Naive + Naive = Smart Bayes?

J.M. Martínez-Otzeta, B. Sierra, E. Lazkano, E. Jauregi and Y. Yurramendi
Department of Computer Science and Artificial Intelligence
University of the Basque Country, Donostia, Spain
ccbmaotj@si.ehu.es

Abstract

Naive Bayes is a well-known and studied algorithm both in statistics and machine learning. Bayesian learning algorithms represent each concept with a single probabilistic summary. This paper presents a novel method to construct a hierarchy of Naive Bayes classifiers, in which two levels of Naive Bayes classifiers are applied. The underlying motivation is to try to correct the mistakes made by a single Naive Bayes classifier applying Naive Bayes again over its output. We have tested this algorithm over 20 UCI datasets. The results suggest that the chance of obtaining a better performance than with the original Naive Bayes approach are much greater than the opposite.

Keywords: data mining, Naive Bayes, combination of classifiers

1 Introduction

In the supervised learning task [1], the main goal is to induce a classification model from a given training database for which we know the labelling of every sample. Existing paradigms come from the area of the Artificial Intelligence and are grouped in the family of *Machine Learning* (ML).

The goal of supervised learning is to predict the class labels of examples that have not been seen; to achieve this goal a number of different techniques have been developed, among them Bayesian models, decision trees, decision rules, neural networks or SVMs.

The Naive Bayes algorithm is one of the oldest and most well-known machine learning approaches [2]. To make the probability based classification computation feasible, Naive Bayes assume independence among the variables, in a *naive* fashion. Such assumption in real problems is certainly a simplistic one. But, despite this objection, Naive Bayes works rather well when tested on actual databases, particularly when combined with some attribute selection procedure.

On the other hand, several lines of research have dealt with the best way of combining classifiers in order to improve the accuracy of a single classifier.

There are very different manners to combine classifiers, most of which have been found to improve accuracy over single classifiers [3]. Several approaches have been studied, among them: bagging [4], that uses more than one model of the same paradigm in order to reduce errors; boosting [5], in which a different weight is given to different training examples looking for a better accuracy; bi-layer classifiers [6] where several models from different paradigms are combined in a parallel mode, and after the individual decision of each one is used as predictor variable for a new classifier which takes the final decision; or some other combination approach in serial or semi-parallel architectures [7, 8].

In this paper a new variant of the Naive Bayes algorithm is presented and tested. A hierarchy consisting of two levels of Naive Bayes classifiers is built, where the classifiers in the second level act over the predictions of the first level. The idea behind this procedure is that the mistakes a classifier makes are suitable to be the input to another classifier, in such a way that this one could discover some patterns in them.

Experimental results have been obtained over 20 UCI datasets, and show that the presented paradigm outperforms standard Naive-Bayes in 12 out of the 20 datasets.

The rest of the paper is organised as follows. Section 2 describes the Naive Bayes classifier; in section 3 the new proposed approach is presented. Experimental results are shown in section 4. Final section is dedicated to conclusions and points out the future work.

2 Naive Bayes

Given a database of cases consisting of a set of predictor variables $X_1, X_2, \cdots, X_n$ and a special variable Y which value must be predicted, being each of the N cases in the form

$$\text{Case}_i : X_1 = x_{1i}, X_2 = x_{2i}, \cdots, X_n = x_{ni}, Y = y_l$$

and being, without lost of generality, $Y = y_1, Y = y_2, \cdots, Y = y_m$ the m possible values the class variable can take (the m categories considered in the classification problem), theoretically, Bayes' rule

minimises error by selecting the class y_j with the largest posterior probability for a given example $\mathbf{X}$ of the form $X =< X_1 = x_1, X_2 = x_2, \cdots, X_n = x_n >$, as indicated below:

$$P(Y = y_j|X) = \frac{P(Y = y_j)P(\mathbf{X}|Y = y_j)}{P(\mathbf{X})}.$$

Since $\mathbf{X}$ is a composition of n discrete values, one can expand this expression to:

$$P(Y = y_j|X_1 = x_1, ..., X_n = x_n) =$$

$$\frac{P(Y = y_j)P(X_1 = x_1, ..., X_n = x_n|Y = y_j)}{P(X_1 = x_1, ..., X_n = x_n)}$$

where $P(X_1 = x_1, ..., X_n = x_n|Y = y_j)$ is the conditional probability of the instance $\mathbf{X}$ given the class y_j; $P(Y = y_j)$ is the a priori probability that one will observe class y_j; $P(\mathbf{X})$ is the prior probability of observing the instance $\mathbf{X}$. All these parameters are estimated from the training set. However, a direct application of these rules is difficult due to the lack of sufficient data in the training set to reliably obtain all the conditional probabilities needed by the model. One simple form of the previous model has been studied [2] that assumes independence of the observations of feature variables $X_1, X_2, ..., X_n$ given the class variable Y, which allows us to use the next equality

$$P(X_1 = x_1, ..., X_n = x_n|Y = y_j) \propto$$

$$P(Y = y_j) \prod_{i=1}^{n} P(X_i = x_i|Y = y_j)$$

where $P(X_i = x_i|Y = y_j)$ is the probability of an instance of class y_j having the observed attribute value x_i. In the core of this paradigm there is an assumption of independence between the occurrence of features values, that is not true in many tasks; however, it is empirically demonstrated that this paradigm gives good results in many real domains, typically in medical tasks. Some modifications of the Naive Bayes have been proposed, like hybrid methods [9], training set edition [10], or new attributes construction [11] [12]. A review of the health of Naive Bayes method at its fortieth birthday can be found in [13].

3 Smart Bayes

In this section we present the way a two-level hierarchy of Naive Bayes classifiers works, as well as the method we have used to construct it. We have named this approach Smart Bayes, as it is intended to overcome some limitations of Naive Bayes.

3.1 Smart Bayes at Work

In a hierarchy of Naive Bayes classifiers single classifiers are arranged in a tree, as depicted in figure 1. Thereafter, a new case runs the following path: starting at the root node, the instance is classified by the top level Naive Bayes classifier; according to the result, a branch is selected and the instance is classified again according to the Naive Bayes model corresponding to that branch and the final outcome is returned.

The main idea of the presented approach is to take into account the fact that a Naive Bayes classifier (as any other ML paradigm) performs different when classifying cases as belonging to a class or to another. In other words, for the presented positive-negative example, the amount of false positives and false negatives could be different. Therefore, another Naive Bayes classifier can be used to correct some of the errors made in the previous classification, i.e., to turn true negatives some of the false positives (classified as *Yes* in the first attempt) and to turn also true positives some of the false negatives (badly classified as *No* in the first layer).

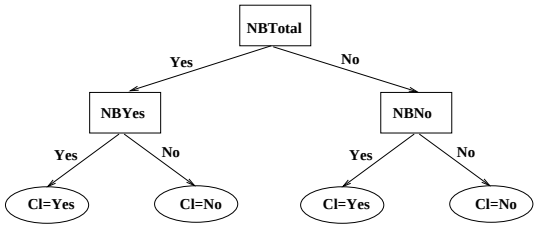

Figure 1: A hierarchy of Naive Bayes classifiers in a Yes-No problem.

3.2 Smart Bayes under Construction

To construct a two-level classifier it needs to be inferred, from a unique training set, all the models that form the hierarchy. While it is more or less obvious how to build a suitable model for the root node (just using the whole training set), it is not so obvious how to get the model for non-root nodes. When a case reaches a node, that means that it has been classified by the classifier in the parent node, as belonging to some class (say *Yes*). Therefore, to build the classifier at that node, a training set fulfilling the previous constraints (cases classified as *Yes* for the parent node) is needed. The best approach would be to perform a leave-one-out validation over the training set, in order to have an accurate estimation of the cases that the classifier assigns to class *Yes*. But, as this is quite time-consuming, in our experiments we use a n-fold cross validation [14]; in order to study the behaviour of Smart Bayes with different values for n, we have experimented with the values from $n = 2$ to $n = 9$ as shown in section 4.

The following example will help to understand the construction process. Given the hierarchy in figure 1, let

Table 1: Accuracy of Naive Bayes single classifier and Smart Bayes.

Database	NB	SB 2	SB 3	SB 4	SB 5	SB 6	SB 7	SB 8	SB 9	Worst	Best
Banding	78.57	**71.41**	70.18	70.54	68.08	*70.16*	70.96	69.71	70.94	↓70.16	↓71.41
Breast	96.14	*92.99*	**93.56**	93.42	93.42	93.42	**93.56**	93.28	**93.56**	↓92.99	↓93.56
Cars1	68.84	*72.24*	74.04	**74.78**	74.03	74.28	72.76	74.03	72.49	↑72.24	↑74.78
Chess	87.64	91.65	91.52	**91.68**	*91.49*	**91.68**	91.58	*91.49*	91.58	↑91.49	↑91.68
Churn	88.42	*89.50*	**89.70**	89.56	89.62	89.54	89.62	89.52	89.58	↑89.50	↑89.70
Cleve	83.51	81.54	*80.86*	80.88	*80.86*	**82.84**	81.86	*80.86*	82.53	↓80.86	↓82.84
Corral	84.42	*81.28*	82.05	**84.42**	83.59	82.05	83.59	83.59	83.59	↓81.28	=84.42
Crx	77.68	*79.85*	80.29	80.00	80.00	80.29	**80.58**	80.14	80.14	↑79.85	↑80.58
Diabetes	75.66	**74.36**	73.06	*72.66*	73.32	73.57	73.44	73.45	73.57	↓72.66	↓74.36
Flare	80.67	*81.89*	82.26	82.73	82.55	82.45	82.64	82.73	**82.92**	↑81.89	↑82.92
Hepatitis	83.88	85.87	*84.62*	*84.62*	85.92	85.92	85.87	**87.21**	86.54	↑84.62	↑87.21
Horse-colic	78.49	81.23	**83.42**	81.24	*80.97*	81.78	81.50	81.51	81.25	↑80.97	↑83.42
Lenses	63.33	63.33	63.33	63.33	63.33	63.33	63.33	63.33	63.33	=63.33	=63.33
Sonar	68.79	*76.02*	77.90	77.90	77.90	77.95	76.48	**78.45**	76.48	↑76.02	↑78.45
Tic-tac-toe	69.72	*71.08*	72.33	**72.96**	72.33	72.33	72.54	72.33	72.02	↑71.08	↑72.96
Vehicle	45.41	*60.99*	61.24	63.02	62.41	62.65	62.89	**63.60**	63.12	↑60.99	↑63.60
Vote	90.34	*94.01*	94.93	94.94	94.93	94.71	94.70	**95.40**	**95.40**	↑94.01	↑95.40
Vote1	87.33	89.88	89.64	**90.11**	**90.11**	*89.41*	**90.11**	**90.11**	*89.41*	↑89.41	↑90.11
Wine	**97.19**	*96.63*	*96.63*	**97.19**	**97.19**	**97.19**	**97.19**	**97.19**	**97.19**	↓96.63	=97.19
Zoo	87.09	87.09	87.09	87.09	87.09	87.09	87.09	87.09	87.09	=87.09	=87.09

us suppose that we are working in a two-class problem, being the two classes *Yes, No*. It is necessary to build the classifier model for each of the nodes in the tree that forms the hierarchy. Let us name *Examples.data* the datafile where all the available instances corresponding to our classification problem are collected. The procedure is as follows:

- The classifier in the root node of the hierarchy is constructed using all the cases in *Examples.data*; let us name the classifier model *NBTotal*.
- The classifier in every branch just below the root is constructed in the following way:
  - Get the database which was used to build the classifier at the root node.
  - Divide the database into *n* folds of the same size.
  - For each fold,
    classify each case in the fold using the Naive Bayes induced by the remaining *n-1* folds.
  - If *class of case* = *yes*, add it to *Examplesyes.data*
    else add it to *Examplesno.data*.
  - If the node is reached from the root node through a *Yes* edge, then use *Examplesyes.data* to build the model associated at that node (*NBYes*). If it is not (that is, if the node is reached through a *No* edge), use *Examplesno.data* to build the model (*NBNo*).

4 Experimental Setup and Results

In order to evaluate the performance of the proposed approach, we have performed an experiment over 20 databases from the UCI repository [15]. In table 2 the characteristics of these databases are drawn. The number of cases ranges from 24 to 5000, the number of attributes from 4 to 59 and the number of classes from 2 to 7, so a wide variety of problems are represented in these databases.

In our experimental setup we have made use of MLC++ libraries [16] to implement Naive Bayes.

Let us remember that in the process of construction of this two-level classifier it is necessary to choose a number of folds in order to build the databases below the root node. In our experiments we have worked with *n* ranging from 2 to 9. The estimation of the accuracy of our proposed hierarchy has been obtained by means of a ten-fold cross validation.

In table 1 is shown the performance of Naive Bayes over the databases we have worked with, along with the performance of Smart Bayes with folds in the range $(2..9)$. *NB* stands for Naive Bayes and *SB n* stands for Smart Bayes with *n* folds. The column labelled *Best* is the best performance among all the number of folds partitions we have considered, while the column labelled *Worst* is the worst performance among all the folds partitions. An upward arrow means that the performance is better than Naive Bayes, while a downward arrow means the opposite.

Table 2: Characteristics of the databases used in the experimental setup.

Database	#Cases	#Attributes	#Classes
Banding	238	29	2
Breast	699	10	2
Cars1	392	7	3
Chess	3196	36	2
Churn	5000	20	2
Cleve	303	14	2
Corral	129	6	2
Crx	690	15	2
Diabetes	768	8	2
Flare	1066	10	2
Hepatitis	155	19	2
Horse-colic	368	28	2
Lenses	24	4	3
Sonar	208	59	2
Tic-tac-toe	958	9	2
Vehicle	846	18	4
Vote	435	16	2
Vote1	435	15	2
Wine	178	13	3
Zoo	101	18	7

Table 3: Summary of the results of table 1.

NB versus SB	Wins	Ties	Losses
Best fold partition	12	4	4
Worst fold partition	12	2	6

Let us note that, taking into account even the worst partition, Smart Bayes outperforms Naive Bayes in 12 out of 20 databases, tying twice and losing in 6 databases. When we choose the best partitions these figures raise to 12 wins, 4 ties and 4 losses. From these numbers we can conclude that whatever the number of folds partitions we choose the chances of Smart Bayes outperforming Naive Bayes are much greater than the opposite. These results are summarised in table 3.

So far we have shown that choosing at random the value of the number of folds (n), even in the worst case the behaviour of Smart Bayes is better than Naive Bayes. Anyway, it is always desirable to know for which value of n our chances of obtaining the best results are the highest. In this way, as n is the only parameter of Smart Bayes, this process can be seen as parameter tuning. Thus, to achieve the goal of selecting the best number of folds, we have constructed the table 4, in which the cell in row i and column j contains two numbers, being the first one the number of times (over the 20 databases) Smart Bayes with i folds outperforms Smart Bayes with j folds, and the second one the number of times the result is the opposite. This sum is not 160 (20 databases X 8 number of folds) because there are some ties.

As it was expected, the best performance is achieved with a high number of folds. Performances are similar to $n = 7,8$ or 9. Let us note that for $n = 4$ the performance is very good too. The best result is achieved with $n = 7$. The number of wins is the same than with $n = 9$, but the number of losses is one less; in the particular match $n = 7$ wins $n = 9$ for $(7,6)$.

Once the parameter n is selected, the comparison of individual classifiers can be done. As shown in table 5, the hierarchy constructed by using 7 fold cross-validation in order to obtain the low-level classifier, outperforms standard Naive Bayes classifier in 12 out of 20 databases, losing in four of them. If the mean of the 20 accuracies is used (only for confirmation purposes, as it is not considered a remarkable value for the authors), a slight increase can be seen also, from the 79.66% of the Naive Bayes to the 81.61% mean obtained by the presented new approach.

Table 5: Comparison between Naive Bayes and Smart Bayes for $n = 7$.

Database	NB	SB (n=7)	SB - NB
Banding	78.57	70.96	-7.61
Breast	96.14	93.56	-2.58
Cars1	68.84	72.76	+3.92
Chess	87.64	91.58	+3.94
Churn	88.42	89.62	+1.20
Cleve	83.51	81.86	-1.65
Corral	84.42	83.59	-0.83
Crx	77.68	80.58	+2.90
Diabetes	75.66	73.44	-2.22
Flare	80.67	82.64	+1.97
Hepatitis	83.88	85.87	+1.99
Horse-colic	78.49	81.50	+3.01
Lenses	63.33	63.33	0.00
Sonar	68.79	76.48	+7.69
Tic-tac-toe	69.72	72.54	+2.82
Vehicle	45.41	62.89	+17.48
Vote	90.34	94.70	+4.36
Vote1	87.33	90.11	+2.78
Wine	97.19	97.19	0.00
Zoo	87.09	87.09	0.00
Mean	79.66	81.61	+1.96

From these figures we conclude that for the presented experiment the best election is $n = 7$.

5 Conclusions and Further Work

In this paper we have presented a new way to combine several Naive Bayes classifiers in order to improve the performance of a single Naive Bayes. The method has been implemented and tested over 20 databases from the UCI repository.

It has been shown that over the databases tested the new method outperforms single Naive Bayes in a significant number of them.

Table 4: Results of matches between Smart Bayes with *i* folds and Smart Bayes with *j* folds.

	SB 2	**SB 3**	**SB 4**	**SB 5**	**SB 6**	**SB 7**	**SB 8**	**SB 9**	**Total**
SB 2	(0, 0)	(6, 11)	(4, 14)	(5, 13)	(3, 15)	(3, 14)	(4,14)	(4,14)	(29, 95)
SB 3	(11, 6)	(0, 0)	(5, 11)	(7, 7)	(6, 9)	(5, 12)	(7, 9)	(7, 10)	(48, 64)
SB 4	(14, 4)	(11, 5)	(0, 0)	(10, 3)	(9, 6)	(8, 8)	(8, 7)	(6, 11)	(66, 44)
SB 5	(13, 5)	(7, 7)	(3, 10)	(0, 0)	(5, 9)	(4, 10)	(2, 9)	(5, 11)	(39, 61)
SB 6	(15, 3)	(9, 6)	(6, 9)	(9, 5)	(0, 0)	(8, 9)	(9, 7)	(7, 8)	(63, 47)
SB 7	(14, 3)	(12, 5)	(8, 8)	(10, 4)	(9, 8)	(0, 0)	(7, 8)	(7, 6)	(67, 42)
SB 8	(14, 4)	(9, 7)	(7, 8)	(9, 2)	(7, 9)	(8, 7)	(0, 0)	(7, 7)	(61, 44)
SB 9	(14, 4)	(10, 7)	(11, 6)	(11, 5)	(8, 7)	(6, 7)	(7, 7)	(0, 0)	(67, 43)

As future work it would be interesting to extend the current two-level model to a more general model with a higher number of levels. In this work we have worked with Naive Bayes classifiers, but other paradigms, as decision trees or neural networks could be used too.

6 Acknowledgements

This work has been supported by the Ministerio de Ciencia y Tecnología under grant TSI2005-00390 and by the AMIGUNE ETORTEK Project.

7 References

[1] T. Mitchell, *Machine Learning*, McGraw-Hill, New York, USA (1997).

[2] M. Minsky, "Steps toward artificial intelligence", *Proceedings of the Institute of Radio Engineers*, 49, pp 8-30 (1961).

[3] T.K. Ho, J.J. Hull, and S.N. Srihari, "Decision combination in multiple classifier systems", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(1), pp 66-75 (1994).

[4] L. Breiman, "Bagging predictors", *Machine Learning*, 24(2), pp 123-140 (1996).

[5] Y. Freund and R.E. Schapire, "A short introduction to boosting", *Journal of Japanese Society for Artificial Intelligence*, 14(5), pp 771-780 (1999).

[6] D.H. Wolpert, "Stacked generalization", *Neural Networks*, 5, pp 241-259 (1992).

[7] J.M. Gama, *Combining Classification Algorithms*, PhD thesis, Universidade do Porto (2000).

[8] V. Gunes, M. Ménard, P. Loonis, and S. Petit-Renaud, "Combination, cooperation and selection of classifiers: a state of the art", *International Journal of Pattern Recognition*, 17, pp 1303-1324 (2003).

[9] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid", *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Portland, USA, pp 202-207 (1996).

[10] J.M. Martínez-Otzeta, B. Sierra, E. Lazkano, M. Ardaiz, and E. Jauregi, "Edited naive bayes", *Ibero-American Journal of Artificial Intelligence*, 10(31), pp 63-69 (2006).

[11] I. Kononenko, "Semi-naive bayesian classifier", *European Working Session on Learning-EWSL91*, Porto, Portugal, pp 206-219 (1991).

[12] M. Pazzani, "Constructive induction of cartesian product attributes", *Proc. Conf. Information, Statistics and Induction in Science (ISIS96)*, Melbourne, Australia, pp 66-77 (1996).

[13] D.D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval", *Proceedings of the 10th European Conference on Machine Learning*, Chemnitz, Germany, Springer-Verlag, pp 4-15 (1998).

[14] M. Stone, "Cross-validation choice and assessment of statistical procedures", *Journal of the Royal Statistical Society*, 36, pp 111-147 (1974).

[15] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz, "UCI repository of machine learning databases", http://www.ics.uci.edu/∼mlearn/MLRepository.html, visited on 24/07/2006.

[16] R. Kohavi, D. Sommerfield, and J. Dougherty, "Data mining using MLC++ : A machine learning library in C++", *International Journal on Artificial Intelligence Tools*, 6(4), pp 537-566 (1997).