Pre-publication draft of a paper which appeared in the Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99), pages 493-502. Morgan Kaufmann

Lazy Bayesian Rules: A Lazy Semi-Naive Bayesian Learning Technique Competitive to Boosting Decision Trees

Zijian Zheng, Geoffrey I. Webb, Kai Ming Ting
School of Computing and Mathematics
Deakin University
Victoria 3217 Australia
{zijian,webb,kmting}@deakin.edu.au

Abstract

Lbr is a lazy semi-naive Bayesian classifier learning technique, designed to alleviate the attribute interdependence problem of naive Bayesian classification. To classify a test example, it creates a conjunctive rule that selects a most appropriate subset of training examples and induces a local naive Bayesian classifier using this subset. Lbr can significantly improve the performance of the naive Bayesian classifier. A bias and variance analysis of Lbr reveals that it significantly reduces the bias of naive Bayesian classification at a cost of a slight increase in variance. It is interesting to compare this lazy technique with boosting and bagging, two well-known state-of-the-art non-lazy learning techniques. Empirical comparison of Lbr with boosting decision trees on discrete valued data shows that Lbr has, on average, significantly lower variance and higher bias. As a result of the interaction of these effects, the average prediction error of Lbr over a range of learning tasks is at a level directly comparable to boosting. Lbr provides a very competitive discrete valued learning technique where error minimization is the primary concern. It is very efficient when a single classifier is to be applied to classify few cases, such as in a typical incremental learning scenario.

1 INTRODUCTION

Among the commonly used and well studied classifier learning techniques, the simple and computationally efficient naive Bayesian classifier (Duda & Hart, 1973; Kononenko, 1990; Langley & Sage, 1994) has been shown in many domains to be surprisingly accurate compared to alternatives including decision tree learning, rule learning, neural networks, and instance-based learning (Cestnik, Kononenko, & Bratko, 1987; Kononenko, 1990; Domingos & Pazzani, 1996). Naive Bayesian classification is based on Bayes' theorem and an assumption that all attributes are mutually independent within each class. This technique predicts that a test example with a vector of attribute values $V = \langle v_1, v_2, \cdots, v_n \rangle$ belongs to the class C_i that maximizes the posterior probability $P(C_i|V) \propto P(C_i) \prod_j P(v_j|C_i)$, where $P(C_i)$ is the prior probability of class C_i, and $P(v_j|C_i)$ is the conditional probability that v_j occurs in class C_i examples. The prior probability and conditional probabilities are usually directly estimated from frequency counts over a given training set.

The attribute independence assumption makes the application of Bayes' theorem to classification practical in many domains. Domingos & Pazzani (1996) argue that the naive Bayesian classifier is optimal even when the independence assumption is violated, so long as the ranks of the conditional probabilities of classes given an example are correct. However, previous research has shown that semi-naive techniques and Bayesian networks that explicitly adjust the naive strategy to allow for violations of the independence assumption, can improve upon the prediction accuracy of the naive Bayesian classifier in many domains (Friedman & Goldszmidt, 1996; Kohavi, 1996; Kononenko, 1991; Langley, 1993; Langley & Sage, 1994; Pazzani, 1998; Sahami, 1996; Singh, & Provan, 1996; Webb & Pazzani, 1998). This suggests that the ranks of conditional probabilities are frequently not correct.

To tackle this problem, the naive Bayesian tree learner, NBTree (Kohavi, 1996), combines naive Bayesian classification and decision tree learning (Breiman, Friedman, Olshen & Stone, 1984; Quinlan, 1993). It uses a tree structure to split the instance space into sub-spaces defined by the paths of the tree. It generates one naive Bayesian classifier in each sub-space. The decision nodes of these trees contain the univariate tests of conventional decision trees. Each leaf of a naive Bayesian tree contains a local naive Bayesian classifier that does not consider attributes involved in tests on the path leading to the leaf, and is used to classify examples that reach this leaf. It has been shown that NBTree frequently achieves higher accuracy than either a naive Bayesian classifier or a decision tree learner (Kohavi, 1996).

Although NBTree can alleviate the attribute interdependence problem of naive Bayesian classification to some extent, NBTree suffers from the replication and fragmentation problems (Pagallo & Haussler, 1990) as well as the small disjunct problem (Holte, Acker, & Porter, 1989; Ting, 1994) due to the tree structure. To solve these problems and further improve the performance of naive Bayesian classification, we previously proposed a lazy Bayesian rule (Lbr) learning technique (Zheng & Webb, 1998). LBR can be thought of as applying lazy learning techniques (Aha, 1997) to naive Bayesian tree induction. At classification time, for each test example, Lbr builds a most appropriate rule with a conjunction of conditions as its antecedent and a local naive Bayesian classifier as its consequent. Our experimental results (Zheng & Webb, 1998) have shown that Lbr is, on average, more accurate than a naive Bayesian classifier, C4.5 (Quinlan, 1993), our implementation (Zheng & Webb, 1998) of NBTree (Kohavi, 1996), a constructive Bayesian classifier that eliminates attributes and constructs new attributes using Cartesian products of existing nominal attributes (Pazzani, 1998), a selective naive Bayesian classifier (Langley & Sage, 1994), and a lazy decision tree learning algorithm (Friedman, Kohavi & Yun, 1996).

While these comparisons with directly related algorithms reveal good performance, it is interesting to hold a new contender up against the state-of-the-art. Recent studies have shown that committee learning techniques, notably boosting and bagging, achieve very low prediction error, especially for decision tree learning (Bauer & Kohavi, 1999; Breiman, 1996; Freund & Schapire, 1997; Quinlan, 1996; Schapire, Freund, Bartlett & Lee, 1997). These techniques generate multiple models to form a committee by repeated application of a single base learning algorithm. At classification time, the committee members vote to make the final decision. Here, we compare Lbr with boosting and bagging decision trees, as previous research showed that they achieve very high average accuracy (Bauer & Kohavi, 1999; Quinlan, 1996). It is interesting to note that boosting naive Bayes is not very effective (Ting & Zheng, 1999) as boosting requires that the base learner be sensitive to variations in the training set while naive Bayes is very stable across training samples, resulting in low variance but high bias.

The following section describes the algorithms used in this paper, Lbr, AdaBoost, and bagging. Section 3 contains empirical studies. We first carry out extensions to our previous analysis of Lbr (Zheng & Webb, 1998), and then compare Lbr to AdaBoost and bagging. We will show that Lbr can significantly reduce the bias of naive Bayesian classification at a cost of a slight increase in variance. Furthermore, Lbr has, on average, significantly lower variance and higher bias than AdaBoost with decision trees. As a result of the interaction of these effects, the average prediction error of Lbr over a range of learning tasks is at a level directly comparable to AdaBoost. Finally, Section 4 summarizes our findings.

2 THE LBR, ADABOOST, AND BAGGING ALGORITHMS

We describe the Lbr algorithm in the following sub-section. Sub-sections 2.2 and 2.3 present short descriptions of AdaBoost and bagging respectively. Readers are referred to elsewhere for more detailed descriptions of these two algorithms (Freund & Schapire, 1997; Breiman, 1996). The empirical studies in this paper also involve two other algorithms: a naive Bayesian classifier, NB, and a decision tree learner, C4.5. NB was described briefly in Section 1. More detail is available elsewhere (Duda & Hart, 1973; Kononenko, 1990; Langley & Sage, 1994). In our implementation of NB, when the probability of an attribute value conditional on a class is estimated from a training set, the m-estimate (Cestnik, 1990) with m = 2 is used. When the probability of a class is estimated, the Laplace estimate (Cestnik, 1990) is used. We assume that readers are familiar with C4.5, since it is widely used. A complete description of C4.5 is given by Quinlan (1993).

2.1 LBR

Lbr uses lazy learning to alleviate the attribute interdependence problem of naive Bayesian classification, as well as to avoid the replication, fragmentation, and small disjunct problems from which NBTree (Kohavi, 1996) may suffer. It retains all training examples until classification time. Before classifying a test example, Lbr generates a rule (called a *Bayesian rule*) that is most appropriate to the test example. This contrasts with creating a model at training time, such as NBTree's single tree, that is, on average, most appropriate to all examples. The antecedent of a Bayesian rule is a conjunction of attribute-value pairs (conditions) each of the form 'attribute = value'. The current version of Lbr can only deal directly with discrete valued attributes. Numeric attributes are discretized as a pre-process. The consequent of a Bayesian rule is a local naive Bayesian classifier created from *local training examples* that satisfy the antecedent of the rule.

Table 1: The Lazy Bayesian Rule learning algorithm

```
LBR($Att, D, E_{test}$)
  INPUT: $Att$: a set of attributes,
         $D$: a set of training cases described
              using $Att$ and classes,
         $E_{test}$: a test case described using $Att$.
  OUTPUT: a predicted class for $E_{test}$.
LocalNB = a NB classifier trained using $Att$ on $D$
Errors = errors of LocalNB estimated using N-CV on $D$
Cond = true
REPEAT
  $TempErrors_{best}$ = the number of cases in $D + 1$
  FOR each attribute $A$ in $Att$ whose value $v_A$ on
      $E_{test}$ is not missing DO
    $D_{subset}$ = cases in $D$ with $A = v_A$
    TempNB = a NB classifier trained using
             $Att - \{A\}$ on $D_{subset}$
    TempErrors = errors of TempNB estimated using
                 N-CV on $D_{subset}$ + the portion
                 of Errors in $D - D_{subset}$
    IF ((TempErrors < $TempErrors_{best}$) AND
       (TempErrors is significantly lower than Errors))
    THEN
      $TempNB_{best}$ = TempNB
      $TempErrors_{best}$ = TempErrors
      $A_{best} = A$
  IF (an $A_{best}$ is found)
  THEN
    Cond = Cond $\wedge$ ($A_{best} = v_{A_\mathrm{best}}$)
    LocalNB = $TempNB_{best}$
    $D = D_{subset}$ corresponding to $A_{best}$
    $Att = Att - \{A_{best}\}$
    Errors = errors of LocalNB estimated using
             N-CV on $D$
UNTIL (no $A_{best}$ is found)
classify $E_{test}$ using LocalNB
RETURN the class
```

This local naive Bayesian classifier uses only those attributes that do not appear in the rule's antecedent.

Table 1 outlines the LBR algorithm. During the generation of a Bayesian rule, only attribute-value pairs that utilize attribute values of the test example are considered. The objective is to grow the antecedent of a Bayesian rule that ultimately decreases the errors of the local naive Bayesian classifier in the consequent of the rule. The antecedent of the Bayesian rule defines a sub-space of the instance space to which the test example belongs, selecting a subset of the available training instances. All instances in this sub-space have the attribute-values specified in the antecedent. In consequence, these attributes are removed from the local naive Bayesian classifier.

For each test example, LBR uses a greedy search to generate a Bayesian rule with an antecedent that matches the test example. The growth of the rule starts from a special Bayesian rule whose antecedent is *true*. The rule's local naive Bayesian classifier is trained on the entire training set using all attributes. This classifier is identical to a conventional naive Bayesian classifier. At each step of the greedy search, LBR tries to select and add, to the antecedent of the current Bayesian rule, the best attribute-value pair ($A_i = v_{A_i}$) where v_{A_i} is the value of A_i on the test example. During this search, those attributes are ignored that are already included in the antecedent of the current rule or whose value on the test example is missing. The objective is to determine whether including this attribute-value pair into the Bayesian rule can significantly improve the estimated accuracy. This search method is similar to forward selection which has been used for relevant attribute subset selection (John, Kohavi & Pfleger, 1994).

The utility of adding each possible $A_i = v_{A_i}$ to the current rule is evaluated as follows. A subset of examples D_{subset} that satisfies $A_i = v_{A_i}$ is identified from the current local training set D, and is used to train a temporary naive Bayesian classifier using all attributes other than A_i that do not occur in the antecedent of the current Bayesian rule. The error of this classifier on D_{subset} is estimated by N-fold cross-validation (N-CV).[1] This estimate together with an N-CV estimate of the errors of the local naive Bayesian classifier of the current Bayesian rule on $D - D_{subset}$ are used to assess the value of adding $A_i = v_{A_i}$ to the current Bayesian rule.[2] If this measure is lower than the estimated errors of the local naive Bayesian classifier on D at a significance level better than 0.05 using a one-tailed pairwise sign-test, this attribute-value pair becomes a candidate condition to be added to the current Bayesian rule. The sign-test is used to control the likelihood of adding conditions that reduce error by chance. After evaluating all possible conditions, the candidate condition with the lowest error estimate is added to the antecedent of the current Bayesian rule.

Training cases that do not satisfy the antecedent of the rule are then discarded, and the above process repeated. This continues until no more candidate conditions are found. This happens when no damaging attribute inter-dependencies exist for the local naive Bayesian classifier, or the local training set is too small

[1]We use N-CV because N-CV errors are more reliable estimates of true errors than re-substitution errors (Breiman, Friedman, Olshen & Stone, 1984) and N-CV on a naive Bayesian classifier is computationally efficient.

[2]Here we evaluate the utility of each attribute-value pair on the whole local training set, D. Since different attribute-value pairs cover different subsets of D, estimated errors for different attribute-value pairs on their corresponding subsets of training examples, D_{subset}, are not comparable.

to further reduce the instance sub-space by specializing the antecedent of the Bayesian rule. In such cases, further growing the Bayesian rule would not significantly reduce its errors. Finally, the local naive Bayesian classifier of this Bayesian rule is used to classify the test example under consideration.

2.2 ADABOOST

Boosting (Freund & Schapire, 1997) is a general framework for improving a base learning algorithm. During training, it sequentially builds different classifiers to form a committee by adaptively modifying the distribution of the training set based on the performance of previously created classifiers. The objective is to make the generation of the next classifier concentrate on the training examples misclassified by the previous classifiers.

Our implementation of AdaBoost uses C4.5 (Quinlan, 1993) as the base learner. Given a training set D consisting of m instances and a specified number of trials T, AdaBoost builds T pruned trees over T trials by repeatedly invoking C4.5. Let $w_t(x)$ denotes the weight of instance x in D at trial t. At the first trial, each instance has weight 1, that is, $w_1(x) = 1$ for each x. At trial t, decision tree H_t is built using D under the distribution w_t. The training error ϵ_t of H_t is then calculated by summing up the weights of the instances that H_t misclassifies and dividing by m. For the next trial, the weight $w_{t+1}(x)$ is set to $w_t(x)/2\epsilon_t$ if H_t incorrectly classifies x, and $w_t(x)/2(1-\epsilon_t)$ otherwise. If $w_{t+1}(x) < 10^{-8}$, set it to 10^{-8} to solve the numerical underflow problem (Bauer & Kohavi, 1999). These weights are then renormalized so that they sum to m.

Following the example of Bauer & Kohavi (1999), when ϵ_t is greater than 0.5, $w_t(x)$ is re-initialized to allow the boosting process to continue. Following Webb (forthcoming), we also reweight examples to continue boosting if ϵ_t reaches zero. This allows the possibility that more than one committee member with zero error will be derived, enabling these to vote against each other (with very large but finite weight) and giving other committee members a casting vote in case of a draw between committee members with zero resubstitution error. We employ Webb's (forthcoming) reweighting scheme in these contexts, setting $w_t(x)$ to random values from the continuous Poisson distribution, generated by:

$$Poisson() = -log\left(\frac{Random(1\ldots999)}{1000}\right), \qquad (1)$$

where $Random(min\ldots max)$ returns a random integer value between min and max inclusive. After values are assigned using this random process, the vector of weights is normalized to sum to the size of the training set. Then, the boosting process continues to build the next tree. Note that any tree with $\epsilon_t > 0.5$ is discarded and this trial is repeated with the re-initialized instance weights, while a tree with $\epsilon_t = 0$ is accepted by the committee.

At the classification stage, for each test example, all decision trees in the committee perform weighted voting to make the prediction. The vote of decision tree H_t is worth $\log(1/\beta_t)$ units, where $\beta_t = \epsilon_t/(1-\epsilon_t)$.

This implementation of AdaBoost is very similar to that of Bauer & Kohavi (1999). The only difference is that Bauer & Kohavi (1999) halt induction and treat H_t as having infinite weight if $\epsilon_t = 0$, and use the bootstrap sampling method to re-initialize the weights when $\epsilon_t > 0.5$, whereas our implementation sets $\beta_t = 10^{-10}$ when $\epsilon_t = 0$, and re-initializes the weights using the continuous Poisson distribution in both situations. The continuous Poisson distribution is chosen because our experiments show that this weight re-initialization method has a small advantage over the bootstrap sampling method.

2.3 BAGGING

Like boosting, bagging (Breiman, 1996) generates committees by changing the distribution of the training set. However, for bagging, the change of the instance distribution is stochastic. The primary idea is to generate a committee of classifiers with each induced from a bootstrap sample of the original training set. Given a committee size T and a training set D consisting of m instances, bagging generates T bootstrap samples with each being created by uniformly sampling m instances from D with replacement. It then applies C4.5 to build one decision tree from each bootstrap sample. All decision trees in the committee vote with equal weight to classify a test example.

3 EVALUATION

In this section, we analyze the behavior of LBR using the bias and variance decomposition to gain insight into how it improves the performance of its base learning algorithm. Then we compare LBR with boosting and bagging. Section 3.1 describes the bias and variance decomposition. Section 3.2 illustrates the experimental domains and methods used in this paper. The following two subsections present the results of the analysis with LBR, and the comparison with boosting and bagging. Section 3.5 gives a comparison in terms of compute time.

3.1 BIAS AND VARIANCE

Bias and *variance* analyses can provide useful insights into the generalization performance of a learning algorithm. Bias is a measure of error due to the central tendency and variance is a measure of error due to disagreements between the classifiers formed by an algorithm across a distribution of training sets. Among several definitions of bias and variance, in this study we adopt Kohavi & Wolpert's (1996) definitions that decompose error into intrinsic noise (optimal Bayes error), squared bias, and variance. Due to the difficulty of estimating the intrinsic noise in practical experiments with real domains, we follow Kohavi & Wolpert's (1996) strategy of generating a bias term that includes both intrinsic noise and squared bias. The values are calculated directly from the performance of each classifier on each test case in the experiments described below.

3.2 EXPERIMENTAL DOMAINS AND METHODS

This study uses the same twenty-nine natural domains used in previous studies of LBR (Zheng & Webb, 1998). These cover a wide variety of domains from the UCI machine learning repository (Blake, Keogh & Merz, 1998). This test suite includes all the twenty-eight domains used by Domingos & Pazzani (1996) for studying naive Bayesian classification, as well as the Tic-Tac-Toe domain. The last domain was chosen for previous research because it contains inter-dependent attributes and its target concept is known.

We employ the bias/variance estimation process developed by Webb (forthcoming). For each domain ten three-fold cross-validations (Breiman, Friedman, Olshen & Stone, 1984) are carried out for each algorithm. All the algorithms are run with their default option settings on the same training and test set partitions in every domain. Only the error rate on the test set is reported. An error (or bias, or variance) rate reported in the following subsections is an average over the 30 runs for an algorithm.

Since the current implementation of LBR and NB can only deal with discrete valued attributes, numeric attributes are discretized as a pre-process in the experiments using an entropy-based discretization method (Fayyad & Irani, 1993) for all learning algorithms. In other words, the algorithms are being compared on performance when learning from discrete valued data. For each pair of training set and test set, both training set and test set are discretized by using cut points found from the training set alone.

3.3 BIAS AND VARIANCE ANALYSIS FOR LBR

LBR is designed to improve the performance of the naive Bayesian classifier NB. A bias and variance decomposition of LBR's performance provides interesting insights into how it achieves this effect.

Table 2: Error rates (%) of LBR, NB, C4.5, AdaBoost, and bagging

Domain	LBR	NB	C4.5	AdaBoost	bagging
Annealing	2.8	3.0	10.4	7.8	8.9
Audiology	28.0	28.4	23.7	19.4	21.5
Breast(W)	2.6	2.6	5.1	3.8	4.4
KR-KP	2.7	12.5	0.8	0.5	0.7
Credit(A)	14.4	14.5	14.5	15.9	14.3
Echocar.	30.8	30.8	36.2	33.9	35.1
Glass	34.1	34.0	35.8	33.1	32.8
Heart(C)	17.6	17.6	24.0	22.3	20.7
Hepatitis	14.7	14.7	21.0	17.7	18.9
Horse	20.3	21.0	17.0	21.3	16.8
HouseVote	6.7	9.9	5.1	4.9	4.6
Hypo	1.5	1.8	1.3	1.2	1.2
Iris	6.6	6.6	7.5	7.5	7.1
Labor	10.7	10.7	19.5	18.2	18.6
LED 24	39.7	39.5	36.9	36.4	31.8
Liver	36.2	36.2	35.7	36.0	36.0
LungC.	49.7	49.4	59.4	55.0	61.3
Lympho	18.5	18.3	23.6	17.7	20.7
Pima	25.1	25.2	25.5	27.0	25.0
Postop.	35.9	36.0	29.9	39.7	31.1
P.Tumor	52.4	52.4	61.0	58.4	57.8
Promoters	10.7	10.7	26.3	8.4	17.2
Solar	17.1	19.1	15.7	17.0	15.9
Sonar	25.4	26.5	32.1	22.1	24.9
Soybean	7.0	9.8	10.3	6.9	8.2
Splice	4.3	4.5	6.5	4.7	5.5
TicTacToe	19.1	29.5	16.5	1.3	9.3
Wine	2.5	2.5	11.8	4.7	8.9
Zoology	6.2	6.2	8.5	6.2	8.4
mean	18.7	19.8	21.4	18.9	19.6
e.r. mean		1.12	1.21	.93	1.07
w/t/l		4/11/14	10/0/19	13/1/15	14/0/15
p. of wtl		.0304	.1360	.8506	1.0000

e.r. mean: geometric mean of error ratios.

Table 2 presents the error rates of LBR and NB (results for C4.5, AdaBoost, and bagging are discussed in the following section). The penultimate row shows the geometric mean error ratio of NB over LBR. Geometric rather than arithmetic mean is used as it is more appropriate for averaging ratio values. The last row in the table shows the numbers of wins, ties, and losses between the error rates of NB against those of LBR in the 29 domains, and the significance level of a two-tailed pairwise sign-test on this win/tie/loss outcome. We consider the comparison significant if the sign-test reveals a probability of less than 0.05 that the observed result or more extreme should be obtained by chance. To illustrate the impact to bias and variance of applying LBR's semi-naive Bayesian approach in place of NB's naive Bayesian approach, Figure 1 presents a bar chart that shows for each domain the decrease in error due to bias reduction and increase in error due

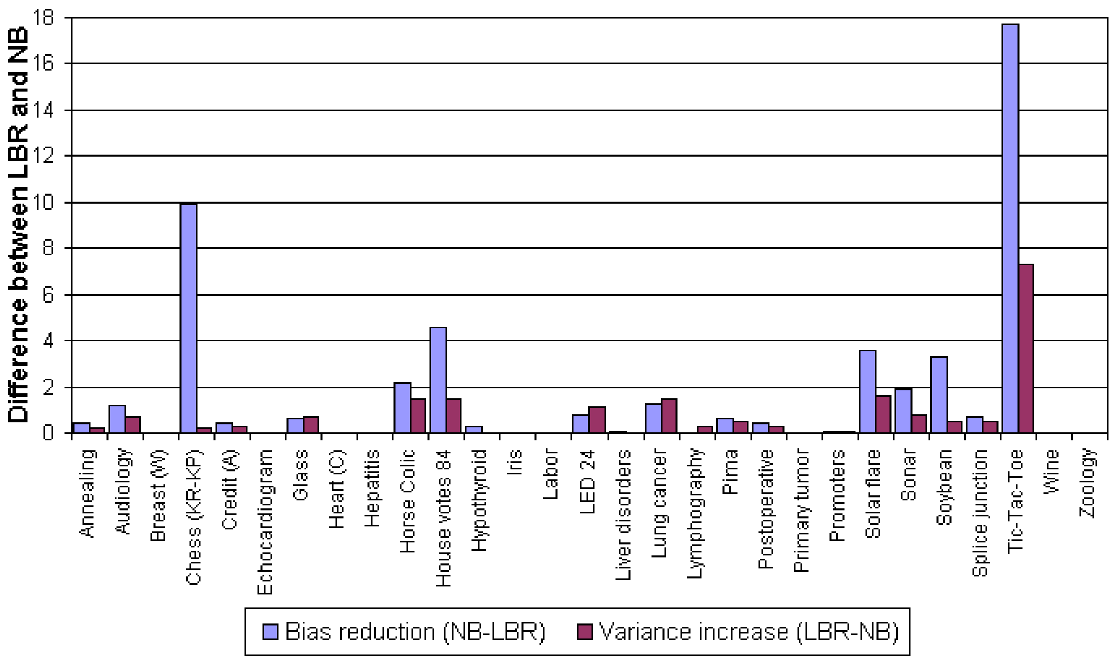

Figure 1: Bias and variance comparison between Lbr and NB

to increased variance. Figure 2 summarizes this result across all domains by displaying the average error of all algorithms decomposed into average bias and variance.

It is apparent that Lbr outperforms NB in terms of average error rate, error ratio, and w/t/l record. The relative error reduction of Lbr over NB is 12%. The sign-test shows that Lbr obtains lower error rates significantly more often than the reverse in comparison to NB.

It is also apparent that the effect of Lbr is to substantially reduce the bias of naive Bayes. Figure 1 shows that Lbr does not obtain higher bias than NB in any of the experimental domains. However, Lbr never reduces the variance of NB. The reason is that introducing rules reduces the instance space and the number of training instances. This is very likely to increase the variance. The bias term strongly dominates the error of NB and hence the bias reduction usually outweighs the variance increase. This accounts for Lbr's lower error in most domains.

3.4 COMPARING LBR WITH C4.5, ADABOOST, AND BAGGING

We turn now to a comparison of Lbr against boosting and bagging decision trees. As C4.5 is the base learning algorithm used in the committee techniques, its performance is also considered.

We use AdaBoost and bagging with 100 trees in our experiments since it has been shown that increasing the committee size usually further improves the performance, especially with AdaBoost (Schapire, Freund, Bartlett & Lee, 1997). The results of C4.5, AdaBoost and bagging are provided in Table 2 and Figure 2.

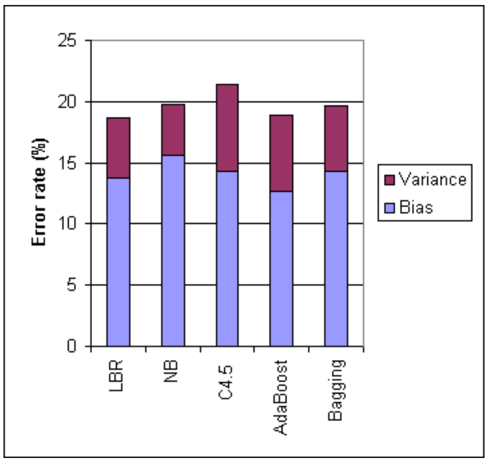

Figure 2: Error rate with bias and variance decomposition

The relative error reduction of Lbr over C4.5 is 21%. Although the sign-test fails to show that Lbr is significantly better than C4.5, Lbr has lower error rates than C4.5 almost twice as often as the reverse. Also, Lbr has lower bias and variance than C4.5 on average.

Note that, despite the discretization of numeric attributes, the results of AdaBoost and bagging reported here are either comparable to or better than earlier published results (Bauer & Kohavi, 1999; Breiman, 1996; Quinlan, 1996) since we use a higher value of T. With reference to their base learning algorithm C4.5, AdaBoost reduces bias from 14.3% to 12.7%, and variance from 7.1% to 6.2%. Bagging has the same bias as C4.5, but reduces variance from 7.1% to 5.3%. Note also that we have repeated the experiments with committees of size 10 and that the results are less favorable to boosting and bagging than the results reported here. For example, the average error rate of AdaBoost with 10 trees is 20.2%, substantially higher than that of AdaBoost with 100 trees (18.9%). With 10 trees, the average error of bagging is 20.3% compared to 19.6% with 100 trees.

Comparing LBR with AdaBoost and bagging, we have the following observations.

- LBR has lower average error than either AdaBoost or bagging. In terms of geometric mean error ratio, AdaBoost has lower error than LBR by 7%. However, when the geometric mean *accuracy* ratio is calculated, it shows that AdaBoost has lower accuracy than LBR by 1%. This seemingly incompatible outcome results from AdaBoost's tendency to better performance where error is low (and hence accuracy is high) while LBR tends to perform well in the reverse context. AdaBoost wins in 13 domains, loses in 15 domains and draws in 1 domain. All this indicates that LBR is very competitive to AdaBoost.

  The geometric mean error ratio indicates the bagging has 7% higher average error than LBR. Bagging wins in 14 domains and loses in 15 domains.
- LBR has lower average variance than either AdaBoost or bagging—4.9% compared to 6.2% for AdaBoost and 5.3% for bagging. The frequency with which LBR achieves lower variance is significant at the 0.05 level with respect to AdaBoost but not bagging.
- LBR has higher average bias than AdaBoost (13.8% versus 12.7%), but lower average bias than bagging (13.8% versus 14.3%). The frequency with which LBR has higher bias is significant at the 0.05 level with respect to AdaBoost but not bagging.

Since C4.5 can deal directly with numeric attributes, it might impede C4.5 to carry out discretization as a pre-process when running C4.5. This also applies for AdaBoost and bagging with C4.5. In an additional experiment, with exactly the same experimental method, C4.5, AdaBoost, and bagging were run without discretization. The average error rates of C4.5, AdaBoost, and bagging without discretization in these 29 domains using the same training and test set partitions are 21.3%, 17.9%, and 18.6% respectively. These are slightly lower than the corresponding values with discretization. The accuracy of C4.5 without discretization is also worse than that of LBR, with 1.18 as the geometric mean error ratio and 11/1/18 as the win/tie/loss record of the former over the latter. AdaBoost without discretization shows a slight advantage over LBR. The win/tie/loss record of AdaBoost without discretization over LBR is 16/1/12 in the 29 domains. This difference is not significant at the level 0.05 using a two-tailed pairwise sign-test ($p = .572$). The geometric mean error ratio of AdaBoost without discretization over LBR is 0.86. While this seems a big advantage to AdaBoost, the geometric mean accuracy ratio of AdaBoost without discretization over LBR is only 1.006, a very slight advantage to AdaBoost. As discussed in the context of AdaBoost with discretization, this results from AdaBoost's tendency to better performance where error is low (and hence accuracy is high) and LBR's tendency to perform well in the reverse context. The geometric mean error ratio and the win/tie/loss record of bagging without discretization over LBR are 1.004 and 14/1/14 respectively, suggesting that bagging even without discretization has no advantage over LBR on average. On the other hand, these results suggest that the performance of LBR might be further improved by carrying out discretization during the generation of Bayesian rules instead of pre-processing or by employing techniques for handling undiscretized numeric values such as those proposed by John & Langley (1995).

3.5 COMPUTATION TIME

In this section, we discuss LBR's computational profile and compare its computation time with that of AdaBoost with 100 decision trees. Since AdaBoost and bagging have similar execution times, the result of the latter is omitted.

Lazy learners have very different computation time profiles to eager learners that generate an explicit model at training time. If we consider the total computation time for learning from a given training set and classifying objects in a given test set, the computation time of an eager learner is strongly dominated by the requirements for learning a model from the training set. The computation associated with applying the model to classify test objects is usually relatively trivial. In particular, the computational profile is relatively insensitive to the number of objects to be classified. The total compute time will differ little irrespective of whether one or 1000 objects are to be classified. In contrast, lazy learners delay computation to classification time and the amount of computation is dependent on the number of objects to be classified.

The major computational overhead in LBR is the evaluation of each potential addition of a condition to the antecedent of the rule being constructed. This requires N-fold cross-validation of the resulting local naive Bayesian classifier. As N-fold cross-validation of a naive Bayesian classifier can be implemented very efficiently, the computational requirements for classifying a single test case are reasonable. However, lazy learning requires that the same process is repeated for each test example, and the cumulative computation for a large test set can be substantial. The computational overhead can be substantially reduced by caching useful information. In the current implementation of LBR, the evaluation function values of attribute-value pairs that have been examined are retained from one test example to the next. This avoids re-calculation of the evaluation function values of the same attribute-value pairs when classifying test examples that appear later in the test set, thus reducing the entire execution time. Our experiment shows that caching this information reduces the execution time of LBR by 96% on average on the 29 datasets used in the experiment. This is because the evaluation of attribute-value pairs for different test examples are often repeated, including repeated generation of identical rules for different test examples. LBR could be made even more efficient by caching further information such as local classifiers and indices for training examples in different stages of the growth of rules. Of course, this would increase memory requirements.

Table 3 shows the execution time (in seconds) of LBR and AdaBoost, running on a Sun UltraSPARC 2 computer. AdaBoost uses more time than LBR in 24 domains. Although, in terms of mean execution time over the 29 domains, LBR is about five time slower than AdaBoost, this is dominated by 5 domains in which AdaBoost is much faster than LBR. In these 5 domains, large numbers of test instances contribute to the long execution in LBR. For example, in the Splice junction domain, the test set size used in the current experiment is 1059. When we change the experiment from three-fold cross-validation to ten-fold cross-validation (the test set size is reduced to 318), the execution time of LBR reduces from 266.30 seconds to 180.39 seconds, while AdaBoost's execution time increases from 40.20 seconds to 46.37 seconds. This suggests that the test set size has a great effect on the execution time of LBR, and has a very little effect on the execution time of AdaBoost.

When interpreting the results in Table 3 it should be noted that they relate to contexts, created by three-fold cross-validation, in which two thirds of the available data are used for training and one third for testing. The results would better favor LBR if the proportion of data used for testing was decreased and better favor AdaBoost if this proportion was increased.

Table 3: Compute time (in seconds) of LBR, and AdaBoost

Domain	AdaBoost	LBR
Annealing	6.74	3.84
Audiology	8.50	21.75
Breast(W)	1.65	0.35
KR-KP	31.10	388.74
Credit(A)	3.67	1.30
Echocar.	0.41	0.02
Glass	1.22	0.16
Heart(C)	1.44	0.20
Hepatitis	1.09	0.09
Horse	4.25	0.77
HouseVote	1.44	0.86
Hypo	13.29	54.14
Iris	0.22	0.01
Labor	0.31	0.01
LED 24	2.98	0.97
Liver	0.64	0.16
LungC.	0.61	0.15
Lympho	1.02	0.12
Pima	2.90	0.64
Postop.	0.48	0.03
P.Tumor	6.43	1.42
Promoters	1.13	0.10
Solar	5.06	2.92
Sonar	2.88	1.30
Soybean	8.95	55.50
Splice	40.20	266.30
TicTacToe	3.52	1.94
Wine	0.54	0.04
Zoology	0.45	0.03
mean	5.28	27.72
e.r. mean		1.34
w/t/l		24/0/5
p. of wtl		.0004

4 CONCLUSIONS

This paper evaluates the Lazy Bayesian Rule learning algorithm. This algorithm seeks to overcome the attribute interdependence problem of the naive Bayesian classifier by forming a customized naive Bayesian classifier for each case to be classified. This customized naive Bayesian classifier is formed by selecting a subset of the available training examples that appear most relevant to classifying the current test example. In practice this often builds a more accurate naive Bayesian classifier for the test example than the naive Bayesian classifier trained using all training examples.

This paper shows that the lazy Bayesian rule induction algorithm is an effective technique for improving upon the naive Bayesian classifier. We show that this effect is achieved by significantly and substantially reducing the bias of naive Bayesian classification, though this is achieved at the cost of a slight increase in variance.

Current implementation constraints, notably that it accepts only discrete attributes and is restricted to antecedent conditions of the form 'attribute = value', suggest that there is room to further improve the performance of the basic lazy Bayesian rule induction algorithm described in this paper. This is a promising direction for future research.

We also show that the prediction error of LBR for discrete valued data is very competitive to state-of-the-art non-lazy techniques. Due to the use of lazy learning, the computation profile of the technique is dominated by the number of cases to be classified. In consequence, it is very efficient when there are only a small number of test cases for each training set, as is the case in contexts where incremental learning is traditionally employed.

Acknowledgments

The authors are grateful to J. Ross Quinlan for providing C4.5. Many thanks to the anonymous reviewers for their very valuable comments.

References

Aha, D.W. (ed.) (1997) *Lazy Learning.* Dordrecht: Kluwer Academic.

Bauer, E. & Kohavi, R. (1999) An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. To appear in *Machine Learning.*

Blake, C., Keogh, E. & Merz, C.J. (1998) UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/~mlearn/MLRepository.html].
Irvine, CA: University of California, Department of Information and Computer Science.

Cestnik, B., Kononenko, I., & Bratko, I. (1987) ASSISTANT 86: A knowledge-elicitation tool for sophisticated users. In *Proc. 2nd European Working Session on Learning*, 31-45, Wilmslow, UK: Sigma Press.

Breiman, L., Friedman, J.H., Olshen, R.A., & Stone, C.J. (1984) *Classification and Regression Trees.* Wadsworth, Belmont.

Breiman, L. (1996) Bagging predictors. *Machine Learning*, 24: 123-140.

Cestnik, B. (1990) Estimating probabilities: A crucial task in machine learning. In *Proc. European Conf. on Artificial Intelligence*, 147-149.

Domingos, P. & Pazzani, M. (1996) Beyond independence: Conditions for the optimality of the simple Bayesian classifier. In *Proc. 13th Intl. Conf. on Machine Learning*, pp. 105-112,. San Francisco, CA: Morgan Kaufmann.

Duda, R.O. & Hart, P.E. (1973) *Pattern Classification and Scene Analysis.* New York: John Wiley.

Fayyad, U.M. & Irani, K.B. (1993) Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. 13th Intl. Joint Conf. on Artificial Intelligence*, 1022-1027, San Mateo, CA: Morgan Kaufmann.

Freund, Y. & Schapire, R. E. (1997) A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1): 119-139.

Friedman, J., Kohavi, R., & Yun, Y. (1996) Lazy decision trees. In *Proc. 13th Natl. Conf. on Artificial Intelligence*, 717-724, Menlo Park, CA: AAAI Press.

Friedman, N. & Goldszmidt, M. (1996) Building classifiers using Bayesian networks. In *Proc. 13th Natl. Conf. on Artificial Intelligence*, 1277-1284, Menlo Park, CA: AAAI Press.

Holte, R.C., Acker, L.E., & Porter, B.W. (1989) Concept learning and the problem of small disjuncts. In *Proc. 11th Intl. Joint Conf. on Artificial Intelligence*, 813-818, San Mateo, CA: Morgan Kaufmann.

John, G., Kohavi, R., Pfleger, K. (1994) Irrelevant features and the subset selection problem. In *Proc. 11th Intl. Conf. on Machine Learning*, 121-129, San Mateo, CA: Morgan Kaufmann.

John, G. & Langley, P. (1995) Estimating continuous distributions in Bayesian classifiers. In *Proc. 11th Conf. on Uncertainty in Artificial Intelligence*, 338-345, San Mateo, CA: Morgan Kaufmann.

Kohavi, R. (1996) Scaling up the accuracy of naive-Bayes classifiers: A decision-tree hybrid. In *Proc. 2nd Intl. Conf. on Knowledge Discovery and Data Mining*, 202-207, Menlo Park, CA: AAAI Press.

Kohavi, R. & Wolpert, D. (1996) Bias plus variance decomposition for zero-one loss functions. In *Proc. 13th Intl. Conf. on Machine Learning*, 275–283, Morgan Kaufmann.

Kononenko, I. (1990) Comparison of inductive and naive Bayesian learning approaches to automatic knowledge acquisition. In B. Wielinga *et al.* (Eds.), *Current Trends in Knowledge Acquisition.* Amsterdam: IOS Press.

Kononenko, I. (1991) Semi-naive Bayesian classifier. In *Proc. of European Conf. on Artificial Intelligence*, 206-219.

Langley, P. (1993) Induction of recursive Bayesian classifiers. In *Proc. European Conf. on Machine Learning*, 153-164, Berlin: Springer-Verlag.

Langley, P. & Sage, S. (1994) Induction of selective Bayesian classifiers. In *Proc. 10th Conf. on Uncertainty in Artificial Intelligence*, 339-406, Seattle, WA: Morgan Kaufmann.

Pagallo, G. & Haussler, D. (1990) Boolean feature discovery in empirical learning. *Machine Learning*, 5: 71-100.

Pazzani, M.J. (1998) Constructive inductive of Cartesian product attributes, In Liu, H, & Motoda, H. (eds.) *Feature Extraction, Construction and Selection - A Data Mining Perspective*, 341-354, Boston, MA: Kluwer Academic.

Quinlan, J.R. (1993) *C4.5: Programs for Machine Learning*, San Mateo, CA: Morgan Kaufmann.

Quinlan, J.R. (1996) Bagging, boosting, and C4.5. In *Proc. 13th Natl. Conf. on Artificial Intelligence*, 725-730, Menlo Park, CA: AAAI Press.

Sahami, M. (1996) Learning limited dependence Bayesian classifiers. In *Proc. 2nd Intl. Conf. on Knowledge Discovery and Data Mining*, 334-338, Menlo Park, CA: AAAI Press.

Schapire, R.E., Freund, Y., Bartlett, P., & Lee, W.S. (1997) Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proc. 14th Intl. Conf. on Machine Learning*, 322-330, Morgan Kaufmann.

Singh, M. & Provan, G.M. (1996) Efficient learning of selective Bayesian network classifiers. In *Proc. 13th Intl. Conf. on Machine Learning*, 453-461, San Francisco, CA: Morgan Kaufmann.

Ting, K.M. (1994) The problem of small disjuncts: Its remedy in decision trees. In *Proc. 10th Canadian Conf. on Artificial Intelligence*, 91-97, Canadian Soc. for Comp. Studies of Intelligence.

Ting, K.M. & Zheng, Z. (1999) Improving the performance of boosting for naive Bayesian classification. In *Proc. 3rd Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, Berlin: Springer-Verlag.

Webb, G. (forthcoming) MultiBoosting: A technique for combining boosting and bagging. Accepted for publication in *Machine Learning*.

Webb, G.I. & Pazzani, M.J. (1998) Adjusted probability naive Bayesian induction. In *Proc. 11th Australian Joint Conf. on Artificial Intelligence*, Berlin: Springer-Verlag.

Zheng, Z. & Webb, G.I, (1998) *Lazy Bayesian rules*, Deakin University Computing Technical Report TR C98-17 (Available at "http://www3.cm.deakin.edu.au/~zijian/Papers/lbr-tr-c98-17.ps.gz").