Praktika5 Attribute Selection

7.1 ATTRIBUTE SELECTION

Most machine learning algorithms are designed to learn which are the most appropriate attributes to use for making their decisions. For example, decision tree methods choose the most promising attribute to split on at each point and should—in theory—never select irrelevant or unhelpful attributes. Having more features should surely—in theory—result in more discriminating power, never less. “What’s the difference between theory and practice?” an old question asks. The answer goes, “There is no difference between theory and practice—in theory. But in practice, there is.” Here there is too: In practice, adding irrelevant or distracting attributes to a dataset often confuses machine learning systems.

Experiments with a decision tree learner (C4.5) have shown that adding to standard datasets a random binary attribute generated by tossing an unbiased coin impacts classification performance, causing it to deteriorate (typically by 5 to 10% in the situations tested). This happens because at some point in the trees that are learned, the irrelevant attribute is invariably chosen to branch on, causing random errors when test data is processed. How can this be when decision tree learners are cleverly designed to choose the best attribute for splitting at each node? The reason is subtle. As you proceed further down the tree, less and less data is available to help make the selection decision. At some point, with little data, the random attribute will look good just by chance. Because the number of nodes at each level increases exponentially with depth, the chance of the rogue attribute looking good somewhere along the frontier multiplies up as the tree deepens. The real problem is that you inevitably reach depths at which only a small amount of data is available for attribute selection. If the dataset were bigger it wouldn’t necessarily help—you’d probably just go deeper.

Divide-and-conquer tree learners and separate-and-conquer rule learners both suffer from this effect because they inexorably reduce the amount of data on which they base judgments. Instance-based learners are very susceptible to irrelevant attributes because they always work in local neighborhoods, taking just a few training instances into account for each decision. Indeed, it has been shown that the number of training instances needed to produce a predetermined level of performance for instance-based learning increases exponentially with the number of irrelevant attributes present. Naïve Bayes, by contrast, does not fragment the instance space and robustly ignores irrelevant attributes. It assumes by design that all attributes are conditionally independent of one another, an assumption that is just right for random “distracter” attributes. But through this very same assumption, Naïve Bayes pays a heavy price in other ways because its operation is damaged by adding redundant attributes.

The fact that irrelevant distracters degrade the performance of state-of-the-art decision tree and rule learners is, at first, surprising. Even more surprising is that relevant attributes can also be harmful. For example, suppose that in a two-class dataset a new attribute was added that had the same value as the class to be predicted most of the time (65%) and the opposite value the rest of the time, randomly distributed among the instances. Experiments with standard datasets have shown that this can cause classification accuracy to deteriorate (by 1 to 5% in the situations tested). The problem is that the new attribute is (naturally) chosen for splitting high up in the tree. This has the effect of fragmenting the set of instances available at the nodes below so that other choices are based on sparser data.

Because of the negative effect of irrelevant attributes on most machine learning schemes, it is common to precede learning with an attribute selection stage that strives to eliminate all but the most relevant attributes. The best way to select relevant attributes is manually, based on a deep understanding of the learning problem and what the attributes actually mean. However, automatic methods can also be useful. Reducing the dimensionality of the data by deleting unsuitable attributes improves the performance of learning algorithms. It also speeds them up, although this may be outweighed by the computation involved in attribute selection. More important, dimensionality reduction yields a more compact, more easily interpretable representation of the target concept, focusing the user's attention on the most relevant variables.

Scheme-Independent Selection
When selecting a good attribute subset, there are two fundamentally different approaches. One is to make an independent assessment based on general characteristics of the data; the other is to evaluate the subset using the machine learning algorithm that will ultimately be employed for learning. The first is called the filter method because the attribute set is filtered to produce the most promising subset before learning commences. The second is called the wrapper method because the learning algorithm is wrapped into the selection procedure. Making an independent assessment of an attribute subset would be easy if there were a good way of determining when an attribute was relevant to choosing the class. However, there is no universally accepted measure of relevance, although several different ones have been proposed.

One simple scheme-independent method of attribute selection is to use just enough attributes to divide up the instance space in a way that separates all the training instances. For example, if just one or two attributes are used, there will generally be several instances that have the same combination of attribute values. At the other extreme, the full set of attributes will likely distinguish the instances uniquely so that no two instances have the same values for all attributes. (This will not necessarily be the case, however; datasets sometimes contain instances with the same attribute values but different classes.) It makes intuitive sense to select the smallest attribute subset that serves to distinguish all instances uniquely. This can easily be found using an exhaustive search, although at considerable computational expense. Unfortunately, this strong bias toward consistency of the attribute set on the training data is statistically unwarranted and can lead to overfitting—the algorithm may go to unnecessary lengths to repair an inconsistency that was in fact merely caused by noise.
Machine learning algorithms can be used for attribute selection. For instance, you might first apply a decision tree algorithm to the full dataset and then select only those attributes that are actually used in the tree. While this selection would have no effect at all if the second stage merely built another tree, it will have an effect on a different

learning algorithm. For example, the nearest-neighbor algorithm is notoriously susceptible to irrelevant attributes, and its performance can be improved by using a decision tree builder as a filter for attribute selection first. The resulting nearest-neighbor scheme can also perform better than the decision tree algorithm used for filtering.

As another example, the simple 1R scheme described in Chapter 4 has been used to select the attributes for a decision tree learner by evaluating the effect of branching on different attributes (although an error-based method such as 1R may not be the optimal choice for ranking attributes, as we will see later when covering the related problem of supervised discretization). Often the decision tree performs just as well when only the two or three top attributes are used for its construction—and it is much easier to understand. In this approach, the user determines how many attributes to use for building the decision tree.

Another possibility is to use an algorithm that builds a linear model—for example, a linear support vector machine—and ranks the attributes based on the size of the coefficients. A more sophisticated variant applies the learning algorithm repeatedly. It builds a model, ranks the attributes based on the coefficients, removes the lowest-ranked one, and repeats the process until all attributes have been removed. This method of recursive feature elimination has been found to yield better results on certain datasets (e.g., when identifying important genes for cancer classification) than simply ranking attributes based on a single model. With both methods it is important to ensure that the attributes are measured on the same scale; otherwise, the coefficients are not comparable. Note that these techniques just produce a ranking; another method must be used to determine the appropriate number of attributes to use.

Attributes can be selected using instance-based learning methods too. You could sample instances randomly from the training set and check neighboring records of the same and different classes—“near hits” and “near misses.” If a near hit has a different value for a certain attribute, that attribute appears to be irrelevant and its weight should be decreased. On the other hand, if a near miss has a different value, the attribute appears to be relevant and its weight should be increased. Of course, this is the standard kind of procedure used for attribute weighting for instance-based learning, described in Section 6.5. After repeating this operation many times, selection takes place: Only attributes with positive weights are chosen. As in the standard incremental formulation of instance-based learning, different results will be obtained each time the process is repeated, because of the different ordering of examples. This can be avoided by using all training instances and taking into account all near hits and near misses of each.
A more serious disadvantage is that the method will not detect an attribute that is redundant because it is correlated with another attribute. In the extreme case, two identical attributes would be treated in the same way, either both selected or both rejected. A modification has been suggested that appears to go some way toward addressing this issue by taking the current attribute weights into account when computing the nearest hits and misses.
Another way of eliminating redundant attributes as well as irrelevant ones is to select a subset of attributes that individually correlate well with the class but have little intercorrelation. The correlation between two nominal attributes A and B can be measured using the symmetric uncertainty, where H is the entropy function described in Section 4.3.

The entropies are based on the probability associated with each attribute value; H(A, B), the joint entropy of A and B, iscalculated from the joint probabilities of all combinations of values of A and B. The symmetric uncertainty always lies between 0 and 1. Correlation-based featureselection determines the goodness of a set of attributes using where C is the class attribute and the indices i and j range over all attributes in the set. If all m attributes in the subset correlate perfectly with the class and with oneanother, the numerator becomes m and the denominator m 2 , which is also m. Thus, the measure is 1, which turns out to be the maximum value it can attain (the minimum is 0). Clearly, this is not ideal, because we want to avoid redundant attributes. However, any subset of this set will also have value 1. When using this criterion to search for a good subset of attributes, it makes sense to break ties in favor of the smallest subset.

$$U(A,B) = 2 * ((H(A) + H(B) - H(A,B))/(H(A) + H(B)))$$

$$U(A, B) = 2\frac{H(A) + H(B) - H(A, B)}{H(A) + H(B)}$$

$$\sum_j U(A_j, C) \Big/ \sqrt{\sum_i \sum_j U(A_i, A_j)}$$

Searching the Attribute Space
Most methods for attribute selection involve searching the space of attributes for the subset that is most likely to predict the class best. Figure 7.1 illustrates the attribute space for the—by now all-too-familiar—weather dataset. The number of possible attribute subsets increases exponentially with the number of attributes, making an exhaustive search impractical on all but the simplest problems.
Typically, the space is searched greedily in one of two directions: top to bottom and bottom to top in the figure. At each stage, a local change is made to the current attribute subset by either adding or deleting a single attribute. The downward direc-tion, where you start with no attributes and add them one at a time, is called forward selection. The upward one, where you start with the full set and delete attributes one at a time, is backward elimination.
In forward selection, each attribute that is not already in the current subset is tentatively added to it, and the resulting set of attributes is evaluated—using, for example, cross-validation, as described in the following section. This evaluation produces a numeric measure of the expected performance of the subset. The effect of adding each attribute in turn is quantified by this measure, the best one is chosen, and the procedure continues. However, if no attribute produces an improvement when added to the current subset, the search ends. This is a standard greedy search procedure and guarantees to find a locally—but not necessarily globally—optimal set of attributes.

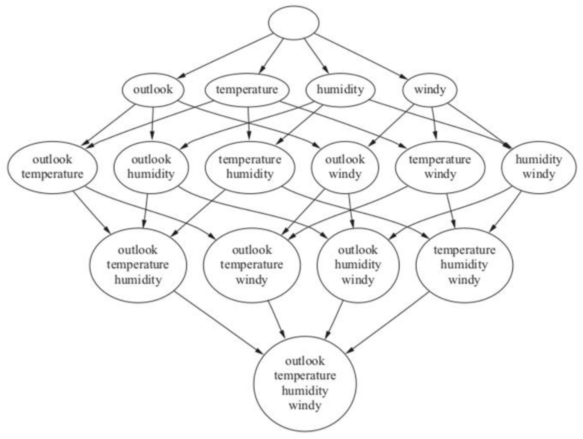

FIGURE 7.1
Attribute space for the weather dataset.

Backward elimination operates in an entirely analogous fashion. In both cases a slight bias is often introduced toward smaller attribute sets. This can be done for forward selection by insisting that if the search is to continue, the evaluation measure must not only increase, but must increase by at least a small predetermined quantity. A similar modification works for backward elimination.

More sophisticated search schemes exist. Forward selection and backward elimination can be combined into a bidirectional search; again, one can begin either with all the attributes or with none of them. Best-first search is a method that does not just terminate when the performance starts to drop but keeps a list of all attribute subsets evaluated so far, sorted in order of the performance measure, so that it can revisit an earlier configuration instead. Given enough time it will explore the entire space, unless this is prevented by some kind of stopping criterion. Beam search is similar but truncates its list of attribute subsets at each stage so that it only contains a fixed number—the beam width—of most promising candidates. Genetic algorithm search procedures are loosely based on the principle of natural selection: They "evolve" good feature subsets by using random perturbations of a current list of candidate subsets and combining them based on performance.

Scheme-Specific Selection

The performance of an attribute subset with scheme-specific selection is measured in terms of the learning scheme's classification performance using just those attributes. Given a subset of attributes, accuracy is estimated using the normal procedure of cross-validation described in Section 5.3. Of course, other evaluation methods such as performance on a holdout set (Section 5.3) or the bootstrap estimator (Section 5.4) could be equally well used.

The entire attribute selection process is rather computation intensive. If each evaluation involves a tenfold cross-validation, the learning procedure must be executed 10 times. With m attributes, the heuristic forward selection or backward elimination multiplies evaluation time by a factor proportional to m 2 in the worst case. For more sophisticated searches, the penalty will be far greater, up to 2 m for an exhaustive algorithm that examines each of the 2 m possible subsets.

Good results have been demonstrated on many datasets. In general terms, backward elimination produces larger attribute sets than forward selection but better classification accuracy in some cases. The reason is that the performance measure is only an estimate, and a single optimistic estimate will cause both of these search procedures to halt prematurely—backward elimination with too many attributes and forward selection with not enough. But forward selection is useful if the focus is on understanding the decision structures involved, because it often reduces the number of attributes with only a small effect on classification accuracy. Experience seems to show that more sophisticated search techniques are not generally justified, although they can produce much better results in certain cases.

One way to accelerate the search process is to stop evaluating a subset of attributes as soon as it becomes apparent that it is unlikely to lead to higher accuracy than another candidate subset. This is a job for a paired statistical significance test, performed between the classifier based on this subset and all the other candidate classifiers based on other subsets. The performance difference between two classifiers on a particular test instance can be −1, 0, or 1 depending on, respectively, whether the first classifier is worse than, the same as, or better than the second on that instance. A paired t-test (described in Section 5.5) can be applied to these figures over the entire test set, effectively treating the results for each instance as an independent estimate of the difference in performance.

Then the cross-validation for a classifier can be prematurely terminated as soon as it turns out to be significantly worse than another, which, of course, may never happen. We might want to discard classifiers more aggressively by modifying the t-test to compute the probability that one classifier is better than another classifier by at least a small user-specified threshold. If this probability becomes very small, we can discard the former classifier on the basis that it is very unlikely to perform substantially better than the latter. This methodology is called race search and can be implemented with different underlying search strategies. When it is used with forward selection, we race all possible single-attribute additions simultaneously and drop those that do not perform well enough. In backward elimination, we race all single-attribute deletions. Schemata search is a more complicated method specifically designed for racing; it runs an iterative series of races that each determine whether or not a particular attribute should be included. The other attributes for this race are included or excluded randomly at each point in the evaluation. As soon as one race has a clear winner, the next iteration of races begins, using the winner as the starting point. Another search strategy is to rank the attributes first using, for example, their information gain (assuming they are discrete), and then race the ranking. In this case the race includes no attributes, the top-ranked attribute, the top two attributes, the top three, and so on.

A simple method for accelerating a scheme-specific search is to preselect a given number of attributes by ranking them first using a criterion like the information gain and discarding the rest before applying scheme-specific selection. This has been found to work surprisingly well on high-dimensional datasets such as gene expression and text categorization data, where only a couple of hundred of attributes are used instead of several thousands. In the case of forward selection, a slightly more sophisticated variant is to restrict the number of attributes available for expanding the current attribute subset to a fixed-sized subset chosen from the ranked list of attributes—creating a sliding window of attribute choices—rather than making all (unused) attributes available for consideration in each step of the search process. Whatever way you do it, scheme-specific attribute selection by no means yields a uniform improvement in performance. Because of the complexity of the process, which is greatly increased by the feedback effect of including a target machine learning algorithm in the attribution selection loop, it is quite hard to predict the conditions under which it will turn out to be worthwhile. As in many machine learning situations, trial and error using your own particular source of data is the final arbiter.

There is one type of classifier for which scheme-specific attribute selection is an essential part of the learning process: the decision table. As mentioned in Section 3.1, the entire problem of learning decision tables consists of selecting the right attributes to be included. Usually this is done by measuring the table's cross-validation performance for different subsets of attributes and choosing the best-performing subset. Fortunately, leave-one-out cross-validation is very cheap for this kind of classifier. Obtaining the cross-validation error from a decision table derived from the training data is just a matter of manipulating the class counts associated with each of the table's entries, because the table's structure doesn't change when instances are added or deleted. The attribute space is generally searched by best-first search because this strategy is less likely to get stuck in a local maximum than others, such as forward selection.

Let's end our discussion with a success story. Naïve Bayes is a learning method for which a simple scheme-specific attribute selection approach has shown good results. Although this method deals well with random attributes, it has the potential to be misled when there are dependencies among attributes, and particularly when redundant ones are added. However, good results have been reported using the forward selection algorithm—which is better able to detect when a redundant attribute is about to be added than the backward elimination approach—in conjunction with a very simple, almost "naïve," metric that determines the quality of an attribute subset to be simply the performance of the learned algorithm on the training set. As was emphasized in Chapter 5, training set performance is certainly not a reliable indicator of test set performance. Nevertheless, experiments show that this simple modification to Naïve Bayes markedly improves its performance on those standard datasets for which it does not do so well as tree- or rule-based classifiers, and does not have any negative effect on results on datasets on which Naïve Bayes already does well. Selective Naïve Bayes, as this learning method is called, is a viable machine learning technique that performs reliably and well in practice.