# SCALABLE FEATURE SELECTION FOR LARGE SIZED DATABASES

**Huan Liu & Rudy Setiono**
**Department of Information Systems and Computer Science**
**National University of Singapore**
**Kent Ridge, Singapore 119260**
**{liuh,rudys}@iscs.nus.sg**
**Tel: (+65) 772-6563; Fax: (+65) 779-4580**
March 16-20, 1998
WCES'98, Mexico City

## ABSTRACT

Feature selection determines relevant features in the data. It is often applied in pattern classification. A special constraint for feature selection nowadays is that the size of a database is normally very large. An effective method is needed to accommodate the practical demands. A scalable probabilistic algorithm is presented here as an alternative to the exhaustive and heuristics approaches. The scalable probabilistic algorithm is designed and implemented to meet the needs arising from real-world data mining applications. Through experiments, we show that (1) the probabilistic algorithm is effective in obtaining optimal/suboptimal subsets of features; (2) its scalable version expedites feature selection further and can scale up without sacrificing the quality of selected features.

## INTRODUCTION

The problem of feature selection can be defined as finding $M$ relevant attributes among the $N$ original attributes, where $M \leq N$, to describe the data in order to minimize the error probability or some other reasonable selection criteria. This problem has long been an active research topic within statistics and pattern recognition [17, 4], but most work in this area has dealt with linear regression [7] and is under assumptions that do not apply to most learning algorithms [5]. Researchers pointed out that the most common assumption is monotonicity, that increasing the number of features can only improve the performance of a learning algorithm[1]. Recently feature selection has received considerable attention from machine learning and knowledge discovery researchers interested in improving the performance of their algorithms and in cleaning data. In handling large databases, feature selection is even more important since many learning algorithms may falter or take too long to run before the data is reduced.

[1]The monotonicity assumption is not valid for many induction algorithms used in machine learning. See for dataset 1 (CorrAL) in Section 4 which is reproduced from [5].

Most feature selection methods (refer to [6, 7, 5]) can be grouped into two categories: exhaustive or heuristic search of an optimal set of $M$ attributes. For example, Almuallim and Dietterich's FOCUS algorithm [1] starts with an empty feature set and carries out exhaustive search until it finds a minimal combination of features that are sufficient to construct a hypothesis consistent with a given set of examples. It works on binary, noise-free data. Its time complexity is $O(\min(N^M, 2^N))$. They proposed three heuristic algorithms to speed up the searching.

There are many heuristic feature selection algorithms. The Relief algorithm [6] assigns a "relevance" weight to each feature, which is meant to denote the relevance of the feature to the target concept. Relief samples instances randomly from the training set and updates the relevance values based on the difference between the selected instance and the two nearest instances of the same and opposite classes. According to Kira and Rendell, Relief assumes two-class classification problems and does not help with redundant features. If most of the given features are relevant to the concept, it would select most of them even though only a fraction are necessary for concept description. The PRESET algorithm [10] is another heuristic feature selector that uses the theory of Rough Sets to heuristically rank the features, assuming a noise-free binary domain. In order to consider higher order information among the attribute, Liu and Wen suggest [8] to use high order information gains to select features. Since the last two algorithms do not try to explore all the combinations of features, it is certain that they fail on problems whose attributes are highly correlated such as the parity problem where the combinations of a small number of attributes do not help in finding the relevant attributes. Another common understanding is that some learning algorithms have built-in feature selection, for example, ID3 [13], FRINGE [11] and C4.5 [14]. The results in [1] suggest that one should not rely on ID3 or FRINGE to filter out irrelevant features.

To sum up, the exhaustive search approach is infeasible in practice; and the heuristic search approach can reduce the search time significantly, but will fail on hard problems (e.g., the parity problem) or cannot remove redundant attributes. A probabilistic approach is proposed as an alternative that and can select the optimal/suboptimal subset(s) of relevant features. In the context of large sized databases, however, it would still take considerably long time to check if a subset is valid or not[2]. As our probabilistic system was dispatched to a local institute[3] for on-site usage, all the evidence showed that reducing data in size can significantly speed up the selection. Hence, a scalable probabilistic method is designed and implemented. In the following, we describe the probabilistic method first, then the scalable one, followed by an empirical study and a conclusion.

## PROBABILISTIC FEATURE SELECTION

The proposed probabilistic approach is a Las Vegas Algorithm [3]. Las Vegas algorithms make probabilistic choices to help guide them more quickly to a correct solution. Another similar type of algorithms is Monte Carlo algorithms in which it is often possible to reduce the error probability arbitrarily at the cost of a slight increase in computing time (refer to page 341 in [3]). One kind of Las Vegas algorithms uses randomness to guide their search in such a way that a correct solution is guaranteed even if unfortunate choices are made. As we mentioned earlier, heuristic search methods are vulnerable to the datasets of high order correlations. Las Vegas algorithms free us from worrying about such situations by evening out the time required on different situations. The time performance of a Las Vegas algorithm may not be better than that of some heuristic algorithms.

The LVF[4] algorithm below generates a random subset, $S$, from $N$ features in every round. If the number of features (C) of $S$ is less than the current best, i.e., $C < C_{best}$, the data $D$ with the features prescribed in $S$ is checked against the inconsistency criterion (to be explained later). If its inconsistency rate is below a pre-specified one ($\gamma$), $C_{best}$ and $S_{best}$ are replaced by $C$ and $S$ respectively; the new current best ($S$) is printed. If $C = C_{best}$ and the inconsistency criterion is satisfied, then an equally good current best is found and printed. MAX_TRIES is set to $77 \times N$ in our experimental study [5] following the rule-of-thumb that the more attributes a dataset has (in other words, the larger $N$ is), the harder the problem of feature selection (parity-5 is more difficult than parity-2, e.g.), and hence more tries are needed. When LVF loops MAX_TRIES times, it stops.

The *inconsistency* criterion (InconCheck($S, D$) $< \gamma$) is the key to LVF. The criterion specifies to what extent the dimensionally reduced data can be accepted. The inconsistency rate of the data described by the selected features is checked against a pre-specified rate ($\gamma$). If it is smaller than $\gamma$, it means the dimensionally reduced data is acceptable. The default value of $\gamma$ is 0 unless otherwise specified. The inconsistency rate of a dataset is calculated as follows: (1) two patterns are considered inconsistent if they match all but their class labels; (2) the inconsistency count is the number of all the matching patterns minus the largest number of patterns of different class labels: for example, there are $n$ matching patterns, among them, $c_1$ patterns belong to label$_1$, $c_2$ to label$_2$, and $c_3$ to label$_3$ where $c_1 + c_2 + c_3 = n$. If $c_3$ is the largest among the three, the inconsistency count is $(n - c_3)$; (3) the inconsistency rate is the sum of all the inconsistency counts divided by the total number of patterns ($\mathcal{N}$).

The inconsistency criterion is in line with information-theoretic considerations [16] which suggest that using a good feature of discrimination provides compact descriptions of each of the two classes, and that these descriptions are maximally distinct. Geometrically, this constraint can be interpreted [9] to mean that (i) such a feature takes on nearly identical values for all examples of the same class, and (ii) it takes on some different values for all examples of the other class. The inconsistency criterion aims to keep the discriminating

[2] The checking can be done in $O(\mathcal{N})$, where $\mathcal{N}$ is the number of patterns, by using a hashing method.

[3] Japan-Singapore AI Centre, Singapore.

[4] F stands for a filter version of Las Vegas algorithms.

[5] 77 is chosen by experiment. MAX_TRIES can be defined according to applications in hand or based on the experience from experimentation. Too small or too big a MAX_TRIES will affect the performance of LVF. The compromise is made between good and fast solutions. The longer LVF runs, the better its results are. Refer to the analysis below.

power of the data for multiple classes after feature selection. In other words, the criterion is supported by information-theoretic considerations.

---

**LVF algorithm**
**Input:**
  MAX-TRIES, $\gamma$ - allowable inconsistency rate,
  $D$ - dataset of $N$ attributes;
**Output:**
  sets of $M$ features satisfying the inconsistency criterion

```
Cbest = N;
for i=1 to MAX-TRIES
   S = randomSet(seed);
   C = numOfFeatures(S);
   if (C < Cbest)
     if (InconCheck(S, D) < γ)
        Sbest = S; Cbest = C;
        print_Current_Best(S) ;
     else if
      ((C = Cbest) ∧ (InconCheck(S, D)< γ))
     print_Current_Best(S)
end for
```

---

Some analysis shows that LVF can give a good solution, or an optimal solution if MAX_TRIES is sufficiently large. With a good pseudo random number generator [12], the selection of an optimal subset of $M$ features can be considered non-replacement experiments. The probability of finding the optimal subset at the $(k+1)$th experiment is $\frac{1}{2^N-k}$, and the probability of having to conduct $(k+1)$ experiments before finding the optimal subset is still $\frac{2^N-1}{2^N} \times \frac{2^N-2}{2^N-1} \times ... \times \frac{1}{2^N-k} = \frac{1}{2^N}$, where $N$ is the number of original features. When $N$ is large, MAX_TRIES $\ll 2^N$. Here we assume there is only one optimum. If there exist $l$ optima as in many applications, at the $(k+1)$th tossing, the probability of finding one optimum is $\frac{l}{2^N-k}$.

## SCALABLE PROBABILISTIC FEATURE SELECTION

Although LVF can generate optimal/suboptimal solutions (see the experimental results below), when datasets are huge, checking whether a dataset is consistent or not still takes time due to its $O(\mathcal{N})$ complexity. It is only natural to think about a scalable version of LVF that can significantly reduce the number of checkings. Studying the LVF algorithm, we notice that if we reduce the data, we can decrease the number of checkings. However, features selected from the reduced data may not be suitable for the whole data. The following algorithm is designed to achieve that features selected from the reduced data will not generate more inconsistencies than using the whole data. Furthermore, this can be done without sacrificing the quality of features which is measured by the number of features and whether they are relevant or not.

The scalable algorithm (LVS) starts with a portion of data ($p$%) and a given allowable inconsistency rate ($\gamma$, usually set to 0 if there is no prior knowledge). It splits the data $D$ into $D_0$ and $D_1$ where $D_0$ is $p$% of $D$ and $D_1$ is the remaining. It finds a subset of features (*subset*) for $D_0$ and checks *subset* on $D_1$. If the inconsistency rate does not exceed $\gamma$, LVS stops. Otherwise, it appends those patterns (*inconData*) of $D_1$, which cause the inconsistency, to $D_0$, and deletes *inconData* from $D_1$. The selection process repeats until a solution is found. If no subset is found, the whole set of attributes is returned as a solution. In LVS, checkIncon() is similar to InconCheck() of LVF, but also saves the inconsistent patterns of $D_1$ to *inconData*. The experiments below are designed to demonstrate the claims made above on LVS.

---

**LVS algorithm**
**Input:**
  $p$ - the percentage of the data used for feature selection,
  $D$ - dataset of $N$ attributes, $\gamma$ - the allowable inconsistency rate;
**Output:**
  sets of $M$ features satisfying the inconsistency criterion

```
D0 = p% of D randomly chosen ;
D1 = D − D0 /* the rest data */
loop
   LVF (D0, γ, subset)
   if (checkIncon(subset,D1,inconData)≤ γ)
     return subset;
   else
     append(inconData, D0);
     remove(inconData, D1);
end loop
```

---

## EMPIRICAL STUDY

Two sets of data are used. The first set is used to show the effectiveness of LVF. These datasets are either commonly used for testing feature selection

| Dataset | N | S | $S_d$ | $S_t$ |
|---|---|---|---|---|
| Credit | 15 | 690 | 490 | 200 |
| Labor | 16 | 57 | 40 | 17 |
| Vote | 16 | 435 | 300 | 135 |
| Mushroom | 22 | 8125 | 7125 | 1000 |
| ParityMix | 20 | $2^{20}$ | 10,000 | - |

TABLE 1. Notations: N - the number of attributes, S - the size of the dataset, $S_d$ - the size of the training data, $S_t$ - the size of the testing data. Training and testing datasets are split randomly if not specified.

algorithms or artificially designed so that relevant attributes are known. This set consists of both artificial and real-world data. The artificial datasets are (1) **CorrAL** The data set was designed in [5]. There are six binary features, $A_0, A_1$, $B_0, B_1, I$, and $C$. Feature $I$ is irrelevant, feature $C$ is correlated to the class label 75% of the time. The Boolean target concept is $(A_0 \wedge A_1) \vee (B_0 \wedge B_1)$. Both ID3 and C4.5 chose feature $C$ as the root. This is an example of datasets in which if a feature like $C$ is removed, a more accurate tree will result; (2) **Monk1, Monk2, Monk3** [15], these datasets of 6 attributes can be used to show that a feature selector only selects the relevant features, while keeping all the relevant ones; and (3) **Parity5+5**, the target concept is the parity of five bits. The dataset contains 10 features, of which 5 are uniformly random (irrelevant) ones. The training set contains 100 patterns randomly selected from all 1024 patterns. Most heuristic feature selectors will fail on this sort of problems since an individual feature is usually not a good indicator. The real-world datasets are (4) **Credit** (or CRX), (5) **Labor**, (6) **Vote**, and (7) **Mushroom**. The choices of training and testing datasets (4 - 6) are the same as in [5]. The details are in Table 1. The Mushroom data is randomly separated into training and testing sets. Since most datasets above are relatively small, we only choose Vote and Mushroom plus ParityMix to show the effectiveness of LVS in relation to the size of a dataset below. ParityMix is composed by having two Parity5+5's side by side.

### A. Effectiveness of LVF

For the artificial datasets, the evaluation of LVF is simple since the relevant attributes are known. However, for the real-world datasets, it is not clear what the relevant features are. Therefore, whether the selected features are relevant or not can only be determined indirectly. One way is to see the effect of feature selection through a learning algorithm. Among many choices, we chose C4.5 [14] and ID3 [13] in our experiments because (1) C4.5 works well on most data sets as reported by many researchers; and (2) it employs a heuristic to find simplest tree structures. LVF is run 100 times on each training dataset. The numbers of selected features and frequencies are reported in Table 2 under the condition that the inconsistency criterion be satisfied. Also reported is a sample of these selected features for each dataset which can be directly used by readers.

For the artificial datasets, the relevant attributes are always selected, albeit a few of irrelevant ones are also chosen sometimes. For the problem like Parity5+5, LVF correctly identifies the correct attributes all the time, plus one irrelevant attribute sometimes. For the real-world datasets, the number of attributes is reduced at least by half to less than one fifth of the original. Table 2 shows that those features in the last column are necessary in order to satisfy the inconsistency criterion (the inconsistency rate is 0 except for Monk3). These features are used by C4.5 to test if its performance improves comparing to C4.5 using all attributes. Ten-fold cross validation is applied and the default setting of C4.5 is used in the experiment. For the experiments of "after feature selection", only the features shown in the last column of Table 2 are used. Given in Table 3 are the average accuracy rates of C4.5 before and after applying feature selection to the datasets. ID3 results are also given by taking unpruned trees of C4.5.

Results in Table 3 suggest that the performance of both ID3 and C4.5 has improved in general. That is, the tree size is getting smaller and the accuracy higher. For the artificial datasets, this experiment further shows that with the relevant attributes, ID3 and C4.5 are doing better than that with the full set of attributes. For the real-world datasets, ID3 and C4.5 are also doing better with the selected attributes. This indicates that LVF has selected relevant attributes for these datasets. In particular, ID3 and C4.5 did poorly on Parity5+5 before feature selection. Nevertheless, ID3 and C4.5 did as well on Mushroom with 22 attributes as with 4 attributes. This demonstrates that ID3 and C4.5 do select relevant features for some datasets, though not for all.

[6] Allowing 5% inconsistency. If not, four attributes are selected: the above chosen 3 plus A1.

| Dataset | # Att | # Selected Att (Frequency) | Features (1 minimum set) |
|---|---|---|---|
| CorrAL | 6 | 4 (86), 5 (14) | A0, A1, B0, B1 |
| Monk1 | 6 | 3 (100) | A1, A2, A5 |
| Monk2 | 6 | 6 (100) | A1 - A6 |
| Monk3[6] | 6 | 3 (100) | A2, A4, A5 |
| Parity5+5 | 10 | 5 (71), 6 (29) | A1 - A5 |
| Credit (CRX) | 15 | 5 (100) | A2, A3, A4, A9, A14 |
| Labor | 16 | 3 (44), 4 (56) | A2, A10, A11 |
| Vote | 16 | 8 (7), 9 (53), 10 (30), 11 (10) | A1 - A4, A9, A11, A13, A16 |
| Mushroom | 22 | 4 (57), 5 (43) | A4, A5, A12, A22 |

TABLE 2. Results of 100 runs of LVF on the datasets with one example of the minimum set of features for each dataset.

| | ID3 | | | | C4.5 | | | |
|---|---|---|---|---|---|---|---|---|
| | TreeSize | | ErrorRate(%) | | TreeSize | | ErrorRate (%) | |
| Dataset | Before | After | Before | After | Before | After | Before | After |
| CorrAL | 10.6 | 11.4 | 37.5 | 20.8 | 10.6 | 11.4 | 37.5 | 20.8 |
| Monk1 | 103.7 | 41.0 | 4.5 | 0.0 | 43.0 | 41.0 | 0.7 | 0.0 |
| Monk2 | 217.4 | 217.4 | 37.0 | 37.0 | 16.3 | 16.3 | 21.1 | 21.1 |
| Monk3 | 20.6 | 19.0 | 1.4 | 1.1 | 19.0 | 19.0 | 1.1 | 1.1 |
| Parity5+5 | 79.0 | 62.8 | 48.5 | 1.5 | 47.0 | 62.8 | 45.5 | 1.5 |
| Credit (CRX) | 139.8 | 144.8 | 16.8 | 19.7 | 54.0 | 38.0 | 15.5 | 15.2 |
| Labor | 15.2 | 8.4 | 15.3 | 14.0 | 7.3 | 7.0 | 17.3 | 14.3 |
| Vote | 29.8 | 27.1 | 5.3 | 5.3 | 14.5 | 6.1 | 5.3 | 5.5 |
| Mushroom | 29 | 29 | 0.0 | 0.0 | 29 | 29 | 0.0 | 0.0 |

TABLE 3. 10-fold cross validation results on Tree Size and Error Rates of ID3 and C4.5 before and after applying LVF to the datasets.

### B. Effectiveness of LVS

For this set of experiments, we want to verify three claims: (1) LVS may not be suitable for small datasets; (2) LVS can run faster than LVF on large datasets; and (3) LVS does not sacrifice the quality of the selected features. The experiments are conducted on a dedicated Sun Sparc20 as follows. For each dataset, starting with 10% of the data ($D_0$) for feature selection, we run 30 times of LVS, recording the number of attributes, attributes, and selection time in each run. Subsequently, we do the same experiments with 20%, 30%, ..., 90%, and 100% of the data. The average time and number of attributes are computed for each experiment. Using 100% of the data as the reference, we calculate the P-values for each sized $D_0$. The lower a P-value is, the more likely the NULL hypothesis that the two averages are the same will be rejected. Refer to Figures 1 and 2 for the details.

We observe the following:

1. The effectiveness of LVS becomes more obvious when the data size is larger. LVS performs well on all the three datasets. If the data size is small (around a few hundred) as in Vote even the time saving for the best $D_0$ is not much. However, the saving is significant in the case of ParityMix and a clear trend can be observed in Figure 1. This is due to the overheads required by scalable feature selection. Since our inconsistency checking is fast ($O(\mathcal{N})$), if $\mathcal{N}$ is not sufficiently large, the time saving will not be apparent, may be negative if $\mathcal{N}$ is too small. This is why LVS is more suitable for large sized datasets.

2. Another issue is the number of patterns with which LVS should start for a dataset. Either too few or too many patterns affects the LVS's performance. If too few patterns are used ($D_0$ is too small), LVF could select few features that cannot pass the inconsistency check on the remaining data ($D_1$), that is, *inconData* can be large. The worst case is that after the first loop, $D_0$ becomes $D$ (the whole dataset). One case of too small a $D_0$ can be observed in Figure 1 for Vote when 10% of the data was used; it took longer time than using 20% - 50% of $D$. If too many patterns are used ($D_0$ is large), the overheads (i.e., the time spent on those steps inside the loop of the LVS algorithm after the LVF call) plus the time on LVF may exceed the time of simply running LVF on $D$. In the cases of Vote and Mushroom, the difference in times is not statistically significant when 70% or more of $D$ is used.

3. The scalable algorithm does not have to sacrifice the quality of feature selection. The time saving is mainly due to (1) small $D_0$ with respect to large $D$, and (2) by remembering inconsistent patterns, LVS can avoid checking wrong guesses twice. The quality is measured here in two dimensions. One is the number of attributes, and the other is the relevance of the features. As shown in Figure 2, if there is some statistically significant difference (light grey bars) in the number of attributes selected between various sized $D_0$'s against $D$, the numbers of attributes are lower than that using 100% of the data; otherwise, there is no statistically significant difference according to $t$-test. For ParityMix, the relevant 5 attributes are always selected plus 1 or 2 irrelevant/redundant ones. For the Vote and Mushroom, the relevance test is done through a learning algorithm (C4.5 and ID3 here). If their performances do not deteriorate or even improve, we conclude that these features are relevant. Some experimental results shown in Table 3 have verified the sets of features for the two datasets via 10-fold cross validations. When the quality of the selected features can be warranted, the reduction can simplify data analysis, rule induction, as well as data collection in future.

4. LVS can easily scale up. Time complexity of a feature selection algorithm can be described along two dimensions: number of attributes ($N$) and number of patterns ($\mathcal{N}$). By approximating MAX_TRIES of LVF with $77 \times N$ (reduced from $2^N$), the time complexity of LVF is mainly determined by $\mathcal{N}$ since $N$ is relatively very small. The scalable version, LVS, makes it possible to start with a fixed small number of patterns (e.g., a few thousand for $D_0$), no matter how large the original dataset is. The experimental results show that the time saving by so doing is statistically significant when $\mathcal{N}$ is large. Thus, LVS is scalable.

## CONCLUSION

The time performance of LVF is not reported for the first set of data because (1) LVF completes its run fast (in few seconds of measured elapsed time); (2) there is not much to compare with among the

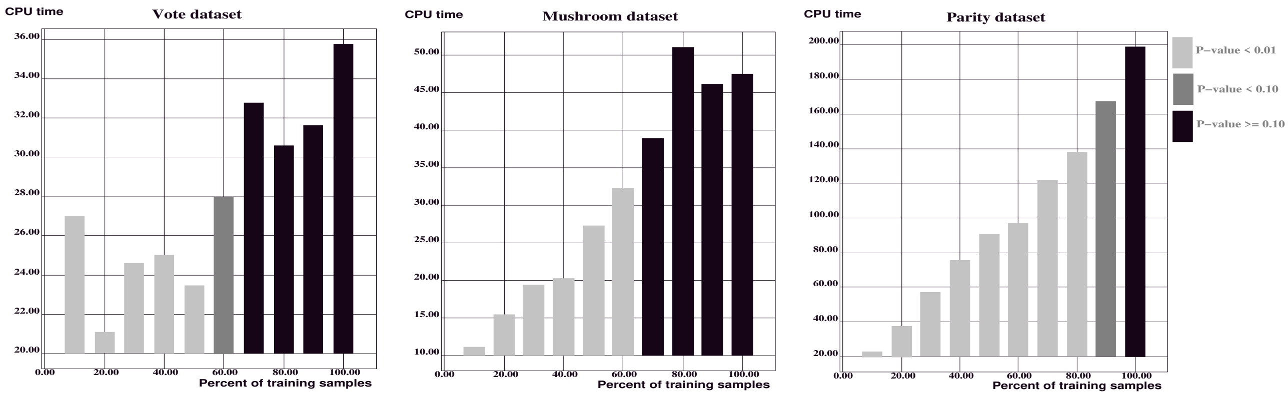


FIGURE 1. Average CPU time: light grey - most significant; dark grey - significant; black - insignificant.

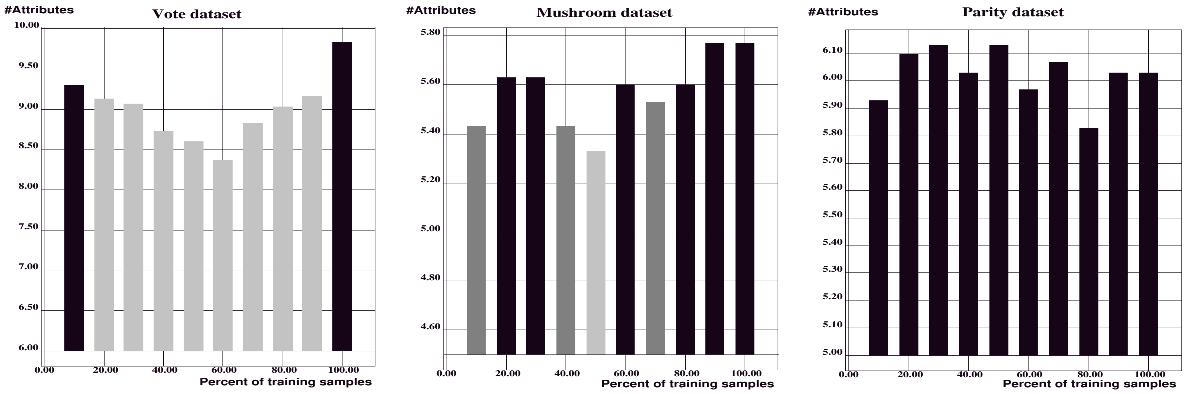


FIGURE 2. Average number of selected attributes

small datasets; and (3) the time measurements of LVS for the large datasets indicate the time performance of both LVF and LVS. By predefining $\gamma$ according to prior knowledge, LVS and LVF can handle noisy data, as shown in the case of Monk3. Both can deal with multiple class values. Another feature of LVF is related to so-called *anytime algorithms* [2] that are algorithms whose quality of results improves gradually as computational time increases. LVF prints out a possible solution whenever it is found; afterwards it reports either a better one or equally good ones. This is really nice a feature because while it works hard to find the optimal solution, it provides those near optimal solutions. The longer it runs, the better the solutions it produces. The salient feature LVS has added is its scaling capability without losing the quality of selected features. More large sized datasets will be sought and experimented. The program is available upon request to the public for trial purpose.

LVS and LVF can find other uses as well. As mentioned earlier that Las Vegas algorithms may not be as fast as some domain specific heuristic methods, LVS can still play a role as a reference in design of a domain specific heuristic method. This is because it is not an easy task to verify a heuristic method, especially when datasets involved are huge. LVS can be most helpful in this case. Another feature with little coverage due to the space limit is that LVF may produce many equally good solutions on many real-world datasets. To choose which solution can be based on the application of a learning algorithm. Hence, one straightforward extension of this work is a combined filter and wrapper [5] model of this scalable probabilistic algorithm. Another line of research is to search for other criteria in addition to the inconsistency criterion used in this work.

**Acknowledgments:**

Thanks H.Y. Lee for the suggestions on an earlier version of this paper; H.L. Ong and A. Pang for providing the results on their applying LVF to huge datasets at Japan - Singapore AI Center.

## REFERENCES

[1] H. Almuallim and T.G. Dietterich. Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305, November 1994.

[2] M. Boddy and T.L. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–285, 1994.

[3] G. Brassard and P. Bratley. *Fundamentals of Algorithms.* Prentice Hall, New Jersey, 1996.

[4] P.A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice Hall International, 1982.

[5] G.H. John, R. Kohavi, and K. Pfleger. Irrelevant feature and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129. Morgan Kaufmann Publisher, 1994.

[6] K. Kira and L.A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *AAAI-92, Proceedings Ninth National Conference on Artificial Intelligence*, pages 129–134. AAAI Press/The MIT Press, 1992.

[7] P. Langley. Selection of relevant features in machine learning. In *Proceedings of the AAAI Fall Symposium on Relevance.* AAAI Press, 1994.

[8] H. Liu and W.X. Wen. Concept learning through feature selection. In *Proceedings of First Australian and New Zealand Conference on Intelligent Information Systems*, 1993.

[9] P. Mehra, L.A. Rendell, and B.W. Wah. Principled constructive induction. In *Proceedings of IJCAI*, pages 651–656, August 1989.

[10] M Modrzejewski. Feature selection using rough sets theory. In P.B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning*, pages 213–226, 1993.

[11] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

[12] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C.* Cambridge University Press, Cambridge, 1992.

[13] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[14] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[15] S.B. Thrun and et al. The monk's problems: A performance comarison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

[16] S. Watanabe. *Pattern Recognition: Human and Mechanical.* Wiley Interscience, 1985.

[17] N. Wyse, R. Dubes, and A.K. Jain. A critical evaluation of intrinsic dimensionality algorithms. In E.S. Gelsema and Kanal L.N., editors, *Pattern Recognition in Practice*, pages 415–425. Morgan Kaufmann Publishers, Inc., 1980.