Towards an Evolutionary Algorithm: A Comparison of Two Feature Selection Algorithms

Kan Chen[1] and **Huan Liu**[2]
[1]Dept of Computational Science, kan_chen@nus.edu.sg
[2]School of Computing, liuh@comp.nus.edu.sg
National University of Singapore, Kent Ridge, Singapore 119260

Abstract- In order to deal with a large number of attributes, probabilistic feature selection algorithms have been proposed. Pure random walk entails mediocre performance in search time. Introducing adaptiveness into a probabilistic algorithm can lead to more focused search that means better search time. We compare two algorithms here in search of an efficient but not myopic algorithm for feature selection. Based on the comparative study, we suggest some ways of improvement towards an evolutionary feature selection algorithm for data mining.

1 Introduction

Researchers show that optimal feature selection is NP-hard [22]. Because of its practical importance, feature selection has been a fertile field for new algorithms development [13, 2, 7, 12]. Having more features implies more discriminating power in inductive learning. However, it renders an inductive algorithm impractical for problems with a large number of features as the time requirements for the algorithm often grow dramatically with the number of features. Moreover, adopting the Occam's Razor bias often leads many algorithms to prefer a small number of relatively predictive features to a large number of features [16]. The benefits of reducing features for inductive learning are: it considerably decreases the run-time of an inductive algorithm, and increases the accuracy of the resulting model.

This work focuses on two feature selection algorithms: a Las Vegas one and an adaptive one. The objective is to study in detail the two algorithms and observe how the problems of one algorithm can be overcome by the other and what new problems are. Based on the experimental results, we will evaluate, in the context of data mining, the pros and cons of an evolutionary algorithm for feature selection yet without using classification accuracy as a part of a fitness function [23, 12, 17]. We investigate the factors that should be considered for such an evolutionary algorithm and propose directions of evolutionary feature selection for data mining.

2 Basic Concepts

A data instance is typically described to a classification learning algorithm as an assignment of values $(a_1, a_2, ..., a_N)$ to a set of features $(A_1, A_2, ..., A_N)$ with a class label l. If f is the underlying function to be learned, $l = f(a_1, a_2, ..., a_N)$. In presence of irrelevant and/or redundant features, the task of feature selection is to select M relevant features from N originally given features where $M \leq N$.

We use a string of bits to indicate corresponding features' inclusion or exclusion in the resulting feature set. This bit representation is commonly used in many optimization problems solving [11]. A solution can be represented by a string of bits $S = \{b_i\}, i = 1, 2, ..., N$ in which $b_i = 1$ denotes the entity's inclusion, and $b_i = 0$ exclusion. For example, this bit-representation has been used directly in the study of the spin glass problem [4] and graph-partitioning problem [6, 5]. For the feature selection problem, if the bit has value 1, the feature is selected; otherwise the bit has value 0, the feature is not selected. Given selected features, we can evaluate the number of inconsistencies IC (to be explained later) for a given data set.

Classically, feature selection is treated as a search problem to find the minimal number of bits that have value 1 [22, 23, 1] with which some criterion is satisfied. The most intuitive approaches are those of sequential forward and backward search (SFS and SBS). SFS starts with an empty set and adds the best unselected feature into the set one at a time. SBS begins with the full set and removes the least relevant feature one at a time. The goodness of a subset varies with the measure used. Depending on what is used for evaluation, filter and wrapper models are designed. A filter model normally uses a measure that is simple and fast to compute; a wrapper model engages a learning algorithm to measure classification accuracy [15] and it is usually more expensive computationally. If we consider accuracy suggested by a classifier as a normal measure *per se*, the difference of the two models diminishes [18]. Different measures have their own merits. If high accuracy is the objective, a classifier should be used as the measure. If the concerns are about dimensionality reduction and removing the redundant, a cheaper measure should be chosen. In the context of data mining, it is natural for us to assume that our objective is the latter and consider a filter model only.

In this paper, we will focus on probabilistic algorithms using a goodness measure called *inconsistency rate*. This measure has been shown *monotonic* in [19]. That is, let $\text{IR}_S(D)$ be the inconsistency rate of the data D measured on a feature subset S, if $S_1 \supset S_2$, then $\text{IR}_{S_1}(D) \leq \text{IR}_{S_2}(D)$. The lower the rate, the better a subset of features. Hence, the task of feature selection is to find the smallest subset of features that can maintain the lowest inconsistency rate. The inconsistency rate of a dataset is calculated as follows: (1) two instances are considered inconsistent if their attribute-values are the same (i.e., they match) except for their class labels; (2) for all the matching instances (without considering their class labels), the inconsistency count is the number of the instances minus the largest number of instances of 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 $(n - c_3)$; and (3) the inconsistency rate is the sum of all the inconsistency counts (IC_k) divided by the total number of instances (**P**), or $IC = (\sum_k IC_k)/\mathbf{P}$.

3 Two Algorithms

The objective of this study is to see if an adaptive version of the Las Vegas algorithm can be fast in obtaining optimal or near-optimal subsets of features. A subsequent goal is to explore an evolutionary algorithm with the inconsistency measure based on the findings. It is an evolutionary procedure to evolve a new algorithm.

3.1 Las Vegas algorithm - LVF

Las Vegas algorithms are a type of probabilistic algorithms that always produce correct results, with the only variation from one run to another being in its running time [21, 3]. Probabilistic choices are used only to guide the algorithm more quickly towards the desired solution, in which case the answer obtained is always correct. If a solution is found to be wrong, the Las Vegas algorithm admits that it fails and keeps trying a new solution. A general form of Las Vegas (LV) algorithms for feature selection can be described as:

function *LVFS(S, K)*

1. Input: K
2. $k = 0$
3. $S_{new} = LV(S)$ and k++
4. **if** S_{new} is better than S_{best}
   $S_{best} = S_{new}$
5. **if** $k \leq K$ Go to Step 3;
   otherwise stop
6. Output: the best solution S_{best}

S is the full feature set, S_{new} is a randomly generated subset, S_{best} is the best subset found so far. For simplicity purpose, we use K, the predefined number of runs, as our sentinel for the loop. As LVFS keeps S_{best}, we may interrupt LVFS anytime and request for S_{best}. Depending on the goodness measure and/or stopping criterion we choose, many versions of LV algorithms can be designed. LVF [20] is an algorithm that uses the inconsistency measure to evaluate the goodness of a feature subset. A subset is good if its inconsistency rate is less than or equal to a pre-specified value (by default, it is 0). LVF generates a random subset of features in each run and the subset is evaluated using the inconsistency measure.

As a particular probabilistic choice of S_{new} being bad is very unlikely repeated (with a probability of $\frac{1}{2^N}$), and there are many good choices due to the monotonicity of the inconsistency measure, LVF should find a good solution quickly. However, the optimal case is that LVF can efficiently return a good subset with the minimum number of features. As the calculation of the inconsistency rate can be done efficiently ($O(\mathbf{P})$ [20]), if we insist that a new subset be smaller than a found one, we can expect that the longer LVF runs, the better the result it returns.

Some analysis shows that LVF should get slower and slower in approaching the smallest subset - a problem of *degradation*. Assuming that there are 10 features and only three are relevant, the number of good subsets in the beginning is $2^7 = 128$, or we should expect a good solution after about 10 runs ($= 128/1024$). If a good solution consists of 5 features (a super set of the 3 relevant features), the number of equally good or better subsets is only $\binom{7}{(5-3)} + \binom{7}{1} + 1 = 29$, or the chance is about $2.8\% (= 29/1024)$. This analytical result has also been confirmed in experiments reported in [8]. In other words, LVF is efficient in the beginning to reduce the number of features and less so when the number of selected features becomes smaller.

3.2 An adaptive algorithm - WLVF

If we look into the source of the degradation problem, we can find that it is caused by assuming that every feature has a 0.5 probability to be on/off even when we have some knowledge about features after generating many subsets of features. The crux of the matter is how we modify this probability for each feature without significantly increasing the complexity of LVF. The work of Relief [14] and the work on the Spin-Glass problem [4] have inspired us to approach the degradation problem with LVF via another perspective. Relief samples the data to assign/update a weight to each feature relying on concepts of near-miss and near-hit which are two nearest neighbors of I (one from each class). Ideally, a feature is relevant if its values are the same between an instance I and its near-hit, and different between I and its near-miss, where I is a randomly picked instance. Spin glasses are disordered magnetic materials that have been studied by physicists for many years in their own right. The existence of many local energy minima makes it difficult to find the ground state (lowest energy state) of a spin glass system. Instead of keeping all configurations, the learning algorithm in [4] maintains probability weights that are associated with each spin in a configuration. These weights help generate new configurations probabilistically. This allows many configurations to be kept implicitly, yet learning improves over time.

The intuition is that we could capture the knowledge gained in generating and testing feature subsets by updating the *on/off* probability of each feature according to its involvement in each run. We can increase a feature's on probability if it occurs in a good feature subset (recall that the goodness of a feature set is determined by the inconsistency measure), otherwise increase its *off* probability, or equivalently decrease its on probability. Features frequently tested as good should have higher on probabilities, others have lower probabilities. When all features' probabilities converge to 1 or 0, feature selection is completed. This idea can only work when relevant features have more opportunities to be included in good subsets. This requirement does not cause a problem. As we understand in our analysis of the degradation problem in the previous subsection, continuing from the above 10-feature example, the chances for the 3 relevant features to be considered as good are not 1 but 128 out of 1024. This is due to the monotonicity of our inconsistency measure - any superset of the relevant feature set is also a good set.

We now describe the adaptive algorithm. We associate a weight w_v with each bit b_v, such that the probability of choosing $b_v = 1$, $p(b_v = 1)$, is $w_v/(1 + w_v)$ and the probability of choosing $b_v = 0$, $p(b_v = 0)$, is $1/(1 + w_v)$. The adjustment of a weight w_v is linked to a cost determined by two terms: (1) number of features selected in a good subset; and (2) number of inconsistencies due to the selected subset. We intend to minimize both terms for the lowest cost. Naturally, without removing any features, there is usually no inconsistency. While maintaining 0 inconsistency, the smaller the set, the lower the cost. By having the second term, we attempt to further reduce features by allowing some inconsistencies. This is particularly useful in presence of noise. A history list H contains last m solutions found so far and m is a predetermined number. Let N_s be the number of attributes selected, IC, N and **P** are as defined before, we define the following cost function for each solution:

$$\text{if } IC = 0,\ c = N_s,$$
$$\text{else } c = N + N\frac{\log(IC)}{\log(\mathbf{P})}.$$

The algorithm WLVF is outlined as follows

---

1. Input: cost function c.
2. Initialize weights w_v's with 1 ($i.e.$, all $p(b_v = 1) = .5$)
3. Generate feasible solutions for H with $p(b_v)$'s.
4. i) Generate a feasible solution S_f based on $p(b_v)$'s
   ii) Compare S_f with those in H and adjust w_v's to favor the better solution
   iii) Update H with the above solution
5. Go back to Step 4 until the stopping criterion is met.
6. Output: The solution with the lowest cost.

Given the weights that are initialized with 1, we can generate feasible solutions (bit strings) by setting the bits probabilistically according to their weights. After a new feasible solution S is generated, it is compared with each solution S' in the history list H. Let $S = \{b_1, b_2, ...\}$ and $S' = \{b'_1, b'_2, ...\}$. Let the corresponding values of the function be c and c' for S and S', respectively. The weights are then modified as follows

$$w_i^{new} = w_i^{old} e^{-\alpha(c-c')(b_i - b'_i)},\ \ i = 1, ..., N, \qquad (1)$$

where α represents the modification rate of weights (or the learning rate of the algorithm). It is not easy to determine an appropriate value for α in practice so we incorporate an adaptive learning rate in WLVF. This can be done by having $\alpha = \alpha_0/\sigma$, where σ is the standard deviation of the costs for the previous m solutions in the history list H ($m = 10$ in our implementation). σ can also be used in the stopping criterion of the algorithm: WLVF stops when $\sigma/\bar{c}$, (where $\bar{c}$ is the average cost for the solutions in H) is less than a certain value ϵ (e.g., $\epsilon = 0.0001$) which is small enough to ensure that the convergence is reached. m and ϵ are rather robust and we used the above values in experiments. As the learning of weights advances, the individual bits will gradually be "frozen" to form a near-optimal bit string.

4 Experimental Results

Based on the above analysis, we expect that WLVF should perform better in terms of time spent to find relevant features. We experimented with data sets of various sorts to test if our expectation is realistic: some with known relevant features (Monks), some with large numbers of features (Soybean, Dermatology), some with many classes (Led24, Soybean), and some with known redundant features (Par10+10). This variety in the chosen data sets allows us to observe how probabilistic feature selection algorithms behave. The characteristics of the data sets used are shown in Table 1. Par5+5 and Par10+10 are parity-5 and parity-10 respectively having varying numbers of irrelevant and redundant features. Par5+5 has 10 features but only its first 5 features determine the class of an instance. Par10+10 has 5 of its 10 relevant features duplicated with another 5 irrelevant features. For data with training and test sets, we use the training data for selection; for other data sets, if the number of instances is less than 2000, we use all for selection, otherwise, we use 2/3 for selection after shuffling.

Table 1: Data characteristics. #I - number of instances, #I$_F$ - number of instances used in feature selection, #F - number of features, and #C - number of classes.

Data	#I	#I$_F$	#C	#F
Soybean	307	307	15	35
Par5+5	1024	1024	2	10
Mush	8142	5416	2	22
Vote	135	135	2	16
Monk1	432	124	2	6
Monk2	432	169	2	6
Monk3	432	122	2	6
Hepat	155	155	2	19
Led24	20000	13332	10	24
Derma'y	366	366	6	34
Par10+10	32768	21845	10	20
Zoo	67	67	7	16
TicTacToe	639	639	2	9

As both LVF and WLVF are probabilistic methods, we run 30 times independently and obtain an average number of selected features to report its average case behavior. WLVF stops automatically (when it stabilizes), but LVF can run forever. So, we impose a time limit in the experiment. Therefore, LVF sometimes has two numbers, the smaller number is the time needed to get the number of features reported in Table 2 and the other number is the time limit. OPT in the table is an optimal number of features selected by an exhaustive feature selection algorithm (ABB) [19] using the inconsistency measure. Using the same inconsistency measure, the smaller the difference of the average number and the optimal number of features for a data set, the better a probabilistic algorithm is. The conclusion drawn from the results is that WLVF is faster and chooses fewer features. It is a promising step to move away from random walk into an adaptive manner. In the next section we will discuss how WLVF compares with a genetic algorithm and whether we should move further towards an evolutionary algorithm.

Table 2: Numbers of features selected and times (in seconds of cpu time) by LVF and WLVF.

	#Features			Time	
Data	LVF	WLVF	Opt	LVF	WLVF
Soybean	15.0	9.4	9	2.1/2.7	1.3
Par5+5	5.7	5.0	5	2.8/3.7	0.7
Mush	5.1	4.2	4	14.5/15.5	15.5
Vote	5.5	4.0	4	0.36	0.16
Monk1	3.0	3.0	3	0.16	0.06
Monk2	6.0	6.0	6	0.015	0.015
Monk3	4.03	4.0	4	0.11	0.06
Hepa	4.0	3.0	3	0.17	0.17
Led24	9.7	5.0	5	150.1/162.1	58.1
Derma'y	9.5	6.3	6	1.5	1.5
Par10+10	11.7	10.0	10	104.1	50.1
Zoo	4.9	4.0	4	0.11	0.05
TicTacT	7.0	7.0	7	1.7/4.3	0.10
AVE	7.01	5.45	5.38	21.36	9.83

5 Towards an Evolutionary Algorithm

At the first glance, WLVF may look similar to a genetic algorithm (GA) [10]. Both algorithms can have many solutions - equally good feature subsets. WLVF keeps a list of solutions in H that were formerly generated, while GA maintains a population with fittest solutions. However, similarity ceases here as new solutions are generated in different ways for the two algorithms. WLVF's generation of new solutions is not based on the existing solutions in H. Instead, it is based on the probability $p(b_v = 1)$ which is a function of w_v. The probability weights w_v's are updated. WLVF's behavior is independent of the size of H since the evolution is about the weights, unlike in a GA implementation, its population size makes a difference. The direct advantage is that we are able to keep the size of the history list H small (as small as 1) even for a large instance size **P**; this is not possible for GA that is based on the evolution of a large population of solutions. Some comparative studies of GA and other methods can be found in [9, 12]. Observations by these studies suggest that (1) the search space GA needs to explore increases faster than SFFS - a floating SFS, or the performance of GA degrades as the dimensionality increases [9], and (2) the GA method suggested by [23] has several parameters for which no guidance is available on how to specify their values as reported by [12], and their conclusion is that it is difficult to compare its performance with others due to its heavy influence by the subset size.

In addition to the above differences, we wish to point out another distinct difference between the two algorithms: the cost function vs. a fitness function. Almost all, if not all, fitness functions for feature selection algorithms contain an important component - classification accuracy. This makes it necessary to train a classification algorithm for every genetically manipulated feature subset. In the context of data mining, the requirement of training a classification algorithm may not be feasible, especially when feature selection is needed for data reduction. One of the fundamental reasons to perform feature selection is that a classification algorithm cannot even run due to the sheer size of huge data. Our work suggests an alternative to a conventional fitness function. In our previous work [19, 8], we showed (1) that feature subsets selected using the inconsistency measure are consistent with the known relevant features, and (2) that features selected using the inconsistency measure usually lead to improved predictive accuracy. Therefore, in the context of data mining, we can consider incorporating the inconsistency measure into a fitness function and taking away the component of classification accuracy.

On the other hand, an adaptive algorithm will have a possibility to converge to a local optimum due to its adaptive nature, although we have not clearly observed this tendency in our experiments. However, such a reality is indicated by the differences between the optimal numbers of selected features and the the numbers of features selected by WLVF for various data sets. Two main issues are worthy of pursuing further. One is to avoid the local optima and the other is to speed up further. Many successful applications of genetic operators lead us to investigate if they can help in both accounts. The consequence of using these operators is that more parameters need to be set for using these genetic operators. It may not be bad in practice as a human expert can intervene more in the feature selection process according to various requirements and different circumstances. Nevertheless, it also means that the degree to which the process is automated is decreased. To solve this dilemma is our immediate future work by introducing some genetic operators and observing the subsequent changes in designing an evolutionary algorithm.

6 Conclusion

Adopting an evolutionary strategy, we studied two algorithms for feature selection. One is a Las Vegas algorithm, the other is the one with an adaptive mechanism. The differences between the two were observed and analyzed. We confirmed the positive aspects of the second algorithm and gains obtained due to its adaptiveness. We pointed out that this new adaptive algorithm is more appropriate for data mining applications. Encouraged by the successful use of genetic operators in many applications [25, 24], we set out our new task to introduce necessary genetic operators to further improve the performance of WLVF. The future version of WLVF in that direction will entail an evolutionary algorithm with its gains and losses checked.

Acknowledgments

We thank Mohammad Mahmud Hasan for helping in conducting experiments and compiling results for this work, and Manoranjan Dash for careful proof-reading.

Bibliography

[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] A.L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271, 1997.

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

[4] K. Chen. A general learning algorithm for solving optimization problems and its application to the spin glass problem. *Europhysics Letters*, 43:635 – 640, 1998.

[5] K. Chen. An effective learning algorithm for the graph partitioning problem. *IEEE Transactions on Evolutionary Computation*, submitted.

[6] C. B. Chua and K. Chen. General learning algorithm for the graph-partitioning problem. *International Journal of Modern Physics C*, 9:331–339, 1998.

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

[8] M. Dash and H. Liu. Hybrid search of feature subsets. In H.Y. Lee and H. Motoda, editors, *Proceedings of the Fifth Pacific Rim International Conference on AI (PRICAI'98), November 22 - 27, 1998*, Singapore, 1998. Springer-Verlag.

[9] F.J. Ferri, P. Pudil, M. Hatef, and J. Kittler. Comparative study of techniques for large-scale feature selection. In *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, pages 403 – 413. Elsevier, 1994.

[10] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc, 1989.

[11] D.S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997.

[12] 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.

[13] G.H. John, R. Kohavi, and K. Pfleger. Irrelevant feature and the subset selection problem. In W.W. asnd Hirsh H. Cohen, editor, *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129, New Brunswick, N.J., 1994. Rutgers University.

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

[15] R. Kohavi and G.H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[16] D. Koller and M. Sahami. Toward optimal feature selection. In L. Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference (ICML-96), July 3-6, 1996*, pages 284–292, Bari, Italy, 1996. San Francisco: Morgan Kaufmann Publishers.

[17] H. Liu and H. Motoda, editors. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Boston: Kluwer Academic Publishers, 1998.

[18] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery Data Mining*. Boston: Kluwer Academic Publishers, 1998.

[19] H. Liu, H. Motoda, and M. Dash. A monotonic measure for optmial feature selection. In C. Nedellec and C. Rouveirol, editors, *Machine Learning: ECML-98, April 21 - 23, 1998*, pages 101–106, Chemnitz, Germany, April 1998. Berlin Heidelberg: Springer-Verlag.

[20] 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), July 3-6, 1996*, pages 319–327, Bari, Italy, 1996. San Francisco: Morgan Kaufmann Publishers, CA.

[21] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[22] 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.

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

[24] H. Vafaie and K. De Jong. *Evolutionary Feature Space Transformation*, pages 307 – 323. In Liu and Motoda [17], 1998.

[25] J. Yang and V. Honavar. *Feature Subset Selection Using A Genetic Algorithm*, pages 117 – 136. In Liu and Motoda [17], 1998.