Pre-publication draft of a paper which appeared in the Proceedings of Expert Systems '99, The Nineteenth SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence pp 122-131
Publisher: Springer

Learning Lazy Rules to Improve the Performance of Classifiers

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

Abstract

Based on an earlier study on lazy Bayesian rule learning, this paper introduces a general lazy learning framework, called LazyRule, that begins to learn a rule only when classifying a test case. The objective of the framework is to improve the performance of a base learning algorithm. It has the potential to be used for different types of base learning algorithms. LazyRule performs attribute elimination and training case selection using cross-validation to generate the most appropriate rule for each test case. At the consequent of the rule, it applies the base learning algorithm on the selected training subset and the remaining attributes to construct a classifier to make a prediction. This combined action seeks to build a better performing classifier for each test case than the classifier trained using all attributes and all training cases. We show empirically that LazyRule improves the performances of naive Bayesian classifiers and majority vote.

1 Introduction

Lazy learning [2] is a class of learning techniques that spend little or no effort during training and delay the computation to the classification time. No concise models, such as decision trees or rules, are created at training time. When classifying a test case, a lazy learning algorithm performs its computation in two stages. First, it selects a subset of the training cases that are relevant to classifying the case in question. Then, a classifier is constructed using this training subset; and the classifier is ultimately employed to classify the test case. The case selection process in the first stage is a crucial part in lazy learning that ultimately influences the classifier to be constructed in the second stage.

The archetypal example of a lazy learning algorithm is the k-nearest neighbor algorithm or instance-based learning algorithm [1, 8, 10]. In its basic form, the k-nearest neighbor algorithm stores all training cases. At classification time, it computes a distance measure between the test case and each of the training cases, and selects the nearest k training cases from the first stage. A simple majority vote is used in the second stage—the majority class of the k nearest training cases is predicted to be the class for the test case. Another example is LazyDT [12], which creates decision rules at classification time to select a subset of training cases, and then performs majority vote to make a prediction.

Lbr [20] uses a lazy learning technique developed to improve the performance of naive Bayesian classification. For each test case, it generates a most appropriate rule with a conjunction of attribute-value pairs as its antecedent and a local naive Bayesian classifier as its consequent. The local naive Bayesian classifier is built using the subset of training cases that satisfy the antecedent of the rule, and is used to classify the test case. The main objective of creating rules is to alleviate the attribute inter-dependence problem of naive Bayesian classification.

There are several variations, especially on the method to select a training subset. For example, the Optimized Set Reduction (Osr) algorithm [5] first identifies a set of plausible rules R, based on an entropy measure, that cover the case X to be classified. The set of training cases S is then formed, containing all training cases covered by any rule in R. X is then classified using Bayesian classification with probability estimates derived from the distributions of attribute values in S. Fulton *et al.* [13] describe a variation of the k-nearest neighbor algorithm that selects more than one subset. For a given test case, a sequence of k decision trees is induced using 1,2,...,k nearest cases. Then a weighted voting scheme is employed to make the final prediction. Fulton *et al.* [13] also explore two other alternative techniques to select a single training subset. One or more decision trees are generated in all these techniques. Because all of these three techniques always produce the same training subset for a test case no matter what base learning algorithm is used in the second stage, they are unlikely to be amenable for different types of base learning algorithm. The Learning All Rules approach [19] performs lazy learning of decision rules.

The lazy learning algorithms described so far are meant to be used as a stand-alone classifier. There is a lack of a general framework of lazy learning that can be used to improve the performance of a chosen learning algorithm which is to be employed to produced a classifier in the second stage of the lazy classification process. In the crucial stage of training subset selection, the criteria, usually heuristics, used by these lazy learning algorithms except Lbr are not directly relevant to the base classifiers employed in the second stage. This paper introduces a lazy learning framework, as a generalization of Lbr [20], that performs both attribute elimination and training case selection. When doing these, the chosen learning algorithm, which is to be employed in the second stage, is utilized in the evaluation process. This framework is intended to improve the performance of the chosen base learning algorithm.

The following section describes the LazyRule framework. Section 3 contains the empirical evaluation to investigate whether the framework can be used to improve the performance of two types of base learning algorithms. Section 4 discusses the advantages and limitations of LazyRule. The final section summarizes our findings and describes possible future work.

2 The Lazy Rule Learning Framework

This section describes the lazy learning framework, called LazyRule. Like most of the other lazy learning algorithms, LazyRule stores all training cases, and begins to compute only when a classification is required.

To classify a test case, LazyRule generates a rule that is most appropriate to the test case. The antecedent of a lazy rule is a conjunction of attribute-value pairs or conditions, and each condition is in the form of 'attribute=value'. The current version of LazyRule can only directly deal with nominal attributes. Numeric attributes are discretized as a pre-process. The consequent of a lazy rule is a local classifier created from those training cases (called *local training cases*) that satisfy the antecedent of the rule. The local classifier is induced using only those attributes that do not appear in the antecedent of the rule.

During the generation of a lazy rule, the test case to be classified is used to guide the selection of attributes for creating attribute-value pairs—only values that appear in the test case are being considered in the selection process. The objective is to grow the antecedent of a rule that ultimately decreases the errors of the local classifier in the consequent of the rule. The antecedent of the rule defines a sub-space of the instance space to which the test case belongs, and selects a subset of the available training instances. For all instances in the instance sub-space, each of the attributes occurring in the antecedent has an identical value which is the same as the one in the antecedent, thus not affecting the behavior of the local classifier. These attributes are removed from the local classifier for computational efficiency. Finally, the local classifier of the rule classifies the test case, since this case satisfies the antecedent of the rule. Table 1 outlines the LazyRule framework. One must choose a base learning algorithm for inducing local classifiers before using this framework.

For each test case, LazyRule uses a greedy search to generate a rule of which the antecedent matches the test case. The growth of the rule starts from a special rule whose antecedent is *true*. The local classifier in its consequent part is trained on the entire training set using all attributes. At each step of the greedy search, LazyRule tries to add, to the current rule, each attribute that has not already been in the antecedent of the rule, so long as its value on the test case is not missing. The objective is to determine whether including this attribute-value pair on the test case into the rule can significantly improve the estimated accuracy.

The utility of every possible attribute-value pair to be added to the antecedent of a rule is evaluated in the following manner. A subset of examples D_{subset} that satisfies the attribute-value pair is identified from the current local training set $D_{training}$, and is used to train a temporary classifier using all attributes that do not occur in the antecedent of the current rule and are not the attribute being examined. Cross-validation (CV) is performed to obtain the estimated errors of both the local and temporary classifiers.[1] Estimated errors of the temporary classifier on D_{subset} together with estimated errors of the local

[1]We choose cross-validation as the evaluation method because cross-validated errors are more reliable estimates of true errors than re-substitution errors [4].

Table 1: The LazyRule Framework

```
Given a base learning algorithm **Alg**.
**LazyRule**($Att, D_{training}, E_{test}$)
    INPUT: $Att$: a set of attributes,
            $D_{training}$: a set of training cases described using $Att$ and classes,
            $E_{test}$: a test case described using $Att$.
    OUTPUT: a predicted class for $E_{test}$.
$LocalClr$ = a classifier induced by Alg using $Att$ on $D_{training}$
$Errors$ = errors of $LocalClr$ estimated using CV on $D_{training}$
$Cond = true$
REPEAT
    $TempErrors_{best}$ = the number of cases in $D_{training} + 1$
    FOR each attribute $A$ in $Att$ whose value $v_A$ on $E_{test}$ is not missing DO
        $D_{subset}$ = cases in $D_{training}$ with $A = v_A$
        $TempClr$ = a classifier induced by Alg using $Att - \{A\}$ on $D_{subset}$
        $TempErrors$ = errors of $TempClr$ estimated using CV on $D_{subset}$ +
                    the portion of $Errors$ in $D_{training} - D_{subset}$
        IF (($TempErrors < TempErrors_{best}$) AND
            ($TempErrors$ is significantly lower than $Errors$))
        THEN
            $TempClr_{best} = TempClr$
            $TempErrors_{best} = TempErrors$
            $A_{best} = A$
    IF (an $A_{best}$ is found)
    THEN
        $Cond = Cond \wedge (A_{best} = v_{A best})$
        $LocalClr = TempClr_{best}$
        $D_{training} = D_{subset}$ corresponding to $A_{best}$
        $Att = Att - \{A_{best}\}$
        $Errors$ = errors of $LocalClr$ estimated using CV on $D_{training}$
    ELSE
        EXIT from the REPEAT loop
classify $E_{test}$ using $LocalClr$
RETURN the class
```

classifier of the current rule on $D_{training} - D_{subset}$ are used as the evaluation measure of the attribute-value pair for growing the current rule. If this measure is lower than the estimated errors of the local classifier on $D_{training}$ at a significance level better than 0.05 using a one-tailed pairwise sign-test [7], this attribute-value pair becomes a candidate condition to be added to the current 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 measure (errors) is added to the antecedent of the current 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 better local classifier can be formed, or the local training set is too small (i.e., ≤ 30 examples) to further reduce the instance sub-space by specializing the antecedent of the rule. In such cases, further growing the rule would not significantly reduce its errors. Finally, the local classifier of this rule is used to classify the test case under consideration.

LazyRule is a generalization of Lbr [20]. In principle, the general framework can be used with any base classifier learning algorithms.

3 Does LazyRule improve the performance of classifiers?

In this section, we evaluate whether the LazyRule framework can be used to improve the performance of a base learning algorithm. In order to show the generality of the framework, two different types of base learning algorithm are used in the following experiments. They are majority vote (MV) and the naive Bayesian classifier (NB). MV classifies all the test cases as belonging to the most common class of the training cases.

NB [16, 17, 18] is an implementation of Bayes' rule:

$$P(C_i|V) = P(C_i)P(V|C_i)/P(V)$$

for classification, where P denotes probability, C_i is class i and V is a vector of attribute values describing a case. By assuming all attributes are mutually independent within each class, $P(V|C_i) = \prod_j P(v_j|C_i)$ simplifies the estimation of the required conditional probabilities. NB is simple and computationally efficient. It has been shown that it is competitive to more complex learning algorithms such as decision tree and rule learning algorithms on many datasets [9, 6].

Because the current version of LazyRule only accepts nominal attribute inputs, continuous-valued attributes are discretized as a pre-process in the experiments. The discretization method is based on an entropy-based method [11]. For each pair of training set and test set, both the training set and the test set are discretized by using cut points found from the training set alone.

LazyRule with MV or NB uses the N-fold cross-validation method (also called leave-one-out estimation) [4] in the attribute evaluation process because both MV and NB are amenable to efficiently adding and subtracting one case. We denote LR-NB as the LazyRule framework that incorporates NB as its base learning algorithm; likewise for LR-MV. Note that LR-NB is exactly the same as Lbr [20].

Ten commonly used natural datasets from the UCI repository of machine learning databases [3] are employed in our investigation. Table 2 gives a brief summary of these domains, including the dataset size, the number of classes, the number of numeric and nominal attributes. Two stratified 10-fold cross-validations [15] are conducted on each dataset to estimate the performance of each algorithm.

Table 3 reports the average test classification error rate for each of the experimental datasets. To summarize the performance comparison between an

Table 2: Description of learning tasks

Domain	Size	No. of Classes	No. of Attributes	
			Numeric	Nominal
Annealing	898	6	6	32
Breast cancer (Wisconsin)	699	2	9	0
Chess (King-rook-vs-king-pawn)	3196	2	0	36
Credit screening (Australia)	690	2	6	9
House votes 84	435	2	0	16
Hypothyroid diagnosis	3163	2	7	18
Pima Indians diabetes	768	2	8	0
Solar flare	1389	2	0	10
Soybean large	683	19	0	35
Splice junction gene sequences	3177	3	0	60

Table 3: Average error rates (%) of LazyRule and its base learning algorithms.

Datasets	NB	LR-NB	MV	LR-MV
Annealing	2.8	2.7	23.8	8.2
Breast(W)	2.7	2.7	34.5	10.3
Chess(KR-KP)	12.2	2.0	47.8	4.5
Credit(Aust)	14.0	14.0	44.5	15.0
House-votes-84	9.8	5.6	38.6	4.5
Hypothyroid	1.7	1.6	4.7	2.3
Pima	25.2	25.4	34.9	26.4
Solar-flare	19.4	16.4	15.7	15.7
Soybean	9.2	5.9	86.6	23.2
Splice-junction	4.4	4.0	48.1	14.7
mean	10.1	8.0	37.9	14.5
ratio		.73		.32
w/t/l		7/2/1		9/1/0
p. of wtl		.0352		.0020

algorithm and LazyRule with it, Table 3 also shows the geometric mean of error rate ratios, the number of wins/ties/losses, and the result of a two-tailed pairwise sign-test. An error rate ratio for LR-NB versus NB, for example, is calculated using a result for LR-NB divided by the corresponding result for NB. A value less than one indicates an improvement due to LR-NB. The result of the sign test indicates the significance level of the test on the win/tie/loss record.

We summarize our findings as follows. LazyRule improves the predictive accuracy of NB and MV. The framework achieves a 68% relative reduction in error rate for MV, and 27% relative reduction for NB. The improvement is significant at a level better than 0.05 for both MV and NB. LazyRule improves the performance of MV on all datasets. It improves the performance of NB on

Table 4: Average rule lengths of LazyRule.

Dataset	LR-NB	LR-MV
Annealing	0.20	1.90
Breast(W)	0.00	1.63
Chess(KR-KP)	4.10	4.00
Credit(Aust)	0.10	2.55
House-votes-84	0.90	2.23
Hypothyroid	0.40	4.21
Pima	0.10	2.13
Solar-flare	1.10	2.65
Soybean	0.90	2.35
Splice-junction	0.70	2.14
mean	0.85	2.58

7 datasets, and keeps the same performance on 2 datasets. Only on the Pima dataset does LR-NB slightly increase the error rate of NB.

Table 4 shows the average length of all rules produced by LR-NB and LR-MV. The average rule length is the ratio of the total of conditions produced for all test cases and the total number of test cases, averaged over all runs.

The mean values across all datasets are 0.85 and 2.58 for LR-NB and LR-MV, respectively. Examining the figures on each dataset indicates that LazyRule only produces rules when it is possible to improve the performance of the classifier trained using all training cases and all attributes. On average, LR-MV produces a rule with more than 1.5 conditions for each test case on each of the experimental datasets. This is an indication that LazyRule could improve the performance of MV on all of these datasets. Small values of average rule length indicate either no or minor improvement. This is shown by LR-NB on the Annealing, Breast(W), Credit(Aust), Hypothyroid and Pima datasets, which have average rule lengths less than 0.5.

LazyRule is expected to require more compute time than the base learning algorithm. For example, in the Breast(W) dataset in which LR-NB produces no rule, the execution time is 0.241 seconds as compared to .005 seconds for NB. In the Chess dataset in which LR-NB produces the longest rule, LR-NB requires 213.13 seconds whereas NB requires only 0.034 seconds. The time is recorded from a 300MHz Sun UltraSPARC machine.

Being a lazy learner, another important factor that affects LazyRule's execution time is the test set size. The execution time of LazyRule is proportional to the size of the test set. For example, in the Chess dataset, the test size used in the current experiment is 319. When we change the experiment from ten-fold cross-validation to three-fold cross-validation (the test set size is increased to 1066), the execution time of LR-NB increases from 213 seconds to 299 seconds.

4 The Advantages and Limitations of LazyRule

LazyRule's primary action is to eliminate attributes and select training cases that are most relevant to classifying the current test case. This builds a better performing classifier for the test case than the classifier trained using all attributes and all training cases. This flexible nature of LazyRule stretches the base learning algorithm to its best potential under these two variables: attribute elimination and training case selection.

The key advantage of LazyRule over a previous system LazyDT [12] is the use of the cross-validation method for attribute elimination and training case selection. The use of this technique allows different types of learning algorithm to be incorporated into the LazyRule framework. LazyDT uses an entropy measure for attribute elimination which leads to selecting cases with the same class. As a result, only majority vote can be used to form the local classifier.

The idea of using cross-validation and the learning algorithm, which is to be used to induce the final classifier, in the evaluation process is called the wrapper method [14]. This method was initially proposed solely for the purpose of attribute selection/elimination. LazyRule uses the method for both attribute elimination and training case selection.

The major computational overhead in LazyRule is the cross-validation process used in the evaluation of an attribute. The nature of the lazy learning mechanism requires that the same process is repeated for each test case. This computational overhead can be substantially reduced by caching the useful information. In the current implementation of LazyRule, the evaluation function values of attribute-value pairs that have been examined are retained from one test case to the next. This avoids re-calculation of the evaluation function values of the same attribute-value pairs when classifying unseen cases that appear later, thus reducing the entire execution time. Our experiment shows that caching this information reduces the execution time of LazyRule with the naive Bayesian classifier by 93% on average on the 10 datasets used in the experiment. This happens, because the evaluation of attribute-value pairs for different test cases are often repeated, including repeated generation of identical rules for different test cases. LazyRule could be made even more efficient by caching further information such as local classifiers and indices for training cases in different stages of the growth of rules. Of course, this would increase memory requirements.

Caching the local classifiers has an added advantage apart from computational efficiency. Now, the number of different rules together with local classifiers induced thus far are ready to be presented to the user in any stage during the classification time.

In theory, decision tree learning algorithm is a candidate to be used in the LazyRule framework. There are reasons why we did not include it in our experiments. First, given a test case, only one path is needed, not the entire tree. Second, the process of growing a lazy rule is similar to the process of growing a tree. Only the criterion for attribute selection is different. Lastly, building a tree/path at the consequent of the rule would actually use different criteria for two similar processes. This seems undesirable.

5 Conclusions and Future Work

We introduce the LazyRule framework based on an earlier work for learning lazy Bayesian rules, and show that it can be used to improve the performance of a base classifier learning algorithm. The combined action of attribute elimination and training case selection of LazyRule, tailored for the test case to be classified, enables it to build a better performing classifier for the test case than the classifier trained using all attributes and all training cases. We show empirically that LazyRule improves the performance of two base learning algorithms, the naive Bayesian classifier and majority vote.

Our future work includes extending LazyRule to accept continuous-valued attribute input, and experimenting with other types of learning algorithm such as k-nearest neighbors. It is interesting to see how it will perform when a lazy learning algorithm such as k-nearest neighbors is incorporated in this lazy learning framework. The current implementation of LazyRule only considers attribute-value pairs each in the form of 'attribute = value'. Alternatives to this form are worth exploring. Applying this framework to regression tasks is also another interesting avenue for future investigation.

References

[1] Aha, D.W., Kibler, D., & Albert, M.K. Instance-based learning algorithms. *Machine Learning, 6*, 37-66, 1991.

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

[3] Blake, C., Keogh, E. & Merz, C.J. 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, 1998.

[4] Breiman, L., Friedman, J.H., Olshen, R.A., & Stone, C.J. *Classification And Regression Trees*, Belmont, CA: Wadsworth, 1984.

[5] Briand, L.C. & Thomas, W.M. A pattern recognition approach for software engineering data analysis. *IEEE Transactions on Software Engineering, 18*, 931-942, 1992.

[6] Cestnik, B. Estimating probabilities: A crucial task in machine learning. *Proceedings of the European Conference on Artificial Intelligence*, pages 147-149, 1990.

[7] Chatfield, C. *Statistics for Technology: A Course in Applied Statistics.* London: Chapman and Hall, 1978.

[8] Cover, T.M. & Hart, P.E. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory, 13*, 21-27, 1967.

[9] Domingos, P. & Pazzani, M. Beyond independence: Conditions for the optimality of the simple Bayesian classifier. *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 105-112, 1996. San Francisco, CA: Morgan Kaufmann.

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

[11] Fayyad, U.M. & Irani, K.B. Multi-interval discretization of continuous-valued attributes for classification learning. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022-1027, 1993. San Mateo, CA: Morgan Kaufmann.

[12] Friedman, J., Kohavi, R., & Yun, Y. Lazy decision trees. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 717-724, 1996. Menlo Park, CA: The AAAI Press.

[13] Fulton, T., Kasif, S., Salzberg, S., and Waltz, D. Local induction of decision trees: Towards interactive data mining. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 14-19, 1996. Menlo Park, CA: AAAI Press.

[14] John, G.H., Kohavi, R., & Pfleger, K. Irrelevant features and the subset selection problem. *Proceedings of the Eleventh International Conference on Machine Learning*, pages 121-129, 1994. San Francisco, CA: Morgan Kaufmann.

[15] Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1137-1143, 1995. San Mateo, CA: Morgan Kaufmann.

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

[17] Langley, P., Iba, W.F., & Thompson, K. An analysis of Bayesian classifiers. *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 223-228, 1992. Menlo Park, CA: The AAAI Press.

[18] Langley, P. & Sage, S. Induction of selective Bayesian classifiers. *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 339-406, 1994. Seattle, WA: Morgan Kaufmann.

[19] Viswanathan, M. & Webb, G.I. Classification learning using all rules. *Proceedings of the Tenth European Conference on Machine Learning*, pages 149-159, 1998. Berlin: Springer-Verlag.

[20] Zheng, Z. & Webb, G.I. Lazy Learning of Bayesian rules. To appear in *Machine Learning.*

[21] Zheng, Z., Webb, G.I. & Ting, K.M. Lazy Bayesian rules: A lazy semi-naive Bayesian learning technique competitive to boosting decision trees. *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 493-502, 1999. Morgan Kaufmann.