Chapter 10
Some Practical Aspects to Know About

The engineer who wants to avoid disappointment has to be aware of certain machine-learning aspects that, for the sake of clarity, our introduction to the basic techniques had to neglect. To present some of the most important ones is the task for this chapter.

The first thing to consider is bias: to be able to learn, the learner has to build on certain assumptions about the problem at hand, thus reducing the size of the search space. The next important point has to do with the observation that an increase in the size of the training set can actually hurt the learner's chances if most of the training examples belong only to one class. After this, we will discuss the question how to deal with classes whose definitions tend to change with context or in time. The last part focusses on some more mundane aspects such as unknown attribute values, the selection of the most useful sets of attributes, and the problem of multi-label examples.

10.1 A Learner's Bias

Chapter 7 boldly declared that there is "no learning without bias." The point was that an unconstrained (and hence extremely large) hypothesis space is bound to contain many hypotheses that only by mere chance correctly classify the entire training set while still erring a lot on future examples. There is another, more practical side to it. To be able to find something, you need to know where to look; and the smaller the place where that something is hidden, the higher the chances of finding it.

A Simple Example Suppose we are to identify the property shared by the following set of integers: $\{2, 3, 10, 20, 12, 21, 22, 28\}$. In the language of machine learning, these constitute positive examples. Alongside these, also negative examples are provided: $\{1, 4, 5, 11\}$. In these, the property is *not* present.

M. Kubat, *An Introduction to Machine Learning*,
DOI 10.1007/978-3-319-63913-0_10

A student trying to find the answer typically explores various notions offered by the number theory, such as primes, odd numbers, integers whose values exceed a certain threshold, results of arithmetic operations, and so on. After a lot of effort, some property satisfying the training examples is found. Usually, however, the discovered rule is ridiculously complicated and awkward to say the least.

And yet, there is a simple solution which the students almost never hit upon. Thing is, the underlying property does not come from the realm of arithmetics. What the positive examples have in common (and the negative examples lack) is that they all begin with the letter `t`: *t*wo, *t*hree, . . ., all the way to *t*wenty-eight. Conversely, none of the integers in the set of negative examples begins with a `t`.

The reason this simple solution is so difficult to find is that most people search for it in the wrong place: arithmetics. Expressed in the language of machine learning, they rely on the wrong *bias*. Once they are given the correct answer, their mindset will be willing to take this experience into account in the future. If you give them another puzzle of a similar nature, they will subconsciously think not only about arithmetics, but also about the English vocabulary—they will incorporate into their thinking also this new bias.

Representational Bias Versus Procedural Bias As far as machine learning is concerned, biases come in different forms. A so-called *representational bias* is determined by the language in which we want the classifier to be formulated. For instance, in domains with continuous-valued attributes, one possible representational bias can consist in the choice of a linear classifier; another can be the preference for polynomials, and yet another the preference for neural networks. If all attributes are discrete-valued, the engineer may prefer conjunctions of attribute values, or perhaps even decision trees. Of course, all of these biases then have its advantages as well as shortcomings.

Apart from this, the engineer usually also has to opt for a certain *procedural bias*. By this we mean preference to a certain method of searching for the solution, the selection of a specific machine-learning procedure. For instance, one such bias relies on the assumption that pruning will improve the classification performance of a decision tree on future data. Another procedural bias is the choice of a concrete set of parameters in a neural-network's training. And yet another, in the field of linear classifiers, can be the engineer's decision to employ the *perceptron learning* instead of WINNOW—or the other way round.

The Strength of a Bias Versus the Correctness of a Bias Suppose the engineer wants to decide whether to approach the given machine-learning problem with a linear classifier or with a neural network. If the positive and negative examples are linearly separable, then the linear classifier is clearly the better choice. While both paradigms are capable of finding the solution, neural networks tend to overfit the training set, thus poorly generalizing to future data. On the other hand, if the tentative boundary separating the two classes is highly non-linear, then the linear classifier will lack the necessary flexibility, whereas neural networks will probably manage quite easily. The reader now begins to understand that each bias has two critical aspects: strength, and correctness.

A bias is *strong* if it defines only a narrow class of classifiers. In this sense, the bias of linear classifiers is much stronger than that of neural networks: the former only allow for linear decision surfaces, while the latter can model virtually *any* decision surface.

A bias is *correct* if it is the right one for the task at hand. For instance, the linear classifier's bias is correct only in a domain where the positive examples are linearly separable from the negative ones. A conjunction of boolean attributes is correct only if the underlying class can indeed be described by a conjunction of attributes. Of course, the opposite term, *incorrect* bias, is not a crisp concept. Some gradation is involved; some biases are only slightly incorrect, others significantly so.

A Useful Rule of Thumb: Occam's Razor Ideally, the engineer wants to use a bias (representational or procedural) that is correct. And if there is a possibility to choose between two or more biases that are all correct, the stronger bias is to be preferred because it has a higher chance of success—this is what we learned in Chap. 7 where the advice to choose the simpler solution was presented under the name of the *Occam's Razor*.

Unfortunately, we rarely know in advance the correctness/incorrectness of all possible biases. An educated guess is the best we can hope for. In some paradigms, say, high-order polynomials, the bias is so weak that there is a high probability that a classifier from this class will classify the entire training set with zero error rate; and yet its performance on future data is uncertain on account of its problems with PAC-learnability. Strengthening the bias (say, by reducing a polynomial's order) will reduce the VC-dimension, increasing the chances on future data—but only as long as the bias remains correct. At a certain moment, further strengthening of the bias will do more harm than good because the bias becomes incorrect, perhaps very much so.

What we need to remember is the existence of an almost inescapable trade-off: a mildly incorrect but strong bias can be better than a correct but very weak bias. But what the term, "mildly incorrect bias," means in a concrete application can usually be decided only based on the engineer's experience or by additional experimentation (see Chap. 11).

"Lifelong Learning" In some applications, the machine-learning software is to learn a series of concepts or classes, all of which are expected to have a solution within the realm of the same specific bias. In this event, it makes sense to organize the learning procedure in two tiers. At the lower level, the task is to identify the most appropriate bias; at the higher level, the software induces the classifier using this bias. The term used for this strategy, "lifelong learning," reminds us of something typical of our own human difficulties in learning: the need to "learn how to learn" in a given field.

Two Sources of the Classifier's Errors The observations made so far will help us get a better grasp of the two main sources of a classifier's errors.

The first is the *variance* in the training examples. Thing is, the data used for the induction of the concrete classifier almost never capture all aspects of the

underlying class. This is partly due to the way the training set has been created. In some applications, the training set has been created at random. In other domains, it consists of examples available at the given moment, which involves a great deal of randomness, too. And in yet others, the training set has been created by an expert who has chosen the examples which he believes best represent the given class. The last case is inevitably subjective, and thus no less unreliable than the previous two. In view of all this, one can easily imagine that a different training set might be created for the same domain. And here is the point. From a different training set, a somewhat different classifier will be induced, and this different classifier will make somewhat different errors on future data. This is what we mean by saying that variance in the training data is an important source of errors. Its negative effect can often be reduced if we use really large training sets.

The second source of error is *bias-related*. If the two classes, `pos` and `neg`, are not linearly separable, then any linear classifier is bound to misclassify certain minimum percentage of future examples. Bias-related errors cannot be reduced below a certain limit because they are inherent in the very nature of the selected type of classifier.

It is instructive to give some thought to the *trade-off* between the two sources. For one thing, the bias-related error can be reduced if we resort to a machine-learning paradigm known to have a weaker bias. Unfortunately, one of the unintended consequences of such a decision is higher variance. Conversely, variance can in principle be reduced by strengthening the bias—which, if incorrect, will increase the frequency of bias-related errors.

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.

- Explain the difference between the representational bias and the procedural bias. Illustrate each type by examples.
- Explain the difference between the strong and weak bias. Explain the difference between the correct and incorrect bias. Discuss the interrelation of the two dichotomies.
- What has this section taught us about the two typical causes of a classifier's underperformance on future data?

10.2 Imbalanced Training Sets

When discussing the oil-spill domain, Sect. 8.2 pointed out that well-documented images of oil spills are relatively rare. Indeed, the project could rely only on a few dozen positive examples while the negative examples were abundant. Such

imbalanced representation of the two classes is not without serious consequences for machine learning. This section will explain the cause of the difficulties, then proceed to some very simple ways of reducing their negative impact.

A Simple Experiment Suppose we have at our disposal a training set that is so small that it consists of only 50 positive examples and 50 negative examples. Let us subject this set to a fivefold crossvalidation[1]: we divide it into five equally sized parts; then, in five different experimental runs, we always remove one of the parts, induce a classifier from the union of the remaining four, and test the classifier on the removed part. In this manner, we eliminate, or at least reduce, the effect of randomness in the choice of the concrete training set. At the end, we write down the average results of the testing: classification accuracy on the positive examples, classification accuracy on the negative examples, and the geometric mean of the two classification accuracies.

Suppose now that we realize we have many more negative examples at our disposal than we originally thought. In order to find out how this newly discovered bounty is going affect the learning, we add to the previous training set another 50 negative examples (the positive examples remain the same), repeat the experimental procedure, and then write down the new result. We then continue in the same spirit, always adding to the training set another batch of 50 negative examples while keeping the same original 50 positive examples.

Observation If we plot the results of the above series of experiments, we will obtain a graph that, in all likelihood, will look very much like the one shown in Fig. 10.1 where the 1-NN classifier was used. The reader can see that as the number of the majority-class examples increases, the induced classifiers become biased toward this class, gradually converging to a situation where the classification accuracy on the negative examples (the majority class) approaches 100%, while the classification accuracy on the positive examples (the minority class) drops to well below 20%. The geometric mean of the two values keeps dropping, too.

The observation may appear somewhat counterintuitive. Surely the induced classifiers should become more powerful if more training examples are made available, even if these added examples all happen to belong to the same class? In turns out, however, that the unexpected behavior described above is typical of many machine-learning techniques. Engineers usually call it the problem of *imbalanced class representation*.

Majority-Class Undersampling (The Mechanical Approach) The experiment has convinced us that adding more examples from the majority class may cause degradation of the induced classifier's performance on the minority class. This may be a serious shortcoming. Thus in the oil-spill domain, the minority class represents the oil spills, the primary target of the machine-learning undertaking. In medical diagnosis, any disorder we want to recognize is typically a minority class, too. And

[1] An evaluation methodology introduced in Sect. 11.5.

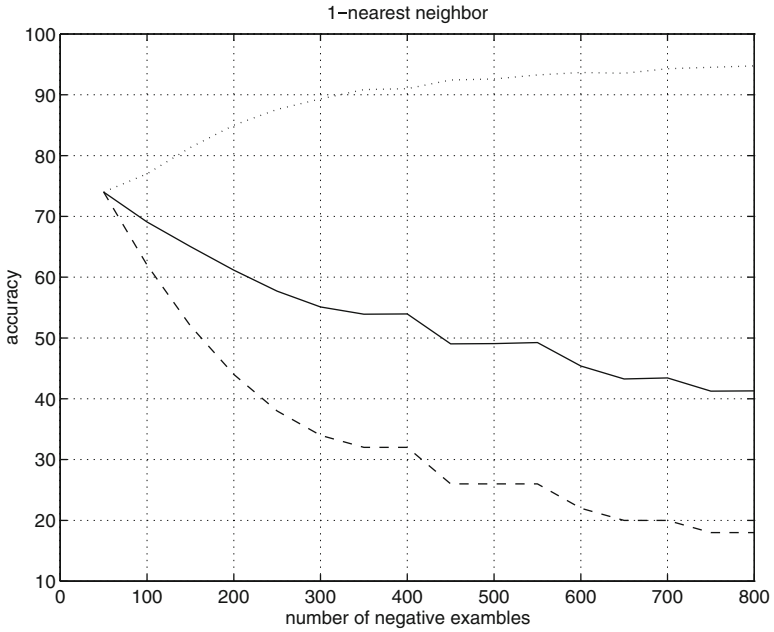

Fig. 10.1 *Dotted curve*: classification accuracy on the `neg` class; *dashed curve*: classification accuracy on the `pos` class; *solid curve*: geometric means of the two classification accuracies

the same applies to software whose task is to alert a company to misuse of its product (e.g., a wrongful use of calling cards, or credit-card fraud). In domains of this kind, it is the minority class that interests us. We now know that blindly adding more and more majority-class examples to the training set is likely to do more harm than good.

Suppose we are provided with a heavily imbalanced training set where, say, nine out of ten examples are negative. In this event, we will often benefit from the removal of many negative examples. In the simplest possible approach, this removal can be made at random: for instance, each negative example will face a 50% chance of being deleted from the training set. As we noticed above, the classifier induced from this reduced set is likely to outperform a classifier induced from the entire training set.

Identifying the Cause The mechanical solution indicated in the previous paragraph will hardly satisfy the thoughtful engineer who wants to understand *why* the data-removing trick worked—or, conversely, why increasing the number of majority-class examples may have such detrimental consequences.

Suppose the 1-NN classifier uses a training set where the vast majority of the examples are negative, and only a few are positive. Moreover, the data are known to suffer from a considerable amount of class-label noise. Limiting itself to an easy-to-visualize two-class domain, the left part of Fig. 10.2 shows one such training set.

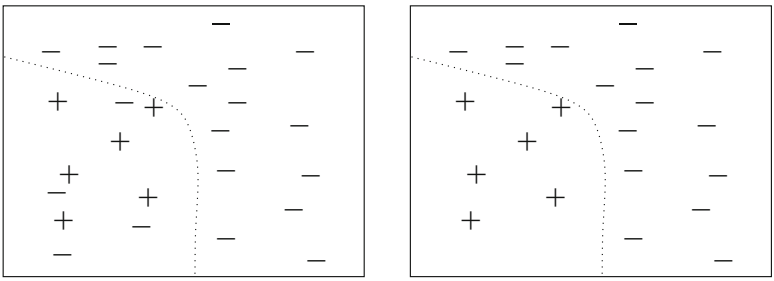

Fig. 10.2 In noisy domains where negative examples heavily outnumber positive examples, the removal of negative examples that participate in Tomek links may increase classification performance

The reader is sure to have noticed the point: in consequence of the noise, the nearest neighbor of almost every positive example is negative. In reality, these neighbors are probably positive, their negative labels being explained by errors made in the course of the creation of the training set. Be it as it may, the 1-NN classifier misclassifies these positive examples, and this is why there are so many false negatives, and only a few (if any) false positives.

Of course, not all machine-learning paradigms will suffer from this situation as dramatically as the 1-NN classifier. But most of them *will* suffer to some degree, and we now understand the reason why.

An Informed Solution: One-Sided Selection Knowing the source of our troubles, we are ready to suggest a remedy. To wit, the cause of our woes is the presence of many class-noisy examples in the positive region; the situation should therefore improve if we remove primarily *these* examples (rather than resorting to a random selection as in the mechanical approach suggested above).

In Chap. 3, we encountered a simple algorithm capable of identifying "suspicious" examples: the technique of Tomek links. The reader will recall that two examples, $(\mathbf{x}, \mathbf{y})$, are said to participate in a Tomek link if three conditions are satisfied: (1) each of the two examples has a different class label; (2) the nearest neighbor of $\mathbf{x}$ is $\mathbf{y}$; and (3) the nearest neighbor of $\mathbf{y}$ is $\mathbf{x}$. In the situation depicted in Fig. 10.2, many of the noisy examples on the left indeed do participate in Tomek links. This indicates that we may do improve the classifier's behavior if we delete from the training set the negative participants of each Tomek-link pair. The principle is known as *one-sided* selection because only one side of the Tomek link is selected for inclusion in the training set.

Applying the technique to the training set shown in the left part of Fig. 10.2, we will obtain something like the smaller training set shown in the right part. It is easy to see that the frequency of false negatives is now going to be lower. The efficiency of this methods is usually higher than just mechanical removal of randomly picked negative examples.

The Opposite Solution: Oversampling the Minority Class In some domains, however, the training set is so small that any further reduction of its size by undersampling is impractical. Even the majority-class examples are sparse, here; the deletion of any one of them may remove some critical aspect of the learning task, and thus jeopardize the performance of the induced classifier.

In this event, the opposite approach is sometimes preferred. Rather than removing majority-class examples, we *add* examples representing the minority class. Since we do not have at our disposal real examples from this class, we have to create them artificially. This can be done in two fundamental ways:

1. For each example from the minority class, create one copy and add this copy into the training set. Alternatively, two or more copies for each example can thus be created and added.
2. For each example from the minority class, create its slightly modified version and add it into the training set. The modification is made by minor (random) changes in continuous attributes; much less useful, though still possible, are changes in discrete attribute values.

The left part of Fig. 10.2 helps us understand why this works. To the neighborhood of some of the "afflicted" positive examples (those whose nearest neighbors have been turned negative by noise), minority-class oversampling inserts additional positive examples; as a result, the 1-NN classifier is no longer misled. The principle helps improve the behavior of other types of classifiers, too.

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 does the term *imbalanced training set* refer to? Explain the main reason why induction from imbalanced training sets so often leads to disappointing results.
- What is the essence of majority-class undersampling? Explain the mechanical approach, and then proceed to the motivation and principle of the *one-sided selection* that uses Tomek links.
- Explain the principle of minority-class oversampling. Describe and discuss the two alternative ways of creating new examples that are to be added to the training set.

10.3 Context-Dependent Domains

Up till now, we have tacitly assumed that the underlying "meaning" of a given class is fixed and immutable, that a single classifier, once induced, will under all circumstances exhibit the same (or at least similar) behavior. This, however, is not always the case.

Context-Dependent Classes Some classes change their essence with circumstances. If you think of that, this is the case of many concepts used in daily life. Thus the meaning of "fashionable dress" changes in time, and different cultures have a different idea of what they want to wear. "State-of-the-art technology" was something else a 100 years ago that it is today. Even the intended meaning of such notorious terms as "democracy" or "justice" depends on political background and historical circumstances. And if you want a more technical example, consider the problems encountered by speech-recognition software: everybody knows that the same word is often pronounced differently in England than in North America; but the software should "understand" speakers from both backgrounds.

Context-Dependent Features For the needs of this book, *context* is understood as a "a feature that has no bearing on the class if taken in isolation, but still affects the class when combined with other features."

For instance, suppose you want to induce a classifier capable of suggesting medical diagnosis, of recognizing X based on a set of symptoms. Some attributes, say, `gender`, do not have any predictive power; the patient being `male` is no proof of `prostate-cancer`; but the attribute value `gender=female` is a clear indication that the class is *not* `prostate-cancer`. This, of course, was an extreme sample. In other diagnoses, the impact of `gender` will be limited to influencing the critical values of certain laboratory tests, say, $p = 0.5$ being a critical threshold for male patients and $p = 0.7$ for female patients. Alternatively, the prior probabilities will be affected, `breast-cancer` being more typical of females, although men can suffer from it, too.

Induction in Context-Dependent Domains Suppose you want to induce a speech-recognition system, and you have a set of training examples coming both from British and American speakers. Suppose the attribute vector describing each example contains the "context" attribute, the speaker's origin. The other attributes capture the properties of the concrete digital signal. Each class label represents a different phoneme.

For the induction of a classifier that for each attribute vector decides which phoneme it represents, the engineer can essentially follow two different strategies. The first takes advantage of the contextual attribute, and divides the training examples into two subsets, one for British speakers and one for American speakers; then it induces a separate classifier from each of these training subsets. The second strategy mixes all examples in one big training set, and induces one "universal" classifier.

Practical experience indicates that, in applications of this kind, the first strategy performs better, provided that the real-time system in which the classifiers are embedded knows which of them to use. This decision can be assisted by an additional two-valued classifier that is trained to distinguish British speakers from American speakers.

Concept Drift Sometimes, the context changes in time. The "fashionable dress" example mentioned earlier belongs to this category, as do the political terms. In this event, machine-learning specialists talk about a so-called *concept drift*. What they have in mind is that, in the course of time, the essence or meaning of a class drifts from one context to another.

The drift has many aspects. One of them is the extent to which the context has changed the meaning of the class. In some rare domains, this change is so substantial that the induced classifier becomes virtually useless, and a new one has to be induced. Much more typical, however, is a less severe change that results only in a minor reduction of the classification performance. The old classifier can then still be used, perhaps after some fine-tuning.

Another feature worth consideration is the "speed" of the drift. At one extreme is an abrupt change. At a certain moment, one context is simply replaced, at it were, by another. More typically, however, the change is gradual in the sense that there is a certain "transition" period during which one context is, step by step, replaced by the other. In this event, the engineer may ask how fast the transition is, and whether (or when) the concept drift will necessitate special actions.

Induction of Time-Varying Classes Perhaps the simplest scenario in which concept drift is encountered is the one shown in Fig. 10.3. Here, a classifier is faced with a stream of examples that arrive one at a time, either in regular or irregular intervals. Each time an example arrives, the classifier labels it with a class. There may or may not be a feedback loop that tells the system (immediately or after some delay) whether the classification was correct, and if not, what the correct class label should have been.

If there is a reason to suspect the possibility of an occasional concept drift, it may be a good idea to take advantage of a sliding window such as the one shown in the picture. The classifier is then induced only from the examples "seen through the window." Each time a new example arrives, it is added to the window. Whenever deemed appropriate, older examples are removed, either one at a time, or in groups,

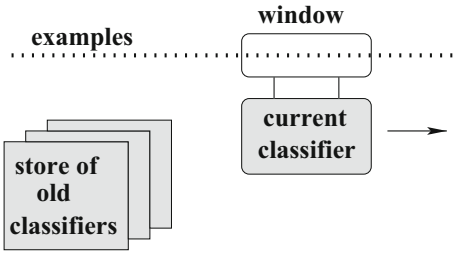

Fig. 10.3 A window passes over a stream of examples; "current classifier" is periodically updated to reflect changes in the underlying class. Occasionally, the system can retrieve some of the previous classifiers if the underlying context recurs

such as "the oldest 25% examples." The motivation for the deletion is simple: the engineer wants the window to contain only recent examples, suspecting that older ones may belong to an outdated context.

As already mentioned, the classifier is supposed to reflect only the examples contained in the window. In the simplest implementation, the classifier is re-induced each time the window contents change. Alternatively, the change in the window contents may only trigger a modification/adaptation of the classifier.

Figure 10.3 shows yet another aspect of this learning paradigm: sometimes, an older context may reappear (e.g., due to a certain "seasonality"). For this reason, it may be a good idea to store previously induced versions of the classifier, just in case they might be re-used in the future.

Engineering Issues in the Sliding-Window Approach There are certain essential issues that the engineer wishing to implement the sliding-window approach needs to consider. The first is the question of the size of the window. If it is too small, then the examples it contains may not be sufficient for successful learning. If it is too big, then it may contain examples that come from outdated contexts. Ideally, then, the window should grow (no old examples deleted) as long as it can be assumed that the context has not changed. When a change is detected, a certain number of the oldest examples should be deleted because they are no longer trusted.

This leads us to the next important question: how to recognize that a context has changed? One simple solution relies on the feedback about the classifier's behavior. The change of context is then identified by a sudden drop in the classification performance.

Finally, there is the question of how many of the oldest examples to delete from the window. The answer will depend on how gradual the context change is, and also on the extent of this change. At one extreme, an abrupt and considerable change will call for the deletion of *all* examples. At the other extreme, a very slow transition between two very similar contexts will necessitate the deletion of only a few of the oldest examples.

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.

- Give examples of domains where the meaning of a given class varies in time and/or geographical location. Give examples of domains where previous meanings recur in time.
- Describe the basic scenario that is based on a stream of time-ordered examples. Explain the principle of the sliding-window approach to induction in time-varying domains.
- Discuss briefly the basic engineering issues encountered in the sliding-window approach.

10.4 Unknown Attribute Values

In many domains, certain attribute values are not known. A patient refused to give his age, a measurement device failed, and some information got lost—or is unavailable for any other reason. As a result, we get an imperfect training set such as the one shown in Table 10.1 where some of the values are replaced with question marks. In some domains, the question marks represent a considerable portion of all attribute-value fields, and this may complicate the learning task. The engineer needs to understand what kind of damage the unknown values may cause, and what solutions exist.

Adverse Effects In the case of the plain version of the k-NN classifier, the distance between two vectors can only be calculated if all values in the vectors are known. True, the distance metric can be modified so that it quantifies also the distance between, say, `red` and `unknown`; but distances calculated in this manner tend to be rather ad hoc.

The situation is not any better in the case of linear and polynomial classifiers. Without the knowledge of all attribute values, it is impossible to calculate the weighted sum, $\Sigma w_i x_i$, whose sign tells the classifier which class label to choose. Likewise, unknown attribute values complicate the use of Bayesian classifiers and neural networks.

Decision trees are more flexible, in this sense. When classifying an example, it is quite possible that the attribute whose value is unknown will not have to be tested (will not find itself on the path from the root node to the terminal node).

Trivial Approaches to Filling-In Missing Values In a domain with a sufficiently large training set that contains only a few question marks, there is usually no harm in removing all examples that have unknown attribute values. This, however, will become impractical in domains where the number of question marks is so high

Table 10.1 Training examples with missing attribute values

		Crust		Filling			
Example	Shape	Size	Shade	Size	Shade	Weight	Class
ex1	Circle	Thick	Gray	Thick	Dark	7	`pos`
ex2	Circle	Thick	White	Thick	Dark	2	`pos`
ex3	Triangle	Thick	Dark	Thick	Gray	2	`pos`
ex4	Circle	Thin	White	?	Dark	3	`pos`
ex5	Square	Thick	Dark	?	White	4	`pos`
ex6	Circle	Thick	White	Thin	Dark	?	`pos`
ex7	Circle	Thick	Gray	Thick	White	6	`neg`
ex8	Square	Thick	?	Thick	Gray	5	`neg`
ex9	Triangle	Thin	Gray	Thin	Dark	5	`neg`
ex10	Circle	Thick	Dark	Thick	?	?	`neg`
ex11	Square	Thick	White	Thick	Dark	9	`neg`
ex12	Triangle	Thick	White	Thick	Gray	8	`neg`

that the removal of all affected examples would destroy most of the training set, disposing in the process of valuable information.

In this event, we may try to replace the question marks with *some* values, even though these may be incorrect. This is easily done. When the attribute is discrete, then we may simply replace the question mark with the attribute's most frequent value. Thus in Table 10.1, example $\texttt{ex}_8$, the unknown value of `crust-shade` will be replaced with `white` because this is the most frequent value of this attribute in this particular training set. In the case of a continuous-valued attribute, the average value can be used. In $\texttt{ex}_6$ and $\texttt{ex}_{10}$, the value of `weight` is unknown. Among the 10 examples where it *is* known, the average value is `weight=5.1`, and this is the value we will use in $\texttt{ex}_6$ and $\texttt{ex}_{10}$.

When doing so, caution is called for. The reader has to keep in mind that using the most frequent or average values will render the examples' description unreliable, perhaps even dubious. The technique should therefore be used sparingly. When many values are missing, more sophisticated methods (such as the one below) should be used.

Learning to Fill-in Missing Values Sometimes, using the most common or average values can mislead the learning program. A better idea of how to fill the empty slots is built around the observation that attributes are rarely independent from each other. For instance, the taller the man, the greater his body weight. If the `weight` of someone with `height=6.5` is unknown, it would be foolish to use the average weight calculated over the whole population; after all, our rather tall individual is certainly heavier than the average person. Seeking a way out, we will probably do better calculating the average weight among those with `height` > 6.

So much for a pair of mutually dependent attributes. Quite often, however, the interrelations are more complicated than that, easily involving three or more attributes. One simple mechanism to predict unknown values in situations of this kind will rely on the idea of decision-tree induction. A pseudocode of the technique is provided in Table 10.2.

The idea is quite simple. Suppose that *at* is an attribute that has, in the training set, many question marks. We want to replace the question marks with the most likely values. We decide to so by means of a decision tree. To this end, we convert

Table 10.2 An algorithm to determine unknown attribute values

Let T be the original training set.

Let *at* be the attribute with unknown values.

1. Create a new training set, T', in which *at* becomes the class label; the examples are described by all the remaining attributes, the former class label (e.g., `pos` versus `neg`) being treated like just another attribute.
2. Remove from T' all examples in which the value of *at* is unknown.
3. From this final version of T', induce a decision tree.
4. Use the decision tree thus induced to determine the values of *at* in those examples in which its values were unknown.

the original training set, T, into a new training set, T', where the original class label (e.g., `pos` or `neg`) becomes one of the attributes, whereas *at* will be treated as a class label. From this training set, we remove all examples whose values of *at* are unknown. From the rest, we induce the decision tree, and then use the decision tree to fill in the missing values.

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 are the main difficulties posed by unknown attribute values? What are their typical consequences for classifier induction?
- Describe the trivial ways of dealing with examples with some unknown attribute values. Discuss their limitations.
- Explain how the idea of decision-tree induction can be used when we want to determine the unknown attribute values.

10.5 Attribute Selection

In many domains, the training examples are described by great many attributes: tens of thousands, or even more. Learning from data sources of this kind can be prohibitively expensive. Besides, we have to face problems with learnability, and also with issues related to irrelevant or redundant attributes. To avoid unnecessary disappointments, the engineer needs to be acquainted with methods to select the most appropriate attributes.

Irrelevant and Redundant Attributes Not all attributes are created equal. Some of them are irrelevant in the sense that their values do not have any effect on an example's class. Others are redundant in the sense that their values can be obtained from values of other attributes: for instance, `age` can be obtained from the value of `date-of-birth`. Attributes of these kinds can mislead certain induction techniques. For instance, irrelevant attributes (and, to a lesser degree, also redundant attributes) distort the vector-to-vector distances calculated by the k-NN classifier. Other paradigms, such as decision trees, are less vulnerable, but even they may suffer from excessive computational costs.

Extremely Long Attribute Vectors Some domains, such as automated text categorization, are marked by tens of thousands of attributes, and this often causes problems. One of the difficulties is the reduced learnability: the induced classifiers are prone to overfit the training data, disappointing the user when tested on future examples. Also the computational costs can be impractical, especially when a

multilayer neural network is used: each additional attribute increases the number of weights to be trained, thus adding to the calculations.

Moreover, examples described by thousands of attributes are inevitably sparse, which is known to mislead many machine-learning approaches. For instance, the problem of sparsity in k-NN classifiers was explained in Chap. 3.

And yet it is known that, for all intents and purposes, most of the attributes are useless, and as such, should be disposed of.

Filter Approaches to Attribute Selection Perhaps the simplest approach to attribute selection is based on what machine learning calls *filtering*. The idea is to calculate for each attribute its "utility" for the classification task at hand, and then order them according to this criterion. The intention to select the top N percent. The choice of the "cut-off" point, N, is usually made by trial and error.

When ordering the attributes, the *information gain* from Sect. 6.3 can be used if the attributes are discrete. Thanks to the existence of mechanisms for binarization (see Sect. 6.4), information gain can actually be employed even in the case of continuous-valued attributes; for this, however, statistical approaches to correlation measurement are usually preferred.

One reason to criticize attribute filtering is that this approach ignores the relations that exist among the attributes. This makes it difficult, almost impossible, to identify redundant attributes. As we know, a redundant attribute does not bring any additional information beyond that provided by the other attributes; and yet its information gain can be high.

There is a relatively simple way to overcome this weakness. We induce a decision tree, and then use only those attributes that are encountered in the tests in the tree's internal nodes. The careful reader will recall that this approach was used in some of the simple applications discussed in Chap. 8.

Wrapper Approaches to Attribute Selection More powerful, but also more computationally expensive, is the so-called *wrapper* approach to attribute selection. Here is the underlying principle. Suppose we want to compare the quality of two attribute sets, A_1 and A_2. From the original training set, T, we create two training sets, T_1 and T_2. In both, all examples have the same class labels as in T. However, T_1 describes the examples by A_1 and T_2 uses A_2. From the two newly created training subsets, two classifiers are induced and evaluated on some independent evaluation set, T_E. The attribute set that results in the higher performance is better.

This is the principle used in the search-based algorithm whose pseudocode is provided in Table 10.3. The input consists of a training set, T, and of a set of attributes, A. The output is a subset, $S \in A$, of the most useful attributes. At the beginning, S is empty. At each step, the approach chooses the best attribute from A, and adds it to S. What is "best" is determined by the classification performance (on an independent testing set) of the classifier induced from the examples described by the attributes from S. The algorithm is terminated if no addition to S leads to an improvement of the classification performance—or if there are no more attributes to be added to S.

Table 10.3 An wrapper approach to sequential attribute selection

Divide the available set of pre-classified examples into two parts, T_T and T_E. Let A be the set of attributes. Create an empty set, S.

1. For every attribute, $at_i \in A$:
   (i) add at_i to S; let all examples in T_T and T_E be described by attributes from S;
   (ii) induce a classifier from T_T, then evaluate its performance on T_E; denote this performance by p_i;
   (iii) remove at_i from S.
2. Identify the attribute that resulted in the highest value of p_i. Remove this attribute from A and add it to S.
3. If $A = \emptyset$, stop; if the latest addition did not increase performance, remove this last attribute from S and stop, too. In both cases, S is the final set of attributes.
4. Return to step 1.

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.

- In what sense do we say that some attributes are less useful than others? Why is the engineer often incapable of choosing the right attributes for example description?
- Explain and discuss the principle of filter-based approaches to attribute selection.
- Describe the principle of the *wrapper* approaches to attribute selection. Explain how the principle can be employed in a simple search-based approach to attribute selection.

10.6 Miscellaneous

Some issues worth knowing about do not merit a separate section in an introductory text, and yet they cannot be ignored. Let us briefly summarize them here.

Lack of Regularity in the Data Suppose you are asked to induce a classifier from training examples that have been created by a random-number generator; all attribute values are random, and so are the class labels. Obviously, there is no regularity in such data—and yet—machine learning techniques are often capable of inducing from them a classifier with zero error rate on the training set. Of course, this perfect behavior on the training set will not translate into similar behavior on future examples.

This observation suggests a simple mechanism to be used when measuring the amount of regularity in data. The idea is simply to divide the data in two subsets—one for training and one for testing. The classifier is induced from the training set, and then applied to the testing set. In the case of random data, we will observe only a small (if any) error rate on the training examples but the results on the testing examples will be dismal. Conversely, the more regularity there is, in the data, the better the results on the testing set.

Classes That Can Be Linearly Ordered In some domains, each example is labeled with one out of several (perhaps even many) classes. In this context, we have to mention the special case where the different class labels can be ordered. For instance, suppose that the output class is `month`, with values `january` through `december`. In domains of this kind, it would be a great (and misleading) simplification to assume that misclassifying `june` for `may` is the same as misclassifying `june` for `december`.

Not only in performance evaluation, but also during the induction of such classes, some attention should therefore be devoted to the ordering of the individual classes. One possibility is to begin by grouping neighboring class labels; only after inducing a classifier for each group should we consider the possibility of fine-tuning within the group.

In the case of class `month`, we may perhaps first induce classifier for `seasons`, each comprising 3 months, and only after this, a separate classifier for each month.

Regression In some applications, the expected output is not a discrete-valued class, but rather a number from a continuous range. For instance, this can be the case when the software is to predict a value of a stock-market index. Problems of this kind are called *regression*. In this book, we do not address them. The simplest way of dealing with them within the framework of machine learning is to replace the continuum with subintervals, and then treat each subinterval as a separate class. Note that the task would then belong to the category of "classes that can be linearly ordered" mentioned in the previous paragraph.

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.

- Explain how to use machine learning when measuring the amount of regularity in data. Give examples of domains where this regularity can be expected to be low.
- What are *multi-label* domains? What is the simplest approach to induction in these domains? What typical problems does the machine-learning engineer encounter in them?

10.7 Summary and Historical Remarks

- Chapter 7 offered mathematical arguments supporting the claim that "there is no learning without bias." Certain practical considerations have convinced us that this is indeed the case.
- Sometimes, the meaning of the underlying class depends on a concrete context; this context can change in time, in which case we are facing the problem of time-varying classes.
- The classical machine-learning techniques from the earlier chapters of this book assume that both (or all) classes are adequately represented in the training set. Quite often, however, this requirement is not satisfied, and the engineer has to deal with the difficulties caused by the problem of *imbalanced training sets*.
- The most typical approaches to the problem of *imbalanced training sets* are majority-class undersampling and minority-class oversampling.
- In many training sets, some attribute values are unknown, and this complicates the use of certain induction techniques. One possible solution is to use (in place of the unknown values) the most frequent or the average values of the given attributes.
- Quite often, the engineer is faced with the necessity to select the most appropriate set of attributes. Two fundamental approaches can be used here: the *filtering* techniques and the *wrapper* techniques.
- In domains with more than two classes, it sometimes happens that the individual classes can be ordered. This circumstance can affect performance evaluation. For instance, if the task is to recognize a concrete month, then it is not the same thing if the classifier's output missed the target by 1 month or by 5. Even the learning procedure should then perhaps be modified accordingly.
- Sometimes, the output is not a discrete-valued class, but rather a value from a continuous range. This type of problem is called *regression*. This book does not address regression explicitly.

Historical Remarks The idea to distinguish different biases in machine learning was pioneered by Gordon and desJardin [34]. The principle of lifelong learning was first mentioned by Thrun and Mitchell [88]. The early influential papers addressing the issue of context were published by Turney [90] and Katz et al. [40]. Induction of time-varying concepts was introduced by Kubat [49] and some early algorithms were described by Widmer and Kubat [97]. The oldest paper on multi-label classification known to the author of this book was published by McCallum [57]. The Wrapper approach to attribute selection is introduced by Kohavi [44].

10.8 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.

Table 10.4 A simple exercise in "unknown values"

Example	Shape	Crust		Filling		Class
		Size	Shade	Size	Shade	
ex_1	Circle	Thick	Gray	Thick	Dark	pos
ex_2	Circle	Thick	White	Thick	Dark	pos
ex_3	Triangle	Thick	Dark	?	Gray	pos
ex_4	Circle	Thin	White	Thin	?	pos
ex_5	Square	Thick	Dark	Thin	White	pos
ex_6	Circle	Thick	White	Thin	Dark	pos
ex_7	Circle	Thick	Gray	Thick	White	neg
ex_8	Square	Thick	White	Thick	Gray	neg
ex_9	Triangle	Thin	Gray	Thin	Dark	neg
ex_{10}	Circle	Thick	Dark	Thick	White	neg
ex_{11}	Square	Thick	White	Thick	Dark	neg
ex_{12}	Triangle	?	White	Thick	Gray	neg

Exercises

1. Consider the training set shown in Table 10.4. How will you replace the missing values (question marks) with the most frequent values? How will you use a decision tree to this end?
2. Once you have replaced the question marks in Table 10.4 with concrete values, identify the two attributes that offer the highest information gain.

Give It Some Thought

1. The text emphasized the difference between two basic types of error: those caused by the wrong bias(representational or procedural), and those that are due to variance in the training data. Suggest an experimental procedure that would give the engineer an idea about how much of the overall error rate can in a given domain be explained by either of these two sources.
2. Boosting algorithms are known to be relatively robust with respect to variance-based errors. Explain why this is the case. Further on, *non-homogeneous boosting* presented in Sect. 9.4 is known to reduce bias-based errors. Again, offer some explanation.
3. Suppose that a Bayesian classifier is to be employed in an imbalanced two-class domain where examples from one class heavily outnumber examples from the other class. Will this classifier be as sensitive to this situation as the nearest-neighbor approach? Support your answer by concrete arguments. Suggest an experimental verification.

4. In this section, the problem of imbalanced training sets was explored only within the framework of two-class domains where each example is either positive or negative. How does the same problem generalize to domains that have more than two classes? Suggest some concrete situations where imbalanced classes in such multi-class domains are or are not a problem.
5. Consider the case of linearly ordered classes mentioned in Sect. 10.6. Using the hint provided in the text, suggest a machine-learning scenario addressing this issue.

Computer Assignments

1. Write a computer program that accepts as input a training set where many attribute-values are missing, and outputs an improved training set where the missing values of discrete attributes have been replaced with the most frequent values, and the missing values of continuous attributes with the average values. Implement a computer program that will experimentally ascertain whether the missing-values replacement helps or harms the performance of a decision tree induced from such data.
2. Choose some public-domain data, for instance from the UCI repository.[2] Make sure this domain has at least one binary attribute. The exercise suggested here will assume that this binary attribute represents a context. Divide the training data into two subsets, each for a different context (a different value of the binary attribute). Then induce from each subset the corresponding context-dependent classifier. Assuming that it is at each time clear which of the two classifiers to use, how much will the average performance of these two classifiers be better than that of a "universal" classifier that has been induced from the original training set?

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