2 K-Nearest Neighbors

2.1 Introduction

This chapter gives an introduction to pattern recognition and machine learning via K-nearest neighbors. Nearest neighbor methods will have an important part to play in this book. The chapter starts with an introduction to foundations in machine learning and decision theory with a focus on classification and regression. For the model selection problem, basic methods like cross-validation are introduced. Nearest neighbor methods are based on the labels of the K-nearest patterns in data space. As local methods, nearest neighbor techniques are known to be strong in case of large data sets and low dimensions. Variants for multi-label classification, regression, and semi-supervised learning settings allow the application to a broad spectrum of machine learning problems. Decision theory gives valuable insights into the characteristics of nearest neighbor learning results.

2.2 Classification

Classification is the problem to predict discrete class labels for unlabeled patterns based on observations. Let $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be the set of observations of q-dimensional patterns $\mathcal{X} = \{\mathbf{x}\}_{i=1}^N \subset \mathbb{R}^q$ and a corresponding set of labels $\mathcal{Y} = \{\mathbf{y}_i\}_{i=1}^N \subset \mathbb{R}^d$. The goal in classification is to learn a functional model f that allows a reasonable prediction of class label y' for an unknown pattern $\mathbf{x}'$. Patterns without labels should be assigned to labels of patterns with known assignment that are similar, e.g., that are close to the target pattern, come from the same distribution, or lie on the same side of a separating function. But learning from observed patterns can be difficult. Training sets can be noisy, important features may be unknown, similarities between patterns may not be easy to define, and observations may not be sufficiently described by simple distributions. Further, learning simple functional models can be a difficult undertaking, as classes may not be linearly

DOI: 10.1007/978-3-642-38652-7_2

separable and may be difficult to separate with simple rules or mathematical equations.

Famous classification methods are decision tress, e.g., ID3 and C4.5 [81], neural networks [93, 96], and SVMs [98, 101]. For an introduction to these methods, we refer the reader to books like Bishop [12] and Hastie *et al.* [40]. SVMs will also briefly be introduced in Chapter 3. In the following, we concentrate on a simple yet powerful method: the nearest neighbor classifier.

2.2.1 KNN Classifier

Nearest neighbor classification, also known as K-nearest neighbors (KNN), is based on the idea that the nearest patterns to a target pattern $\mathbf{x}'$, for which we seek the label, deliver useful label information. KNN assigns the class label of the majority of the K-nearest patterns in data space. For this sake, we have to be able to define a similarity measure in data space. In $\mathbb{R}^q$, it is reasonable to employ the Minkowski metric (p-norm)

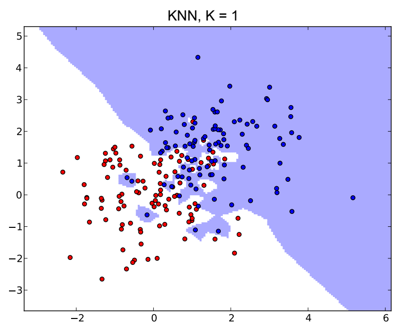
$$\|\mathbf{x}' - \mathbf{x}_j\|^p = \left(\sum_{i=1}^{q} |(x_i)' - (x_i)_j|^p \right)^{1/p}, \tag{2.1}$$

which corresponds to the Euclidean distance for $p = 2$. In other data spaces, adequate distance functions have to be chosen, e.g., the Hamming distance in $\mathbb{B}^q$. In the case of binary classification, the label set $\mathcal{Y} = \{1, -1\}$ is employed, and KNN is defined as

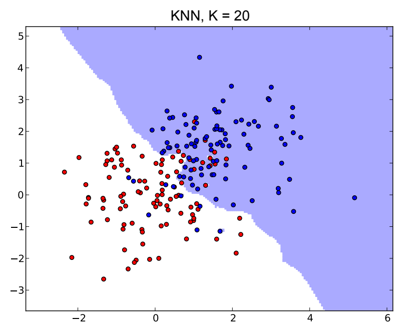
$$f_{\text{KNN}}(\mathbf{x}') = \begin{cases} 1 & \text{if } \sum_{i \in \mathcal{N}_K(\mathbf{x}')} y_i \geq 0 \\ -1 & \text{if } \sum_{i \in \mathcal{N}_K(\mathbf{x}')} y_i < 0 \end{cases} \tag{2.2}$$

with neighborhood size K and with the set of indices $\mathcal{N}_K(\mathbf{x}')$ of the K-nearest patterns.

The choice of K defines the *locality* of KNN. For $K = 1$, little neighborhoods arise in regions, where patterns from different classes are scattered. For larger neighborhood sizes, e.g. $K = 20$, patterns with labels in the minority are ignored. Figure 2.1 illustrates the difference in classification between KNN with $K = 1$ and $K = 20$ on a simple 2-dimensional data set consisting of two overlapping data clouds of each 50 Gaussian-sampled red and blue points. The data space locations that would be classified as *blue* are shown in bright blue, while areas classified as *red* are shown in white. For $K = 1$, the prediction is *local.* For example, a blue point that is an outlier from the blue class is located at the center of the red cloud. For large K, the classifier generalizes ignoring small agglomerations of patterns. KNN induces a Voronoi tessellation in data space. In case of large data sets, KNN has to search for the K-nearest patterns in the whole space, but can already yield a good approximation based on the K-nearest neighbors in a scanned subset.



(a)



(b)

Fig. 2.1 A comparison of KNN classification on two Gaussian-based data clouds for two neighborhood sizes ((a) $K = 1$ and (b) $K = 20$). For small neighborhoods, KNN tends to overfit becoming *local*, while KNN generalizes for larger K ignoring small agglomerations of patterns.

The question arises, how to choose K, i.e., which neighborhood size achieves the best classification result. This problem is also known as model selection, and various techniques like cross-validation can be employed to choose the best model and parameters (cf. Section 2.5).

2.2.2 Multi-class K-Nearest Neighbors

KNN can also be applied to multi-class classification problems. For an unknown pattern $\mathbf{x}'$, KNN for multi-class classification predicts the class label of the majority of the K-nearest patterns in data space

$$f_{\text{KNN}}(\mathbf{x}') = \arg\max_{y \in \mathcal{Y}} \sum_{i \in \mathcal{N}_K(\mathbf{x}')} \mathcal{I}(y_i = y) \tag{2.3}$$

with indicator function $\mathcal{I}(\cdot)$ that returns *one*, if its argument is true[1] and *zero* otherwise. This definition will also be used for the ensemble classifier in Section 3.4.

2.3 KNN Regression

Closely related to classification is regression. Functional regression models map patterns to continuous labels, i.e., to a subspace of $\mathbb{R}^d$. Although in practice, machine accuracy only allows a mapping to a discrete set of numbers, the difference becomes obvious: the set of labels is very large in comparison to classification problems, where the set of labels is restricted to

[1] i.e., label y_i of pattern $\mathbf{x}_i$ is y.

few elements. The problem in regression is to predict labels $\mathbf{y}' \in \mathbb{R}^d$ for new patterns $\mathbf{x}' \in \mathbb{R}^q$ based on a set of N observations, i.e., labeled patterns $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$. The regression learning problem will be derived from a statistical learning perspective in Section 2.4.

The goal is to learn a function $\mathbf{f} : \mathbb{R}^q \rightarrow \mathbb{R}^d$ known as regression function. For an unknown pattern $\mathbf{x}'$, KNN regression computes the mean of the function values of its K-nearest neighbors

$$\mathbf{f}_{KNN}(\mathbf{x}') = \frac{1}{K} \sum_{i \in \mathcal{N}_K(\mathbf{x}')} \mathbf{y}_i \tag{2.4}$$

with set $\mathcal{N}_K(\mathbf{x}')$ containing the indices of the K-nearest neighbors of $\mathbf{x}'$. The idea of averaging in KNN is based on the assumption of locality of functions in data and label space. In local neighborhoods of $\mathbf{x}_i$, patterns $\mathbf{x}'$ are expected to have similar continuous labels $\mathbf{f}(\mathbf{x}')$ like $\mathbf{y}_i$. Hence, for an unknown $\mathbf{x}'$ the label must be similar to the labels of the closest patterns, which is modeled by the average of the label of the K-nearest patterns. KNN has been proven well in various applications, e.g., in the detection of quasars based on spectroscopic data [33].

Also in regression, the neighborhood size K of KNN is an important parameter. For $K = 1$, KNN regression *overfits* to the label of the nearest neighbor of $\mathbf{x}'$, for $K = N$ it averages over all patterns Figure 2.2 shows a comparison between KNN regression with the two neighborhood sizes (a) $K = 2$ and (b) $K = 5$. Weighted KNN regression induces plateaus.

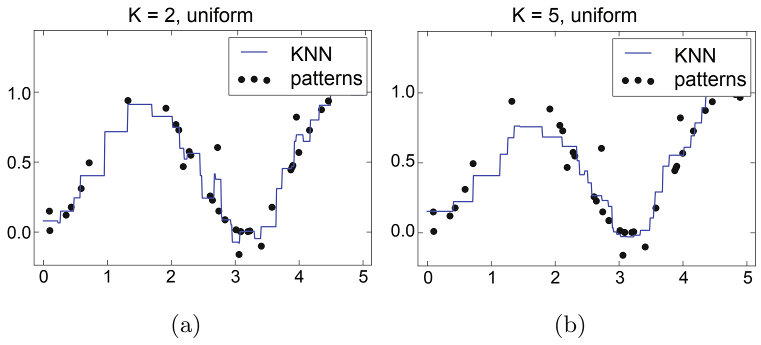


Fig. 2.2 Illustration of uniform KNN regression for (a) $K = 2$ and (b) $K = 5$

2.4 Statistical Learning Theory

From statistical learning theory, we can get insights into the quality a functional model should have. We assume that observed patterns and labels can be modeled with random variables. This allows to model noise and fluctuations. In the following, we consider the univariate case. We assume two continuous random variables: X for patterns and Y for labels, which can be described by probability distribution functions, i.e,. $p(X = x)$ and $p(Y = y)$, as well as the joint distribution $p(X = x, Y = y)$. The task in supervised learning is to learn a functional model f, which predicts the correct label y for a given pattern x. A loss function $L(y, f(x))$ allows to measure the error, which is the deviation of the correct label and the prediction of the functional model. The task is to learn a model with minimal error. Often, the square loss is employed

$$L(y, f(x)) = (y - f(x))^2. \tag{2.5}$$

The probability distribution functions allow the evaluations of the quality of the functional model f with the expected prediction error

$$E_{exp}(f) = \langle L(Y, f(X)) \rangle = \int \int L(y, f(x)) p(x, y) dx dy, \tag{2.6}$$

while $\langle \cdot \rangle$ is defined as expected value of a random variable. Now, we can replace the joint probability density function by the Bayes expression

$$p(x, y) = p(y|x)p(x) = p(Y = y|X = x)p(X = x), \tag{2.7}$$

becoming

$$E_{exp}(f) = \int \left(\int (y - f(x))^2 p(y|x) dy \right) p(x) dx \tag{2.8}$$

with the square loss. We seek for the best functional model f^* that minimizes the *empirical risk* $E_{exp}(f)$. Minimization of E_{exp} can be accomplished by point-wise minimization of the inner integral for each x yielding the regression function

$$\hat{f}^*(x) = \int y p(y|x) dy = \int y \frac{p(x, y)}{p(x)} dy = \langle y|x \rangle. \tag{2.9}$$

The consistency criterium postulates that the empirical risk converges to *zero* in the limit of an infinite number of training examples. The problem to find the best functional model is also known as model selection problem. The optimal functional model f^* could easily be found, if we knew $P(x, y)$, and if we searched in the set of all functions $\mathcal{F}$. But as we have to restrict to minimizing Equation 2.8, it is not reasonable to search in $\mathcal{F}$. Instead, we choose a parameterized model (e.g. KNN) that represents a smaller function space $F \subset \mathcal{F}$ and to minimize the empirical risk w.r.t. this model

$$f^* = \arg \min_{f \in F} E_{emp}(f). \tag{2.10}$$

It is not very probable that the true optimal model f^* lies in F. On the one hand, it is reasonable to increase the function space. On the other hand, we have to avoid functions that only reconstruct the observations, e.g., that return $f(\mathbf{x}_i) = \mathbf{y}_i$ for $i = 1, \ldots, N$ and yield false values, e.g., the opposite class label in binary classification for all patterns we have not observed yet. Such an effect is known as overfitting.

The increase of function space F, from which f is chosen, may be reasonable to get closer to the true functional model. This is also known as increasing the capacity of the model. But to avoid overfitting, it is often sufficient to penalize the complexity of model f with a regularizer, which can be a functional norm $\|f\|$. Then, the regularized risk

$$E_{reg}(f, \lambda) = E_{emp}(f) + \lambda \|f\| \tag{2.11}$$

is minimized, where $\lambda \in \mathbb{R}^+$ is known as regularization parameter balancing between empirical risk minimization and smoothness of the function. From the perspective of KNN, we get a direct solution of this formulation. The expectation is approximated by averaging over training patterns. For large training set sizes N, conditioning of f is better than for small N, as points are likely to be close to $\mathbf{x}$. As Bishop [12] states, for $N, k \to \infty$, i.e., $k/N \to 0$ we get $f_{KNN}(x) \to E(Y|X = x)$. But the curse of dimensionality (cf. Section 2.6) weakens this argument. KNN is an excellent model for (1) low-dimensions and (2) large training set sizes. But in case of high-dimensional data spaces or few patterns, extensions of KNN are necessary [12, 40].

2.5 Cross-Validation

The problem in supervised learning is to find an adequate model f and its parameterizations. To avoid overfitting, an often employed model selection strategy is cross-validation. The idea of cross-validation is to split up the N observations $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ into training, validation, and test set. The training set is used as basis for the learning algorithm given a potential parameter set. The validation set is used to evaluate the model given the parameter set. The optimized model $\hat{f}^*$ based on a training process with cross-validation is finally evaluated on an independent test set.

Minimization of the empirical risk on the validation set is basis of the training phase. For optimization, weak strategies like grid search are often sufficient. Finally, the approach is evaluated on the test set that has not been used for training model f. An advanced strategy to avoid overfitting is k-fold cross-validation that repeats the learning process k times with different training and validation sets. For this sake, the data set is slit up into k disjoint sets. In each step, model f employs $k - 1$ sets for training and is evaluated on the remaining validation set. The error is aggregated to select the best parameters of model f on all k validation sets and is called cross-validation score. Advantage of this procedure is that all observations have been used for training the model and not only a subset of a small data set.

In case of tiny data sets, the number of patterns might be too small to prevent that model f is not biased towards the training and validation set. In this case, the k-fold cross-validation variant $k = N$ called leave-one-out cross-validation (LOO-CV) is a recommendable strategy. In LOO-CV, one pattern is left out for prediction based on the remaining $N-1$ training patterns. The whole procedure is repeated N times.

2.6 Curse of Dimensionality

Many machine learning methods have problems in high-dimensional data spaces. The reason is an effect also know as *curse of dimensionality* or *Hughes effect*. In high-dimensional data spaces, many patterns are required to cover the whole data space, and our intuition often breaks down. Hastie *et al.* [40] gives interesting arguments for this effect that we review in the following.

Let us assume we sample points uniformly in the unit hypercube. The volume of the q-dimensional unit hypercube is 1^q. Let r be the hypercube edge length of a smaller hypercube corresponding to the volume of the unit hypercube we want to capture. Then, r^q is its volume and at the same time the volume fraction v of the unit hypercube $v = r^q$. Hence, $r = v^{1/q}$, which for example means that in $q = 10$ dimensions we have to cover $r = 0.1^{1/10} \approx 0.8$ of each variable to cover 10% of the volume of the unit hypercube with labeled patterns. In other words, covering 0.8 of each dimension with points means that still 90% of the 10-dimensional hypercube is empty.

2.7 Nearest Neighbor Queries

The search for nearest neighbors is a frequent problem in machine learning. If we have a nearest neighbor query for a pattern $\mathbf{x}'$, a simple, brute-force approach is to test the distance to each other pattern $\|\mathbf{x}' - \mathbf{x}_i\|^2$ for $i = 1, \dots N$, with $\mathbf{x}' \neq \mathbf{x}_i$ in $O(N)$ time. But there are various possibilities to accelerate the nearest neighbor queries. If more than $\log N$ queries are necessary, sorting all pattern w.r.t. their distance may be reasonable, e.g., if $K > \log N$ for KNN. Sorting can be accomplished in the average case in $O(N \log N)$, e.g., with Quicksort, but in $O(N^2)$ in worst case. Another option might be appropriate for high-dimensional patterns. Computing the distances usually means to sum up distance values per dimension. The computation of a nearest neighbor request can be stopped, if the maximum distance to the previously computed K-nearest patterns is exceeded.

The employment of efficient data structures like *k-d* trees [9] and balltrees that partition the data space can result in $O(\log N)$ neighborhood requests for certain data distributions and lower dimensions. Figure 2.3 illustrates binary space partitioning trees. Every non-leaf node of a *k-d* tree induces a hyperplane that splits the data space into two parts. Each subtree represents the corresponding subspace. A *k-d* tree is constructed by first sorting the patterns w.r.t. the first dimension. The median of the sorting result is taken as pivot element and represents a node. Elements left of the node belong to the left subtree, elements right of the node to the right subtree. Then, the two groups are sorted w.r.t. the second dimension. The process is recursively repeated until the corresponding sets consist of one element. If the data is smoothly (e.g. uniformly) distributed, half of the patterns can be excluded for each node resulting in a neighbor query lying in $O(\log N)$. Building a k-d tree takes $O(N \log N)$ time in average. Adding and removing of elements also take $O(\log N)$ time.

Experiments have shown that balltrees show better results than k-d trees, when the data is clustered, sparse, or has extra structure [84]. In this case, k-d trees may fall back to $O(N)$ runtime for one neighbor query. A ball in Euclidean space is the region bound by a hypersphere. It can be represented by center coordinates and radius. A balltree is a binary tree, whose nodes correspond to the smallest balls that contain all balls that belong to nodes of its subtrees. The balls of a balltree may intersect and do not need to cover the entire data space, which makes them applicable to sparse and non-smooth data. For a discussion on balltree construction algorithms, we refer to Omohundro [84] and Hastie *et al.* [40].

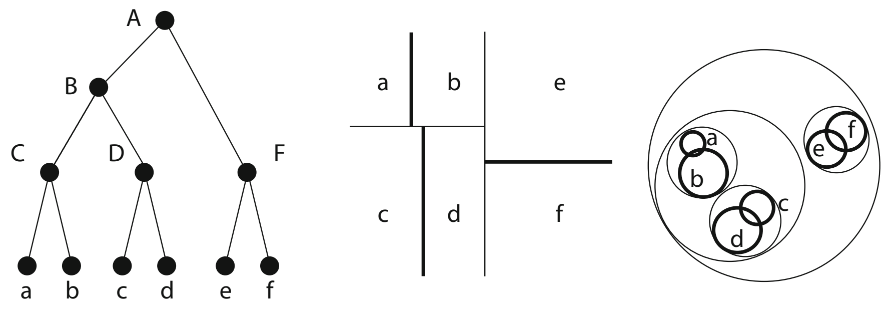


Fig. 2.3 Illustration of binary space partitioning trees: (a) If the binary tree is a k-d tree, (b) its nodes induce linear borders and divide the data space into *half-spaces*, while (c) a balltree defines a hierarchy of balls

2.8 Nearest Neighbor Variants

KNN is a technique with a long tradition. It has first been mentioned by Fix and Hodges [28] in the fifties in an unpublished US Air Force School of Aviation Medicine report as non-parametric classification technique. Cover and Hart [21] investigated the approach experimentally in the sixties. Interesting properties have been found, e.g., that for $K = 1$ and $N \to \infty$, KNN is bound by twice the Bayes error rate. Many variants of KNN have been presented in the past. Two variants are presented in the following, and a semi-supervised KNN modification is presented.

2.8.1 Model-Based KNN

The idea of model-based KNN is to replace the training set by a set of reference points (or codebook vectors) that achieve the same prediction results. This concept is related to the landmark variant of unsupervised kernel regression, which will be introduced in Section 4.9.5. The collection of landmark points is called model. The selection of a set of landmarks is treated as optimization problem, i.e., we have to search for the optimal subset of landmark vectors that achieve the same nearest neighbor result as KNN on the complete set of patterns. First, a similarity matrix from the data set is computed. All labels of $\mathbf{y}_i$ are set to *ungrouped.* Then, we seek the neighborhood that covers the largest number of neighbors with the same label. Their label is set to *grouped.* The last steps are repeated until all labels are set to *grouped.* The resulting model contains a selection for landmark vectors that can be employed as surrogate for the original KNN model.

2.8.2 Distance-Weighted KNN

KNN induces locally constant outputs. From the optimization perspective, this means we get an output space with plateaus: for neighborhood size K and N patterns in KNN regression, $\binom{N}{K}$ different output values are possible. Plateaus can hinder optimization methods from a fast approximation of the optimal solution, as not much information about promising search directions can be gained during optimization. Bailey and Jain [5] introduced distance-weighted KNN rules in the late seventies to smooth the prediction function weighting the prediction with the similarity $\Delta(\mathbf{x}', \mathbf{x}_i)$ of the nearest patterns $\mathbf{x}_i$ with $i \in \mathcal{N}_K(\mathbf{x}')$ to the target $\mathbf{x}'$

$$\mathbf{f}_{wKNN}(\mathbf{x}') = \sum_{i \in \mathcal{N}_K(\mathbf{x}')} \frac{\Delta(\mathbf{x}', \mathbf{x}_i)}{\sum_{j \in \mathcal{N}_K(\mathbf{x}')} \Delta(\mathbf{x}', \mathbf{x}_j)} \mathbf{y}_i. \tag{2.12}$$

Patterns close to the target should contribute more to the prediction than patterns that are further away. Similarity can be defined with the distance between patterns, e.g. by

$$\Delta(\mathbf{x}', \mathbf{x}_i) = 1/\|\mathbf{x}' - \mathbf{x}_i\|^2. \tag{2.13}$$

Model $\mathbf{f}_{wKNN}$ introduces a continuous output. Figure 2.4 shows the KNN prediction based on KNN regression in the weighted variant on the trigonometric function. Weighted KNN regression interpolates between the points in contrast to the uniform variant (cf. Figure 2.2).

Also weighted KNN maps to a discrete number of solutions. Machine accuracy may restrict the output space to, e.g., 2^{64} in case of 64 bits used. Uniform KNN restricts the number of possible output values to $\binom{N}{K}$. As a final remark, we state that for $K = N$ we take every pattern into account

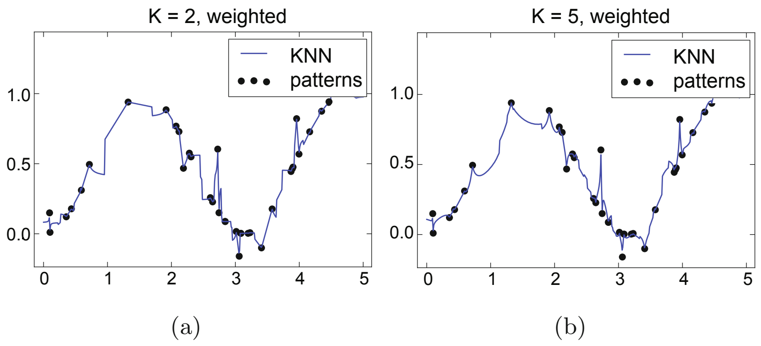


Fig. 2.4 Illustration of weighted KNN regression for (a) $K = 2$ and (b) $K = 5$

$$\mathbf{f}_{wKNN_{K=N}}(\mathbf{x}') = \sum_{i=1}^{N} \frac{\|\mathbf{x}' - \mathbf{x}_i\|^2}{\sum_{j=1}^{N} \|\mathbf{x}' - \mathbf{x}_j\|^2} \mathbf{y}_i. \tag{2.14}$$

resulting in a simplification that does not afford the computation of the nearest neighbors.

2.8.3 Propagating 1-Nearest Neighbor

Propagating 1-nearest neighbor is an approach for semi-supervised learning [116]. In semi-supervised learning, we have given a set of labeled patterns $L = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ and a set of unlabeled patterns $U = \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_u\}$. The idea of semi-supervised learning is that the unlabeled data enriches the learning process by yielding implicit information about the underlying data distributions. Propagating 1-nearest neighbor works as follows. In each step, the unlabeled pattern closest to any of the (already) labeled patterns is selected

$$\tilde{\mathbf{x}}^* = \min_{\tilde{\mathbf{x}} \in U} \min_{\mathbf{x} \in L} \|\tilde{\mathbf{x}} - \mathbf{x}\|^2. \tag{2.15}$$

Its label y is determined by the label of the nearest neighbor $\mathbf{x}$ in the set of labeled patterns L. Pattern $(\tilde{\mathbf{x}}^*, y)$ is added to L and removed from U. The algorithm terminates, when U is empty.

2.9 Conclusions

In this chapter, we gave an introduction to basic principles in machine learning, concentrating on supervised learning and nearest neighbor methods. Classification is the prediction of discrete class labels based on observed pattern-label pairs. Regression is the prediction of continuous values based on pattern-label observations. Nearest neighbor approaches for classification and regression rely on the label of the K-nearest patterns in data space. In supervised learning, overfitting may occur. It can be prevented by regularization and cross-validation. Regularization is a method to avoid that functional models become too complex, while cross-validation avoids overfitting to small data sets. LOO-CV is a variant with $k = N$. Classification becomes difficult in high-dimensional data spaces, known as curse of dimensionality or Hughes effect. In case of nearest neighbor methods, the training set size has to be increased to improve the learning result. KNN variants have been introduced: from distance-weighted KNN to propagating 1-nearest neighbor for semi-supervised learning scenarios.