CHAPTER 5. PAPER 1:

FAST RULE-BASED CLASSIFICATION USING P-TREES

5.1. Abstract

Lazy classification does not require construction of a classifier. Decisions can, therefore, be based on the properties of the particular unknown sample rather than the set of all possible samples. We introduce a rule-based classifier that uses neighborhood membership rather than static intervalization to break up continuous domains. A distance function that makes neighborhood evaluation efficient for the P-tree data structure is chosen. We discuss information gain and statistical significance in a unified framework and observe that the commonly used formulation of information gain can be seen as an approximation to the exact entropy. A new algorithm for constructing and combining multiple classifiers is introduced and its improved performance demonstrated, especially for data sets with many attributes.

5.2. Introduction

Many data mining tasks involve predicting a discrete class label, such as whether a patient is sick, whether an e-mail message is spam, etc. Eager classification techniques construct a classifier from the training data before considering unknown samples. Eager algorithms include decision tree induction, which fundamentally views data as categorical and discretizes continuous data as part of the classifier construction [1]. Lazy classification predicts the class label attribute of an unknown sample based on the entire training data. Lazy versions of decision tree induction improve classification accuracy by using the information gain criterion based on the actual values of the unknown sample [2]. This modification reduces the fragmentation, replication, and small disjunct problems of standard decision trees [3]. Previous work has, however, always assumed that the data were discretized as a preprocessing step.

Many other lazy techniques use distance information explicitly through window or kernel functions that select and weight training points according to their distance from the unknown sample. Parzen window [4], kernel density estimate [5], Podium [6], and k-nearest neighbor algorithms can be seen as implementations of this concept. The idea of using similarity with respect to training data directly is not limited to lazy classification. Many eager techniques exist that improve on the accuracy of a simple kernel density estimate by minimizing an objective function that is a combination of the classification error for the training data and a term that quantifies complexity of the classifier. These techniques include support vector machines and regularization networks as well as many regression techniques [7]. Support vector machines often have the additional benefit of reducing the potentially large number of training points that have to be used in classifying unknown data to a small number of support vectors.

Kernel- and window-based techniques, at least in their original implementation, do not normally weight attributes according to their importance. For categorical attributes, this limitation is particularly problematic because it only allows two possible distances for each attribute. In this paper, we assume distance 0 if the values of categorical attributes match and 1 otherwise. Solutions that have been suggested include weighting attribute dimensions according to their information gain [8]; optimizing attribute weighting using genetic algorithms [9]; selecting important attributes in a wrapper approach [10]; and, in a more general kernel formulation, boosting as applied to heterogeneous kernels [11].

We combine the concept of a window that identifies similarity in continuous attributes with the lazy decision tree concept of [2] that can be seen as an efficient attribute selection scheme. A constant window function (step function as a kernel function) is used to determine the neighborhood in each dimension with attributes combined by a product of window functions. The window size is selected based on information gain. A special distance function, the HOBbit distance, allows efficient determination of neighbors for the P-tree data structure [12] that is used in this paper. A P-tree is a bit-wise, column-oriented data structure that is self-indexing and allows fast evaluation of tuple counts with given attribute value or interval combinations. For P-tree algorithms, database scans are replaced by multi-way AND operations on compressed bit sequences.

Section 5.3 discusses the Algorithm with Section 5.3.1 summarizing the P-tree data structure, Section 5.3.2 motivating the HOBbit distance, and Section 5.3.3 explaining how we map to binary attributes. Section 5.3.4 gives new insights related to information gain; Section 5.3.5 discusses our pruning technique; and Section 5.3.6 introduces a new way of combining multiple classifiers. Section 5.4 presents our experimental setup results, with Section 5.4.1 describing the data sets, Sections 5.4.2-5.4.4 discussing the accuracy of different classifiers, and Section 5.4.5 demonstrating performance. Section 5.5 concludes the paper.

5.3. Algorithm

Our algorithm selects attributes successively using information gain based on the subset of points selected by all previously considered attributes. The procedure is very similar to decision tree induction, e.g., [1], and, in particular, lazy decision tree induction

[2] but has a few important differences. The main difference lies in our treatment of continous attributes using a window or kernel function based on the HOBbit distance function. Since a window function can be seen as a lazily constructed interval, it is straightforward to integrate into the algorithm.

Eager decision tree induction, such as [1], considers the entire domain of categorical attributes to evaluate information gain. Lazy decision tree induction, including [2], in contrast, calculates information gain and significance based on virtual binary attributes that are defined through similarity with respect to a test point. Instead of constructing a full tree, one branch is explored in a lazy fashion; i.e., a new branch is constructed for each test sample. The branch is part of a different binary tree for each test sample. Note that the other branches are never of interest because they correspond to one or more attributes being different from the test sample. We will, therefore, use the term rule-based classification since the branch can more easily be seen as a rule that is evaluated.

5.3.1. P-trees

The P-tree data structure was originally developed for spatial data [13] but has been successfully applied in many contexts [9,14]. The key ideas behind P-trees are to store data column-wise in hierarchically compressed bit-sequences that allow efficient evaluation of bit-wise AND operations based on an ordering of rows that benefits compression. The row-ordering aspect can be addressed most easily for data that show inherent continuity such as spatial and multi-media data. For spatial data, for example, neighboring pixels will often represent similar color values. Traversing an image in a sequence that keeps neighboring points close will preserve the continuity when moving to the one-dimensional storage representation. An example of a suitable ordering is Peano, or recursive raster ordering. If data show no natural continuity, sorting it according to generalized Peano order can lead to significant speed improvements. The idea of generalized Peano order sorting is to traverse the space of data mining-relevant attributes in Peano order while including only existing data points. Whereas spatial data do not require storing spatial coordinates, all attributes have to be represented if generalized Peano sorting is used, as is the case in this paper. Only integer data types are traversed in Peano order since the concept of proximity does not apply to categorical data. The relevance of categorical data for the ordering depends on the number of attributes needed to represent it. Figure 5.1 shows how two numerical attributes are traversed together with the P-tree that is constructed from the sequence of the highest order bit of x.

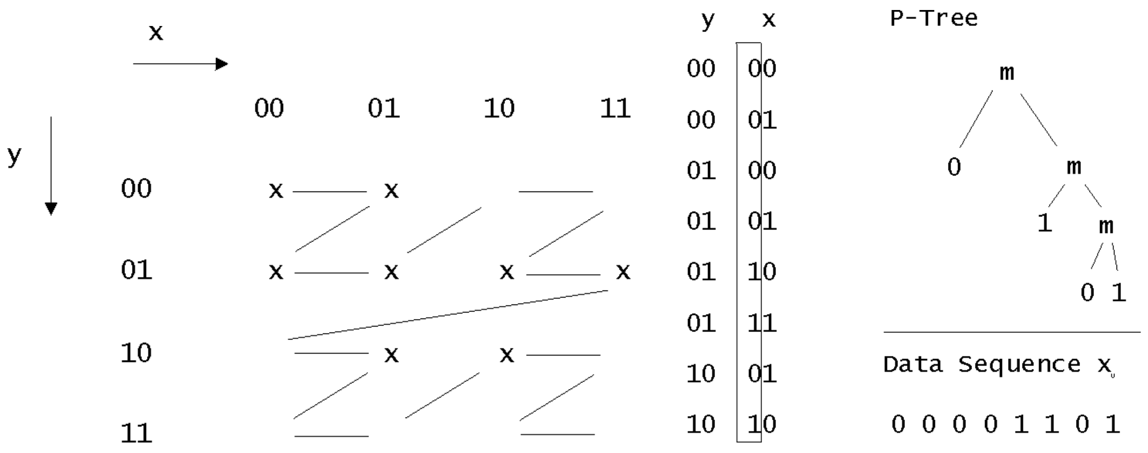

Figure 5.1. Peano order sorting and P-tree construction.

P-trees are data structures that use hierarchical compression. The P-tree graph shows a tree with fan-out 2 in which 1 stands for nodes that represent only 1 values, 0 stands for nodes that represent only 0 values, and m ("mixed") stands for nodes that represent a combination of 0 and 1 values. Only "mixed" nodes have children. For other nodes, the data sequence can be reconstructed based on the purity information and node level alone. The P-tree example is kept simple for demonstration purposes. The implementation has a fan-out of 16 and uses an array rather than pointers to represent the tree structure. It, furthermore, stores the count of 1-bits for all nodes to improve ANDing speed. For data mining purposes, we are mainly interested in the total number of 1-bits for the entire tree, which we will call root count in the following sections.

5.3.2. HOBbit Distance

The nature of a P-tree-based data representation with its bit-column structure has a strong impact on the kinds of algorithms that will be efficient. P-trees allow easy evaluation of the number of data points in intervals that can be represented by one bit pattern. For 8-bit numbers, the interval [0,128) can easily be specified by requiring the highest-order bit to be 0 and not putting any constraints on the lower-order bits. This specification is derived as the root count of the P-tree corresponding to the highest-order bit. The interval [128,194) can be defined by requiring the highest order bit to be 1 and the next bit to be 0, and not putting constraints on any other bits. The number of points within this interval is evaluated by computing an AND of the basic P-tree corresponding to the highest-order bit and the complement of the next highest-order bit. Arbitrary intervals can be defined using P-trees but may require the evaluation of several ANDs.

This observation led to the definition of the HOBbit distance for numerical data. Note that for categorical data, which can only have either of two possible distances, distance 0 if the attribute values are the same and 1 if they are different, bits are equivalent.

The HOBbit distance is defined as

$$d_{HOBbit}(a_s, a_t) = \begin{cases} 0 & \text{for} \quad a_s = a_t \\ \max\limits_{j=0}^{\infty}\left(j+1 \mid \left\lfloor \frac{a_s}{2^j} \right\rfloor \neq \left\lfloor \frac{a_t}{2^j} \right\rfloor\right) & \text{for} \quad a_s \neq a_t \end{cases} \qquad (1)$$

where a_s and a_t are attribute values, and $\lfloor \; \rfloor$ denotes the floor function. The HOBbit distance is, thereby, the number of bits by which two values have to be right-shifted to make them equal. The numbers 32 (10000) and 37 (10101) have HOBbit distance 3 because only the first two digits are equal, and the numbers, consequently, have to be right-shifted by 3 bits.

5.3.3. Virtual Binary Attributes

For the purpose of the calculation of information gain, all attributes are treated as binary. Any attribute can be mapped to a binary attribute using the following prescription. For a given test point and a given attribute, a virtual attribute is assumed to be constructed, which we will call *a*. Virtual attribute *a* is 1 for the attribute value of the test sample and 0 otherwise. For numerical attributes, *a* is 1 for any value within the HOBbit neighborhood of interest and 0 otherwise. Consider, as an example, a categorical attribute that represents color and a sample point that has attribute value "red." Attribute *a* would (virtually) be constructed to be 1 for all "red" items and 0 for all others, independently of how large the domain is. For a 5-bit integer attribute, a HOBbit neighborhood of $d_{\text{HOBbit}}=3$, and a sample value of 37, binary attribute *a* will be imagined as being 1 for values in the interval [32,48) and 0 for values outside this interval. The class label attribute is a binary attribute for all data sets that we discuss in this paper and therefore does not require special treatment.

5.3.4. Information Gain

We can now formulate information and information gain in terms of imagined attribute a as well as the class label. The information concept was introduced by Shannon in analogy to entropy in physics [15]. The commonly used definition of the information of a binary system is

$$Info(p_0, p_1) = -(p_0 \log p_0 + p_1 \log p_1), \tag{2}$$

where p_0 is the probability of one state, such as class label 0, and p_1 is the probability of the other state, such as class label 1. The information of the system before attribute a has been used can be seen as the expected information still necessary to classify a sample. Splitting the set into those tuples that have a = 0 and those that have a = 1 may give us useful information for classification and, thereby, reduce the remaining information necessary for classification. The following difference is commonly called the information gain of attribute a

$$InfoGain(a) = Info\left(\frac{t_0}{t}, \frac{t_1}{t}\right) - \frac{x}{t} Info\left(\frac{x_0}{x}, \frac{x_1}{x}\right) - \frac{y}{t} Info\left(\frac{y_0}{y}, \frac{y_1}{y}\right) \tag{3}$$

with x (y) being the number of training points for which attribute a has value 0 (1) and $t = x + y$. x_0 refers to the number of points for which attribute a and the class label are 0; y_0 refers to the number of points for which attribute a is 1 and the class label is 0; etc.

Appendix B shows how this definition of information gain can be derived from the probability associated with a 2x2 contingency table. It is noted that, in this derivation, an approximation has to be made, namely Stirling's approximation of a factorial ($\log n! \cong n \log n - n$), which is only valid for large numbers. An exact expression for the information is derived as

$$ExactInfo(x, x_0, x_1) = \frac{1}{x}\left(\log x! - \log x_1! - \log x_2!\right) \qquad (4)$$

The derivation suggests that the natural logarithm be used in (4). Using the binary logarithm leads to a result that differs by a constant scaling factor that can be ignored. The corresponding definition of information gain is

$$ExactInfoGain(a) = ExactInfo(t, t_0, t_1) - \frac{x}{t} ExactInfo(x, x_0, x_1) - \frac{y}{t} ExactInfo(y, y_0, y_1) \qquad (5)$$

Figure 5.2 compares *InfoGain* (4) and *ExactInfoGain* (5) for a system with x = 250, y = 50, x_1 = 50, and y_1 varying from 0 to 50. Note that the expression for information (4) strictly follows from the equivalence between information and entropy that Shannon noted [15]. In a physical system, entropy is defined as

$$S = k \log \Omega, \qquad (6)$$

where k is Bolzmann's constant and Ω is the number of microstates of the system, leading an expression that is proportional to (4) for a binary system. Since Stirling's approximation is valid for most physical systems and log*n*! could not easily be evaluated for the 10^{23} particles that are typical for physical systems, (2) is more commonly quoted than the exact expression. Note that, in decision tree induction where information gain is used in branches with very few samples, numbers are not necessarily as large as they are in physical systems. It could, therefore, be practical from a computational perspective to use the exact definition. The logarithm of a factorial is, in fact, commonly used for significance calculations in decision tree induction and is, thereby, readily available. The difference in performance between both definitions is clearly noticeable for our algorithm, but unfortunately, we will see that the approximate definition performs better under most circumstances. This poor performance may, however, be due to the setup of our algorithm.

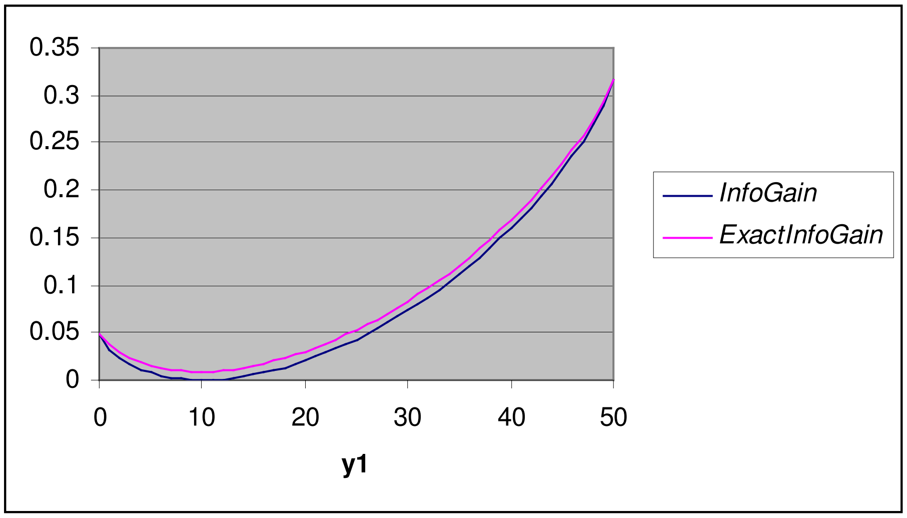

Figure 5.2. *InfoGain* and *ExactInfoGain* for x = 250, y = 50, and $x_1 = 50$.

One noteworthy difference between both formulations of information gain is that (3) vanishes for $x_1/x = y_1/y = t_1/t$, whereas (5) does not, except in the limit of large numbers. In the exact formulation, information is, therefore, gained for any split, which is correct: If a set contains two data points with class label 0 and two with class label 1, then there are $\binom{4}{2} = 6$ ways in which these class labels could be distributed among the four data points. Assume that attribute *a* is 0 for one data point of each class label. The number of configurations after the split will then only be $2\binom{2}{1} = 4$. We have gained information about the precise system of the training points. Whether the exact formulation generally helps reduce the generalization error, which is the quantity of interest in classification of an unknown sample, is yet to be determined.

It is interesting to analyze systematically in what way *ExactInfoGain* differs from *InfoGain*. Figure 5.3 shows the difference *ExactInfoGain*(x,y,x_1,y_1) - *InfoGain*(x,y,x_1,y_1) for x = 250, y = 50, $x_1 = r\ y_1$ with y_1 varying from 1 to 50, and different values for r.

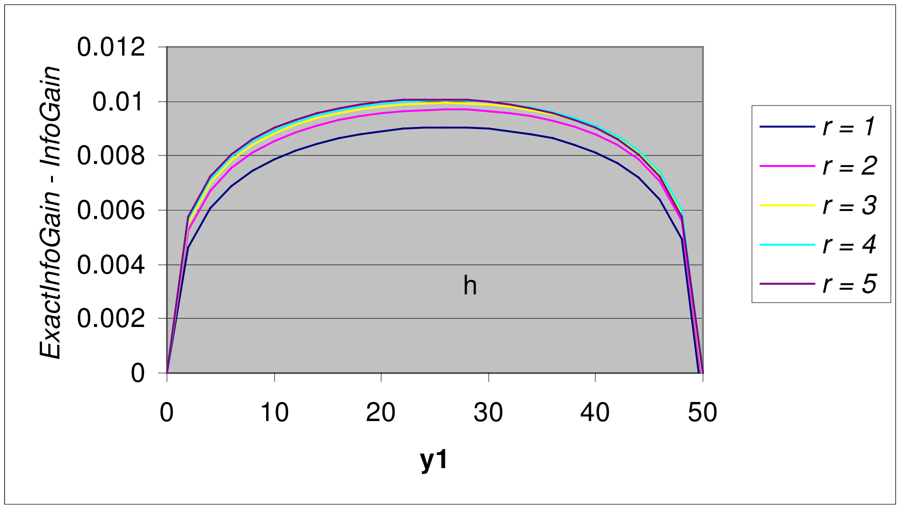

Figure 5.3. Difference between *ExactInfoGain* and *InfoGain.*

It can clearly be seen that *ExactInfoGain* favors attributes that split the training set evenly. Note that the contribution depends very little on the absolute value of information gain. r = 5 corresponds to a split for which *InfoGain* = 0 independently of y_1 while r = 1 corresponds to a split with high information gain (Both *InfoGain* and *ExactInfoGain* are high.) The difference between both, however, differs by less than 10% over the entire range.

It should finally be noted that Shannon [15] strictly introduced the information *Info* based on its mathematical properties, i.e., as a function that satisfies desirable conditions, and only noted in passing that it also was the well-known formulation of physical entropy.

5.3.5. Pruning

It is well known that decision trees have to be pruned to work successfully [1]. Information gain alone is, therefore, not a sufficient criterion to decide which attributes to use for classification. We use statistical significance as a stopping criterion, which is similar to decision tree algorithms that prune during tree construction. The calculation of significance is closely related to that of information gain. Appendix B shows that not only significance, but also information gain can be derived from contingency tables. Information gain is a function of the probability of a particular split under the assumption that a is unrelated to the class label, whereas significance is commonly derived as the probability that the observed split or any more extreme split would be encountered; i.e., it represents the p-value. This derivation explains why information gain is commonly used to determine the best among different possible splits, whereas significance determines whether a particular split should be done at all.

In our algorithm, significance is calculated on different data than information gain to get a statistically sound estimate. The training set is split into two parts, with two-thirds of the data being used to determine information gain and one-third to test significance through Fisher's exact test, where the two-sided test was approximated by twice the value of the one-sided test [16]. A two-sided version was also implemented, but test runs showed that it was slower and did not lead to higher accuracy. An attribute is considered relevant only if it leads to a split that is significant, e.g., at the 1% level. The full set is then taken to determine the predicted class label through majority vote.

5.3.6. Pursuing Multiple Paths

The number of attributes that can be considered in a decision tree like setting while maintaining a particular level of significance is limited. This limitation is a consequence of the "curse of dimensionality" [5]. To get better statistics, it may, therefore, be useful to construct different classifiers and combine them. A similar approach is taken by bagging algorithms [17]. We use a very simple alternative in which several branches are pursued, each starting with a different attribute. The attributes with the highest information gain are picked as starting attributes, and branches are constructed in the standard way from thereon. Note that the number of available attributes places a bound on the number of paths that can be pursued. The votes of all branches are combined into one final vote. This modification leads to a particularly high improvement for data sets with many attributes. For such data sets, each rule only contains a small subset of the attributes. Deriving several rules, each starting with a different attribute, gives some attributes a vote that would not otherwise have one.

5.4. Implementation and Results

We implemented all algorithms in Java and evaluated them on 7 data sets. Data sets were selected to have at least 3000 data points and a binary class label. Two-thirds of the data were taken as a training set and one-third as a test set. Due to the consistently large size of data sets, cross-validation was considered unnecessary. All experiments were done using the same parameter values for all data sets.

5.4.1. Data Sets

Five of the data sets were obtained from the UCI machine learning library [18] where full documentation on the data sets is available. These data sets include the following

- adult data set: Census data are used to predict whether income is greater than $50,000.
- spam data set: Word and letter frequencies are used to classify e-mail as spam.
- sick-euthyroid data set: Medical data are used to predict sickness from thyroid disease.
- kr-vs.-kp (king-rook-vs.-king-pawn) data set: Configurations on a chess board are used to predict if "white can win."
- mushroom data set: Physical characteristics are used to classify mushrooms as edible or poisonous.

Two additional data sets were used. A gene-function data set was generated from yeast data available at the web site of the Munich Information Center for Protein Sequences [19]. This site contains hierarchical categorical information on gene function, localization, protein class, complexes, pathways and phenotypes. One function was picked as a class label, "metabolism." The highest level of the hierarchical information of all properties except function was used to predict whether the protein was involved in "metabolism." Since proteins can have multiple localizations and other properties, each domain value was taken as a Boolean attribute that was 1 if the protein is known to have the localization and 0 otherwise.

A second data set was generated from spatial data. The RGB colors in the photograph of a cornfield are used to predict the yield of the field. The data corresponded to the top half of the data set available at [20]. Class label is the first bit of the 8-bit yield information; i.e., the class label is 1 if yield is higher than 128 for a given pixel. Table 5.1 summarizes the properties of the data sets.

No preprocessing of the data was done. Some attributes, however, were identified as being logarithmic in nature, and the logarithm was encoded in P-trees. The following attributes were chosen as logarithmic: "capital-gain" and "capital-loss" of the adult data set, and all attributes of the "spam" data set.

Table 5.1. Properties of data sets.

	Number of attributes without class label			Number of instances		Un-known values	Prob. of minority class label
	total	catego-rical	numeri-cal	training	test		
adult	14	8	6	30162	15060	no	24.57
spam	57	0	57	3067	1534	no	37.68
sick-euthyroid	25	19	6	2108	1055	yes	9.29
kr-vs-kp	36	36	0	2130	1066	no	47.75
mushroom	22	22	0	5416	2708	yes	47.05
gene-function	146	146	0	4312	2156	no	17.86
crop	3	0	3	784080	87120	no	24.69

5.4.2. Results

Table 5.2 compares the results for our rule-based algorithm with the reported results for C4.5. It can be seen that the standard rule-based implementation tends to give slightly poorer results, whereas using the collective vote from 20 runs that use different attributes as first attributes leads to results that are at least comparable to C4.5. The performance on the spam data set is comparable with the reported result of approximately 7%. A comparison with C4.5 for all data sets is still to be done.

Table 5.2. Results of rule-based classification.

	C4.5	Rule-based	(+/-)	*ExactInfoGain*	20 paths
adult	15.54 [UCI]	15.976	0.325703	15.93	14.934
spam		11.536	0.867191	12.125	7.11
sick-euthyroid		2.849	0.519661	3.318	2.938
kr-vs-kp	0.8 [lazy BR]	1.689	0.398049	1.689	0.844
mushroom	0 [lazy DTI]	0	0	0	0
gene-function		15.538	0.848933	15.816	15.492
crop		18.796	0.156117	20.925	19.034

5.4.3. Rule-based Results Compared with 20 Paths

For the rule-based algorithm a significance level of 1% was chosen. Information gain was calculated on two-thirds of the training data with significance being evaluated on the rest. The final vote was based on the entire training set. Accuracy was generally higher when a maximum of 20 paths were pursued. In this setting, accuracy improved when information gain and significance were evaluated on the full data set with the significance level set to 0.1%.

It can be seen in Figure 5.4 that the difference is particularly large for the spam data set that has many continuous attributes, which is a setting in which the algorithm is expected to work particularly well. For the crop data set, only three paths could be pursued corresponding to the number of attributes.

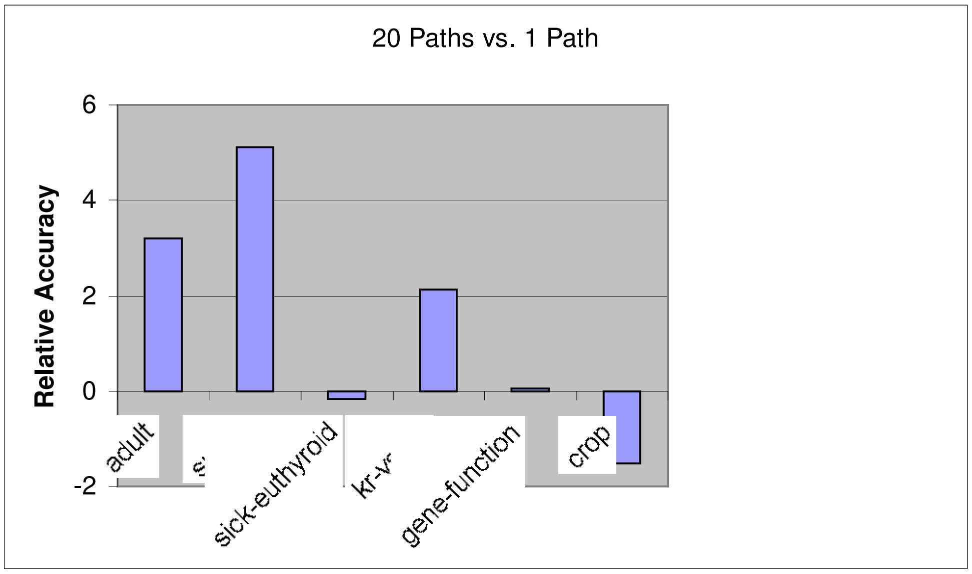

Figure 5.4. Difference between a vote based on 20 paths and a vote based on 1 path in the rule-based algorithm in units of the standard error.

5.4.4. Exact Information Gain

The use of the exact formula for information gain proved to be disappointing as Figure 5.5 shows that results tend to be poorer for the exact information gain formula (5). A reason for this poorer performance may be that our lazy algorithm does not normally benefit from an attribute that splits the data set into similarly large halves. It would, instead, benefit from a split that leaves the available data larger. A standard decision tree, on the other hand, would benefit from such a split because it would keep the tree balanced. It would, therefore, be necessary to evaluate the exact formula within a standard decision tree algorithm to get a final answer as to its usefulness.

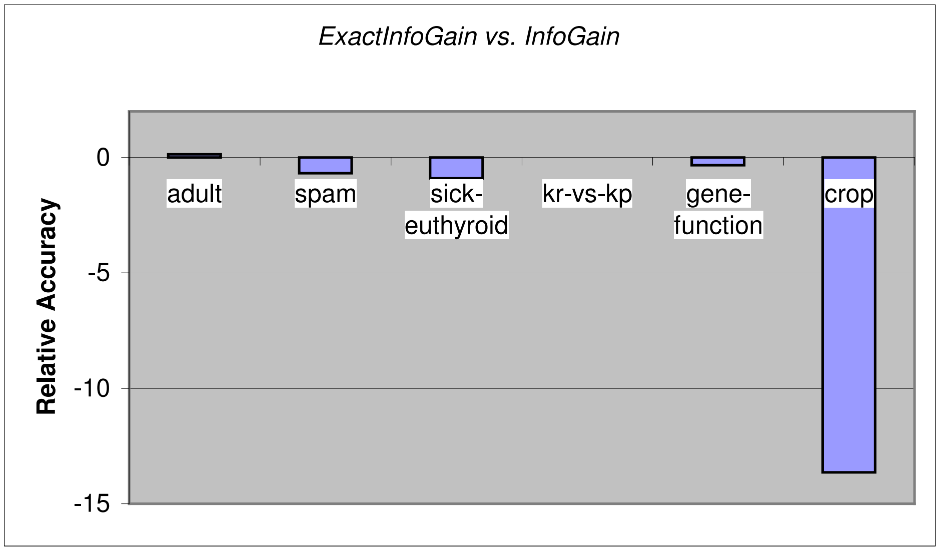

Figure 5.5. Difference between standard and exact information in the rule-based algorithm in units of the standard error.

5.4.5. Performance

Standard decision tree algorithms, whether they are eager or lazy, are based on database scans that scale linearly with the number of data points. The linear scaling is a serious problem in data mining problems that deal with thousands or millions of data points. A main benefit of using of P-trees lies in the fact that the AND operation that replaces database scans benefits from compression at every level. As a consequence, we see a significantly sub-linear scaling. Figure 5.6 shows the execution time for rule-based classification as a function of the number of data points for the adult data set.

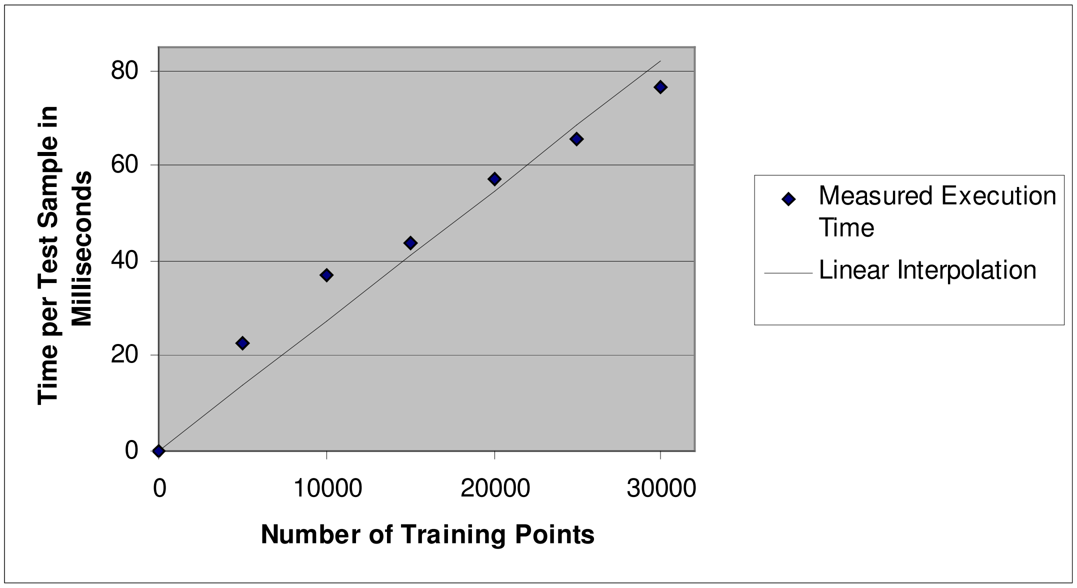

Figure 5.6. Scaling of execution time as a function of training set size.

5.5. Conclusions

We have introduced a lazy rule-based classifier that uses distance information within continuous attributes consistently by considering neighborhoods with respect to the unknown sample. Performance is achieved by using the P-tree data structure that allows efficient evaluation of counts of training points with particular properties. Neighborhoods are defined using the HOBbit distance that is particularly suitable to the bit-wise nature of the P-tree representation. We derive information gain from a 2x2 contingency table and analyze the approximation that has to be done in the process. We, furthermore, show that accuracy of our classifier can be improved by constructing multiple rules. The attributes with highest information gain are determined, and each rule is required to use one of them.

This leads to accuracies that are comparable or better than results of C4.5. Finally, we show that our algorithm has less than O(N) scaling with the number of training points.

5.6. References

[1] J. R. Quinlan, "C4.5: Programs for Machine Learning," Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[2] J. Friedman, R. Kohavi, and Y. Yun, "Lazy Decision Trees," 13th National Conf. on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference, 1996.

[3] Z. Zheng, G. Webb, and K. M. Ting, "Lazy Bayesian Rules: A Lazy Semi-naive Bayesian Learning Technique Competitive to Boosting Decision Trees," 16th Int. Conf. on Machine Learning, pp. 493-502, 1999.

[4] R. Duda, P. Hart, and D. Stork, "Pattern Classification," 2nd edition, John Wiley and Sons, New York, 2001.

[5] D. Hand, H. Mannila, and P. Smyth, "Principles of Data Mining," The MIT Press, Cambridge, MA, 2001.

[6] W. Perrizo, Q. Ding, A. Denton, K. Scott, Q. Ding, and M. Khan, "PINE-Podium Incremental Neighbor Evaluator for Spatial Data Using P-trees," Symposium on Applied Computing (SAC'03), Melbourne, FL, 2003.

[7] N. Cristianini and J. Shawe-Taylor, "An Introduction To Support Vector Machines And Other Kernel-Based Learning Methods," Cambridge University Press, Cambridge, England, 2000.

[8] S. Cost and S. Salzberg, "A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features," Machine Learning, Vol. 10, 57-78, 1993.

[9] A. Perera, A. Denton, P. Kotala, W. Jackheck, W. Valdivia Granda, and W. Perrizo, "P-tree Classification of Yeast Gene Deletion Data," SIGKDD Explorations, Dec. 2002.

[10] R. Kohavi and G. John, "Wrappers for Feature Subset Selection," Artificial Intelligence, Vol. 1-2, pp. 273-324, 1997.

[11] K. Bennett, M. Momma, and M. Embrechts, "A Boosting Algorithm for Heterogeneous Kernel Models," SIGKDD '02, Edmonton, Canada, 2002.

[12] M. Khan, Q. Ding, and W. Perrizo, "K-nearest Neighbor Classification of Spatial Data Streams Using P-trees," Pacific Asian Conf. on Knowledge Discovery and Data Mining (PAKDD-2002), Taipei, Taiwan, May 2002.

[13] Q. Ding, W. Perrizo, and Q. Ding, "On Mining Satellite and Other Remotely Sensed Images," Workshop on Data Mining and Knowledge Discovery (DMKD-2001), Santa Barbara, CA, pp. 33-40, 2001.

[14] W. Perrizo, W. Jockheck, A. Perera, D. Ren, W. Wu, and Y. Zhang, "Multimedia Data Mining Using P-trees," Multimedia Data Mining Workshop, Conf. for Knowledge Discovery and Data Mining, Edmonton, Canada, Sept. 2002.

[15] C. Shannon, "A Mathematical Theory of Communication," Bell Systems Technical Journal, Vol. 27, pp. 379-423 and 623-656, Jul. and Oct. 1948.

[16] W. Ledermann, "Handbook of Applicable Mathematics," Vol. 6, Wiley, Chichester, England, 1980.

[17] L. Breiman, "Bagging Predictors," Machine Learning, Vol. 24, No. 2, pp. 123-140, 1996.

[18] C.L. Blake and C.J. Merz, "(UCI) Repository of Machine Learning Databases," Irvine, CA, 1998 http://www.ics.uci.edu/~mlearn/MLSummary.html, accessed Oct. 2002.

[19] Munich Information Center for Protein Sequences, "Comprehensive Yeast Genome Database," http://mips.gsf.de/genre/proj/yeast/index.jsp, accessed Jul. 2002.

[20] W. Perrizo, "Satellite Image Data Repository," Fargo, ND http://midas-10.cs.ndsu.nodak.edu/data/images/data_set_94/, accessed Dec. 2002.