# NAIVE BAYESIAN LEARNING

Charles Elkan

Department of Computer Science
Harvard University

Adapted from Technical Report No. CS97-557, Department of Computer Science and Engineering, University of California, San Diego, September 1997.

## 1 Introduction

So-called "naive" Bayesian classification is the optimal method of supervised learning if the values of the attributes of an example are independent given the class of the example. Although this assumption is almost always violated in practice, recent work has shown that naive Bayesian learning is remarkably effective in practice and difficult to improve upon systematically [Domingos and Pazzani, 1996].

On many real-world example datasets naive Bayesian learning gives better test set accuracy than any other known method, including backpropagation and C4.5 decision trees. Also, these classifiers can be learned very efficiently. Given $e$ training examples over $f$ attributes, the time required to learn a boosted naive Bayesian classifier is $O(ef)$, i.e. linear. No learning algorithm that examines all its training data can be faster.

## 2 A review of naive Bayesian learning

Let $A_1$ through $A_k$ be attributes with discrete values used to predict a discrete class $C$. Given an example with observed attribute values $a_1$ through $a_k$, the optimal prediction is class value $c$ such that $Pr(C = c \mid A_1 = a_1 \ \wedge \ \ldots \ \wedge \ A_k = a_k)$ is maximal. By Bayes' rule this probability equals

$$\frac{Pr(A_1 = a_1 \ \wedge \ \ldots \ \wedge \ A_k = a_k \mid C = c)}{Pr(A_1 = a_1 \ \wedge \ \ldots \ \wedge \ A_k = a_k)} Pr(C = c).$$

The background probability or base rate $Pr(C = c)$ can be estimated from training data easily. The example probability $Pr(A_1 = a_1 \ \wedge \ \ldots \ \wedge \ A_k = a_k)$ is irrelevant for decision-making since it is the same for each class value $c$. Learning is therefore reduced to the problem of estimating $Pr(A_1 = a_1 \ \wedge \ \ldots \ \wedge \ A_k = a_k \mid C = c)$ from training examples. Using Bayes' rule again, this class-conditional probability can be written as

$$Pr(A_1 = a_1 \mid A_2 = a_2 \ \wedge \ \ldots \ \wedge \ A_k = a_k, C = c) \cdot Pr(A_2 = a_2 \ \wedge \ \ldots \ \wedge \ A_k = a_k \mid C = c).$$

Recursively, the second factor above can be written as

$$Pr(A_2 = a_2 \mid A_3 = a_3 \ \wedge \ \ldots \ \wedge \ A_k = a_k, C = c) \cdot Pr(A_3 = a_3 \ \wedge \ \ldots \ \wedge \ A_k = a_k \mid C = c)$$

and so on. Now suppose we assume for each $A_i$ that its outcome is independent of the outcome of all other $A_j$, given $C$. Formally, we assume that

$$Pr(A_1 = a_1 \mid A_2 = a_2 \ \wedge \ \ldots \ \wedge \ A_k = a_k, C = c) = Pr(A_1 = a_1 \mid C = c)$$

and so on for $A_2$ through $A_k$. Then $Pr(A_1 = a_1 \ \wedge \ \ldots \ \wedge \ A_k = a_k \mid C = c)$ equals

$$Pr(A_1 = a_1 \mid C = c) Pr(A_2 = a_2 \mid C = c) \cdots Pr(A_k = a_k \mid C = c).$$

Now each factor in the product above can be estimated from training data:

$$\hat{Pr}(A_j = a_j \mid C = c) = \frac{count(A_j = a_j \ \wedge \ C = c)}{count(C = c)}. \tag{1}$$

It can be shown that Equation (1) gives "maximum likelihood" probability estimates, i.e. probability parameter values that maximize the probability of the training examples.

The induction algorithm explained above is called naive Bayesian learning. One early reference is Chapter 12 of the celebrated *Perceptrons* book by Minsky and Papert [1969].

## 3 Taking logarithms

Suppose that there are just two possible classes, called 0 and 1, and let $a_1$ through $a_k$ be the observed attribute values for a test example. Let $b_0 = Pr(C = 0)$, let $b_1 = Pr(C = 1) = 1 - b_0$, let $p_{j0} = Pr(A_j = a_j \mid C = 0)$, and let $p_{j1} = Pr(A_j = a_j \mid C = 1)$. Then

$$p = Pr(C = 1 \mid A_1 = a_1 \ \wedge \ \ldots \ \wedge \ A_k = a_k) = \Big(\prod_{j=1}^{k} p_{j1}\Big) b_1 / z$$

and

$$q = Pr(C = 0 \mid A_1 = a_1 \ \wedge \ \ldots \ \wedge \ A_k = a_k) = \Big(\prod_{j=1}^{k} p_{j0}\Big) b_0 / z$$

where $z$ is a normalizing constant. Taking logarithms gives

$$\log p - \log q = (\sum_{j=1}^{k} \log p_{j1} - \log p_{j0}) + \log b_1 - \log b_0.$$

Letting $w_j = \log p_{j1} - \log p_{j0}$ and $b = \log b_1 - \log b_0$ gives

$$\log \frac{1 - p}{p} = - \sum_{j=1}^{k} w_j - b.$$

Exponentiating both sides and rearranging gives

$$p = \frac{1}{1 + e^{-\left(\sum_{j=1}^{k} w_j\right) - b}}.$$

In general, suppose the attribute $A_j$ has $v(j)$ possible values. Let

$$w_{jj'} = \log Pr(A_j = a_{jj'} \mid C = 1) - \log Pr(A_j = a_{jj'} \mid C = 0)$$

where $1 \leq j' \leq v(j)$. Then

$$Pr(C(x) = 1) = \frac{1}{1 + e^{-\left(\sum_{j=1}^{k} \sum_{j'=1}^{v(j)} I(A_j(x) = a_{jj'}) w_{jj'}\right) - b}}. \tag{2}$$

Here $I$ is an indicator function: $I(\phi) = 1$ if $\phi$ is true and $I(\phi) = 0$ if $\phi$ is false.

Naive Bayesian classification generalizes logistic regression, the most widely used statistical method for probabilistic classification from numerical attribute values. Consider again Equation (1). If $A_j$ has discrete values then $count(A_j = a_j \ \wedge \ C = c)$ can be computed directly from training examples. If the values of $A_j$ are numerical, the standard practice is to quantize or discretize the attribute. The simplest quantization method is to divide the range of the attribute into a fixed number $M$ of equal-width bins. In the experiments described below $M = 10$ is chosen arbitrarily. Previous experimental work has shown that the benefits of more complex quantization methods are small at best [Dougherty *et al.*, 1995]. Using bins of equal width tends to work well because they allow good non-parametric approximation of skewed, multimodal, and/or heavy-tailed probability distributions.

Let each $A_j$ be a numerical (integer-valued or real-valued) attribute. Logistic regression assumes the model

$$\log \frac{Pr(C = 1 \mid A_1 = a_1, \ldots A_k = a_k)}{Pr(C = 0 \mid A_1 = a_1, \ldots A_k = a_k)} = \sum_{j=1}^{k} b_j a_j + b_0. \tag{3}$$

This equation describes the functioning of a perceptron with a sigmoid activation function and a single input node for each attribute, i.e. with attribute values encoded as their magnitude. Discretization of $A_j$ corresponds to replacing the linear term $b_j a_j$ by a piecewise constant function of $a_j$. If the range of $A_j$ is divided into $M$ intervals where the $i$th interval is $[c_{j(i-1)}, c_{ji}]$ then this function is

$$f_j(a_j) = \sum_{i=1}^{M} b_{ji} I(c_{j(i-1)} \leq a_j < c_{ji}) \tag{4}$$

where each $b_{ji}$ is a constant. Combining Equations (3) and (4) yields a version of Equation (2). Hence naive Bayesian classification is a nonparametric, nonlinear extension of logistic regression; the standard logistic regression equation can be approximated by setting $b_{ji} = (c_{j(i-1)} + c_{ji})/(2b_j)$.

## 4 Computational complexity

Suppose that examples are over $f$ attributes, each with $v$ values. Then a naive Bayesian classifier as in Equation (2) has $fv+1$ parameters. These parameters are learned by accumulating $2fv+2$ counts.

Each attribute value of each training example leads to incrementing exactly one count. Hence with $e$ examples training time is $O(ef)$ independent of $v$. This time complexity is essentially optimal: any learning algorithm that examines every attribute value of every training example must have the same or worse complexity. For comparison, learning a decision tree without pruning requires $O(ef^2)$ time. (The algorithm that underlies this result is non-trivial.) The time required to update weights under boosting is also $O(ef)$, so $T$ rounds of boosting use $O(Tef)$ time in total, which is also $O(ef)$ if $T$ is constant.

When accumulating the counts on which a naive Bayesian classifier is based, training examples may be processed sequentially directly from disk or tape storage. The amount of random-access memory required is $O(fv)$ independent of the number of training examples. With boosting, the random-access memory needed is $O(Tfv)$. Therefore, boosted naive Bayesian classifiers are well-suited for knowledge discovery in very large databases: these databases can remain on disk or even on tape, and need not fit into main memory.

It is straightforward to show that learning a boosted naive Bayesian classifier is in the parallel complexity class NC. This class consists of problems that can be solved in polylogarithmic time with a polynomial number of processors. With communication patterned after a binary tree of height $\log e$, $e$ processors can compute $count(A_j = a \wedge C = 0)$ and $count(A_j = a \wedge C = 1)$ for a given $j$ and $a$ in $O(\log e)$ time. Therefore naive Bayesian learning requires $O(\log e)$ time using $O(efv)$ processors.

In practice, with $f$ and $v$ constant and many examples allocated to each processor, a parallel implementation could use far fewer than $e$ processors and still run in $O(\log e)$ time. With a constant number of rounds of boosting, boosted naive Bayesian learning is also in NC. A literature search reveals that no other practical learning problem is known to have an NC algorithm.

## References

[Domingos and Pazzani, 1996] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 105–112. Morgan Kaufmann Publishers, Inc., 1996.

[Dougherty *et al.*, 1995] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 194–202. Morgan Kaufmann Publishers, Inc., 1995.

[Minsky and Papert, 1969] Marvin Minsky and Seymour Papert. *Perceptrons; an introduction to computational geometry*. MIT Press, 1969.