + 1$, the number of coefficients defining the fitted hyperplane. Regression learning machines have low variance (meaning stable with respect to input perturbation) but high bias (meaning not tight to the true separation of classes).

– k-NN model: The model complexity of a k-NN classifier is $d \times n$, very large since dependent on the input size n of the training set. The properties of the k-NN classifier is to have low bias since it fits well the class separation boundary but it has high variance since a single point perturbation of the training set can significantly affects the decision boundary of the k-NN classifier.

Figure 9.6 illustrates the bias/variance properties of learning machines according to their model complexities. In practice, one has to choose the proper model complexity of a learning machine. The higher the model complexity the lower the prediction error on the training sample. However, at some point, there is an overfitting phenomenon, and the prediction error on the test sample increases instead of continuing to decrease. Since we are interested

in minimizing the generalization error (and not on minimizing error on the training sample, which can be optimally reaching zero for the k-NN classifier), the ideal model complexity should be chosen so as to minimize the prediction error on the test sample (see Figure 9.6).

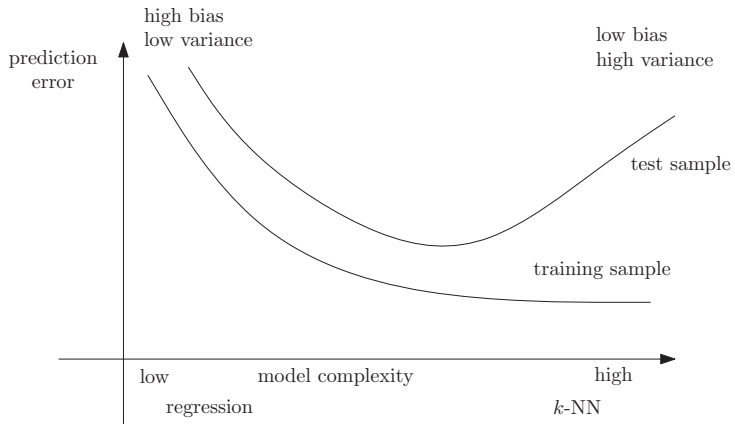

Figure 9.6 Model complexity, bias and variance of learning machines, and prediction error.

9.8 Summary

The k-NN classification rule labels a new observation query q from a test set by choosing the dominant class among the k nearest neighbors of q in the training set. One evaluates the performance of a classifier by calculating its F-score that is the harmonic mean of the precision rate and the recall rate. This yields a single quality value that takes into account the four different cases that can occur when classifying observations in one of either two classes (false/true-positive/negative). In statistical machine learning, a classifier can never beat the optimal Bayes' error (or the probability of error), and the 1-NN guarantees asymptotically an error factor of 2. Since the nearest neighbor queries are decomposable queries, meaning that $\mathrm{NN}_k(q, X_1 \cup X_2) = \mathrm{NN}_k(\mathrm{NN}_k(q, X_1), \mathrm{NN}_k(q, X_2))$, the k-NN classification rule can be easily parallelized on a distributed memory architecture like a computer cluster. One of the drawback of the k-NN rule is that it needs to store all the training set in order to classify new observations.

Processing code for displaying the nearest neighbor classification rule

Figure 9.7 displays a snapshot of the `processing.org` program.

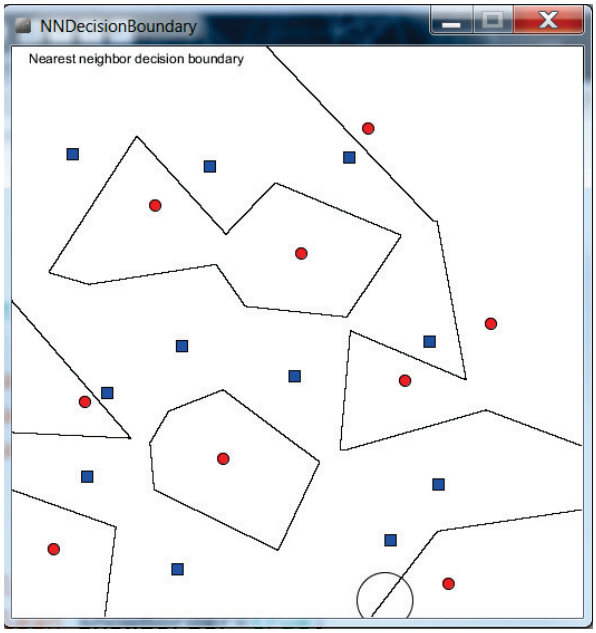

Figure 9.7 Snapshot of the `processing` code for displaying the nearest neighbor decision border.

WWW source code: `NNDecisionBoundary.pde`

9.9 Exercises

Exercise 1: *Pruning the boundary decision of the nearest neighbor classifier*

Prove that when an element of the training set has its neighboring Voronoi cells (called *natural neighbors*) of the same class, then this sample can be safely removed without changing the boundary decision of the NN classification rule. How to prune the training set for the k-NN rule?

Exercise 2: ** Probability of error for conditional Gaussian laws*

Let us consider balanced *a priori* probabilities $w_1 = w_2 = \frac{1}{2}$ and conditional probabilities following univariate Gaussian distributions $X_1 \sim N(\mu_1, \sigma_1)$ and $X_2 \sim N(\mu_2, \sigma_2)$. Recall that the probability density of a normal distribution is $p(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$.

- Calculate exactly the probability of error when P_e when $\sigma_1 = \sigma_2$,
- Calculate P_e using the standard normal cumulative distribution function $\Phi(\cdot)$ when $\sigma_1 \neq \sigma_2$.
- Since it is difficult to compute P_e in closed-form formula, using the mathematical rewriting $\min(a, b) = \frac{a+b-|b-a|}{2}$ with the following inequality $\min(a, b) \leq a^\alpha b^{1-\alpha}, \forall \alpha \in (0, 1)$ when $a, b > 0$, deduce an upper-bound formula in closed-form bounding P_e for Gaussian distributions.

Exercise 3: *** The k-NN rules and the order-k Voronoi diagrams [60]*

Consider a finite point set $X = \{x_1, ..., x_n\}$. Show that the decomposition of the space $\mathbb{R}^d$ induced by the first k nearest neighbors yield convex polyhedral cells. We define the k-order Voronoi diagram as the partition of space induced by all the $\binom{n}{k}$ $X_i \subset 2^X$ sub-sets of X for the following distance function: $D(X_i, x) = \min_{x' \in X_i} D(x', x)$. That is, the k-order Voronoi diagram is the collection of non-empty cells $V_k(X_i)$ defined by $V_k(X_i) = \{x \mid D(X_i, x) \leq D(X_j, x),\ \forall i \neq j\}$ (with $|X_l| = k$ for all l). How one can simplify the boundary decision of the k-NN classification rule?

Exercise 4: *** Sensitivity of the k-NN classification rule with respect to the magnitude order of axes*

The performance of a classifier for the nearest neighbor classification rule is quite sensitive to a rescaling of axis since it may change significantly the Euclidean distance. In practice, one has to find a good weighting rule on the attributes (*feature weighting*) to calibrate the "Euclidean distance": $D_w(p, q) = \sqrt{\sum_{j=1}^{d} w_j (p^j - q^j)^2}$. Study different methods of attribute rewriting and discuss on their performance [43] (refer to Mahalanobis distance with diagonal precision matrix as well).