Selecting Features by Vertical Compactness of Data *

Ke Wang and **Suman Sundaresh**
Department of Information Systems and Computer Science
National University of Singapore
wangk@iscs.nus.edu.sg, sumans@iscs.nus.edu.sg

Abstract

Feature selection is a data preprocessing step for classification and data mining tasks. Traditionally, feature selection is done by selecting a minimum number of features that determine the class label, i.e., by the *horizontal compactness* of data. In this paper, we propose a new selection criterion that aims at the *vertical compactness* of data. In particular, we select a subset of features that yields the least number of projected instances while determining the class label. A hybrid search that is partially DFS and partially BFS is proposed to exploit the pruning potential of the problem. We compare the result induced by C4.5 before and after the feature selection.

Introduction

Traditionally, feature selection is done by selecting fewest features that determine the class label. See, for example, Almuallim and Dietterich 1994, Schlimmer 1993. (There are other definitions of feature selection, such as Kira and Rendell 1992, Koller and Sahami 1996, e.g.) One obvious problem is that in a medical diagnosis task, the patient's social security number (SSN) would be identified as one solution because SSN determines the diagnosis. The rules produced using only SSN would not be able to predict the diagnosis for an unseen patient. The poor generalization lies in the fact that the selection of SSN yields no reduction in the number of projected instances. Of course, one could first remove SSN before the feature selection starts because SSN is well known to have poor generalization. However, in many other less obvious cases, the choice of which features to be removed is not always straightforward. Answering this question is exactly the task of feature selection.

In this paper, we propose a new feature selection criterion that aims at the maximum *vertical compactness* of the data, in contrast to the traditional *horizontal compactness*. For some given tolerance of noise level γ, we select a subset of features that yields fewest projected instances without exceeding the tolerance γ. This criterion returns the most vertically compact subset for the given tolerance. In fact, the more vertically compact the subset is, the more duplicates in the selected features map to the same class, therefore, the more likely the selected features are relevant in describing the class. In the above medical diagnosis example, it is obvious that SSN would be among the first not to be selected since it gives no reduction in the number of projected instances.

The rest of the paper defines the new feature selection criterion, considers issues involved in searching for a solution, and studies the performance of the criterion.

The MIN_INSTANCE Criterion

The user usually has some idea about how much noise in the dataset can be tolerated when performing feature selection. Subject to such tolerance, the feature selection is to select a subset of features that produces the least number of projected instances. First, let us define what is meant by the noise of the data.

Two instances are *inconsistent* if they match except for the class label. For a set of matching instances, the *inconsistency count* is the number of instances in the set minus the number of instances belonging to a major class in the set. For example, there are n matching instances, among them, I_1 instances belong to one class, I_2 instances belong to another class, and I_3 instances belong to the third class, where $I_1+I_2+I_3 = n$. If I_3 is the largest among the three, the inconsistency count is $n - I_3$. The *inconsistency rate* is the sum of all inconsistency counts (for all matching patterns) divided by the total number of instances. In other words, the inconsistency rate measures the misclassification caused by changing minority classes of matching instances to a majority class. The *data size* of a subset of features refers to the number of distinct instances

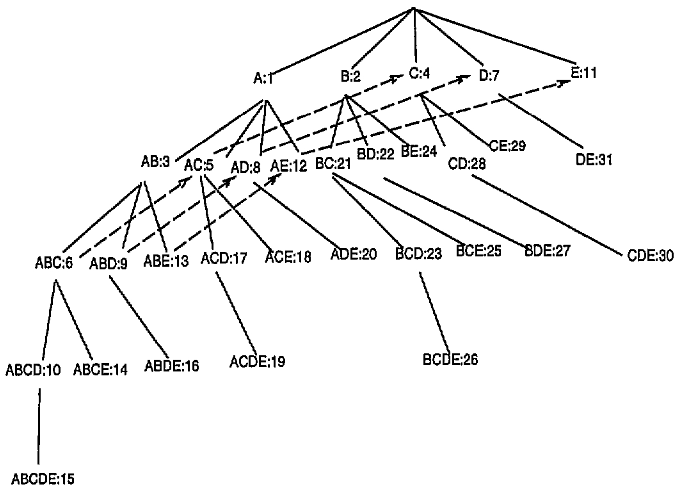

Figure 2: Pruning potential

Hybrid Search Algorithm

We present a search algorithm for HS. At the top level, the search order of HS is guided by DFS. To enforce Principle 1, however, a checking of the logical parent is interleaved with DFS. On visiting a node v (whose physical parent has obviously been evaluated), HS checks the status of the logical parent $v1$ of v, which must be one of *pruned, evaluated, unevaluated.* If $v1$ is unevaluated, $v1$ is pushed onto the stack and the checking goes to the logical parent of $v1$ recursively (note that the physical parent of $v1$ has been evaluated). This repeats until encountering a logical parent that is either pruned or evaluated. The unwinding stage of the recursion pops each node from the stack and evaluates it. When the stack becomes empty, the computation comes back to v and the DFS resumes and determines the next node to visit from unpruned nodes. Instead of presenting the algorithm, we illustrate the idea using an example.

Example 1 *Consider Figure 2. The number at each node denotes the order in which the node is visited following Principles 1 and 2. Let us consider how Prunings I and II are exercised. HS starts with A. AB is the next node to visit in the DFS order. Principle 1 requires to evaluate the logical parent of AB first, i.e., B, and then AB itself. ABC is visited next in the DFS order. The parent checking at ABC pushes ABC, AC, C onto the stack, as indicated by the dotted line starting at ABC, and evaluates C, AC, ABC in the unwinding stage. Suppose that ABC is the current best suboptimal solution. From Pruning I, all nodes under ABC are pruned. ABD is the next unpruned node to be visited in the DFS order. The parent checking at ABD pushes ABD, AD, and D onto the stack. Assume that the data size of D is larger than the current best size (i.e., the data size of ABC). In the unwinding stage, D, AD, ABD are simply popped off the stack and tagged as pruned.*

ABE is the next unpruned node in the DFS order. The parent checking at ABE pushes ABE, AE, E onto the stack. Assume that the data size of E is larger than the current best size, E, AE, ABE all are tagged as pruned and popped off the stack. ACD is visited next. Since the logical parent AD was pruned before, ACD is pruned immediately. ACE is visited next. Similarly, since its logical parent AE was pruned before, ACE is pruned immediately. Then BC becomes the next unpruned node in the DFS order. This traversal goes on until no more unpruned can be reached.

The worst case for HS algorithm is when all features are relevant and no node can be pruned. In this case, $2^n - 1$ nodes will be evaluated, where n is the number of original features. In most cases, however, the class label depends only on a small number of features and much fewer nodes are evaluated.

Experiments

We tested the proposed feature selection on a few datasets in UCI Repository (Murphy and Aha 1994). Our focus is the data size reduction and the quality of rules induced.

Features selected

Table 1 shows the features selected and the data size before and after feature selection. The last three datasets have continuous features. All datasets use consistency rate $\gamma = 0$, except Monk3(5%) which uses $\gamma = 5\%$ because the dataset contains 5% noise. If the training data and testing data are separately specified, such as all Monks, feature selection is applied to the training data; otherwise, to the entire dataset.

For the first seven datasets for which the relevant features are known, exactly the relevant features are selected, as in the table. For all datasets, only a proper subset of original features is selected. Except for Tic-Tac-Toe, there is always some reduction in the data size after feature selection, with Led 7 having the most reduction, i.e., from 2000 instances to 10 instances. For Abalone and Pima, though the reduction in data size is small, three out of the original eight are selected, reducing the data volume by more than 60%.

We have chosen DFS and BFS coupled with Prunings I and Pruning II, denoted DFS(P) and BFS(P), as the benchmark for comparison. The last three columns in Table 1 give the number of dataset scans performed. In most cases, HS makes fewer scans than DFS(P) and BFS(P). For the three Monks, Parity3+3, and Pima,

Dataset	Size	# of Features	Features Selected	New Size	Scan Hybrid	Scan DFS(P)	Scan BFS(P)
Monkl	124	7	$A_1A_2A_5$	35	66	107	94
Monk2	169	7	all except the ID	169	69	126	126
Monk3(5%)	133	7	$A_2A_4A_5$	39	54	101	92
CorrAL	64	6	$A_0A_1B_0B_1$	16	61	61	60
Parity5+5	200	10	$A_1A_2A_3A_4A_5$	32	677	683	672
Parity3+3	64	12	$A_1A_2A_3$	8	320	397	397
Tic-Tac-Toe	958	9	all but A_2	958	510	510	510
Led 7	2000	7	$A_1A_2A_3A_4A_5$	10	124	124	124
Bupa	345	6	$A_3A_4A_5$	337	43	51	51
Abalone	4177	8	$A_2A_6A_8$	4175	79	81	95
Pima	768	8	$A_1A_4A_7$	764	121	164	156

Table 1: Selected features and cost comparison

the reduction in the number of scans ranges between 19% and 45%, and for Tic-Tac-Toe and Led 7, the three methods make the same number of scans. It is expected that HS will prune more scans on real datasets where there are more irrelevant features.

Effectiveness of classification

We run C4.5 (Quinlan 1993) on the above datasets with and without feature selection. Duplicate instances produced by feature selection are kept in running C4.5. In practice, duplicates can be represented by a single copy and a duplication number. For each dataset, the 10-fold cross validation was applied to the whole dataset projected on the features selected in Table 1. Table 2 summarizes the size of pruned decision trees and the error rate on test data. In general, selected features induce better decision trees than original features. For CorrAl (John, Kohavi and Pfleger 1994), Bupa, and Abalone, both tree size and error rate are reduced after feature selection. Without feature selection the decision tree for CorrAl picks the correlated feature C as the root. With feature selection the correlated feature (C) and the irrelevant feature (I) are removed and the induced rules capture the target concept, i.e., $(A_0 \wedge B_0) \vee (A_1 \wedge A_2)$.

For the two Parity datasets, a bigger decision tree is induced, but the error rate is reduced from 40% to nearly 0. Since the error rate is collected on test data, this improvement is not always achievable by having a larger tree. In the case of Tic-Tac-Toe, the tree size has reduced significantly by having a slightly higher error rate. The feature selection does not affect Monk2, Monk3, and Led 7 because C4.5 selects exactly same features. For Monkl, the tree gets slightly bigger but the error rate drops to 0. Unlike discrete features, continuous features have much fewer repeating values, a direct feature selection is less effective, as shown by the last three datasets.

Dataset	Tree Size		Error Rate (%)	
	Bef. FS	Aft. FS	Bef. FS	Aft. FS
Monkl	38.6	41.0	1.1	0.0
Monk2	1.0	1.0	32.9	32.9
Monk3(5%)	19.0	19.0	0.0	0.0
CorrAL	14.6	13.0	6.0	0.0
Parity 5+5	41.8	62.4	45.0	2.0
Parity 3+3	13.4	15.0	39.5	0.0
Tic-Tac-Toe	133.0	116.2	13.8	14.7
Led 7	19.0	19.0	0.0	0.0
Bupa	77.6	55.8	39.1	35.9
Abalone	2174.5	1882.2	79.9	79.0
Pima	125.4	136.2	29.5	38.1

Table 2: Results of C4.5

References

Almuallim, H.; & Dietterich, T. G. 1994. Learning boolean concepts in the presence of many irrelevant features. Artificial Intelligence, 69 (1-2), 279-305.

John, G. H.; Kohavi, R.; & Pfleger, K. 1994. Irrelevant features and the subset selection problem. In Proceedings of the Eleventh International Conference on Machine Learning, 121-129.

Kira, K.; & Rendell, L. A. 1992. The feature selection problem: Traditional methods and a new algorithm. In Proceedings of Ninth National Conference on AI, 129-134.

Koller, D.; & Sahami, M. 1996. Toward optimal feature selection. In Machine Learning: Proceedings of the Thirteenth International Conference.

Murphy, P.; & Aha, D. 1994. Repository of Machine Learning Databases. http://www.ics.uci.edu/mlearn/MLRepository.html

Quinlan, J. R. 1993. C4.5 : Programs for Machine Learning.

Schlimmer, J. C. 1993. Efficiently Inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In Proceedings of Tenth International Conference on Machine Learning, 284-290.