A Naive Bayes Classifier Plug-In for DataEngine™

Christian Borgelt

Dept. of Knowledge Processing and Language Engineering
Otto-von-Guericke-University of Magdeburg
Universitätsplatz 2, D-39106 Magdeburg, Germany

e-mail: borgelt@iws.cs.uni-magdeburg.de

Abstract: Naive Bayes classifiers are an old and well-known type of classifiers that can be seen as a special type of probabilistic networks. They use a probabilistic approach to assign a class to a case or an object and can easily be induced from a dataset of sample cases. In this paper I review this technique in order to demonstrate its simplicity and its power to produce comprehensible results. Since the task to induce a classifier from data turns up frequently in applications (e.g. credit assessment, disease detection etc.) a commercial data analysis tool should provide an implementation of this technique. Nevertheless, the well-known data analysis program DataEngine™ has until recently suffered from lacking such a module. This drawback is now removed by a plug-in consisting of a set of user-defined function blocks I implemented.

1 Introduction

Naive Bayes classifiers [3, 2, 4, 5] are an old and well-known type of classifiers. Classifiers, in turn, are programs which automatically classify a case or an object, i.e. assign it according to its features to one of several predefined classes. For example, if the cases are patients in a hospital, the attributes are properties of the patients (e.g. sex, age etc.) and their symptoms (e.g. fever, high blood pressure etc.), the classes may be diseases or drugs to administer.

Naive Bayes classifiers use a probabilistic approach to assign the classes. That is, they try to compute the conditional probabilities of the different classes given the values of other attributes and then predict the class with the highest conditional probability. Since it is usually impossible to store or even to estimate these conditional probabilities, they exploit Bayes rule and a set of conditional independence statements to simplify the task. A detailed description is given in section 2.

Due to the strong independence assumptions, but also because some attributes may not be able to contribute to the classification accuracy, it is not always advisable to use all available attributes. With all attributes a naive Bayes classifier is more complicated than necessary and sometimes even yields results that can be improved upon by using fewer attributes. Therefore a naive Bayes classifier should be simplified. Two very simple methods to reduce the number of attributes are discussed in section 3.

In section 4 I describe the plug-in I implemented for the well-known data analysis tool DataEngine™. This plug-in consists of three function blocks: one to induce (and simplify) a naive Bayes classifier, one to classify new data, and one to compute a confusion matrix to evaluate the quality of the induced classifier.

2 Naive Bayes Classifiers

As already mentioned in the introduction, naive Bayes classifiers use a probabilistic approach to classify data: They try to compute conditional class probabilities and then predict the most probable class. To be more precise, let C denote a class attribute with a finite domain of m classes, i.e., $\mathrm{dom}(C) = \{c_1, \ldots, c_m\}$, and let $A_1, \ldots, A_n$ be a set of other attributes used to describe a case or an object of the universe of discourse. These other attributes may be symbolic, i.e., $\mathrm{dom}(A_j) = \{a_1^{(j)}, \ldots, a_{m_j}^{(j)}\}$, or numeric, i.e., $\mathrm{dom}(A_j) = \mathbb{R}$. For simplicity, I always use the notation $a_{i_j}^{(j)}$ for a value of an attribute A_j, independent of whether it is a symbolic or a numeric one.[1] With this notation, a case or an object can be described by an instantiation $\omega = (a_{i_1}^{(1)}, \ldots, a_{i_n}^{(n)})$ of the attributes $A_1, \ldots, A_n$ and thus the universe of discourse is $\Omega = \mathrm{dom}(A_1) \times \ldots \times \mathrm{dom}(A_n)$.

For a given instantiation ω, a naive Bayes classifier tries to compute the conditional probability

$$P(C = c_i \mid \omega) = P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)})$$

for all c_i and then predicts the class for which this probability is highest. Of course, it is usually impossible to store all of these conditional probabilities explicitly, so that a simple lookup would be all that is

[1] To be able to use this notation for numeric attributes, one simply has to choose an appropriate uncountably infinite index set $\mathcal{I}_j$, from which the index i_j is to be taken.

needed to find the most probable class. If there are numeric attributes, this is obvious (some parameterized function is needed then). But even if all attributes are symbolic, such an approach most often is infeasible: A class (or a class probability distribution) has to be stored for each point of the Cartesian product of the attribute domains, whose size grows exponentially with the number of attributes. To circumvent this problem, naive Bayes classifiers exploit—as their name already indicates—Bayes rule and a set of conditional independence assumptions. With Bayes rule the conditional probabilities are inverted, i.e., naive Bayes classifiers consider[2]

$$P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) = \frac{f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)} \mid C = c_i) \cdot P(C = c_i)}{f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)})}$$

Of course, for this inversion to be possible, the probability density function $f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)})$ must be strictly positive.

There are two observations to be made about the inversion carried out above. In the first place, the denominator of the fraction on the right can be neglected, since for a given case or object to be classified, it is fixed and therefore does not have any influence on the class ranking (which is all we are interested in). In addition, its influence can always be restored by normalizing the class distribution, i.e., we can exploit

$$f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) = \sum_{i=1}^{m} f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)} \mid C = c_i) \cdot P(C = c_i).$$

It follows that we only need to consider

$$P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) = \frac{1}{S} f(A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)} \mid C = c_i) \cdot P(C = c_i),$$

where S is a normalization constant.[3]

Secondly, we can see that just inverting the probabilities does not buy us anything, since the probability space is just as large as it was before the inversion. However, here the second ingredient of naive Bayes classifiers, which is responsible for the "naive" in their name, comes in, namely the conditional independence assumptions. To exploit them, we first apply the chain rule of probability:

$$\begin{aligned} P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) \\ = \frac{1}{S} \cdot f(A_n = a_{i_n}^{(n)} \mid A_{n-1} = a_{i_{n-1}}^{(n-1)}, \ldots, \\ A_1 = a_{i_1}^{(1)}, C = c_i) \\ \cdots \\ \cdot f(A_2 = a_{i_2}^{(2)} \mid A_1 = a_{i_1}^{(1)}, C = c_i) \\ \cdot f(A_1 = a_{i_1}^{(1)} \mid C = c_i) \cdot P(C = c_i). \end{aligned}$$

Now we make the crucial assumption that given the value of the class attribute, any attribute A_j is independent of any other. That is, we assume that knowing the class is enough to determine the probability (density) for a value $a_{i_j}^{(j)}$, i.e., that we need not know the values of any other attributes. Of course, this is a pretty strong assumption, which is very likely to fail. However, it considerably simplifies the formula stated above, since with it we can cancel all attributes A_j appearing in the conditions:

$$\begin{aligned} P(C = c_i \mid A_1 = a_{i_1}^{(1)}, \ldots, A_n = a_{i_n}^{(n)}) \\ = \frac{1}{S} \cdot f(A_1 = a_{i_1}^{(1)} \mid C = c_i) \\ \cdot f(A_2 = a_{i_2}^{(2)} \mid C = c_i) \\ \cdots \\ \cdot f(A_n = a_{i_n}^{(n)} \mid C = c_i) \cdot P(C = c_i). \end{aligned}$$

This is the fundamental formula underlying naive Bayes classifiers. For a symbolic attribute A_j the conditional probabilities $P(A_j = a_{i_j}^{(j)} \mid C = c_i)$ are stored as a simple conditional probability table. This is feasible now, since there is only one condition and hence only $m \cdot m_j$ probabilities have to be stored.[4] For numeric attributes it is usually assumed that the probability density is a Gaussian function (a normal distribution) and hence only the expected values $\mu_j(c_i)$ and the variances $\sigma_j^2(c_i)$ need to be stored in this case.

It should be noted that naive Bayes classifiers can be seen as a special type of probabilistic networks, or, to be more precise, of Bayesian networks [7]. Due to the strong independence assumptions underlying them, the corresponding network has a very simple structure: It is star-like with the class attribute being the source of all edges (see figure 1).

Naive Bayes classifiers can easily be induced from a dataset of preclassified sample cases. All one has to do is to estimate the conditional probabilities/probability densities $f(A_j = a_{i_j}^{(j)} \mid C = c_i)$ using, for instance,

[2]For simplicity, we always use a probability density function f, although this is strictly correct only, if there is at least one numeric attribute. If all attributes are symbolic, this should be a probability P. The only exception is the class attribute, since it necessarily has a finite domain.

[3]Strictly speaking, the constant S is dependent on the instantiation $(a_{i_1}^{(1)}, \ldots, a_{i_n}^{(n)})$. However, as already said above, when classifying a given case or object, this instantiation is fixed and hence we need to consider only one value S.

[4]Actually only $m \cdot (m_j - 1)$ probabilities are really necessary. Since the probabilities have to add up to one, one value can be discarded from each conditional distribution. However, in implementations it is usually much easier to store all probabilities.

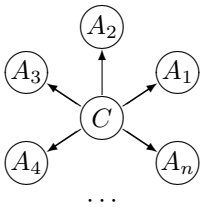

Figure 1: A naive Bayes classifier is a Bayesian network with a star-like structure.

maximum likelihood estimation. For symbolic attributes, this yields

$$\hat{P}(A_j = a_{i_j}^{(j)} \mid C = c_i) = \frac{\#(A_j = a_{i_j}^{(j)}, C = c_i)}{\#(C = c_i)},$$

where $\#(C = c_i)$ is the number of sample cases that belong to the class c_i and $\#(A_j = a_{i_j}^{(j)}, C = c_i)$ is the number of sample cases belonging to class c_i and having the value $a_{i_j}^{(j)}$ for the attribute A_j. To ensure that the probability is strictly positive (see above), it is assumed that there is at least one example for each class in the dataset. Otherwise the class is simply removed from the domain of the class attribute. If an attribute value does not occur given some class, its probability is either set to $\frac{1}{2N}$, where N is the number of sample cases, or a uniform prior of, for example, $\frac{1}{N}$ is always added to the estimated distribution, which is then renormalized (Laplace correction).

For a numeric attribute A_j the standard maximum likelihood estimation functions

$$\hat{\mu}_j(c_i) = \frac{1}{\#(C = c_i)} \sum_{k=1}^{\#(C=c_i)} a_{i_j(k)}^{(j)}$$

for the expected value, where $a_{i_j(k)}^{(j)}$ is the value of the attribute A_j in the k-th sample case belonging to class c_i, and

$$\hat{\sigma}_j^2(c_i) = \frac{1}{\#(C = c_i)} \sum_{k=1}^{\#(C=c_i)} \left(a_{i_j(k)}^{(j)} - \hat{\mu}_j(c_i)\right)^2$$

for the variance can be used.

As an illustrative example, let us take a look at the well-known iris data [6]. The problem is to predict the iris type (iris setosa, iris versicolor, or iris virginica) from measurements of the sepal length and width and the petal length and width. Due to the limited number of dimensions of a sheet of paper we confine ourselves to the latter two measures. The naive Bayes classifier induced from these two measures and all 150 cases is shown in table 1. It is easy to see from this table how different petal lengths and widths provide evidence for the different types of iris flowers. The conditional probability density functions used by this naive Bayes classifier to predict the iris type are shown graphically in figure 2. The ellipses are the 2σ-boundaries of the (bivariate) normal distribution. As a consequence of the strong conditional independence assumptions, these ellipses are axis-parallel: The normal distributions are estimated separately for each dimension and no covariance is taken into account.

iris type	setosa	versicolor	virginica
prior prob.	0.333	0.333	0.333
petal length	1.46 ± 0.17	4.26 ± 0.46	5.55 ± 0.55
petal width	0.24 ± 0.11	1.33 ± 0.20	2.03 ± 0.27

Table 1: A naive Bayes classifier for the iris data. The normal distributions are described by stating $\hat{\mu} \pm \hat{\sigma}$.

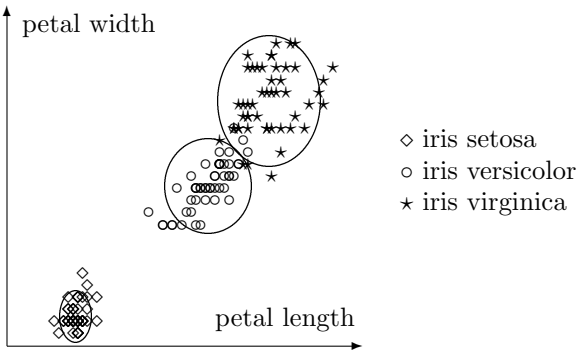

Figure 2: Naive Bayes density functions for the iris data. The ellipses are the 2σ-boundaries of the probability density functions.

3 Classifier Simplification

A naive Bayes classifier makes strong independence assumptions (see above). It is not surprising that these assumptions are likely to fail. If they fail, the classifier may be worse than necessary. In addition, some attributes may not contribute to the classification accuracy, making the classifier more complicated than necessary. To cope with these problems, simplification methods may be used, for instance, simple greedy attribute selection. With this procedure one can hope to find a subset of attributes for which the strong assumptions hold at least approximately.

I consider here two very simple, but effective attribute selection methods: The first method starts with a classifier that simply predicts the majority class and does not use any attribute information. Then attributes are added one by one. In each step that attribute is selected which, if added, leads to the smallest number of misclassifications on the training data. The process stops when adding any of the remaining attributes does not reduce the number of errors.

The second method is a reversal of the first. It starts with a classifier that uses all available attributes and then removes attributes step by step. In each step that attribute is selected which, if removed, leads to the smallest number of misclassifications on the training data. The process stops when removing any of the remaining attributes leads to a larger number of errors.

4 The DataEngine™ Plug-In

I implemented a naive Bayes classifier as a plug-in for the well-known data analysis program DataEngine™ in order to improve this esteemed tool even further. It consists of three user-defined function blocks:

nbi — Naive Bayes classifier induction.

This function block receives as input a table of classified sample cases and induces a naive Bayes classifier. The data types of the table columns (either symbolic or numeric) can be stated in the unit fields of the table columns, which can also be used to instruct the algorithm to ignore certain columns. Although tables passed to user-defined functions blocks may not contain unknown values, this function block provides a facility to specify which table fields should be considered as unknown: In the configuration dialog you may enter a value for the lowest known value. All values below this value are considered to be unknown. In addition the configuration dialog lets you choose a simplification method and you can specify the Laplace correction to be used (as a multiple of the standard value $\frac{1}{n}$, where n is the number of tuples the classifier is induced from) and the name of a file into which the induced naive Bayes classifier should be saved.

nbc — Naive Bayes classification.

This function block receives as input an induced naive Bayes classifier. It executes the naive Bayes classifier for each tuple in the table and adds to it a new column containing the class predicted by the classifier. The configuration dialog lets you enter the name of the classification column, the Laplace correction to be used (again as a multiple of the standard value $\frac{1}{n}$, where n is the number of tuples the classifier was induced from), and (just as described for the block above) a lowest known value. In addition you can request an additional column into which a confidence value is written for each classified tuple. This confidence value is the probability of the predicted class as computed by the classifier.

xmat — Compute a confusion matrix.

This function blocks receives as input a table. Its configuration dialog lets you enter the names of two columns for which a confusion matrix shall be determined. It generates a table containing the confusion matrix (either with absolute or relative numbers) and the sums over lines and columns (excluding the diagonal elements). This function block is identical to the one I described last year for the decision tree plug-in [1]. I include it here, since it is useful for evaluating the quality of classifiers of arbitrary type.

All function blocks dealing directly with naive Bayes classifiers (*nbi* and *nbc*) also comprise a viewer which lets you inspect the constructed naive Bayes classifier using the well-known MS Windows tree view control (used, for example, in the MS Windows explorer to visualize the hierarchic file system). Hence you need not accept the classifier as a black box (as it is usually the case for e.g. neural networks), but you can inspect what evidence it exploits to arrive at its results.

5 Summary

In this paper I reviewed the well-known naive Bayes classifier in order to demonstrate its simplicity and its power to produce comprehensible results, which makes it a useful tool for many data analysis task. I presented an implementation of a naive Bayes classifier induction algorithm as a plug-in for the well-known data-analysis tool DataEngine™. This implementation comprises a simplification procedure to improve the quality of induced classifiers and lets you choose freely the Laplace correction to be used. Two other function blocks allow you to execute an induced naive Bayes classifier to classify a set of cases and to compute a confusion matrix to assess the quality of an induced classifier.

References

[1] C. Borgelt. A Decision Tree Plug-In for DataEngine™. *Proc. 2nd Data Analysis Symposium*, Aachen, Germany 1998

[2] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. J. Wiley & Sons, New York, NY, USA 1973

[3] I.J. Good. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press, MA, USA, 1965

[4] P. Langley, W. Iba, and K. Thompson. An Analysis of Bayesian Classifiers. *Proc. 10th Nat. Conf. on Artificial Intelligence*, 223–228, AAAI Press and MIT Press, USA 1992

[5] P. Langley and S. Sage. Induction of Selective Bayesian Classifiers. *Proc. 10th Conf. on Artificial Intelligence*, 1994

[6] P.M. Murphy and D. Aha, UCI Repository of Machine Learning Databases, ftp from ics.uci.edu, directory pub/machine-learning-databases, 1994

[7] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference (2nd edition)*. Morgan Kaufman, New York 1992