Chapter 3
Similarities: Nearest-Neighbor Classifiers

Two plants that look very much alike probably represent the same species; likewise, it is quite common that patients complaining of similar symptoms suffer from the same disease. In short, similar objects often belong to the same class—an observation that forms the basis of a popular approach to classification: when asked to determine the class of object **x**, find the training example most similar to it. Then label **x** with this example's class.

The chapter explains how to evaluate example-to-example similarities, presents concrete mechanisms that use these similarities for classification purposes, compares the performance of this approach with that of the Bayesian classifier, and introduces methods to overcome some inherent weaknesses.

3.1 The *k*-Nearest-Neighbor Rule

How do we establish that an object is more similar to **x** than to **y**? Some may doubt that this is at all possible. Is a giraffe more similar to a horse than to a zebra? Questions of this kind raise suspicion. Too many arbitrary and subjective factors are involved in answering them.

Similarity of Attribute Vectors The machine-learning task, as formulated in this book, reduces these objections to a minimum. Rather than real objects, the classifier will compare their attribute-based descriptions—hardly an insurmountable problem. Thus in the toy domain from Chap. 1, the similarity of two pies can be established by counting the attributes in which they differ: the fewer the differences, the greater the similarity. The first row in Table 3.1 gives the attribute values of object **x**. For each of the twelve training examples that follow, the rightmost column specifies the number of differences between the given example and **x**. The smallest value being found in the case of $\texttt{ex}_5$, we conclude that this is the training example most similar to **x**, and this suggests that we should label **x** with `pos`, the class of $\texttt{ex}_5$.

DOI 10.1007/978-3-319-63913-0_3

Table 3.1 Counting the numbers of differences between pairs of discrete-attribute vectors

Example	Shape	Crust		Filling		Class	# differences
		Size	Shade	Size	Shade		
x	Square	Thick	Gray	Thin	White	?	–
$\texttt{ex}_1$	Circle	Thick	Gray	Thick	Dark	`pos`	3
$\texttt{ex}_2$	Circle	Thick	White	Thick	Dark	`pos`	4
$\texttt{ex}_3$	Triangle	Thick	Dark	Thick	Gray	`pos`	4
$\texttt{ex}_4$	Circle	Thin	White	Thin	Dark	`pos`	4
$\texttt{ex}_5$	Square	Thick	Dark	Thin	White	`pos`	1
$\texttt{ex}_6$	Circle	Thick	White	Thin	Dark	`pos`	3
$\texttt{ex}_7$	Circle	Thick	Gray	Thick	White	`neg`	2
$\texttt{ex}_8$	Square	Thick	White	Thick	Gray	`neg`	3
$\texttt{ex}_9$	Triangle	Thin	Gray	Thin	Dark	`neg`	3
$\texttt{ex}_{10}$	Circle	Thick	Dark	Thick	White	`neg`	3
$\texttt{ex}_{11}$	Square	Thick	White	Thick	Dark	`neg`	3
$\texttt{ex}_{12}$	Triangle	Thick	White	Thick	Gray	`neg`	4

Of the 12 training examples, $\texttt{ex}_5$ is the one most similar to **x**

Table 3.2 The simplest version of the *k*-NN classifier

Suppose we have a mechanism to evaluate the similarly between attribute vectors. Let **x** denote the object whose class we want to determine.

1. Among the training examples, identify the k nearest neighbors of **x** (examples most similar to **x**).
2. Let c_i be the class most frequently found among these k nearest neighbors.
3. Label **x** with c_i.

Dealing with continuous attributes is just as simple. The fact that each example can be represented by a point in an n-dimensional space makes it possible to calculate the geometric distance between any pair of examples, for instance, by the Euclidean distance (Sect. 3.2 will have more to say about how to measure distance between vectors). And again, the closer to each other the examples are in the instance space, the greater their mutual similarity. This, by the way, is how the *nearest-neighbor classifier* got its name: the training example with the smallest distance from **x** in the instance space is, geometrically speaking, **x**'s nearest neighbor.

From a Single Neighbor to *k* Neighbors In noisy domains, the testimony of the nearest neighbor cannot be trusted. What if this single specimen's class label is incorrect? A more robust approach identifies not one, but several nearest neighbors, and then lets them vote. This is the essence of the so-called k-NN classifier, where k is the number of the voting neighbors—usually a user-specified parameter. The pseudocode in Table 3.2 summarizes the algorithm.

Note that, in a two-class domain, k should be an odd number so as to prevent ties. For instance, a 4-NN classifier might face a situation where the number of positive neighbors is the same as the number of negative neighbors. This will not happen to a 5-NN classifier.

As for domains that have more than two classes, using an odd number of nearest neighbors does not prevent ties. For instance, the 7-NN classifier can realize that three neighbors belong to class C_1, three neighbors belong to class C_2, and one neighbor belongs to class C_3. The engineer designing the classifier then needs to define a mechanism to choose between C_1 and C_2.

An Illustration Certain "behavioral aspects" of this paradigm can be made obvious with the help of a fictitious domain where the examples are described by two numeric attributes, a situation easy to visualize. Figure 3.1 shows several positive and negative training examples, and also some objects (the big black dots) whose classes the k-NN classifier is to determine. The reader can see that objects **1** and **2** are surrounded by examples from the same class, and their classification is therefore straightforward. On the other hand, object **3** is located in the "no man's land" between the positive and negative regions, so that even a small amount of attribute noise can send it to either side. The classification of such *borderline examples* is unreliable.

In the right-hand part of the picture, object **4** finds itself deep in the positive region, but class noise has mislabeled its nearest neighbor in the training set as negative. Whereas the 1-NN classifier will go wrong, here, the 3-NN classifier will give the correct answer because the other two neighbors, which are positive, will outvote the single negative neighbor.

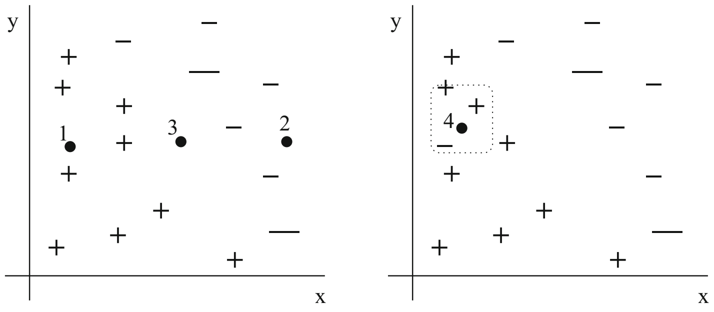

Fig. 3.1 Object **3**, finding itself in the borderline region, is hard to classify. In the noisy domain shown in the *right-hand part*, the 1-NN classifier will misclassify object **4**, but the mistake is corrected if the 3-NN classifier is used

What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- How can we measure example-to-example similarity in domains where all attributes are discrete, and how in domains where they are all continuous?
- Under what circumstances will the k-NN classifier (with $k > 1$) outperform the 1-NN classifier and why?
- Explain why, in 2-class domains, the k in the k-NN classifier should be an odd number.
- How does attribute noise affect the classification of a "borderline example"? What is the impact of class-label noise?

3.2 Measuring Similarity

As indicated, a natural way to find the nearest neighbors of object **x** is to compare the geometrical distances of the individual training examples from **x**. Figure 3.1 illustrated this principleusing a domain so simple that the distances could be

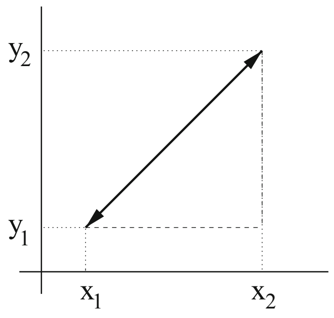

Fig. 3.2 The Euclidean distance between two points in a two-dimensional space is equal to the length of the triangle's hypotenuse

measured by a ruler. Yet the ruler will hardly be our tool of choice if the examples are described by more than two attributes. What we need is an expression to calculate the similarity based on attribute values.

Euclidean Distance In a plane, the geometric distance between two points, $\mathbf{x} = (x_1, x_2)$ and $\mathbf{y} = (y_1, y_2)$, is obtained with the help of the pythagorean theorem as indicated in Fig. 3.2: $d(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$. This formula is easy to generalize to a domain with n continuous attributes where the *Euclidean distance* between $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ is defined as follows:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\Sigma_{i=1}^{n} (x_i - y_i)^2} \tag{3.1}$$

Table 3.3 Using the nearest-neighbor principle in a 3-dimensional Euclidean space

Using the following training set of four examples described by three numeric attributes, determine the class of object $\mathbf{x} = [2, 4, 2]$.

Distance between `ex`$_i$ and $[2, 4, 2]$		
`ex`$_1$	$\{[1, 3, 1], \texttt{pos}\}$	$\sqrt{(2-1)^2 + (4-3)^2 + (2-1)^2} = \sqrt{3}$
`ex`$_2$	$\{[3, 5, 2], \texttt{pos}\}$	$\sqrt{(2-3)^2 + (4-5)^2 + (2-2)^2} = \sqrt{2}$
`ex`$_3$	$\{[3, 2, 2], \texttt{neg}\}$	$\sqrt{(2-3)^2 + (4-2)^2 + (2-2)^2} = \sqrt{5}$
`ex`$_4$	$\{[5, 2, 3], \texttt{neg}\}$	$\sqrt{(2-5)^2 + (4-2)^2 + (2-3)^2} = \sqrt{4}$

Calculating the Euclidean distances between **x** and the training examples, we realize that **x**'s nearest neighbor is `ex`$_2$. Its label being `pos`, the 1-NN classifier returns the positive label.

The same result is obtained by the 3-NN classifier because two of **x**'s three nearest neighbors (`ex`$_1$ and `ex`$_2$) are positive, and only one (`ex`$_4$), is negative.

The way this metric is used in the context of k-NN classifiers is illustrated in Table 3.3 where the training set consists of four examples described by three numeric attributes.

A More General Formulation The reader can see that the term under the square-root symbol is a sum of the squared distances along corresponding attributes.[1] This observation is mathematically expressed as follows:

$$d_M(\mathbf{x}, \mathbf{y}) = \sqrt{\Sigma_{i=1}^{n} d(x_i, y_i)} \tag{3.2}$$

This is how the distance is usually calculated in the case of vectors in which discrete and continuous attributes are mixed (in the symbol, $d_M(\mathbf{x}, \mathbf{y})$, this is indicated by the subscript, M). For instance, we can use $d(x_i, y_i) = (x_i - y_i)^2$ for continuous attributes, whereas for discrete attributes, we put $d(x_i, y_i) = 0$ if $x_i = y_i$ and $d(x_i, y_i) = 1$ if $x_i \neq y_i$.

Note that if all attributes are continuous, the formula is identical to Euclidean distance; and if the attributes are all discrete, the formula simply specifies the number of attributes in which the two vectors differ. In purely Boolean domains, where for any attribute only the values *true* or *false* are permitted (let us abbreviate these values as t and f, respectively), this latter case is called *Hamming distance*, d_H. For instance, the Hamming distance between the vectors $\mathbf{x} = (t, t, f, f)$ and $\mathbf{y} = (t, f, t, f)$ is $d_H(\mathbf{x}, \mathbf{y}) = 2$. In general, however, Eq. (3.2) is meant to be employed in domains where the examples are described by a mixture of discrete and continuous attributes.

[1]One benefit of these differences being squared, and thus guaranteed to be positive, is that this prevents negative differences, $x_i - y_i < 0$, to be subtracted from positive differences, $x_i - y_i > 0$.

Attribute-to-Attribute Distances Can Be Misleading We must be careful not to apply Formula (3.2) mechanically, ignoring the specific aspects of the given domain. Let us briefly discuss two circumstances that make it is easy to go wrong.

Suppose our examples are described by three attributes, `size`, `price`, and `season`. Of these, the first two are obviously continuous, and the last, discrete. If $\mathbf{x} = (2, 1.5, \texttt{summer})$ and $\mathbf{y} = (1, 0.5, \texttt{winter})$, then Eq. (3.2) gives the following distance between the two:

$$d_M(\mathbf{x}, \mathbf{y}) = \sqrt{(2-1)^2 + (1.5-0.5)^2 + 1} = \sqrt{3}$$

Let us first consider the third attribute: `summer` being different from `winter`, our earlier considerations lead us to establish that $d(\texttt{summer}, \texttt{winter}) = 1$. In reality, though, the difference between `summer` and `winter` is sometimes deemed greater than the difference between, say, `summer` and `fall` which are "neighboring seasons." Another line of reasoning, however, may convince us that `spring` and `fall` are more similar to each other than `summer` and `winter`—at least as far as weather is concerned. We can see that the two values, 0 and 1, will clearly not suffice, here. Intermediate values should perhaps be considered, the concrete choice depending on the specific needs of the given application. The engineer who does not pay attention to factors of this kind may fail to achieve good results.

Mixing continuous and discrete attributes can be risky in another way. A thoughtless application of Eq. (3.2) can result in a situation where the difference between two `sizes` (e.g., $\texttt{size}_1 = 1$ and $\texttt{size}_1 = 12$, which means that $d(\texttt{size}_1, \texttt{size}_2) = 11^2 = 121$) can totally dominate the difference between two `seasons` (which, in the baseline version, could not exceed 1). This observation is closely related to the problem of *scaling*—discussed in the next section.

Distances in General The reader is beginning to understand that the issue of similarity is far from trivial. Apart from Eq. (3.2), quite a few other formulas have been suggested, some of them fairly sophisticated.[2] While it is good to know they exist, we will not examine them here because we do not want to digress too far from our main topic. Suffice it so say that any distance metric has to satisfy the following requirements:

1. the distance must never be negative;
2. the distance between two identical vectors, $\mathbf{x}$ and $\mathbf{y}$, is zero;
3. the distance from $\mathbf{x}$ to $\mathbf{y}$ is the same as the distance from $\mathbf{y}$ to $\mathbf{x}$;
4. the metric must satisfy the triangular inequality: $d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \geq d(\mathbf{x}, \mathbf{z})$.

[2]Among these, perhaps the best-known are the polar distance, the Minkowski metric, and the Mahalanobis distance.

What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- What is Euclidean distance, and what is Hamming distance? In what domains can they be used? How is distance related to similarity?
- Write down the distance formula for domains where examples are described by a mixture of continuous and discrete attributes. Discuss the difficulties that complicate its straightforward application in practice.
- What fundamental properties have to be satisfied by any distance metric?

3.3 Irrelevant Attributes and Scaling Problems

By now, the reader understands the principles of the *k*-NN classifier well enough to be able to write a computer program implementing the tool. This is not enough, though. If applied mechanically, the software may disappoint. It is necessary to understand why.

The rock-bottom of the nearest-neighbor paradigm is that, "objects are similar if the geometric distance between the vectors describing them is small." This said, we must be careful not to let this principle degenerate into a dogma. In certain situations, the geometric distance can be misleading. Two of them are quite common.

Irrelevant Attributes It would be a mistake to think that all attributes are created equal. They are not. In the context of machine learning, some are *irrelevant* in the sense that their values have nothing to do with the given example's class. But they do affect the geometric distance between vectors.

A simple illustration will clarify the point. In the training set from Fig. 3.3, the examples are characterized by two numeric attributes: `body-temperature` (the horizontal axis) and `shoe-size` (the vertical axis). The black dot stands for the object that the *k*-NN classifier is expected to label as healthy (`pos`) or sick (`neg`).

As expected, all positive examples find themselves in the shaded area delimited by two critical points along the "horizontal" attribute: temperatures exceeding the maximum indicate fever; those below the minimum, hypothermia. As for the "vertical" attribute, though, we see that the positive and negative examples alike are distributed along the entire range, `show-size` being unable to betray anything about a person's health. The object we want to classify is in the highlighted region, and common sense requires that it should be labeled as positive—despite the fact that its nearest neighbor happens to be negative.

The Lesson If only the first attribute is used, the Euclidean distance between the two examples is $d_E(x, y) = \sqrt{(x_1 - y_1)^2} = |x_1 - y_1|$. If both attributes are used, the Euclidean distance will be $d_E(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$. If the second

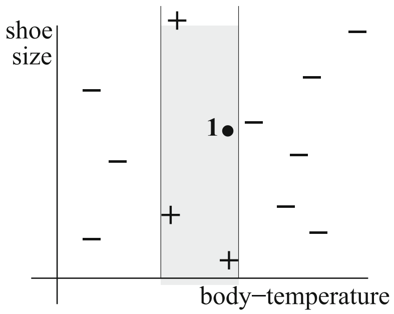

Fig. 3.3 The "vertical" attribute is irrelevant for classification, and yet it affects the geometric distances between examples. Object **1** is positive, even though its nearest neighbor in the 2-dimensional space is negative

attribute is irrelevant, then the term $(x_2 - y_2)^2$ is superfluous—and yet it affects, adversely, k-NN's notion of similarity! This is what occurred in Fig. 3.3, and this is why object **1** was misclassified.

How much damage is caused by irrelevant attributes depends on how many of them are used to describe the examples. In a domain with hundreds of attributes, of which only one is irrelevant, there is no need to panic: one lonely culprit is unlikely to distort the value of $d_E(\mathbf{x}, \mathbf{y})$ in any meaningful way. But things will change as the percentage of irrelevant attributes grows. If the vast majority of the attributes have nothing to do with the class we want to recognize, then the geometric distance will become almost meaningless, and the classifier's performance will be dismal.

The Scales of Attribute Values Suppose we want to evaluate the similarity of two examples, $\mathbf{x} = (t, 0.2, 254)$ and $\mathbf{y} = (f, 0.1, 194)$, described by three attributes, of which the first is boolean, the second is continuous with values from interval [0; 1], and the third is continuous with values from interval [0; 1000]. Using Eq. (3.2), the reader will find it easy to calculate the distance between $\mathbf{x}$ and $\mathbf{y}$, obtaining the following:

$$d_M(\mathbf{x}, \mathbf{y}) = \sqrt{(1-0)^2 + (0.2-0.1)^2 + (254-194)^2}$$

Inspecting this expression, we notice that the third attribute completely dominates, reducing the other two to virtual insignificance. No matter how we modify their values within their ranges, the distance, $d_M(\mathbf{x}, \mathbf{y})$, will hardly change. Fortunately, the situation is easy to rectify. If we divide, in the training set, all values of the third attribute by 1000, thus "squeezing" its range to [0; 1], the impacts of the attributes will become more balanced. We can see that the scales of the attribute values can radically affect the k-NN classifier's behavior.

Another Aspect of Attribute Scaling There is more to it. Consider the following two training examples, $\texttt{ex}_1$ and $\texttt{ex}_2$, and the object $\mathbf{x}$ whose class we want to determine:

$$\begin{aligned}\texttt{ex}_1 &= [(10, 10), \texttt{pos})]\\ \texttt{ex}_2 &= [(20, 0), \texttt{neg})]\\ \mathbf{x} &= (32, 20)\end{aligned}$$

The distances are $d_M(\mathbf{x}, \texttt{ex}_1) = \sqrt{584}$ and $d_M(\mathbf{x}, \texttt{ex}_2) = \sqrt{544}$. The latter being smaller, the 1-NN classifier will label **x** as `neg`. Suppose, however, that the second attribute expresses `temperature`, and does so in centigrades. If we decide to use Fahrenheits instead, the three vectors will change as follows:

$\texttt{ex}_1 = [(10, 50), \texttt{pos})]$
$\texttt{ex}_2 = [(20, 32), \texttt{neg})]$
$\mathbf{x} = (32, 68)$

Recalculating the distances, we obtain $d_M(\mathbf{x}, \texttt{ex}_1) = \sqrt{808}$ and $d_M(\mathbf{x}, \texttt{ex}_2) = \sqrt{1440}$. This time, it is the first distance that is smaller, and 1-NN will therefore classify **x** as positive. This seems a bit silly. The examples are still the same, except that we chose different units for `temperature`; and yet the classifier's verdict has changed.

Normalizing Attribute Scales One way out of this trouble is to *normalize* the attributes: to re-scale them in a way that makes all values fall into the same interval, [0; 1]. From the several mechanisms that have been used to this end, perhaps the simplest is the one that first identifies, for the given attribute, its maximum (*MAX*) and minimum (*MIN*), and then replaces each value, *x*, of this attribute using the following formula:

$$x = \frac{x - MIN}{MAX - MIN} \tag{3.3}$$

A simple illustration will show us how this works. Suppose that, in the training set consisting of five examples, a given attribute acquires the following values, respectively:

$$[7, 4, 25, -5, 10]$$

We see that $MIN = -5$ and $MAX = 25$. Subtracting MIN from each of the values, we obtain the following:

$$[12, 9, 30, 0, 15]$$

The reader can see that the "new minimum" is 0, and the "new maximum" is $MAX - MIN = 25 - (-5) = 30$. Dividing the obtained values by $MAX - MIN$, we obtain a situation where all the values fall into [0; 1]:

$$[0.4, 0.3, 1, 0, 0.5]$$

One Potential Weakness of Normalization Normalization reduces error rate in many practical applications, especially if the scales of the original attributes vary significantly. The downside is that the description of the examples thus becomes distorted. Moreover, the pragmatic decision to make all values fall between 0 and 1 may not be justified. For instance, if the difference between `summer` and `fall` is 1, it will always be bigger than, say, the difference between two normalized body temperatures. Whether this matters or not is up to the engineer's common sense—assisted by his or her experience and perhaps a little experimentation.

What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- Why do irrelevant attributes impair the *k*-NN classifier's performance? How does the performance depend on the number of irrelevant attributes?
- Explain the basic problems pertaining to attribute scaling. Describe a simple approach to normalization.

3.4 Performance Considerations

Having spent all this time exploring various aspects of the *k*-NN classifier, the reader is bound to ask: should I really care? Sure enough, the technique is easy to implement in a computer program, and its function is easy to grasp. But is there a reason to believe that its classification performance is good enough?

The 1-NN Classifier Versus Ideal Bayes Let us give the question some thought. The ultimate yardstick by which to measure *k*+-NN's potential is the Bayesian approach. We will recall that if the probabilities and *pdf*'s employed by the Bayesian formula are known with absolute accuracy, then this classifier—let us call it *Ideal Bayes*—exhibits the lowest error rate theoretically achievable on the given (noisy) data. It would be reassuring to know that the *k*-NN paradigm does not trail too far behind.

The question has been subjected to rigorous mathematical analysis, and here are the results. Figure 3.4 shows what the comparison will look like under certain ideal circumstances such as an infinitely large training set which fills the instance space with infinite density. The solid curve represents the two-class case where each example is either positive or negative. We can see that if the error rate of the *Ideal Bayes* is 5%, the error rate of the 1-NN classifier (vertical axis) is 10%. With the growing amount of noise, the difference between the two approaches decreases, only to disappear when the *Ideal Bayes* suffers 50% error rate—in which event, of course, the labels of the training examples are virtually random, and any attempt at automated classification is doomed anyway. The situation is not any better in multi-class domains, represented in the graph by the dotted curve. Again, the *Ideal Bayes* outperforms the 1-NN classifier by a comfortable margin.

Increasing the Number of Neighbors From the perspective of the 1-NN classifier, the results from Fig. 3.4 are rather discouraging. On the other hand, we know that the classifier's performance might improve when we use the more general k-NN (for $k > 1$), where some of the noisy nearest neighbors get outvoted by better-behaved ones. Does mathematics lend some support to this expectation?

Yes it does. Under the above-mentioned ideal circumstances, the error rate has been proven to decrease with the growing value of k, until it converges to that of *Ideal Bayes* for $k \rightarrow \infty$. At least in theory, then, the performance of the nearest-neighbor classifier is able to reach the absolute maximum.

Practical Limitations of Theories The engineer's world is indifferent to theoretical requirements. In a realistic application, the training examples will but sparsely populate the instance space, and increasing the number of voting neighbors can be counterproductive. More often than not, the error rate *does* improve with the growing k, but only up to a certain point from which it starts growing again—as illustrated in Fig. 3.5 where the horizontal axis represents the values of k, and the vertical axis represents the error rate that is measured on an independent testing set.

The explanation is simple: the more distant "nearest neighbors" may find themselves in regions (in the instance space) that are already occupied by other classes; as such, they only mislead the classifier. Consider the extreme: if the training

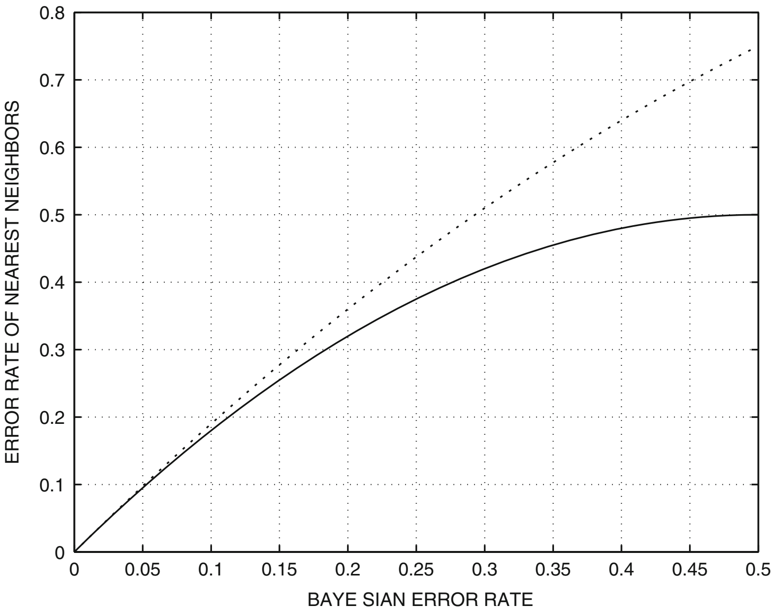

Fig. 3.4 The theoretical error rate of the 1-NN rule compared to that of the *Ideal Bayes*. Two classes: the *solid curve*; many classes: the *dotted curve*

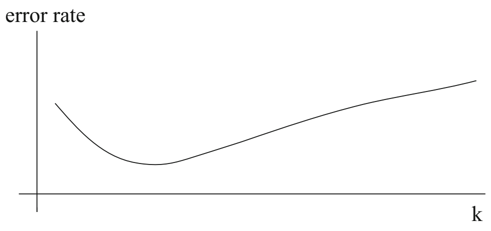

Fig. 3.5 With the growing number of voting neighbors (k), the error rate of the k-NN classifier decreases until it reaches a level from which it starts growing again

set contains 25 training examples, then the 25-NN classifier is useless because it simply labels any object with the class that has the highest number of representatives in the training data.[3]

The Curse of Dimensionality Obviously, when classifying object $\mathbf{x}$, some of its nearest neighbors may *not* be similar enough to $\mathbf{x}$ to deserve to participate in the vote. This often happens in domains marked by many attributes. Suppose that the values of each attribute are confined to the unit-length interval, $[0; 1]$. Using the pythagorean theorem, it would be easy to show that the maximum Euclidean distance in the n-dimensional space defined by these attributes is $d_{MAX} = \sqrt{n}$. For $n = 10^4$ (quite reasonable in such domains as, say, text categorization), this means $d_{MAX} = 100$. In view of the fact that no attribute value can exceed 1, this is perhaps surprising. No wonder that examples then tend to be sparse—unless the training set is really very large.

The term sometimes mentioned, in this context, is the *curse of dimensionality*: as we increase the number of attributes, the number of training examples needed to fill the instance space with adequate density grows very fast, perhaps so fast as to render the nearest-neighbor paradigm impractical.

To Conclude Although the ideal k-NN classifier is capable of reaching the performance of the *Ideal Bayes*, the engineer has to be aware of the practical limitations of both approaches. Being able to use the *Ideal Bayes* is unrealistic in the absence of the perfect knowledge of the probabilities and *pdf*'s. On the other hand, the k-NN classifier is prone to suffer from sparse data, from irrelevant attributes, and from scaling-related problems. The concrete choice has to be based on the specific requirements of the given application.

What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

[3]The optimal value of k (the one with the minimum error rate) is usually established experimentally.

- How does the performance of the k-NN classifier compare to that of the *Ideal Bayes*? Summarize this separately for $k = 1$ and $k > 1$. What theoretical assumptions do these two paradigms rely on?
- How will the performance of a k-NN classifier depend on the growing values of k in theory and in a realistic application?
- What is understood by the *curse of dimensionality*?

3.5 Weighted Nearest Neighbors

So far, the voting mechanism was assumed to be democratic, each nearest neighbor having the same vote. This seems to be a fair enough arrangement, but from the perspective of classification performance, we can do better.

In Fig. 3.6, the task is to determine the class of object **x**, represented by a black dot. Since three of the nearest neighbors are negative, and only two are positive, the 5-NN classifier will label **x** as negative. And yet, something seems to be wrong, here: the three negative neighbors are quite distant from **x**; as such, they may not deserve to have the same say as the two positive examples in the object's immediate vicinity.

Weighted Nearest Neighbors Situations of this kind motivate the introduction of *weighted voting*, in which the weight of each of the nearest neighbors is made proportional to its distance from **x**: the closer the neighbor, the greater its impact.

Let us denote the weights as $w_1, \ldots w_k$. The *weighted k-NN* classifier sums up the weights of those neighbors that recommend the positive class (let the result be denoted by Σ^+) and then sums up the weights of those neighbors that support the negative class (Σ^-). The final verdict is based on which is higher: if $\Sigma^+ > \Sigma^-$, then **x** is deemed positive, otherwise it is deemed negative. Generalization to the case with $n > 2$ classes is straightforward.

For the sake of illustration, suppose the positive label is found in neighbors with weights 0.6 and 0.7, respectively, and the negative label is found in neighbors with weights $0.1, 0.2$, and 0.3. The weighted k-NN classifier will choose the positive

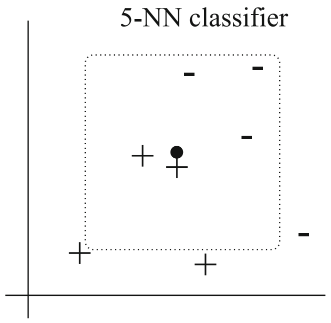

Fig. 3.6 The testimony of the two "positive" neighbors should outweigh that of the three more distant "negative" neighbors

class because the combined weight of the positive neighbors, $\Sigma^+ = 0.6+0.7 = 1.3$, is greater than that of the negative neighbors, $\Sigma^- = 0.1 + 0.2 + 0.3 = 0.6$. Just as in Fig. 3.6, the more frequent negative label is outvoted by the positive neighbors because the latter are closer to $\mathbf{x}$.

A Concrete Formula Let us introduce a simple formula to calculate the weights. Suppose the k neighbors are ordered according to their distances, $d_1, \ldots, d_k$, from $\mathbf{x}$ so that d_1 is the smallest distance and d_k is the greatest distance. The weight of the i-th closest neighbor is calculated as follows:

$$w_i = \begin{cases} \frac{d_k - d_i}{d_k - d_1}, & d_k \neq d_1 \\ 1 & d_k = d_1 \end{cases} \tag{3.4}$$

Obviously, the weights thus obtained will range from 0 for the most distant neighbor to 1 for the closest one. This means that the approach actually considers only $k-1$ neighbors (because $w_k = 0$). Of course, this makes sense only for $k > 3$. If we used $k = 3$, then only two neighbors would really participate, and the weighted 3-NN classifier would degenerate into the 1-NN classifier.

Table 3.4 illustrates the procedure on a simple toy domain.

Another important thing to observe is that if all nearest neighbors have the same distance from $\mathbf{x}$, then they all get the same weight, $w_i = 1$, on account of the bottom part of the formula in Eq. (3.4). The reader will easily verify that $d_k = d_1$ if and only if all the k nearest neighbors have the same distance from $\mathbf{x}$.

Table 3.4 Illustration of the weighted nearest neighbor rule

The task is to use the weighted 5-NN classifier to determine the class of object $\mathbf{x}$. Let the distances between $\mathbf{x}$ and the five nearest neighbors be $d_1 = 1, d_2 = 3, d_3 = 4, d_4 = 5, d_5 = 8$. Since the minimum is $d_1 = 1$ and the maximum is $d_5 = 8$, the individual weights are calculated as follows:

$$w_i = \frac{d_5 - d_i}{d_5 - d_1} = \frac{8 - d_i}{8 - 1} = \frac{8 - d_i}{7}$$

This gives the following values:

$$w_1 = \tfrac{8-1}{7} = 1, w_2 = \tfrac{8-3}{7} = \tfrac{5}{7}, w_3 = \tfrac{8-4}{7} = \tfrac{4}{7}, w_4 = \tfrac{8-5}{7} = \tfrac{3}{7}, w_5 = \tfrac{8-8}{7} = 0.$$

If the two nearest neighbors are positive and the remaining three are negative, then $\mathbf{x}$ is classified as positive because $\Sigma^+ = 1 + \frac{5}{7} \; > \; \Sigma^- = \frac{4}{7} + \frac{3}{7} + 0$.

What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- Why did we argue that each of the voting nearest neighbors should sometimes have a different weight?
- Discuss the behavior of the formula recommended in the text for the calculation of the individual weights.

3.6 Removing Dangerous Examples

As far as the classification procedure is concerned, the value of each individual training example can be different. Some are typical of the classes they represent, others less so, and yet others are outright misleading. This is why it is often a good thing to pre-process the training set before using it: to remove examples suspected of being less than useful.

The concrete method to pre-process the training set is guided by two essential observations that are illustrated in Fig. 3.7. First, an example labeled with one class

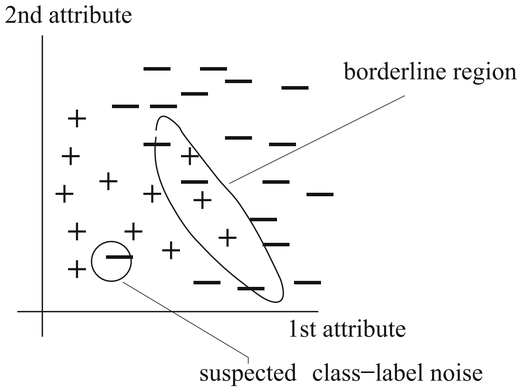

Fig. 3.7 Illustration of two potentially harmful types of examples: those surrounded by examples of a different class, and those in the "borderline region."

but surrounded by examples of another class is likely to be the result of class-label noise. Second, examples from the borderline region between two classes are unreliable because even small amount of noise in their attribute values can shift their locations in the wrong directions, thus affecting classification. In both cases, the examples better be removed.

The Technique of Tomek Links Before the culprit can be removed, however, it has to be detected. One way to do so is by the technique of *Tomek Links*, named so after the mathematician who first used them a few decades ago.[4] A pair of examples, **x** and **y**, are said to form a *Tomek Link* if the following three requirements are satisfied at the same time:

1. **x** is the nearest neighbor of **y**
2. **y** is the nearest neighbor of **x**
3. the class of **x** is not the same as the class of **y**.

These conditions being characteristic of borderline examples, and also of examples surrounded by examples of anotherclass, it makes sense to remove from the

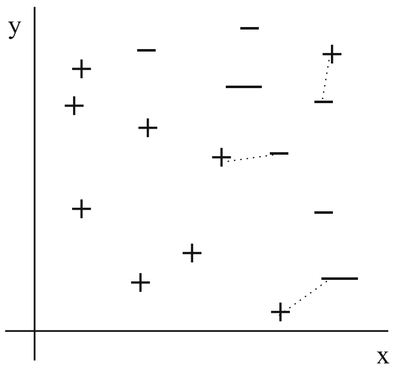

Fig. 3.8 *Dotted lines* connect all Tomek links. Each participant in a Tomek Link is its partner's nearest neighbor, and each of the two examples has a different class

training set all such pairs. Even this may not be enough. Sometimes, the removal of existing *Tomek Links* only creates new ones so that the procedure has to be repeated.

The algorithm is summarized by the pseudocode in Table 3.5, and a few instances of *Tomek Links* are shown in Fig. 3.8. Note that there are only these three; no other pair of examples satisfies here the criteria for being called a *Tomek Link*.

One side-effect perhaps deserves to be mentioned: once the training set has been cleaned, the number (k) of the voting nearest neighbors can be reduced because the main reason for using a $k > 1$ is to mitigate the negative impact of noise—which the removal of *Tomek Links* has reduced. It can even happen that the 1-NN classifier will now be able to achieve the performance of a k-NN classifier that uses the entire original training set.

A Limitation Nothing is perfect. The technique of *Tomek Links* does not identify all misleading examples; and, conversely, some of the removed examples might have been "innocent," and thus deserved to be retained. Still, experience has shown that the removal of *Tomek Links* usually does improve the overall quality of the data.

[4]It is fair to mention that he used them for somewhat different purposes.

Table 3.5 The algorithm to identify (and remove) *Tomek Links*

```
Input: the training set of $N$ examples

1. Let $i = 1$ and let $T$ be an empty set.
2. Let $\mathbf{x}$ be the $i$-th training example and let $\mathbf{y}$ be the nearest neighbor of $\mathbf{x}$.
3. If $\mathbf{x}$ and $\mathbf{y}$ belong to the same class, go to 5.
4. If $\mathbf{x}$ is the nearest neighbor of $\mathbf{y}$, let $T = T \cup \{\mathbf{x}, \mathbf{y}\}$.
5. Let $i = i + 1$. If $i \leq N$, goto 2.
6. Remove from the training set all examples that are now in $T$.
```

The engineer only has to be careful in two specific situations. (1) When the training set is very small, and (2) when one of the classes significantly outnumbers the other. The latter case will be discussed in Sect. 10.2.

What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- What motivates the attempts to "clean" the training set?
- What are *Tomek Links*, are how do we identify them in the training set? Why does the procedure sometimes have to be repeated?
- How does the removal of *Tomek Links* affect the k-NN classifier?

3.7 Removing Redundant Examples

Some training examples do not negatively affect classification performance, and yet we want to get rid of them. Thing is, they do not reduce the error rate, either; they only add to computational costs.

Redundant Examples To find the nearest neighbor in a domain with 10^6 training examples described by 10^4 attributes (and such domains are not rare), the program relying on the Euclidean distance has to carry out $10^6 \times 10^4 = 10^{10}$ arithmetic operations. When the task is to classify thousands of objects at a time (which, again, is not impossible), the number of arithmetic operations is $10^{10} \times 10^3 = 10^{13}$. This is a lot.

Fortunately, training sets are often redundant in the sense that the k-NN classifier's behavior will not change even if some of the training examples are deleted. Sometimes, a great majority of the examples can be removed with impunity because they add to classification costs without affecting classification performance. This is the case of the domain shown in the upper-left corner of Fig. 3.9.

Consistent Subset In an attempt to reduce redundancy, we want to replace the training set, T, with its *consistent subset*, S. In our context, S is said to be a consistent subset of T if replacing T with S does not affect what class labels are returned by the k-NN classifier. Such definition, however, is not very practical because we have no idea of how the k-NN classifier will behave—using either T or S—on future examples. Let us therefore modify the criterion: S will be regarded as a consistent subset of T if any $\mathbf{ex} \in T$ receives the same label no matter whether the k-NN classifier has employed $T - \{\mathbf{ex}\}$ or $S - \{\mathbf{ex}\}$.

It is in the nature of things that a realistic training set will have many consistent subsets to choose from. Of course, the smaller the subset, the better. But a perfectionist who insists on always finding the smallest one should be warned: this ideal can often be achieved only at the price of enormous computational costs. The practically minded engineer doubts that these costs are justified, and will be content with a computationally efficient algorithm that downsizes the original set to a "reasonable size," unscientific though such formulation may appear to be.

Creating a Consistent Subset One such pragmatic technique is presented in Table 3.6. The algorithm starts by choosing one random example from each class.

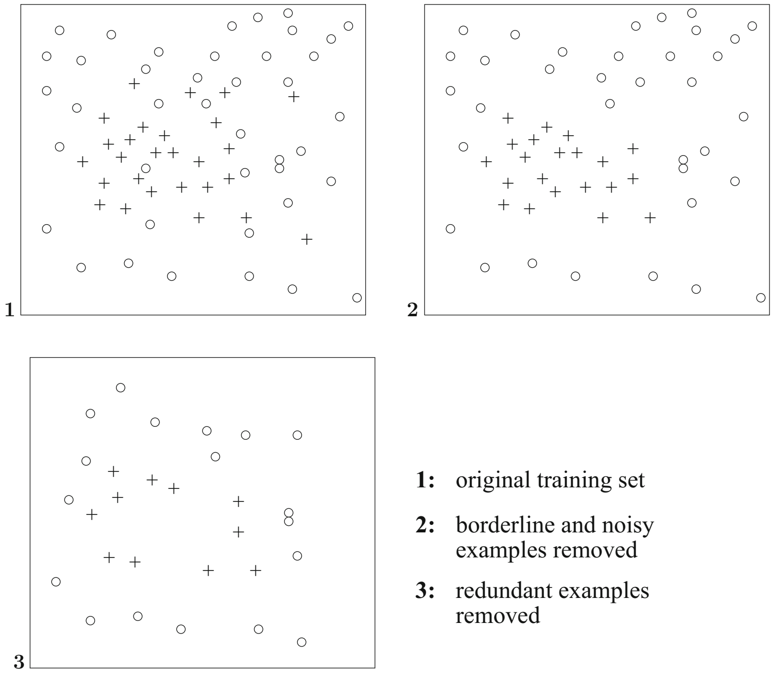

Fig. 3.9 An illustration of what happens when the borderline, noisy, and redundant examples are removed from the training set

Table 3.6 Algorithm to create a consistent training subset by the removal of (some) redundant examples

1. Let S contain one positive and one negative example from the training set, T.
2. Using examples from S, re-classify the examples in T with the 1-NN classifier. Let M be the set of those examples that have in this manner received the wrong class.
3. Copy to S all examples from M.
4. If the contents of S have not changed, in the previous step, then *stop*; otherwise go to step 1.

This initial subset, S, is then used by the 1-NN classifier to suggest the labels of all training examples. At this stage, it is more than likely that some training examples will thus be misclassified. These misclassified examples are added to S, and the whole procedure is repeated using this larger set. This is then repeated again and again until, at a certain moment, S becomes representative enough to allow the 1-NN classifier to label all training examples correctly. This is when the search stops.

What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- What is the benefit of removing redundant examples from the training set?
- What do we mean by the term, "consistent subset of the training set"? Why is it not necessary always to look for the smallest consistent subset?
- Explain the principle of the simple algorithm that creates a reasonably sized consistent subset.

3.8 Summary and Historical Remarks

- When classifying object $\mathbf{x}$, the k-NN classifier finds, in the training set, k examples most similar to $\mathbf{x}$, and then chooses the class label most common among these examples.
- Classification behavior of the k-NN classifier depends to a great extent on how similarities between attribute vectors are calculated. Perhaps the simplest way to establish the similarity between vectors $\mathbf{x}$ and $\mathbf{y}$ is to use their geometric distance obtained by the following formula:

$$d_M(\mathbf{x},\mathbf{y}) = \sqrt{\Sigma_{i=1}^{n} d(x_i, y_i)} \tag{3.5}$$

  Essentially, we use $d(x_i, y_i) = (x_i - y_i)^2$ for continuous attributes, whereas for discrete attributes, we put $d(x_i, y_i) = 0$ if $x_i = y_i$ and $d(x_i, y_i) = 1$ if $x_i \neq y_i$.

- The performance of the k-NN classifier is poor if many of the attributes describing the examples are irrelevant. Another issue is the scaling of the attribute values. The latter problem can be mitigated by normalizing the attribute values to unit intervals.
- Some examples are harmful in the sense that their presence in the training set increases error rate. Others are redundant in that they only add to the computation costs without improving classification performance. Harmful and redundant examples should be removed.
- In some applications, each of the nearest neighbors can have the same vote. In others, the votes are weighted based on the distance of the examples from the classified object.

Historical Remarks The idea of the nearest-neighbor classifier was originally proposed by Fix and Hodges [27], but the first systematic analysis was offered by Cover and Hart [20] and Cover [19]. Exhaustive treatment was then provided by the book by Dasarathy [21]. The weighted k-NN classifier described here was proposed by Dudani [23]. The oldest technique to find a consistent subset of the training set was described by Hart [35]—the one introduced in this chapter is its minor modification. The notion of *Tomek Links* is due to Tomek [89].

3.9 Solidify Your Knowledge

The exercises are to solidify the acquired knowledge. The suggested thought experiments will help the reader see this chapter's ideas in a different light and provoke independent thinking. Computer assignments will force the readers to pay attention to seemingly insignificant details they might otherwise overlook.

Exercises

1. Determine the class of $\mathbf{y} = [1, 22]$ using the 1-NN classifier and the 3-NN classifier, both using the training examples from Table 3.7. Explain why the two classifiers differ in their classification behavior.
2. Use the examples from Table 3.7 to classify object $\mathbf{y} = [3, 3]$ with the 5-NN classifier. Note that two nearest neighbors are positive and three nearest neighbors are negative. Will weighted 5-NN classifier change anything? To see what is going on, plot the locations of the examples in a graph.

Table 3.7 A simple set of training examples for the exercises

x_1	1	1	1	2	3	3	3	4	5
x_2	1	2	4	3	0	2	5	4	3
class	+	−	−	+	+	+	−	−	−

3. Again, use the training examples from Table 3.7. (a) Are there any Tomek links? (b) can you find a consistent subset of the training set by the removal of at least one redundant training example?

Give It Some Thought

1. Discuss the possibility of applying the *k*-NN classifier to the "pies" domain. Give some thought to how many nearest neighbors to use, and what distance metric to employ, whether to make the nearest neighbors' vote depend on distance, and so on.
2. Suggest other variations on the nearest-neighbor principle. Use the following hints:
   (a) Introduce alternative distance metrics. Do not forget that they have to satisfy the axioms mentioned at the end of Sect. 3.2.
   (b) Modify the voting scheme by assuming that some examples have been created by a knowledgeable "teacher," whereas others have been obtained from a database without any consideration given to their representativeness. The teacher's examples should obviously carry more weight.
3. Design an algorithm that uses hill-climbing search to remove *redundant examples*. Hint: the initial state will contain the entire training set, the search operator will remove a single training example at a time (this removal must not affect behavior).
4. Describe an algorithm that uses hill-climbing search to remove *irrelevant attributes*. Hint: withhold some training examples on which you will test 1-NN's classifier's performance for different subsets of attributes.

Computer Assignments

1. Write a program whose input is the training set, a user-specified value of k, and an object, $\mathbf{x}$. The output is the class label of $\mathbf{x}$.
2. Apply the program implemented in the previous assignment to some of the benchmark domains from the UCI repository.[5] Always take 40% of the examples out and reclassify them with the 1-NN classifier that uses the remaining 60%.
3. Create a synthetic toy domain consisting of 1000 examples described by a pair of attributes, each from the interval [0,1]. In the square defined by these attribute values, $[0,1] \times [0,1]$, define a geometric figure of your own choice and label all examples inside it as positive and all other examples as negative. From this initial

[5] www.ics.uci.edu/~mlearn/MLRepository.html.

noise-free data set, create 5 files, each obtained by changing p percent of the class labels, using $p \in \{5, 10, 15, 20, 25\}$ (thus obtaining different levels of class-label noise).

Divide each data file into two parts, the first to be reclassified by the k-NN classifier that uses the second part. Observe how different values of k result in different behaviors under different levels of class-label noise.

4. Implement the Tomek-link method for the removal of harmful examples. Repeat the experiments above for the case where the k-NN classifier uses only examples that survived this removal. Compare the results, observing how the removal affected the classification behavior of the k-NN classifier for different values of k and for different levels of noise.