Some Issues on Scalable Feature Selection*

Huan Liu & Rudy Setiono

Department of Computer Science

School of Computing

National University of Singapore

Singapore 119260

{liuh,rudys}@comp.nus.edu.sg

Tel: (+65) 874-6563; Fax: (+65) 779-4580

Running head: Scalable Feature Selection

Key Word List: Features, Probabilistic Selection, Scalability, Large Databases, Pattern Classification

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

*This is an extended version of the paper presented at the Fourth World Congress of Expert Systems: Application of Advanced Information Technologies held in Mexico City in March 1998.

Some Issues on Scalable Feature Selection

Abstract

Feature selection determines relevant features in the data. It is often applied in pattern classification, data mining, as well as machine learning. A special concern for feature selection nowadays is that the size of a database is normally very large, both vertically and horizontally. In addition, feature sets may grow as the data collection process continues. Effective solutions are needed to accommodate the practical demands. This paper concentrates on three issues: large number of features, large data size, and expanding feature set. For the first issue, we suggest a probabilistic algorithm to select features. For the second issue, we present a scalable probabilistic algorithm that expedites feature selection further and can scale up without sacrificing the quality of selected features. For the third issue, we propose an incremental algorithm that adapts to the newly extended feature set and captures "concept drifts" by removing features from previously selected and newly added ones. We expect that research on scalable feature selection will be extended to distributed and parallel computing and have impact on applications of data mining and machine learning.

1 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 (Wyse, Dubes, & Jain, 1980; Devijver & Kittler, 1982), but most work in this area has dealt with linear regression (Blum & Langley, 1997) and is under assumptions that do not apply to most learning algorithms (John, Kohavi, & Pfleger, 1994). 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 their 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.

Most feature selection methods (refer to (Kira & Rendell, 1992; John et al., 1994; Blum & Langley, 1997)) 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 (Almuallim & Dietterich, 1994) starts with an empty feature set and carries out exhaustive search until it finds a minimal combination 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 $O(\min(N^M, 2^N))$. They proposed three heuristic algorithms to speed up the search.

There are many heuristic feature selection algorithms. The Relief algorithm (Kira & Rendell, 1992) 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

[1]The monotonicity assumption is not valid for many induction algorithms used in machine learning.

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 (Modrzejewski, 1993) is another heuristic feature selector that assumes a noise-free binary domain and uses the theory of Rough Sets to heuristically rank the features. Since PRESET does not try to explore all combinations of the features, it is certain that it will fail on problems whose attributes are highly correlated such as the parity problem where the combinations of a few attributes do not help in finding the relevant attributes. One solution to this problem is to use feedforward neural networks as a feature selector (Setiono & Liu, 1997). The idea is to train a network and then prune it while maintaining its performance. At the end, features with connections to hidden units are chosen as selected features. Another common understanding is that some learning algorithms have built-in feature selection, for example, ID3 (Quinlan, 1986), FRINGE (Pagallo & Haussler, 1990) and C4.5 (Quinlan, 1993). The results in (Almuallim & Dietterich, 1994) suggest that one should not rely on ID3 or FRINGE to filter out irrelevant features.

In summary, the exhaustive search approach may be infeasible in practice; and the heuristic search approach can reduce the search time significantly, but may fail on hard problems (for example, the parity problem) or cannot remove redundant attributes. A probabilistic approach is proposed as an alternative that 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. The above two issues are termed as (1) large number of features and (2) large number of instances. A third issue is about expanding features. For the first two issues, we assume the environment is static, but we do not make the same assumption for the third issue. In other words, we extend feature selection into a slowly changing envi-

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

[3]Japan-Singapore AI Centre, Singapore.

ronment. In the following, we discuss the three issues in turn and suggest some solutions to each problem.

2 Large Number of Features

The proposed probabilistic approach to tackling data sets with large number features is a Las Vegas Algorithm (Brassard & Bratley, 1996). 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 (Brassard & Bratley, 1996)). 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 along the way. As we mentioned earlier, heuristic search methods are vulnerable to the data sets 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, however, may not be better than that of some heuristic algorithms. Data that took a long time deterministically are now solved much faster, but data on which the heuristic algorithm was particularly good are slowed down to average by the Las Vegas algorithm.

The LVF[4] algorithm in Figure 1 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 (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 can be set according to the user's need. It should be proportional to the search space we want to cover. The longer LVF runs, the better its results are. When

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

LVF loops MAX_TRIES times, it stops.

LVF algorithm

Input:

MAX-TRIES, γ - allowable inconsistency rate,

D - data set of N attributes;

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)
     if (InconCheck(S, D) < γ)
        S_best = S; C_best = C;
        print_Current_Best(S) ;
     else if
     ((C = C_best) ∧ (InconCheck(S, D) < γ))
     print_Current_Best(S)
end for
```

Figure 1: A probabilistic algorithm for feature selection: LVF

The *inconsistency* criterion (InconCheck$(S, D) < \gamma$) is the key to LVF. In (Liu, Motoda, & Dash, 1998), it is shown that the inconsistency criterion is monotonic in the sense that if a feature set S_1 is a subset of another feature set S, then InconCheck$(S, D) \leq$ InconCheck(S_1, D). 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 (γ). If it is smaller than γ, it means the dimensionally reduced data is acceptable. The default value of γ is 0 unless otherwise specified. The inconsistency rate of a data set is calculated as follows: (1) two instances are considered inconsistent if they match all but their class labels; (2) the inconsistency count is the number of all the matching instances minus the largest number of instances of different class labels: for example, there are n matching instances, among them, c_1 instances 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 instances (P).

The inconsistency criterion is in line with information-theoretic considerations (Watanabe, 1985) 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 (Mehra, Rendell, & Wah, 1989) 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 power of the data for multiple classes after feature selection. In other words, the criterion is supported by information-theoretic considerations.

The affirmative experimental results for the effectiveness of LVF can be found in (Liu & Setiono, 1996). It is shown that LVF can select relevant features where Focus or Branch-and-Bound are impractical. In other words, LVF complements many existing algorithms nicely (Dash & Liu, 1997). A limitation for LVF is that it works only on discrete data. Other versions extended from LVF inherit this limitation. However, there exist many effective feature discretization algorithms (for example, Chi2 (Liu & Setiono, 1995) which is available to the public for trial purposes). Throughout this paper, therefore, we assume all data sets are discrete.

3 Large Number of Instances

Although LVF can generate optimal/suboptimal solutions, when data sets are huge, checking whether a data set is consistent or not still takes time due to its $O(P)$ 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) described in Figure 2 starts with a portion of data ($p\%$) and a given allowable inconsistency rate (γ, 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 γ, LVS stops. Otherwise, it appends those instances (*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 instances of D_1 to *inconData*. The experiments below are designed to demonstrate the claims made above on LVS.

Data sets for empirical study

Two real-world data sets are **Vote** and **Mushroom**. The details are in Table 1. The Vote and Mushroom data sets are randomly separated into training and test sets. **ParityMix** is composed by having two Parity5+5's side by side. Parity5+5 is a data set whose target concept is the parity of five bits. The data set contains 10 features, of which 5 are uniformly random (irrelevant) ones. Most heuristic feature selectors will fail on this sort of problems

LVS algorithm

Input:

p - the percentage of the data used for feature selection,

D - data set of N attributes, γ - the allowable 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

  LVF $(D_0, \gamma, subset)$

  if (checkIncon($subset$,D_1,$inconData$)$\leq \gamma$)

   return $subset$;

  else

   append($inconData$, D_0);

   remove($inconData$, D_1);

end loop

Figure 2: A scalable algorithm for feature selection: LVS

since an individual feature is usually not a good indicator.

Data set	N	S	S_d	S_t
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 data set, S_d - the size of the training data, S_t - the size of the test data. Training and test data sets are split randomly if not specified.

Experimental results of LVS

For this set of experiments, we want to verify three claims: (1) LVS may not be suitable for small data sets; (2) LVS can run faster than LVF on large data sets; 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 data set, starting with 10% of the data (D_0) for feature selection, we run 30 times of LVS, recording the number of attributes, selected 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 the results obtained from 100% of the data as a reference, we calculate the P-value for each D_0. The lower a P-value is, the more confidence we have in rejecting the NULL hypothesis that the two averages are the same. Results are shown in Figures 3 and 4.

We observe the following:

1. The effectiveness of LVS becomes more obvious when the data size is larger. LVS performs as expected. 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 3. This is due to the

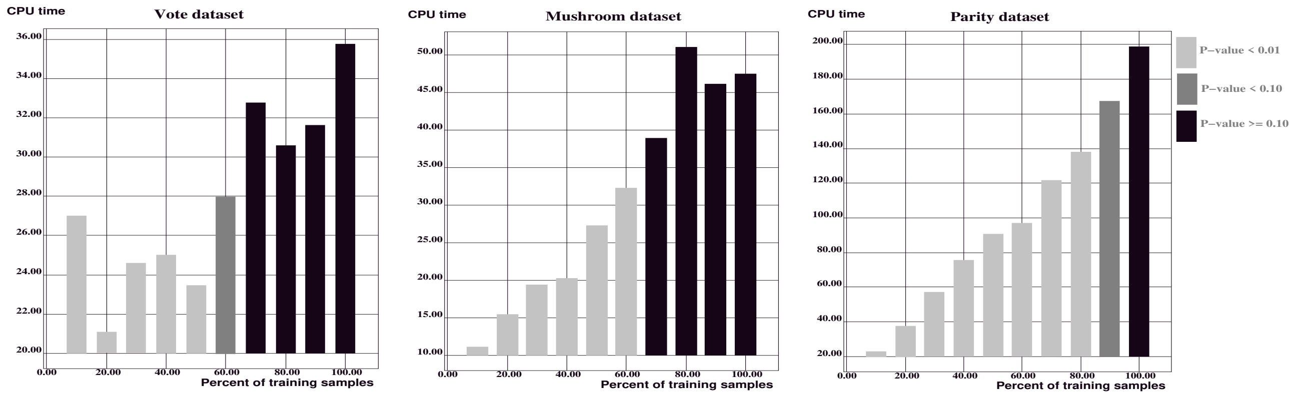

Figure 3: Average CPU time: light grey - most significant; dark grey - significant; black - insignificant.

overheads required by scalable 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 LVS is more suitable for large sized data sets.

2. Another issue is the number of instances with which LVS should start for a data set. Either too few or too many instances affects LVS's performance. If too few instances 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 data set). One case of too small a D_0 can be observed in Figure 3 for Vote when 10% of the data was used; it took longer time than using 20% - 50% of D. If too many instances 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.

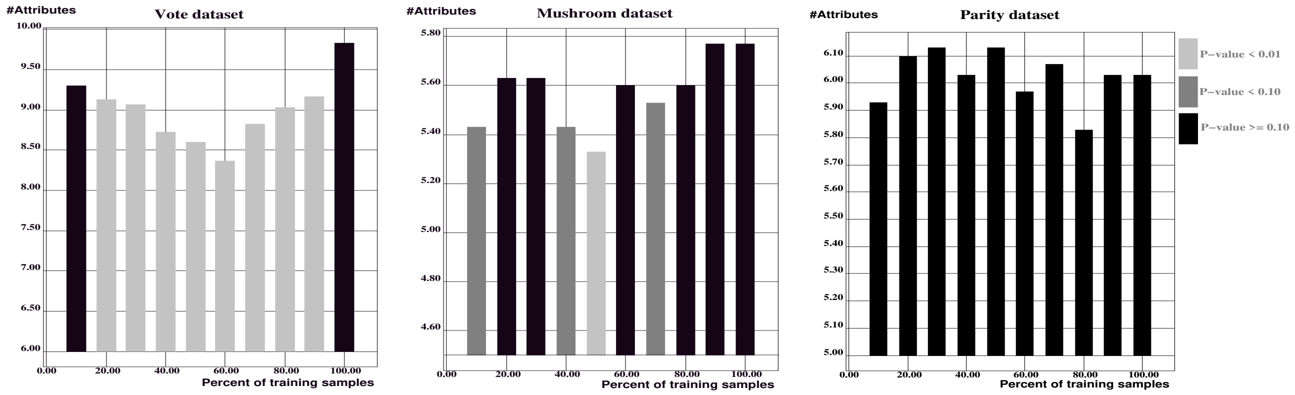

Figure 4: Average number of selected attributes

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 instances, 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 4, if there is any 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 data sets, the relevance test can be done through a learning algorithm. If their performances do not deteriorate or even improve, we can conclude that these features are relevant. We used C4.5 to test whether the selected features for the two data sets are relevant. The 10-fold cross validation experiments were conducted. The average performance is given in tree size and error rate. Comparing the selected features with the full set of features, we have the following results:

	Tree Size		Error Rate	
	Before	After	Before	After
Vote	14.5	6.1	5.3	5.5
Mushroom	29.0	29.0	0.0	0.0

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 instances (P). By approximating MAX_TRIES of LVF with $c \times N$ (reduced from 2^N) (Liu & Setiono, 1996), the time complexity of LVF is mainly determined by P since N is relatively small. The scalable version, LVS, makes it possible to start with a fixed small number of instances (e.g., a few thousand for D_0), no matter how large the original data set is. The experimental results show that the time saving by so doing is statistically significant when P is large. Thus, LVS is scalable.

By predefining γ according to prior knowledge, LVS and LVF can handle noisy data. Both can deal with multiple class values. Another feature of LVF is related to so-called *anytime algorithms* (Boddy & Dean, 1994) 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 a really nice 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 data sets will be sought and experimented. The program is available, upon request, to the public for trial purposes.

4 Expanding Features

In the above two sections, we assume that the features describing data are fixed once and for all. It is only a matter of which features are irrelevant. In some real-world tasks, however, the concepts of interest may not be entirely stable - they change over time (Widmer, 1996). One way of viewing *concept drifts* is to assume that the concepts of interest depend on some context, and that changes in this context include corresponding changes in the target concepts. One type of concept drifts may be captured by changing values of features. In some domains, systematic changes in values of features or combinations of features may indicate a context change: some features that were relevant before may not be relevant now so that they can be removed for a learning algorithm to learn better in a new context. A relevant situation is that new features are introduced, for example, resulting from a better understanding of the domain after preliminary learning. In this case, we need to combine both existing and newly introduced features to capture concept drifts. It is not surprising that some relevant features become irrelevant. It would be ideal if these irrelevant features can be removed.

Thereby, we summarize the above case as a problem of "expanding features": periodically, a feature set F_0 is expanded with a few newly introduced features and a larger feature set F_1 is formed; meanwhile, some existing features thus become irrelevant. Because of the new features, new instances are also collected, now the original data set has missing values for the new features. We consider two situations: (1) the new instances are sufficiently many and (2) they are relatively few compared to the data without new features. If it is case (1), the solution to the problem of expanding features can be straightforward. One way is to apply neural networks. Basically, we recast the problem of filling in these newly introduced features by training neural networks and let the networks do the job: firstly, according to the class labels, we group the new data; secondly, we train a network for each group of the data with the newly introduced features as conventional "class" features; lastly, we apply the trained networks to the old data and predict the "class" feature values. If it is case (2), we cannot simply remove previously collected data because doing so is equivalent to

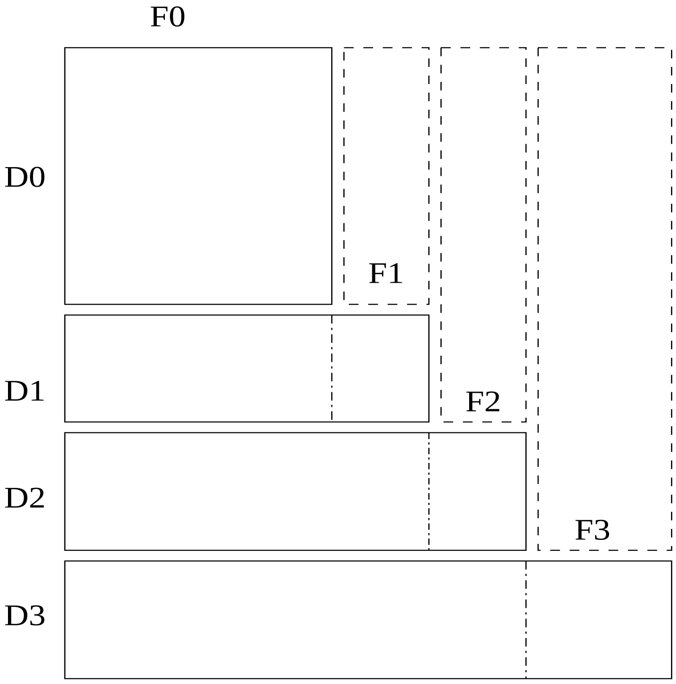

Figure 5: Data accumulates with expanding features. F_{i+1} stands for newly introduced features.

performing feature selection on the data with newly introduced features and we end up with very little data; and some feature-values in the removed data may be critical. This problem of expanding features periodically appears as illustrated in Figure 5. We need a solution to accommodate the newly introduced features and update the data so that data with F_0 and F_1, for example, are in sync with each other.

In this paper, we are experimenting with the assumption that the new data has a significantly smaller number of instances than that of the existing data (for example, $\#(D_0)$ $>> \#(D_1)$, and $\#(D_0 + D_1) >> \#(D_2)$). Another assumption is that the number of newly introduced features is significantly smaller than that of the existing features. In other words, we are considering a feature set that is gradually expanding with new data being collected. After every change, with D_0, we have D_1, and we will try to merge D_0 and D_1 into one data set D'_1. Therefore, we can just consider two data sets D_i - the existing one, and D_{i+1} - the newly collected data with newly introduced features F_{new}. Let D^*_{i+1} be part of the

for each instance I_0 in D_0
  for each I_1^i in D_1
    $d_i = \text{match}(F_0, I_0, I_1^i)$
  find I_1^{max} where its $d_{max} = \max_i\{d_i\}$
  if $d_{max} \geq \theta \wedge \text{Class}(I_1^{max}) = \text{Class}(I_0)$
    fill in missing values of I_0 with values of I_1^{max} for F_1
  else
    P = subset of D_1^* with $Class(I_0)$
    p_{most} = pattern in P with highest frequency
    fill in missing values of I_0 with p_{most}

Figure 6: An algorithm for handling expanding features.

data D_{i+1} described by F_{new}, and D_i^* be the extended part of the data D_i due to F_{new}. Our suggested algorithm is (1) to use D_{i+1}^* to help fill in proper values of D_i^* and (2) perform feature selection on the combined data to remove any irrelevant features.

The second part of the algorithm can be handled by LVF or some other feature selection algorithms (Dash & Liu, 1997). It is the first part that requires special attention here. Whenever we introduce new features and collect some data, we need to merge the existing data D_i with the newly collected data D_{i+1}. The major problem is that there are no values for the newly introduced features F_{new} in D_i. How can we fill in the missing values? Before answering this question, we investigate an alternative that leaves them (rectangles in dashed lines in Figure 5) as they are and consider them as missing. There are two disadvantages: (1) newly introduced features are usually important (that is the reason why they were introduced), and leaving them out would not be wise; and (2) if we keep leaving them out, in a long run, we will end up with a data set with many missing values. Therefore, we need to find proper values for F_{new}. In this context, the conventional approaches (for example, using a new value for all missing values, or using the most frequently occurring values to replace missing values) to handling missing values do not work because we have very few new instances compared to the existing data. Using D_0 and D_1 as an example (existing features are F_0, and F_{new} is F_1), we explain our suggestion in Figure 4.

Function match(F_0, I_0, I_1^i) returns a value d_i which is calculated as the number of feature-values that are the same divided by the number of features F_0. θ is a threshold that can be determined by the user. We set its default value as 0.5, that is, only if more than half of the features have the same values, then we use I_1^{max} for the task. We start with the existing data D_0 because we need to fill in missing values for each of its instances. Through steps 2 - 4, we try to find an instance I_1^{max} in the new data D_1 that has the maximum number of the same values for F_0 as I_0. If we can find such an instance I_1^{max} with d_{max} that is greater than θ, we check whether the class labels of I_1^{max} and I_0 are the same or not (step 5). If they are the same, we fill in values of F_1 for I_0 with those of I_1^{max} (step 6). This is sensible because we assumed earlier that some or most existing features are still relevant even after new features are introduced. If I_1^{max} has a different class label from I_0, it suggests that newly introduced features play a pivotal role in changing the class label of the new instance. This is also consistent with our assumption that new features are introduced only when they are considered potentially useful. Hence, we can concentrate on the new features and disregard the existing ones. Through steps 7 - 9, we focus on features F_1. Let P be the subset of patterns in D_1^* with the same class label as I_0 and p_{most} be the most frequently occurring pattern in P, we fill in the missing values of I_0 by p_{most}. For implementation efficiency, this calculation can actually be done for each class label once and for all. The work described here is still on-going research. We expect to apply this work to image processing related tasks as our first application.

5 Conclusions and Further Work

We have identified three issues related to scalable features selection and proposed solutions to each issue correspondingly. The first two are about handling large number of features and instances. The third is about dealing with expanding features in a case where new features are gradually introduced. In addition to their proposed applications, LVS and LVF can find other uses as well. As mentioned earlier, although Las Vegas algorithms may not be as fast as some domain specific heuristic methods, LVS or LVF can still play a role as a reference in the design of a domain specific heuristic method. This is because to verify a heuristic method is not an easy task, especially when data sets involved are huge. LVS can be most helpful in this case. Another feature is that LVF may produce many equally good solutions on real-world data sets. We can use, for example, a learning algorithm to choose a suitable solution. Hence, one straightforward extension of this work is a model combining both filter and wrapper approaches (John et al., 1994). Another line of research is to search for other criteria as alternatives to the inconsistency criterion.

Parallel feature selection is another line of further research. Instead of relying on one processor for the task, we can make use of multiple processors or cluster systems. Techniques such as sampling, data partitioning, data placement (declustering) have been mentioned in (Freitas & Lavington, 1998) with many possibilities for parallelism. Parallel feature selection will have many uses as data sets are increasingly large. We expect that good results in this research area will greatly increase the scalability of some feature selection algorithms. Data warehousing (Devlin, 1997) can also help boost the scalability of a feature selection algorithm. A data warehouse is simply a single, complete, and consistent store of data obtained from a variety of sources and made available to end users in a way they can understand and use in a business context, through various techniques such as partitioning, aggregation, data marting, and meta data (Anahory & Murray, 1997). The data obtained through data warehousing has gone through a preprocessing process and meta-data is usually available as a by-product. We can use this meta-data as a catalog that describes the data in its business context and acts as a guide to the location and use of this information, and we will be able to better choose and apply feature selection algorithms. This opens another avenue for feature selection among large data.

References

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

Anahory, S., & Murray, D. (1997). *Data warehousing in the real world.* Addison Wesley Longman, Inc.

Blum, A., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, *97*, 245-271.

Boddy, M., & Dean, T. (1994). Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, *67*(2), 245-285.

Brassard, G., & Bratley, P. (1996). *Fundamentals of algorithms.* Prentice Hall, New Jersey.

Dash, M., & Liu, H. (1997). Feature selection methods for classifications. *Intelligent Data Analysis: An International Journal*, *1*(3). (`http://www-east.elsevier.com/ida/free.htm`)

Devijver, P., & Kittler, J. (1982). *Pattern recognition: A statistical approach.* Prentice Hall International.

Devlin, B. (1997). *Data warehouse from architecture to implementations.* Addison Wesley Longman, Inc.

Freitas, A., & Lavington, S. (1998). *Mining very large databases with parallel processing.* Boston/Dordrecht/London: Kluwer Academic Publishers.

John, G., Kohavi, R., & Pfleger, K. (1994). Irrelevant feature and the subset selection problem. In W. a. H. H. Cohen (Ed.), *Machine learning: Proceedings of the Eleventh International Conference* (pp. 121–129). New Brunswick, N.J.: Rutgers University.

Kira, K., & Rendell, L. (1992). The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 129–134). Menlo Park: AAAI Press/The MIT Press.

Liu, H., Motoda, H., & Dash, M. (1998). A monotonic measure for optmial feature selection. In C. Nedellec & C. Rouveirol (Eds.), *Machine Learning: ECML-98, april 21 - 23, 1998* (pp. 101–106). Chemnitz, Germany: Berlin Heidelberg: Springer-Verlag.

Liu, H., & Setiono, R. (1995). Chi2: Feature selection and discretization of numeric attributes. In J. Vassilopoulos (Ed.), *Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence, november 5-8, 1995* (pp. 388–391). Herndon, Virginia: IEEE Computer Society.

Liu, H., & Setiono, R. (1996). A probabilistic approach to feature selection - a filter solution. In L. Saitta (Ed.), *Proceedings of International Conference on Machine Learning (ICML-96), July 3-6, 1996* (pp. 319–327). Bari, Italy: San Francisco: Morgan Kaufmann Publishers, CA.

Mehra, P., Rendell, L., & Wah, B. (1989). Principled constructive induction. In *Proceedings of International Joint Conference on AI* (pp. 651–656). Detroit, Michigan.

Modrzejewski, M. (1993). Feature selection using rough sets theory. In P. Brazdil (Ed.), *Proceedings of the European Conference on Machine Learning* (pp. 213–226). Vienna, Austria.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, *5*, 71-99.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81-106.

Quinlan, J. (1993). *C4.5: Programs for machine learning.* Morgan Kaufmann.

Setiono, R., & Liu, H. (1997). Neural network feature selectors. *IEEE Trans. on Neural Networks*, *8*(3), 654-662.

Watanabe, S. (1985). *Pattern recognition: Human and mechanical.* Wiley Interscience.

Widmer, G. (1996). Recognition and exploitation of contextual clues via incremental meta-learning. In L. Saitta (Ed.), *Machine Learning: Proceedings of the Thirteenth International Conference* (p. 525-533). Bari, Italy: Morgan Kaufmann Publishers.

Wyse, N., Dubes, R., & Jain, A. (1980). A critical evaluation of intrinsic dimensionality algorithms. In E. Gelsema & L. Kanal (Eds.), *Pattern recognition in practice* (pp. 415–425). Morgan Kaufmann Publishers, Inc.