6.9 NAIVE BAYES CLASSIFIER

One highly practical Bayesian learning method is the naive Bayes learner, often called the *naive Bayes classifier*. In some domains its performance has been shown to be comparable to that of neural network and decision tree learning. This section introduces the naive Bayes classifier; the next section applies it to the practical problem of learning to classify natural language text documents.

The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function $f(x)$ can take on any value from some finite set V. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values $\langle a_1, a_2 \ldots a_n \rangle$. The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value, v_{MAP}, given the attribute values $\langle a_1, a_2 \ldots a_n \rangle$ that describe the instance.

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n)$$

We can use Bayes theorem to rewrite this expression as

$$\begin{aligned} v_{MAP} &= \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)} \\ &= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2 \ldots a_n | v_j) P(v_j) \end{aligned} \tag{6.19}$$

Now we could attempt to estimate the two terms in Equation (6.19) based on the training data. It is easy to estimate each of the $P(v_j)$ simply by counting the frequency with which each target value v_j occurs in the training data. However, estimating the different $P(a_1, a_2 \ldots a_n | v_j)$ terms in this fashion is not feasible unless we have a very, very large set of training data. The problem is that the number of these terms is equal to the number of possible instances times the number of possible target values. Therefore, we need to see every instance in the instance space many times in order to obtain reliable estimates.

The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target value of the instance, the probability of observing the conjunction $a_1, a_2 \ldots a_n$ is just the product of the probabilities for the individual attributes: $P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$. Substituting this into Equation (6.19), we have the approach used by the naive Bayes classifier.

Naive Bayes classifier:

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j) \tag{6.20}$$

where v_{NB} denotes the target value output by the naive Bayes classifier. Notice that in a naive Bayes classifier the number of distinct $P(a_i | v_j)$ terms that must

be estimated from the training data is just the number of distinct attribute values times the number of distinct target values—a much smaller number than if we were to estimate the $P(a_1, a_2 \ldots a_n|v_j)$ terms as first contemplated.

To summarize, the naive Bayes learning method involves a learning step in which the various $P(v_j)$ and $P(a_i|v_j)$ terms are estimated, based on their frequencies over the training data. The set of these estimates corresponds to the learned hypothesis. This hypothesis is then used to classify each new instance by applying the rule in Equation (6.20). Whenever the naive Bayes assumption of conditional independence is satisfied, this naive Bayes classification v_{NB} is identical to the MAP classification.

One interesting difference between the naive Bayes learning method and other learning methods we have considered is that there is no explicit search through the space of possible hypotheses (in this case, the space of possible hypotheses is the space of possible values that can be assigned to the various $P(v_j)$ and $P(a_i|v_j)$ terms). Instead, the hypothesis is formed without searching, simply by counting the frequency of various data combinations within the training examples.

6.9.1 An Illustrative Example

Let us apply the naive Bayes classifier to a concept learning problem we considered during our discussion of decision tree learning: classifying days according to whether someone will play tennis. Table 3.2 from Chapter 3 provides a set of 14 training examples of the target concept *PlayTennis*, where each day is described by the attributes *Outlook, Temperature, Humidity*, and *Wind*. Here we use the naive Bayes classifier and the training data from this table to classify the following novel instance:

$$\langle Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong \rangle$$

Our task is to predict the target value (*yes* or *no*) of the target concept *PlayTennis* for this new instance. Instantiating Equation (6.20) to fit the current task, the target value v_{NB} is given by

$$\begin{aligned} v_{NB} &= \underset{v_j \in \{yes,no\}}{\text{argmax}} \; P(v_j) \textstyle\prod_i \; P(a_i|v_j) \\ &= \underset{v_j \in \{yes,no\}}{\text{argmax}} \; P(v_j) \quad P(Outlook = sunny|v_j) P(Temperature = cool|v_j) \\ &\qquad \cdot P(Humidity = high|v_j) P(Wind = strong|v_j) \end{aligned} \tag{6.21}$$

Notice in the final expression that a_i has been instantiated using the particular attribute values of the new instance. To calculate v_{NB} we now require 10 probabilities that can be estimated from the training data. First, the probabilities of the different target values can easily be estimated based on their frequencies over the 14 training examples

$$P(PlayTennis = yes) = 9/14 = .64$$

$$P(PlayTennis = no) = 5/14 = .36$$

Similarly, we can estimate the conditional probabilities. For example, those for $Wind = strong$ are

$$P(Wind = strong|PlayTennis = yes) = 3/9 = .33$$

$$P(Wind = strong|PlayTennis = no) = 3/5 = .60$$

Using these probability estimates and similar estimates for the remaining attribute values, we calculate v_{NB} according to Equation (6.21) as follows (now omitting attribute names for brevity)

$$P(yes)\ P(sunny|yes)\ P(cool|yes)\ P(high|yes)\ P(strong|yes) = .0053$$

$$P(no)\ P(sunny|no)\ P(cool|no)\ P(high|no)\ P(strong|no) = .0206$$

Thus, the naive Bayes classifier assigns the target value $PlayTennis = no$ to this new instance, based on the probability estimates learned from the training data. Furthermore, by normalizing the above quantities to sum to one we can calculate the conditional probability that the target value is *no*, given the observed attribute values. For the current example, this probability is $\frac{.0206}{.0206+.0053} = .795$.

6.9.1.1 ESTIMATING PROBABILITIES

Up to this point we have estimated probabilities by the fraction of times the event is observed to occur over the total number of opportunities. For example, in the above case we estimated $P(Wind = strong|PlayTennis = no)$ by the fraction $\frac{n_c}{n}$ where $n = 5$ is the total number of training examples for which $PlayTennis = no$, and $n_c = 3$ is the number of these for which $Wind = strong$.

While this observed fraction provides a good estimate of the probability in many cases, it provides poor estimates when n_c is very small. To see the difficulty, imagine that, in fact, the value of $P(Wind = strong|PlayTennis = no)$ is .08 and that we have a sample containing only 5 examples for which $PlayTennis = no$. Then the most probable value for n_c is 0. This raises two difficulties. First, $\frac{n_c}{n}$ produces a biased underestimate of the probability. Second, when this probability estimate is zero, this probability term will dominate the Bayes classifier if the future query contains $Wind = strong$. The reason is that the quantity calculated in Equation (6.20) requires multiplying all the other probability terms by this zero value.

To avoid this difficulty we can adopt a Bayesian approach to estimating the probability, using the m-estimate defined as follows.

m-estimate of probability:

$$\frac{n_c + mp}{n + m} \tag{6.22}$$

Here, n_c and n are defined as before, p is our prior estimate of the probability we wish to determine, and m is a constant called the *equivalent sample size*, which determines how heavily to weight p relative to the observed data. A typical method for choosing p in the absence of other information is to assume uniform

priors; that is, if an attribute has k possible values we set $p = \frac{1}{k}$. For example, in estimating $P(Wind = strong | PlayTennis = no)$ we note the attribute $Wind$ has two possible values, so uniform priors would correspond to choosing $p = .5$. Note that if m is zero, the m-estimate is equivalent to the simple fraction $\frac{n_c}{n}$. If both n and m are nonzero, then the observed fraction $\frac{n_c}{n}$ and prior p will be combined according to the weight m. The reason m is called the equivalent sample size is that Equation (6.22) can be interpreted as augmenting the n actual observations by an additional m virtual samples distributed according to p.

6.10 AN EXAMPLE: LEARNING TO CLASSIFY TEXT

To illustrate the practical importance of Bayesian learning methods, consider learning problems in which the instances are text documents. For example, we might wish to learn the target concept "electronic news articles that I find interesting," or "pages on the World Wide Web that discuss machine learning topics." In both cases, if a computer could learn the target concept accurately, it could automatically filter the large volume of online text documents to present only the most relevant documents to the user.

We present here a general algorithm for learning to classify text, based on the naive Bayes classifier. Interestingly, probabilistic approaches such as the one described here are among the most effective algorithms currently known for learning to classify text documents. Examples of such systems are described by Lewis (1991), Lang (1995), and Joachims (1996).

The naive Bayes algorithm that we shall present applies in the following general setting. Consider an instance space X consisting of all possible *text documents* (i.e., all possible strings of words and punctuation of all possible lengths). We are given training examples of some unknown target function $f(x)$, which can take on any value from some finite set V. The task is to learn from these training examples to predict the target value for subsequent text documents. For illustration, we will consider the target function classifying documents as interesting or uninteresting to a particular person, using the target values *like* and *dislike* to indicate these two classes.

The two main design issues involved in applying the naive Bayes classifier to such text classification problems are first to decide how to represent an arbitrary text document in terms of attribute values, and second to decide how to estimate the probabilities required by the naive Bayes classifier.

Our approach to representing arbitrary text documents is disturbingly simple: Given a text document, such as this paragraph, we define an attribute for each word position in the document and define the value of that attribute to be the English word found in that position. Thus, the current paragraph would be described by 111 attribute values, corresponding to the 111 word positions. The value of the first attribute is the word "our," the value of the second attribute is the word "approach," and so on. Notice that long text documents will require a larger number of attributes than short documents. As we shall see, this will not cause us any trouble.

Given this representation for text documents, we can now apply the naive Bayes classifier. For the sake of concreteness, let us assume we are given a set of 700 training documents that a friend has classified as *dislike* and another 300 she has classified as *like*. We are now given a new document and asked to classify it. Again, for concreteness let us assume the new text document is the preceding paragraph. In this case, we instantiate Equation (6.20) to calculate the naive Bayes classification as

$$\begin{aligned} v_{NB} &= \underset{v_j \in \{like, dislike\}}{\operatorname{argmax}} P(v_j) \prod_{i=1}^{111} P(a_i | v_j) \\ &= \underset{v_j \in \{like, dislike\}}{\operatorname{argmax}} P(v_j)\ P(a_1 = \text{“}our\text{”} | v_j) P(a_2 = \text{“}approach\text{”} | v_j) \\ &\qquad \ldots\ P(a_{111} = \text{“}trouble\text{”} | v_j) \end{aligned}$$

To summarize, the naive Bayes classification v_{NB} is the classification that maximizes the probability of observing the words that were actually found in the document, subject to the usual naive Bayes independence assumption. The independence assumption $P(a_1, \ldots a_{111} | v_j) = \prod_1^{111} P(a_i | v_j)$ states in this setting that the word probabilities for one text position are independent of the words that occur in other positions, given the document classification v_j. Note this assumption is clearly incorrect. For example, the probability of observing the word "learning" in some position may be greater if the preceding word is "machine." Despite the obvious inaccuracy of this independence assumption, we have little choice but to make it—without it, the number of probability terms that must be computed is prohibitive. Fortunately, in practice the naive Bayes learner performs remarkably well in many text classification problems despite the incorrectness of this independence assumption. Domingos and Pazzani (1996) provide an interesting analysis of this fortunate phenomenon.

To calculate v_{NB} using the above expression, we require estimates for the probability terms $P(v_j)$ and $P(a_i = w_k | v_j)$ (here we introduce w_k to indicate the kth word in the English vocabulary). The first of these can easily be estimated based on the fraction of each class in the training data ($P(like) = .3$ and $P(dislike) = .7$ in the current example). As usual, estimating the class conditional probabilities (e.g., $P(a_1 = \text{“}our\text{”} | dislike)$) is more problematic because we must estimate one such probability term for each combination of text position, English word, and target value. Unfortunately, there are approximately 50,000 distinct words in the English vocabulary, 2 possible target values, and 111 text positions in the current example, so we must estimate $2 \cdot 111 \cdot 50,000 \approx 10$ million such terms from the training data.

Fortunately, we can make an additional reasonable assumption that reduces the number of probabilities that must be estimated. In particular, we shall assume the probability of encountering a specific word w_k (e.g., "chocolate") is independent of the specific word position being considered (e.g., a_{23} versus a_{95}). More formally, this amounts to assuming that the attributes are independent and identically distributed, given the target classification; that is, $P(a_i = w_k | v_j) =$

$P(a_m = w_k|v_j)$ for all i, j, k, m. Therefore, we estimate the entire set of probabilities $P(a_1 = w_k|v_j)$, $P(a_2 = w_k|v_j)\ldots$ by the single position-independent probability $P(w_k|v_j)$, which we will use regardless of the word position. The net effect is that we now require only $2 \cdot 50,000$ distinct terms of the form $P(w_k|v_j)$. This is still a large number, but manageable. Notice in cases where training data is limited, the primary advantage of making this assumption is that it increases the number of examples available to estimate each of the required probabilities, thereby increasing the reliability of the estimates.

To complete the design of our learning algorithm, we must still choose a method for estimating the probability terms. We adopt the m-estimate—Equation (6.22)—with uniform priors and with m equal to the size of the word vocabulary. Thus, the estimate for $P(w_k|v_j)$ will be

$$\frac{n_k + 1}{n + |Vocabulary|}$$

where n is the total number of word positions in all training examples whose target value is v_j, n_k is the number of times word w_k is found among these n word positions, and $|Vocabulary|$ is the total number of distinct words (and other tokens) found within the training data.

To summarize, the final algorithm uses a naive Bayes classifier together with the assumption that the probability of word occurrence is independent of position within the text. The final algorithm is shown in Table 6.2. Notice the algorithm is quite simple. During learning, the procedure LEARN_NAIVE_BAYES_TEXT examines all training documents to extract the vocabulary of all words and tokens that appear in the text, then counts their frequencies among the different target classes to obtain the necessary probability estimates. Later, given a new document to be classified, the procedure CLASSIFY_NAIVE_BAYES_TEXT uses these probability estimates to calculate v_{NB} according to Equation (6.20). Note that any words appearing in the new document that were not observed in the training set are simply ignored by CLASSIFY_NAIVE_BAYES_TEXT. Code for this algorithm, as well as training data sets, are available on the World Wide Web at http://www.cs.cmu.edu/~tom/book.html.

6.10.1 Experimental Results

How effective is the learning algorithm of Table 6.2? In one experiment (see Joachims 1996), a minor variant of this algorithm was applied to the problem of classifying usenet news articles. The target classification for an article in this case was the name of the usenet newsgroup in which the article appeared. One can think of the task as creating a newsgroup posting service that learns to assign documents to the appropriate newsgroup. In the experiment described by Joachims (1996), 20 electronic newsgroups were considered (listed in Table 6.3). Then 1,000 articles were collected from each newsgroup, forming a data set of 20,000 documents. The naive Bayes algorithm was then applied using two-thirds of these 20,000 documents as training examples, and performance was measured

LEARN_NAIVE_BAYES_TEXT($Examples$, V)

Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms $P(w_k|v_j)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

1. *collect all words, punctuation, and other tokens that occur in Examples*
   - $Vocabulary \leftarrow$ the set of all distinct words and other tokens occurring in any text document from $Examples$
2. *calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms*
   - For each target value v_j in V do
     - $docs_j \leftarrow$ the subset of documents from $Examples$ for which the target value is v_j
     - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
     - $Text_j \leftarrow$ a single document created by concatenating all members of $docs_j$
     - $n \leftarrow$ total number of distinct word positions in $Text_j$
     - for each word w_k in $Vocabulary$
       - $n_k \leftarrow$ number of times word w_k occurs in $Text_j$
       - $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

CLASSIFY_NAIVE_BAYES_TEXT(Doc)

Return the estimated target value for the document Doc. a_i denotes the word found in the ith position within Doc.

- *positions* $\leftarrow$ all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return v_{NB}, where

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

TABLE 6.2
Naive Bayes algorithms for learning and classifying text. In addition to the usual naive Bayes assumptions, these algorithms assume the probability of a word occurring is independent of its position within the text.

over the remaining third. Given 20 possible newsgroups, we would expect random guessing to achieve a classification accuracy of approximately 5%. The accuracy achieved by the program was 89%. The algorithm used in these experiments was exactly the algorithm of Table 6.2, with one exception: Only a subset of the words occurring in the documents were included as the value of the $Vocabulary$ variable in the algorithm. In particular, the 100 most frequent words were removed (these include words such as "the" and "of"), and any word occurring fewer than three times was also removed. The resulting vocabulary contained approximately 38,500 words.

Similarly impressive results have been achieved by others applying similar statistical learning approaches to text classification. For example, Lang (1995) describes another variant of the naive Bayes algorithm and its application to learning the target concept "usenet articles that I find interesting." He describes the NEWSWEEDER system—a program for reading netnews that allows the user to rate articles as he or she reads them. NEWSWEEDER then uses these rated articles as