Incremental Feature Selection*

Huan Liu and Rudy Setiono
Department of Information Systems and Computer Science
National University of Singapore
Kent Ridge, Singapore 119260
{liuh,rudys}@iscs.nus.edu.sg
Tel: (+65) 874-6563; Fax: (+65) 779-4580

Abstract

Feature selection is a problem of finding relevant features. When the number of features of a dataset is large and its number of patterns is huge, an effective method of feature selection can help in dimensionality reduction. An incremental probabilistic algorithm is designed and implemented as an *alternative* to the exhaustive and heuristic approaches. Theoretical analysis is given to support the idea of the probabilistic algorithm in finding an optimal or near-optimal subset of features. Experimental results suggest that (1) the probabilistic algorithm is effective in obtaining optimal/suboptimal feature subsets; (2) its incremental version expedites feature selection further when the number of patterns is large and can scale up without sacrificing the quality of selected features.

Keywords: Pattern Recognition, Machine Learning, Feature Selection, Dimensionality Reduction

*To appear in Applied Intelligence, Kluwer Academic Publishers.

1 Introduction

Feature selection is about finding useful (relevant) features that describe an application domain. The problem of feature selection can formally be defined as selecting a minimum set of M relevant features from N original features where $M \leq N$ such that the probability distribution of different classes given the values for these M features is as close as possible to the original distribution given the values for N features. Mathematically, if F_N is the original feature set and F_M is the chosen feature subset, then the conditional probability, $\mathrm{P}(C \mid F_M = f_M)$ should be as close as possible to $\mathrm{P}(C \mid F_N = f_N)$ for possible classes, C. Here f_M and f_N are value vectors of respective feature vectors F_M and F_N [11]. As the dimensionality of a domain expands, the number of features increases. In general, the role of feature selection is three-fold: 1. simplifying data description; 2. reducing the task of data collection; and 3. improving the quality of problem solving. For the same problem, a representation by three features is generally simpler than one by six features. The benefits of having a simple representation are abundant such as easier understanding of problems, and better and faster problem solving. In the context of data collection, having more features means that more data should be collected. In many applications, this could be time consuming and costly. The quality improvement of problem solving resulting from feature selection can be illustrated via a classical supervised learning task - pattern classification problem: given a training set of labeled patterns, induce a classification model that can predict the class label for a set of previously unseen patterns (the so-called testing set). Although having more features enhances discriminating power in representation, having excessive features would introduce many difficulties for induction algorithms [11]. First, the time required by an induction algorithm often grows dramatically with the number of features, rendering the algorithm impractical for problems with a large number of features. Second, many learning algorithms can be viewed as performing estimation of the probability of the class label given a set of features. With many features, this distribution is of high dimension and becomes very complex. Unless exponential amounts of data are available, it is difficult to obtain a good estimation from a training dataset. Third, irrelevant and redundant features may confuse a learning algorithm by obscuring the distribution of the small set of truly relevant features. In addition, irrelevant and redundant features require an exponential increase in data storage requirements [1]. This is because with more features, much more data is required for effective induction. For instance, in a binary domain, the extra m irrelevant/relevant features would require 2^m times more patterns to describe the whole data. For an induction algorithm, the reduced features can also result in a simpler induction model such as shorter and fewer classification rules.

For N features, if d of them are relevant, an exhaustive approach to finding the optimal d features would require examining $\sum_{i=0}^{d} \binom{N}{i}$ subsets. The number of possible subsets grows exponentially. Researchers have designed different strategies in search of optimal subsets of d features (Branch and Bound [20] and its variations [26], many heuristic and stochastic methods [5, 7]). If we view these feature selection algorithms from the perspective of using an induction algorithm, as pointed out in [8], the work on feature selection can be divided into filter and wrapper models. In a filter model, a feature selector is independent of an induction algorithm and serves as a filter to sieve the irrelevant and/or redundant features; in a wrapper model, a feature selector wraps around an inductive algorithm relying on which relevant features are determined. One problem with the wrapper model is that it is restricted by the time complexity of a learning algorithm [12]. This time complexity is dependent on the number of features. Often the wrapper methods are prohibitively expensive to run and can be intractable for a very large number of features. Recall that, in many cases, feature selection is performed because of the excessive number of features and because a favorite induction algorithm has difficulties in handling so many features. Different models, however, suit for different applications. If a classifier is chosen and it can run for an application at hand, then it may be wise to choose a wrapper model since both feature selection and classifier induction use the same bias. This work considers the cases in which learning a classifier becomes cumbersome or ineffective due to the large size of a dataset. The largeness can be defined by both the number of features (N) and the number of patterns (P). It is the latter that makes some induction algorithms falter. Hence, large datasets in terms of P are the main concern. Naturally, the filter model is adopted. Our aim is to provide a simple and practical method that can select features for large datasets. In the following, we first review related work on feature selection.

2 Related Work

The problem of feature selection has long been an active research topic within statistics and pattern recognition [30, 6, 7], but most work in this area has dealt with linear regression [12] and is under assumptions that do not apply to most machine learning algorithms [8]. Researchers [12, 8] 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 researchers in machine learning and knowledge discovery who are interested in improving the performance of 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 time to run before data is reduced.

Most feature selection methods [9, 12, 8] can be grouped into two categories: exhaustive or heuristic search for an optimal set of M features. For example, Almuallim and Dietterich's FOCUS algorithm [2] starts with an empty feature set and carries out exhaustive search until it finds a minimal set of features that is sufficient to construct a hypothesis consistent with a given set of examples. It works on binary, noise-free data. Its time complexity is $O(\min(N^M, 2^N))$.

[1] The monotonicity assumption is not valid for many induction algorithms used in machine learning. An example is dataset 1 (CorrAL) in Section 5 which is reproduced from [8].

They proposed three heuristic algorithms to speed up the searching [2]. This is because selecting a minimal subset is a known intractable problem, and in practice, we often have to trade off the optimality of a solution for less time spent on searching.

There are many heuristic feature selection algorithms. The Relief algorithm [9] assigns a "relevance"weight to each feature, which is meant to denote the relevance of a feature to the target concept. Relief samples patterns randomly from the training set and updates the relevance values based on the difference between the selected pattern and the two nearest patterns of the same and opposite classes. According to [9], Relief assumes two-class classification problems and does not help with redundant features. If most of the given features are relevant to the concept (including redundant features), it would select most of them even though only a fraction of them is necessary for concept description. The PRESET algorithm [19] is another heuristic feature selector that uses the theory of Rough Sets to rank the features heuristically, assuming a noise-free binary domain. In order to consider higher order relations among the features, Liu and Wen [16] suggest the use of 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 features are highly interdependent such as the parity problem where combining a small number of features does not help in finding the relevant ones. Another common understanding is that some learning algorithms have built-in feature selection, for example, ID3 [23], FRINGE [21] and C4.5 [24]. The results in [2] suggest that one should not rely on ID3 or FRINGE to filter out irrelevant features. A more detailed survey can be found in [5]. The latest development of feature selection in pattern recognition can be found in [7].

To sum up, the exhaustive search approach is infeasible in practice; 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 features. A probabilistic approach is proposed as an alternative [15] in selecting the optimal/suboptimal subset(s) of 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]. We had first-hand experience of this problem when our probabilistic system was dispatched to a local institute for on-site usage. All the evidence showed that reducing data size can significantly speed up the selection of features (see a case study in Section 3.3). Hence, the incremental probabilistic method is designed and implemented. In the following, we describe the probabilistic method first, then the incremental one, followed by an empirical study in which the effectiveness of the algorithms is verified. At the end of the paper, we provide relevant discussion.

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

3 Probabilistic Feature Selection

The proposed probabilistic approach is a Las Vegas Algorithm [4]. Las Vegas algorithms make probabilistic choices to help guide them more quickly to a correct solution. 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 datasets with high interdependency among their features. Las Vegas algorithms free us from worrying about such situations by evening out the time required on different situations. 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 [4]). In this work, LVF (Las Vegas Filter)[3] is more suitable since probabilities of generating distinct subsets are the same. The time performance of a Las Vegas algorithm may not be better than that of some heuristic algorithms.

LVF algorithm
Input: MAX-TRIES, γ - allowed inconsistency rate,
D - dataset of N features;
Output: sets of M features satisfying the inconsistency criterion
$C_{best} = N$;
for i=1 to MAX-TRIES
S = `randomSet`(seed); C = `numOfFeatures`(S);
if ($C < C_{best}$ and `InconCheck`(S, D) $< \gamma$)
$S_{best} = S$; $C_{best} = C$;
`printCurrentBest`(S) ;
else if (($C = C_{best}$) and (`InconCheck`(S, D) $< \gamma$))
`printCurrentBest`(S)
end for

The LVF algorithm 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. If its inconsistency rate (defined later) is below a pre-specified one (γ), 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 in the algorithm is used to control the number of loops. A value of MAX_TRIES can be defined according to applications 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 for good and fast solutions. The longer LVF runs, the better its results are. Refer to the analysis in Section 3.2. MAX_TRIES is set to $77 \times N$ in our experimental study[4] following the rule-of-thumb that the

[3] Its counterpart is LVW - a wrapper feature selector applying a Las Vegas algorithm.
[4] We tried first a constant c alone instead of $c \times N$, then linked it to N. 77 was chosen for c

more features 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. An alternative to this stopping criterion is to let LVF run forever to take full advantage of its "anytime algorithm" nature (more in Section 6). The function `randomSet`(seed) returns a set of features randomly. When the seed is changed dynamically, a different set is generated. The function `numOfFeatures`(S) returns the cardinality of set S. `InconCheck`(S, D) returns the inconsistency rate of data D with selected features specified in S. `printCurrentBest`(S) prints out subset S.

A more sophisticated version of LVF is like this: since we know the cardinality of a better subset can only be smaller than C_{best} - the cardinality of the current best subset, we just need to randomly generate subsets whose cardinalities are smaller than C_{best}. For a new round of selection, we sample features without replacement.

3.1 Measure of feature goodness

The *inconsistency* criterion (`InconCheck`(S, D) $< \gamma$) is the key to LVF. The criterion specifies to what extent the dimensionally reduced data is acceptable. The inconsistency rate of the data described by the selected features is checked against a pre-specified rate (γ). If it is smaller than γ, it means the dimensionally reduced data is acceptable. The default value of γ is 0 unless 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 $(n - c_3)$; and (3) the inconsistency rate is the sum of all the inconsistency counts divided by the total number of patterns (P). It can be easily shown that if the inconsistency rate is 0 for both datasets with M and N features, and F_N is the original feature set and F_M is the chosen feature subset, then the conditional probability $\mathrm{P}(C \mid F_M = f_M)$ exactly equals $\mathrm{P}(C \mid F_N = f_N)$ for different possible classes, C, where f_M and f_N represent vectors of values of respective feature vectors F_M and F_N.

The inconsistency criterion is a conservative way of achieving the "class separability" which is commonly used in pattern recognition as the basic selection criterion [6]. A limited version of this was first proposed by [2] as the MIN-FEATURES bias on a binary domain. Instead of aiming to *maximize* the class separability, our measure tries to *maintain* the original class separability of the data. The inconsistency criterion is also in line with information-theoretic

in all the experiments in this paper. We tried not to use too large c so that for all (small and large) datasets, we could use just one fixed MAX_TRIES. The reader may do as we have done in another version of LVF to link MAX_TRIES to the percentage of the total search space (2^N) according to the desired quality of selected features.

considerations [28] which suggest that using a feature that is good for discrimination provides compact descriptions of each of the two classes, and that these descriptions are maximally distinct. Geometrically, this constraint can be interpreted [17] 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 retain the discriminating power of the data for multiple classes after feature selection.

3.2 Theoretical analysis

Our 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 [22], selecting an optimal subset of M features can be considered as sampling without replacement. 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 $\frac{2^N-1}{2^N} \times \frac{2^N-2}{2^N-1} \times ... \times \frac{1}{2^N-k} = \frac{1}{2^N}$, 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 $\frac{l}{2^N-k}$. Roughly, when the number of optima is doubled, the number of run times can be halved.

Referring to the LVF algorithm, we notice that the inconsistency criterion is checked only when $C \leq C_{best}$. Thus, when C_{best} is reduced due to the random search, the number of inconsistency checking is also reduced. As shown in Section 5.1, for the real-world datasets, C_{best} can be as few as one fifth of the original number of features (Mushroom). In addition, the time complexity of the checking is $O(P)$. Hence, LVF is expected to run fast. If the equivalently good subsets are not required, the last two lines inside the for-loop of the LVF algorithm can be removed, LVF can be made even faster.

3.3 Applying LVF to huge datasets (a practical case)

Feature selection is particularly useful when datasets are huge since many learning algorithms may encounter difficulties. As mentioned earlier, feature selection can help reduce the dimensionality of the datasets so that learning algorithms can be used to induce rules. Hence, huge datasets are also an ultimate test for a feature selection algorithm. LVF had an opportunity to undergo a real test of huge datasets. In Section 5 (Empirical Study) below, the results of LVF on the benchmark datasets are reported.

The datasets involved are related to the service industry. LVF was given to a local institution[5], which was in need of a method to reduce the number of features before applying some machine learning algorithms to the datasets due to the huge size of the datasets. Because the datasets are confidential, we have no access to them. The users at the institution ran LVF independently

[5] Japan-Singapore AI Centre, Singapore.

and without modification and provided the following account: One dataset (let us call it HD1) has 65,000 patterns and 59 features; the other (HD2) has 5,909 patterns and 81 features. Both datasets are discrete, feature values range from 2 to 13. LVF found that 10 and 35 features were relevant for describing HD1 and HD2 respectively without sacrificing their discriminating power, after hours of running LVF on a Sun Sparc workstation. Due to the long waiting time, they did another experiment in which only 10,000 patterns of HD1 were used, it took LVF about 5 minutes to complete its run and obtained the same results. The results are summarized in the table below. The stark difference between hours and minutes inspired us to extend the work of LVF. In short, their findings manifest two points: (1) LVF *significantly* reduced the number of features; and (2) reducing the number of patterns *significantly* reduced the run time. It is the second finding that leads us to incremental feature selection.

Data	#Features	#Patterns	#Selected	Time
HD1	59	65,000	10	hours
HD2	81	5,909	35	hours
HD1'	59	10,000	10	5 minutes

The largeness of a dataset can be differentiated into two types: (1) horizontal largeness - the number of features, and (2) vertical largeness - the number of patterns. In our implementation of LVF, we have considered overcoming the horizontal largeness by applying a Las Vegas algorithm in order to avoid exhaustive search and attack the vertical largeness by using a hash mechanism in order to speed up. *However, the above practical case shows that more can be done in overcoming the vertical largeness.* Hence, in the following, when we mention largeness, we mean the vertical one (P).

4 Incremental Probabilistic Feature Selection

Although LVF can generate optimal/suboptimal solutions (see the experimental results below), when datasets are huge, as shown in Section 3.2, checking whether a dataset is consistent still takes time due to its $O(P)$ complexity. It is only natural to think about an incremental version of LVF that can significantly reduce the number of inconsistency 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 those from the whole data. Furthermore, this is done without sacrificing the quality of feature subsets which is measured by the number of features and by their relevance.

LVI algorithm
Input: p - percentage of the data used for feature selection,
D - dataset of N features, γ - allowed inconsistency rate;
Output: sets of M features satisfying the inconsistency criterion

```
D_0 = p% of D randomly chosen ;
D_1 = D - D_0 /* the rest data */
loop
  β = LVF (D_0, γ, subset)
  if (checkIncon(subset, D_1, inconData) <= γ - β)
    return subset;
  else
    append(inconData, D_0);
    remove(inconData, D_1);
end loop
```

In LVI, `checkIncon()` is similar to `InconCheck()` in LVF. In addition, it saves the inconsistent patterns of D_1 to *inconData*. The experiments below are designed to demonstrate the claims made above on LVI. The incremental algorithm (LVI) starts with a portion of data ($p\%$) and an acceptable inconsistency rate (γ) which is usually set to 0 if there is no prior knowledge, or the minimum value of γ can be obtained from applying `InconCheck`(F, D) in LVF where F is the set of N features. LVI splits the data D into D_0 and D_1 where D_0 is $p\%$ of D and D_1 is the remaining. LVI uses a subset of features (*subset*) for D_0 found by LVF to check *subset* on D_1. The actual inconsistency found in D_0 is $\beta \leq \gamma$. If the inconsistency rate on D_1 does not exceed $\gamma - \beta$, LVI stops. Otherwise, it appends those patterns (*inconData*) of D_1, which cause the additional 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 is returned as a solution.

5 Empirical Study

The error probability plays the most important role in the feature selection algorithms. Ultimately, it is always used as a meta-selection criterion [25]. That is, regardless of different feature selection algorithms, the subset with the lowest estimated error will always be selected for classification tasks. An error is caused by a wrongly classified pattern. The number of errors divided by the number of total patterns in the set gives us the error rate. Each dataset is split into two sets (training vs. testing). Error rates are obtained for both sets. It is the error rate of the testing set that estimates the performance of a classification algorithm. In order to check error rates before and after features selection, both artificial and real-world datasets are used in the study of the effectiveness of LVF and LVI. These datasets are either commonly used in comparison or having known relevant features. All but two (CorrAL and Parity5+5) datasets can be obtained from the UCI Repository [18].

Artificial Data:

- **CorrAL** The data was designed in [8]. 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 $(A_0 \wedge A_1) \vee (B_0 \wedge B_1)$. 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.

- **Monk1, Monk2, Monk3** The datasets were taken from [27]. They have six features. The training datasets provided were used for feature selection. Monk1 and Monk3 only need three features to describe the target concepts, but Monk2 requires all the six. The training data of Monk3 contains some noise. These datasets are used to show that relevant features should always be selected.

- **Led17** This data is generated artificially by a program at the UCI data mining repository. It generates 24 features among which the first 7 are used to display a value between 0 - 9 in the seven segment display system. The remaining 17 features are generated randomly. All the values are binary except the class which takes a value between 0 and 9 (representable in seven segments). The number of patterns to be generated is determined by the user. 20000 patterns were generated for our experiments.

- **Parity5+5** The target concept is the parity of five bits. The dataset contains 10 features, 5 of them are uniformly random (irrelevant). The training set contains 100 patterns randomly selected from all 1024 patterns. Another independent 100 patterns are drawn to form the testing set. Most heuristic feature selectors will fail on this sort of problems since an individual feature does not mean anything.

Real-World Data:

- **LungCan** The Lung Cancer data describes 3 types pathological lung cancers found in UCI repository. This data contains only 32 patterns and 56 features taking the values 0-3.

- **SoybeanL** In the UCI machine learning repository, we found training and testing datasets in two separate files containing 307 and 376 patterns respectively. It contains 35 features describing symptoms of 19 different diseases in soybean plant.

- **Vote** This dataset includes votes from the U.S. House of Representatives Congress-persons on the 16 key votes identified by the Congressional Quarterly Almanac Volume XL. The dataset consists of 16 features, 300 training patterns and 135 test patterns.

Table 1: Notations: C - the number of distinct classes, N - the number of features, 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.

Dataset	C	N	S	S_d	S_t
CorrAL	2	6	64	16	16
Monk1	2	6	432	124	432
Monk2	2	6	432	169	432
Monk3	2	6	432	122	432
LungCan	3	56	32	32	-
SoybeanL	19	35	683	307	376
Parity5+5	2	10	1024	100	100
Vote	2	16	435	300	135
Mushroom	2	22	8125	7125	1000
Led17	10	24	20,000	20,000	-
Krvskp	2	36	3196	3196	-
Parity Mix	2	20	2^{20}	10,000	-

- **Mushroom** The dataset has a total of 8124 patterns, of which 1000 patterns are randomly selected for testing, the rest are used for training. The data has 22 discrete features. Each feature can have 2 to 10 values.
- **Krvskp** This is the data for Chess End-Game – King+Rook versus King+Pawn on a7. The Pawn on a7 means its one square away from queening. Its the King+Rook's side (white) to move. The data contains 3196 patterns and 36 features. The class value 1 indicates white can win, which means white can check the black pawn not to advance and vice versa. Each pattern is a board description for this chess end-game. The first 36 features describe board and the last one is the classification.

The major measurements of these datasets are summarized in Table 1. Since most datasets in the first group do not have a large number of patterns, we choose Vote and Mushroom plus ParityMix, Led17 and Krvskp to form the second group of datasets to show the effectiveness of LVI in relation to the size of datasets (small, medium, large). ParityMix is composed by having two Parity5+5's side by side so that there are 20 features in total.

5.1 Effectiveness of LVF

For the artificial datasets, the evaluation of LVF is simple since the relevant features 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 be only determined indirectly. One way is to see the effect of fea-

Table 2: Results of 100 runs of LVF on the datasets with one example of the minimum set of features for each dataset. N - number of original features, M - number of selected features, F - frequency.

Dataset	N	M (F)	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
LungCan	56	10 (3), 12 (4), 13 (27), 14 (12), 15 (33), 16 (18), 17 (3)	A3, A6, A8, A10, A16, A24, A25, A26, A29, A42
SoybeanL	35	14 (3), 15 (11), 16 (18), 17 (37), 18 (27), 19 (4)	A1, A2, A4 - A7, A9, A16, A18, A21, A23, A26, A31, A32
Parity5+5	10	5 (71), 6 (29)	A1 - A5
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

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

ture selection through a learning algorithm. Among many choices, we chose C4.5 [24] and NBC [29] in our experiments because (1) C4.5 is a decision tree induction algorithm that works well on most datasets as reported by many researchers; and (2) it employs a heuristic to find the simplest tree structures. NBC (Naive Bayesian Classifier) employs the Bayes rule by assuming features are independent of each other and is an approximation of Bayesian Classifiers - the optimal classifier. NBC is chosen because it works in a different way from that of C4.5. 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 directly be used by readers in their experiments.

For the artificial datasets, the relevant features are always selected, albeit a few of irrelevant ones are also chosen sometimes. For the problem like Parity5+5, LVF correctly identifies the correct features all the time, plus one irrelevant feature sometimes. For the real-world datasets, the number of features 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 and NBC to test if its performance improves compared to using all features. Ten-fold cross validation is usually recommended, t-test is used instead of Z-test in calculation of P-values since we need to take into account the small sample effect (10 data in each sample for 10-fold cross validation). The default settings of C4.5 are used in the experiments. For the experiments "after" feature selection, only the features shown in the last column of Table 2 are used. Given in Tables 3 and 4 are the average accuracy rates of C4.5 before and after applying feature selection to the datasets. The same applies to NBC: instead of reporting the tree size, we report the table size.

Table 3: 10-fold cross validation results on Tree Size and Error Rates of NBC before and after applying LVF to the datasets. P-val stands for P-value of t-test; and "-" means that the pooled variances of "before" and "after" are zero.

	NBC					
	Table Size			Error Rate(%)		
Dataset	Before	After	P-val	Before	After	P-val
CorrAL	26.0	18.0	-	16.6	23.3	.5673
Monk1	36.0	22.0	-	25.0	25.0	1
Monk2	36.0	36.0	-	37.4	37.4	1
Monk3	36.0	22.0	-	3.63	3.63	1
LungCan	722.8	95.4	**0.0**	56.66	63.33	.6685
SoybeanL	2542.2	1022.2	**0.0**	7.0	17.05	**0.0**
Parity5+5	42.0	22.0	-	55.5	50.0	.5161
Vote	98.0	62.0	-	9.9	9.9	1
Mushroom	236.0	74.0	-	0.33	1.16	**.0004**

In cases indicated by "-", the comparison between "before" and "after" is obvious. Tables 3 shows that results are consistent with the known fact that there are no bad features from the standpoint of Bayesian decision rules [26]. In all the datasets tested using NBC, only table sizes are all reduced (except Monk2) due to feature selection; error rates are not significantly changed in seven out of nine datasets. For the two datasets (SoybeanL and Mushroom), the latter's error rate increases a little in absolute percentage, but SoybeanL's error rate is much worse after feature selection. This is because the training dataset has fewer patterns than the test dataset (recall that the division is done by the data contributor and because features were selected based on the training data, then both datasets were put together to run 10-fold cross validation). To verify this conjecture, we did another experiment in which features were selected using both data sets (training and testing). Fifteen (instead of fourteen) features were chosen (they are $A_1, A_3 - A_7, A_9, A_{10}, A_{14}, A_{16}, A_{22}, A_{25}, A_{26}, A_{30}, A_{34}$), and results of 10-fold cross validation on NBC and C4.5 are 14.4% (7.0% before) and 9.7% (7.3%) respectively. Thus, error rates are lower with all data used for feature selection. The only improvement of NBC's performance is on Parity5+5, but it is not statistically significant.

Results in Table 4 suggests that the performance of C4.5 improves in general. That is, the tree size is getting smaller and the error rate lower. For the artificial datasets, this experiment further shows with the relevant features, C4.5 does better than that with the full set of features. For the real-world datasets, C4.5 is also doing better with the selected features. This indicates that LVF has selected relevant features for these datasets. In particular, C4.5 did poorly on Parity5+5

Table 4: 10-fold cross validation results on Tree Size and Error Rates of C4.5 before and after applying LVF to the datasets. P-val stands for P-value of t-test; and "-" means that the pooled variances of "before" and "after" are zero.

	C4.5					
	Tree Size			Error Rate (%)		
Dataset	Before	After	P-val	Before	After	P-val
CorrAL	10.6	10.2	.5906	33.3	12.5	.0726
Monk1	41.9	41.0	.0782	1.3	0.0	.1937
Monk2	14.3	14.3	1	35.4	35.4	1
Monk3	19.0	19.0	-	1.1	1.1	1
LungCan	18.3	16.6	**.03**	56.7	57.5	.2627
SoybeanL	96.8	129.8	**0.0**	7.3	15.2	**.0001**
Parity5+5	47.0	62.8	**.0015**	45.5	1.5	**.0001**
Vote	14.5	6.1	**.0001**	5.3	5.5	.8357
Mushroom	29	29	1	0.0	0.0	-

before feature selection. Nevertheless, C4.5 did as well on Mushroom with 22 features as with 4 features. This demonstrates that C4.5 does select relevant features for some datasets, though not for all. The only serious deterioration of C4.5's performance is seen in the results for SoybeanL. The reason is given above in explaining NBC's poor performance on the dataset.

The gain from feature selection differs for NBC and C4.5. The difference is due to the way in which features are used to induce a classifier. C4.5 is a selective induction algorithm that selects the best feature at each test for tree branching. NBC uses all features' conditional probabilities in determining a pattern's class. Since NBC assumes that features are conditionally independent given the class, the conditional probabilities of an irrelevant feature given the class will be approximately the same, so it is not a good discriminant.

5.2 Effectiveness of LVI

For this set of experiments, we want to verify four claims: (1) LVI may not be suitable for small datasets; (2) LVI can run faster than LVF on large datasets; (3) LVI does not sacrifice the quality of the selected features; and (4) if no solution can be found by LVF in the earlier runs, neither it can in later runs (earlier runs start with less data). Five datasets in the second group are chosen for experiments. They are (1) Vote, (2) Mushroom, (3) ParityMix, (4) Krvskp, and (5) Led17.

The experiments are conducted as follows. For each dataset, starting with 10% of the data (D_0) for feature selection, we run LVI 10 times, recording the number of features, features, 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 features are computed for each experiment. Using

100% of data as the reference, we calculate the P-values for each sized D_0. A low P-value (e.g., $< 5\%$) suggests that the NULL hypothesis that the two averages are the same be rejected. Refer to Figures 1 and 2 for varied P-values shaded differently.

We summarize the findings from the experiments as follows.

- The effectiveness of LVI becomes more obvious when the data size is larger. LVI 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. This is due to the overheads required by incremental feature selection. Since our inconsistency checking is fast ($O(P)$), if P is not sufficiently large, the time saving will not be apparent, it may even be negative if P is too small. This is why LVI is more suitable for large sized datasets.

- Another issue is the number of patterns with which LVI should start for a dataset. Having either too few or too many patterns affects the LVI'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 that 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 LVI 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.

- The incremental algorithm does not have to sacrifice the quality of feature selection. The time saving is mainly due to (1) small D_0 which is usually a portion (say 10%) of D, and (2) by remembering inconsistent patterns, LVI can avoid checking wrong guesses twice. The quality is measured here in two dimensions. One is the number of features, and the other is the relevance of the features. As shown in Figure 2, if there is some statistically significant difference in the number of features selected between various sized D_0's against D, the numbers of features are lower than using 100% of the data; otherwise, there is no statistically significant difference according to t-test. For ParityMix, the relevant 5 features 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 NBC here). If their performances do not deteriorate or even improve, we conclude that these features are relevant. Experimental results shown in Tables 3 and 4 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.

- LVI can scale up. Time complexity of a feature selection algorithm can be described along two dimensions: number of features (N) and number of patterns (P). By approximating MAX_TRIES of LVF with $77 \times N$ (reduced from 2^N), the time complexity of LVF is mainly determined by P since N is relatively very small. The incremental version, LVI, 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 P is large.

- For data Krvskp, no feature can be removed from 10% data to 100% data used. It indicates the other side of incremental feature selection by LVF: if LVF cannot reduce features based on a smaller portion of data, then more or all data cannot help reduce features either, other things being equal. It will help if we extend the run time, for example, linking MAX_TRIES to the percentage of the total search space.

6 Discussion and Conclusion

The time performance of LVF is not reported for the first set of data because (1) LVF completes its run fast (in a few seconds of elapsed time); (2) there is not much to compare with among the small datasets; and (3) the time measurements of LVI for the large datasets indicate the time performance of both LVF and LVI. Both algorithms are simple to implement and fast to obtain results. By predefining γ according to prior knowledge, LVI 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* [3] that are algorithms whose quality of results improves gradually as computational time increases. LVF prints out a possible solution whenever it is found; afterwards LVF reports either a better subset or equally good ones. This is a really nice feature because while it works hard to find the optimal solution, it provides near optimal solutions. There is no need for a user to wait for results until the end of search for optimal/suboptimal solutions as other types of search do. The longer LVF runs, the better the solutions it produces. One salient feature of LVI is its scaling capability without losing the quality of selected features. The suggested modification - sampling subsets of features without replacement of selected features and constraining subset generation by the newly found minimum number of features should allow LVF to work faster.

In order to verify that a filter feature selector can easily be turned to a wrapper one, LVW is built to prove the case [14]. If a favorite induction algorithm is available, LVF can be easily transformed into LVW. The experimental results show that LVW is much slower than LVF. This finding is consistent with the results reported in [10].

There may be a problem with using inconsistency as a feature selection criterion when one feature alone (such as social security number) can guarantee that there is no inconsistency in the data. Obviously, this feature is irrelevant for rule induction. The problem can be solved by leaving this feature out of the feature selection process. If there is no prior knowledge, it will just take one run of LVF to locate this kind of features[7]. Another run of LVF with the other features will identify the right set of features.

LVF only works on discrete features since it relies on the inconsistency calculation. One way is to apply a discretization algorithm (e.g., Chi2 [13]) to discretize the continuous features first before one runs LVF. Other possibilities are (1) to simply treat a continuous feature as a discrete one in some cases; and (2) to apply LVF only to the discrete features when the number of features is large. More work is needed. The search for new criteria in addition to the inconsistency continues.

LVI 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, LVI 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. LVI can be most helpful in this case to validate feature subsets found by the heuristic method. Another feature is that LVF may produce a number of equally good solutions for one dataset based on the inconsistency criterion. One solution can be chosen according to its predictive accuracy of a learning algorithm. That is, we choose a solution that generates the best accuracy. This suggests a straightforward extension of this work, i.e., a combined filter and wrapper model of this incremental probabilistic algorithm. The significant advantage of LVI is that we move one step further in handling the large sized datasets. It is a necessary addition to the present repertoire[8]. So far, all algorithms are of automated feature selection. We have not mentioned another important practical issue - using domain knowledge in feature selection. Domain knowledge or expert's understanding of the data can help tremendously in feature selection. For instance, domain knowledge can be used to verify the finding of automated feature selection; domain knowledge can be used to remove some obviously irrelevant or redundant features; and domain knowledge can also help in designing heuristics. When expertise is available, one should always start feature selection from what is known first, and apply automated selection algorithms next.

Acknowledgments:

The authors would like to thank H.Y. Lee for the suggestions on an earlier version of this paper and H.L. Ong and A. Pang for providing the results on their applying LVF to huge datasets at Japan - Singapore AI Center. Thanks also go to Manoranjan Dash and Farhad Hussain for conducting some experiments using

[7] Recall that one run of LVF has MAX_TRIES loops.

[8] Both LVI and LVF are available for research purposes upon request.

LVF and LVI, and Jian Shu for implementing NBC used in the experiments. The suggestions by anonymous referees have also significantly helped improve the paper.

References

[1] D.W. Aha. Tolerating noisy, irrelevant and novel attributes in instance-bassed learning algorithm. *International Journal of Man-Machine Studies*, 36(1), 1992.

[2] 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.

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

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

[5] M. Dash and H. Liu. Feature selection methods for classifications. *Intelligent Data Analysis: An International Journal*, 1(3), 1997.

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

[7] A. Jain and D. Zongker. Feature selection: Evaluation, application, and small sample performance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(2):153–158, 1997.

[8] 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.

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

[10] R. Kohavi. *Wrappers for performance enhancement and oblivious decision graphs.* PhD thesis, Department of Computer Science, Standford University, Stanford, CA, 1995.

[11] D. Koller and M. Sahami. Toward optimal feature selection. In L. Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 284–292. Morgan Kaufmann Publishers, 1996.

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

[13] H. Liu and R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence*, pages 388–391, 1995.

[14] H. Liu and R. Setiono. Feature selection and classification - a probabilistic wrapper approach. In *Proceedings of the Ninth International Conference on Industrial and Engineering Applications of AI and ES*, pages 419–424, 1996.

[15] H. Liu and R. Setiono. A probabilistic approach to feature selection - a filter solution. In L. Saitta, editor, *Proceedings of International Conference on Machine Learning (ICML-96)*, pages 319–327. Morgan Kaufmann Publishers, 1996.

[16] H. Liu and W.X. Wen. Concept learning through feature selection. In *Proceedings of First Australian and New Zealand Conference on Intelligent Information Systems*, pages 293–297, 1993.

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

[18] C.J. Merz and P.M. Murphy. 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, 1996.

[19] 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.

[20] P.M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Trans. on Computer*, C-26(9):917–922, September 1977.

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

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

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

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

[25] T.W. Rauber. *Inductive Pattern Classification Methods - Features - Sensors*. PhD thesis, Dept. of Electrical Engineering, Universidade Nova de Lisboa, Lisboa, 1994.

[26] W. Siedlecki and J. Sklansky. On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2:197–220, 1988.

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

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

[29] Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems That Learn.* Morgan Kaufmann Publishers, San Mateo, California, 1991.

[30] 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.

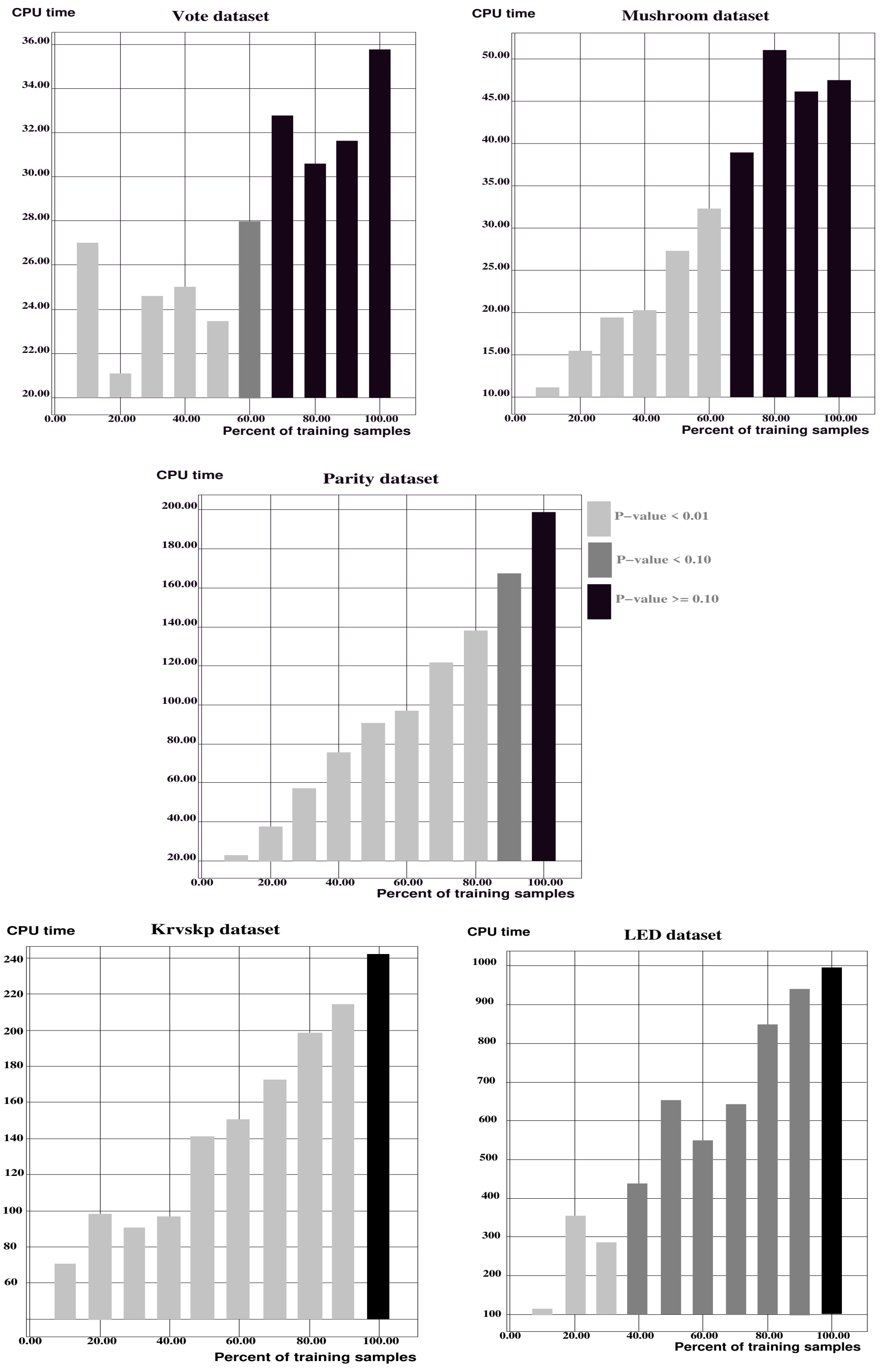

Figure 1: Average CPU time (seconds). The result for 100% data is used as the reference. The difference between any two samples (e.g., 10% vs. 100%, or 20% vs. 100%) is most significant (light grey), significant (dark grey), or insignificant (black).

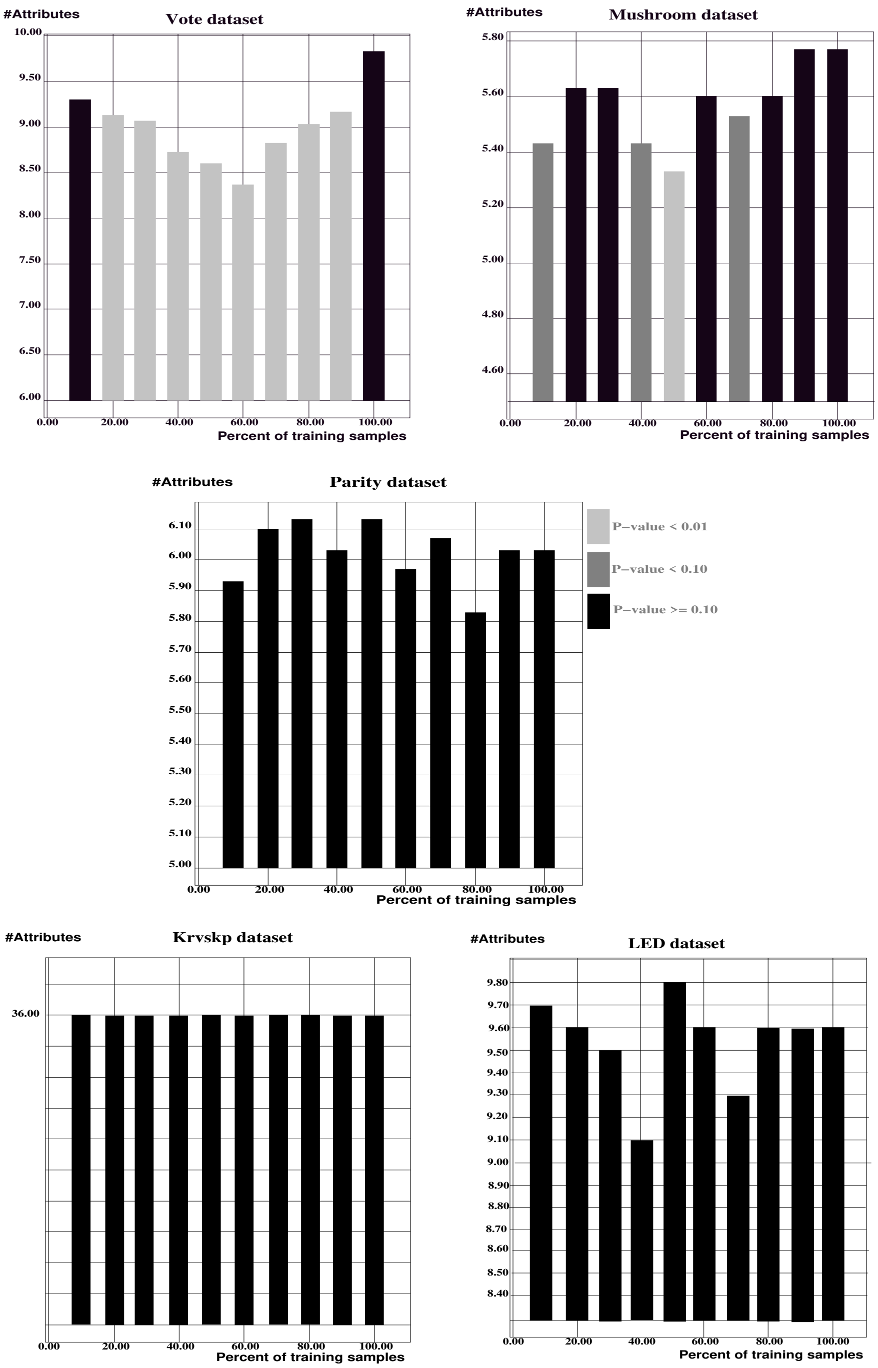

Figure 2: Average number of selected features. The result for 100% data is used as the reference. The difference between any two samples (e.g., 10% vs. 100%, or 20% vs. 100%) is most significant (light grey), significant (dark grey), or insignificant (black).