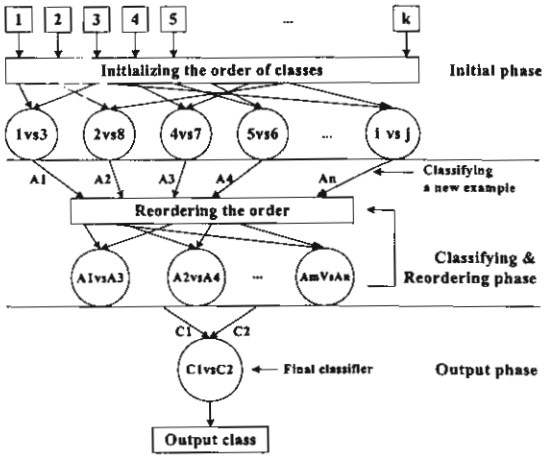


Fig. 4. Classifying process of the RADAG.

concerned with determining the factors that affect the accuracy of a pattern classifier [3]. Generalization performance of SVMs can be approximated by bounding on the generalization error. Define the class F of real-valued functions on the ball of radius R in $\mathbb{R}^n$ as $F = \{\mathbf{w} \mapsto \mathbf{w} \cdot \mathbf{x} : ||\mathbf{w}|| \leq 1, ||\mathbf{x}|| \leq R\}$. There is a constant c such that, for all probability distributions, with probability at least $1 - \delta$ over l independently generated examples $\mathbf{z}$, if a classifier $h = sgn(f) \in sgn(F)$ has margin at least γ on all the examples in $\mathbf{z}$, then the error of h is no more than

$$\frac{c}{l}\left(\frac{R^2}{\gamma^2}log^2 l + log\left(\frac{1}{\delta}\right)\right) \tag{1}$$

Furthermore, in case that the training data cannot be separated by the hyperplane without error, with probability at least $1 - \delta$, every classifier $h \in sgn(F)$ has error no more than

$$\frac{k}{l} + \sqrt{\frac{c}{l}\left(\frac{R^2}{\gamma^2}log^2 l + log\left(\frac{1}{\delta}\right)\right)} \tag{2}$$

where k is the number of labeled examples in $\mathbf{z}$ with margin less than γ. Below we show an example of the generalization error of classifiers. The experiment is based on the English letter image recognition data set from [4], which has 26 classes. Hence there are 325 classifiers. The classifiers are trained by using the Polynomial kernel of degree 3. In Figure 5, the generalization errors of all classifiers expressed by Equation 2 are depicted. The generalization errors of all classifiers are varying. The average of all of them is 28.82.

5.4 Algorithm

For the reason described above, we consider the generalization errors in order to choose the optimal order with less chance to predict the wrong class from

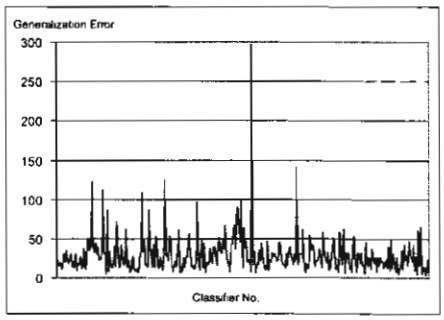


Fig. 5. The generalization errors of 325 classifiers.

all possible $\frac{N!}{2^{\lfloor N/2 \rfloor}\lfloor N/2 \rfloor!}$ orders. Among $N(N-1)/2$ classifiers, $N/2$ classifiers which have small generalization errors will be considered to be used in data classification.

Let $G = (V, E)$ be a graph with node set V and edge set E. Each node in G denotes one class and each edge denotes one binary classifier which has a generalization error from equation 2 (see Figure 6(a)). The output of the reordering algorithm for graph G is a subset of edges with the minimum sum of generalization errors of all edges and each node in G is met by exactly one edge in the subset (see Figure 6(b)). Given a real weight c_e being generalization error for each edge e of G, the problem of reordering algorithm can be solved by the minimum weight perfect matching that finds a perfect matching M of minimum weight $\sum(c_e : e \in M)$.

For $U \subseteq V$, let $E(U) = \{(i,j) : (i,j) \in E, i \in U, j \in U\}$. $E(U)$ is the set of edges with both endpoints in U. The set of edges incident to node i in the node-edge incidence matrix is denoted by $\delta(i)$. The convex hull of perfect matchings on a graph $G = (V, E)$ with $|V|$ even is given by
a) $x \in R_+^m$
b) $\sum_{e \in \delta(v)} x_e = 1$ for $v \in V$
c) $\sum_{e \in E(U)} x_e \leq \lfloor \frac{|U|}{2} \rfloor$ for all odd sets $U \subseteq V$ with $|U| \geq 3$
or by (a),(b) and
d) $\sum_{e \in \delta(U)} x_e \geq 1$ for all odd sets $U \subseteq V$ with $|U| \geq 3$
where $|E| = m$ and $x_e = 1$ means that e is in the matching. So the minimum weight of a perfect matching is at least as large as the value of

$$min \sum_{e \in E} c_e x_e \tag{3}$$

where x satisfies "(a),(b) and (c)" or "(a),(b) and (d)". Therefore, the reordering problem is to solve the linear program in Equation 3.

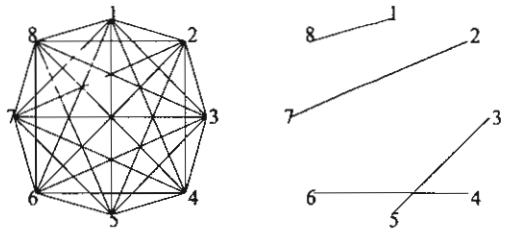


Fig. 6. (a) A graph for an 8-class problem (b) An example of the output of the re-ordering algorithm.

5.5 Estimating the Difference between Means of Generalization Errors

In this subsection we will explain test concerning means of generalization errors of classifiers of interest (the binary classifier of the correct class and other classes in the order of classes), which are selected by the ADAG and the RADAG. For the ADAG, we randomly selected the classifiers to be used in data classification, whereas for the RADAG the classifiers are selected by using the minimum-weight perfect matching algorithm. This may be analyzed using a method called the one tailed paired t test.

To estimate the difference between two means, we wish to test

$$H_0 : \mu_1 - \mu_2 = 0,$$
$$H_1 : \mu_1 - \mu_2 > 0.$$

The point estimate of the difference $\mu_1 - \mu_2$ of two population means is given by $\bar{X}_1 - \bar{X}_2$. The form of the paired t test is

$$t = \frac{\bar{X}_1 - \bar{X}_2}{S_d/\sqrt{n}} \tag{4}$$

where S_d is the standard deviation of the sample of differences $\bar{X}_1 - \bar{X}_2$.

Assume that the distribution of generalization errors is normal distribution. The simulation is based on the dataset which has 26 classes, and hence there are 325 binary classifiers. In the experiment, a generalization error is randomly selected to each classifier with equal probability, uniformly distributed. We examine 5,200 orders of classifiers, where for the first 200 orders we assume that the correct class is class 1, for the second 200 orders we assume that the correct class is class 2 and so on. For the ADAG, we randomly selected 5,200 orders of classifiers. For the RADAG, 5,200 orders are selected by using the minimum-weight perfect matching algorithm so all orders are the same. Then we compute sum of generalization errors of classifiers of interest that correspond to the binary classifier of the correct class and other classes in the order. The experiments are repeated 100 times. Then we calculate the sample mean $\bar{X}_1$ ($\bar{X}_{ADAG}$) and $\bar{X}_2$

($\bar{X}_{RADAG}$). Using $\alpha = 0.001$ with a one-tailed test and degree of freedom = 99, the null hypothesis is rejected. This means that the RADAG obtains classifiers whose generalization errors are statistically significantly lower than the ADAG.

6 Experiments

In the experiments, we compared the accuracy of classification of four algorithms, i.e., the Max Wins, the DDAG, the ADAG, and the RADAG. We also compared the computational time between the ADAG, the RADAG and the Max Wins.

6.1 Data Set and Experimental Setting

The experiments are based on several data sets from the UCI Machine Learning Repository [4] including glass, satimage, segment, shuttle, vowel, soybean, letter and isolet (see Table 1). For the soybean problem, we discarded the last four classes because of missing values. In addition, we examined our methods with Thai printed character recognition [20], which has 68 classes including 44 consonants, and 26 vowels and tonal masks. For the training set, the characters were printed by laser printer with 600 dpi resolution. Then they were copied by a copier machine with saturated ink. There are two test sets. For the test set of Thai printed character 1 data set, the charecters were printed by laser printer with 600 dpi resolution. For the test set of Thai printed character 2 data set, the characters were printed by laser printer with 600 dpi resolution. Then they were copied by a copier machine with pale ink. These data sets are different in the number of classes, the number of dimensions, and sizes. 68 classes is the most classes and 617 dimensions is the most dimensions which are used in the experiment.

In these experiments we scaled both training data and test data to be in [-1,1] and employed Polynomial and RBF kernels. In the training phase, the $N(N-1)/2$ binary classifiers were constructed by using the software package called SVM^{light} version 5.0 [13, 14]. For the DDAG and the ADAG, we examined all possible orders of classes for datasets having not more than 8 classes, whereas we randomly selected 50,000 orders for datasets having more than 8 classes. We then calculated the average of accuracy of these orders.

Table 1. Description of the datasets used in the experiments.

Dataset	Training data	Test data	Classs	Dimension
Glass	214	-	6	9
Satimage	4,435	2,000	6	36
Segment	2,310	-	7	18
Shuttle	43,500	14,500	7	9
Vowel	528	462	11	10
Soybean	290	340	15	35
Letter	15,963	4,037	26	16
Isolet	6,238	1,559	26	617
ThaiPrintedCharacter1	3,264	3,264	68	128
ThaiPrintedCharacter2	3,264	3,264	68	128

6.2 Experimental Results

Table 2 and Table 3 present the results of comparing four methods including the Max Wins, the DDAG, and our methods, the ADAG and the RADAG. We present the optimal parameters, d in the Polynomial kernel $|(\mathbf{x} \cdot \mathbf{y} + 1)|^d$ and c in the RBF kernel $exp(-|\mathbf{x} - \mathbf{y}|^2/c)$. The best accuracy among these methods is illustrated in bold-face. ****, ***, ** and * in the tables mean 99.99%, 99%, 95%, and 90% confidence interval for estimating the difference between accuracies of the ADAG and other methods, using a one-tailed paired t-test. ++++, +++, ++ and + mean 99.99%, 99%, 95%, and 90% confidence interval for estimating the difference between accuracies of the RADAG and other methods using a one-tailed paired t-test.

To estimate the difference between accuracies, we added up the training set and test set into one set. Then we use a k-fold cross-validation method in which the set is partitioned into k disjoint, equal-sized subsets. In this k-fold cross-validation approach, each example from the set is used exactly once in a test set, and k-1 times in a training set [19]. In our experiment, we use 5-fold crsss-validation for the glass dataset and 10-fold crsss-validation for all others.

For estimating the difference between errors of two learning methods, the mean difference $\bar{Y}$ in errors from all disjoint subsets is returned as an estimate of the difference between the two learning algorithms [19]. The approximate $N\%$ confidence interval for estimating the difference using $\bar{Y}$ is given by

$$(\bar{Y} - t_{N,k-1} S_{\bar{Y}}, \bar{Y} + t_{N,k-1} S_{\bar{Y}}) \tag{5}$$

where $\bar{Y}$ is the sample mean defined as

$$\bar{Y} = \frac{1}{k} \sum_{i=1}^{k} Y_i$$

Y_i is the difference in error between two learning methods from the i^{th} subset, and $S_{\bar{Y}}$ is the estimated standard deviation of the sample mean defined as

$$S_{\bar{Y}} \equiv \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} \left(Y_i - \bar{Y}\right)^2}.$$

As shown in the tables, our methods yield higher accuracy than the Max Wins and the DDAG in almost all of datasets. The results show that the ADAG performs statistically significantly better than the DDAG in satimage, shuttle, vowel, soybean and letter problems. In case of the RBF kernel, the ADAG performs statistically significantly better than the DDAG in shuttle, soybean, letter and isolet problems. The results also show that the RADAG performs statistically significantly better than the other methods in the segment, shuttle, vowel and letter problems in case of the Polynomial kernel. In case of the RBF kernel, the RADAG performs statistically significantly better than the other methods in the glass, shuttle, vowel, soybean and letter problems. These show the effectiveness of the ADAG and the RADAG.

6.3 Computational Time

Table 4 and Table 5 present the comparison of the running time for classifying test data between the ADAG, the RADAG and the Max Wins for Polynomial

Table 2. A comparison of the accuracy of classification using the Polynomial kernel.

Dataset	d	Max Wins	d	DDAG	d	ADAG	d	RADAG
Glass	2	71.078	2	71.069	2	71.135	2	71.063
Satimage	6	89.615	6	89.599^{***}	6	89.622	6	**89.681**
Segment	8	97.351_{++}	8	97.360_{++}	8	97.383_{+}	8	**97.533**
Shuttle	8	99.923_{+}	8	99.919^{**}_{++}	8	99.922_{++}	8	**99.930**
Vowel	3	98.901_{+}	3	98.872^{*}_{+}	3	98.894_{+}	2	**98.990**
Soybean	3	92.470	5	92.202^{*}	5	92.281	3	**92.698**
Letter	3	96.512_{+}	3	95.994^{*****}_{++++}	3	96.379_{+++}	4	**96.674**
Isolet	3	97.488	3	97.484	3	97.485	3	**97.499**
ThaiPrintedCharacter1	2	**99.246**	2	99.239	2	99.242	2	99.234
ThaiPrintedCharacter2	2	**99.677**	2	99.657	2	99.670	2	99.617

Table 3. A comparison of the accuracy of classification using the RBF kernel.

Dataset	c	Max Wins	c	DDAG	c	ADAG	c	RADAG
Glass	0.09	73.238_{+}	0.08	72.850_{+}	0.08	72.759_{+}	0.09	**74.319**
Satimage	3.0	92.148	3.0	92.129	3.0	92.141	3.0	**92.152**
Segment	0.7	**97.656**	0.7	97.652	0.7	97.650	0.7	97.576
Shuttle	3.0	99.928	3.0	99.926^{***}_{++}	3.0	99.927_{+}	3.0	**99.931**
Vowel	0.2	98.980_{+}	0.2	98.965_{+}	0.2	98.975_{+}	0.2	**99.091**
Soybean	0.08	92.533_{+}	0.07	91.739^{**}_{++}	0.08	92.570_{+}	0.07	**93.016**
Letter	3.0	96.512_{+}	3.0	95.994^{*****}_{++++}	3.0	96.379_{++++}	3.0	**96.634**
Isolet	0.01	97.527	0.01	97.517^{*}	0.01	97.523	0.003	**97.589**
ThaiPrintedCharacter1	0.003	**99.387**	0.003	**99.387**	0.003	**99.387**	0.003	**99.387**
ThaiPrintedCharacter2	0.004	**99.663**	0.004	**99.663**	0.004	**99.663**	0.004	**99.663**

and RBF kernels by using a 400 MHz Pentium II processor. There is no running time of the glass dataset because it has too few test examples to measure the time. For the segment problem, there is no provided test data so we use 5-fold crsss-validation. The results show that both the ADAG and the RADAG require low running time in all data sets, especially when the number of classes and/or the number of dimensions are relatively large. For an N-class problem, the Max Wins requires $N(N-1)/2$ classifiers for the classification whereas the ADAG and the RADAG require only $N-1$ classifiers. Hence the larger the number of classes the more running time the Max Wins requires than the ADAG and the RADAG. Moreover, the number of dimensions affects the running time of each classifier. As the result, the larger the number of dimensions the more running time the Max Wins requires than the ADAG and the RADAG. For the RADAG, the number of classes affects the running time for reordering. However, it takes a little time even when there are many classes.

The Max Wins needs $O(N^2)$ number of comparisons for the problem with N classes. The DDAG reduces the number of comparisons down to $O(N)$. By reducing the depth of the path, the ADAG requires $O(N)$ comparisons of binary classifiers with accuracy higher than that of the DDAG. The RADAG needs a little time more than the ADAG for reordering the order of classes. Note that, currently, the minimum-weight perfect matching algorithm, which is used in the reordering algorithm, runs in time bounded by $O(N(M + NlogN))$ [6], where N is the number of nodes (classes) in the graph and $M = N(N-1)/2$ is the number of edges (binary classifiers). The RADAG will reorder the order of classes in every level, except for the last level. The order of classes in the top level is

Table 4. A comparison of the computational time using the Polynomial kernel.

Dataset	Test data	Class	Dimension	d	RADAG			Max Wins (seconds)
					ADAG (seconds)	Reordering (seconds)	Total (seconds)	
Satimage	2,000	6	36	6	1.90	0.50	2.40	9.47
Segment	462	7	18	8	0.11	0.08	0.19	0.41
Shuttle	14,500	7	9	8	1.75	3.38	5.13	5.15
Vowel	462	11	10	2	0.12	0.25	0.37	0.61
Soybean	340	15	35	3	0.30	0.25	0.55	1.86
Letter	4,037	26	16	4	8.48	4.20	12.68	125.58
Isolet	1,559	26	617	3	116.02	1.48	117.50	1,671.98
ThaiPrintedCharacter1	3,264	68	128	3	94.63	13.83	108.46	2,996.64
ThaiPrintedCharacter2	3,264	68	128	3	96.41	13.19	109.60	3,042.98

Table 5. A comparison of the computational time using the RBF kernel.

Dataset	Test data	Class	Dimension	c	RADAG			Max Wins (seconds)
					ADAG (seconds)	Reordering (seconds)	Total (seconds)	
Satimage	2,000	6	36	3.0	11.75	0.51	12.26	37.13
Segment	462	7	18	0.7	0.24	0.10	0.34	0.82
Shuttle	14,500	7	9	3.0	3.36	0.63	3.99	9.27
Vowel	462	11	10	0.2	0.10	0.27	0.37	0.61
Soybean	340	15	35	0.07	0.32	0.45	0.77	2.20
Letter	4,037	26	16	3.0	62.27	3.69	65.96	802.85
Isolet	1,559	26	617	0.01	100.42	1.60	102.02	1,369.11
ThaiPrintedCharacter1	3,264	68	128	0.002	86.25	16.46	102.71	2,772.18
ThaiPrintedCharacter2	3,264	68	128	0.001	55.75	14.37	70.12	1,877.77

reordered only one time and we use the order to evaluate every test example. Hence for classifying each test data, we need $log_2 N - 2$ times of reordering, where each time the number of classes is reduced by half. Therefore, the running time of the RADAG is bounded by $O(c_1 N) + O(c_2 N^3 log_2 N)$, where c_1 is much larger than c_2.

7 Conclusion

We have proposed a new approach, Adaptive Directed Acyclic Graph (ADAG), that alleviates the problem of the DDAG caused by its structure which needs an unnecessarily high number of evaluations for the correct class. Using the reversed triangular structure, the ADAG reduces the number of times the correct class is tested against other classes, and thus reduces the cumulative errors. We proved that the expected accuracy of the ADAG is higher than that of the DDAG.

We also proposed an enhancement version of the ADAG, Reordering Adaptive Directed Acyclic Graph (RADAG), to choose an optimal order of classes in the list in the ADAG method. By the use of minimum-weight perfect matching, the RADAG can reorder the order of classes in polynomial time and only binary classifiers which have small generalization errors will be considered to be used in decision nodes.

Our experimental results are also evidence that the ADAG and the RADAG yield higher accuracy of classification than the Max Wins and the DDAG, especially in such a case that the number of classes is relatively large. Moreover, the

running time used by the ADAG and the RADAG is much less than that used by the Max Wins, especially when the number of classes and/or the number of dimensions are relatively large.

Since the DDAG reduces evaluation time while maintaining accuracy compared to the Max Wins, which is one of the SVMs' fastest methods in multiclass classification, this modification of the DDAG will help improve accuracy even further. In our ongoing work, we are studying how to reduce the number of evaluations and how to enhance the performance in terms of both accuracy and running time.

Appendix

In the following analyses of the DDAG and ADAG, we assume that the probability of the correct class being in any position in the list is a uniform distribution. We also assume that the probability of the correct class being eliminated from the list is p, when it is tested against another class, and that the probability of one of any two classes, except for the correct class, being eliminated from the list is 0.5 when they are tested against each other.

We first illustrate the expected accuracy of the DDAG by the following example.

Example: (Expected accuracy of the DDAG for a 4-class problem). Consider a four-class problem. Figure 7 shows all *probability calculation paths* where the correct class will be correctly classified by the DDAG. There are 8 calculation paths for this problem. The correct class will be correctly classified if it is not eliminated from the list. This means that when it is at the edge (the first or the last element) of the list in each calculation path, all other classes have to be excluded from the list.

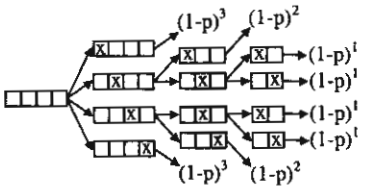


Fig. 7. An example of a four-class problem.

Under a uniform distribution, the probability is 1/4 that the correct class will be at any position of the *initial* list. In the case that the correct class (indicated by 'X' in the figure) is at the edge of the current list, it will be correctly classified if all other classes are eliminated from the list. The probability of this is $(1-p)^{N-1}$, where N is the number of elements in the current list. In the case that the correct class is not at the edge, we have two possible choices, i.e. to remove the first element and to remove the last element from the list. This reduces the number of elements one by one. From the above example, the probability that the correct class is correctly classified is:

$(1/4)(1-p)^3 + (1/4)(1/2)^1(1-p)^2 + (1/4)(1/2)^2(1-p)^1 + (1/4)(1/2)^2(1-p)^1 +$
$(1/4)(1/2)^2(1-p)^1 + (1/4)(1/2)^2(1-p)^1 + (1/4)(1/2)^1(1-p)^2 + (1/4)(1-p)^3$
$= (1/4)[(1-p)/p + (1-p)^3 - (1-p)^4/p]$ □

We now give the theorem for expected accuracy of the DDAG as follows.

Proof of Theorem 1 (Expected accuracy of the DDAG). As shown in the above example, the correct class will be correctly classified if when it is at the edge of the list, all other classes have to be excluded from the list. Consider the cases when we *first* obtain a list with i elements where the correct class is at the left-most position, where $2 \leq i \leq N-1$. The list can be written as $XO \ldots O$, where X and O represent the correct class and a wrong class, respectively. This list is obtained from a list containing $i+1$ elements with one wrong class preceding the correct class as shown by $OXO \ldots O$. Before the list $OXO \ldots O$ is obtained, $N-i-1$ wrong classes must be excluded from an initial list. Thus beginning from all possible different initial lists, the number of possible calculation paths ending with a list of i elements where the correct class is at the left-most position is 2^{N-i-1} (as there are two possible choices for one wrong class; to remove the first element or to remove the last element of the list). Therefore the number of possible calculation paths ending with a list of i elements where the correct class is at the edge (the left-most and right-most positions) is 2^{N-i}. The probability of obtaining the list of i elements where the correct class is at the edge is equal to $(1/N)(0.5)^{N-i}(1-p)^{i-1}$, as $N-i$ wrong classes must be eliminated from the list and after that the correct class is at the edge and $i-1$ wrong classes must be excluded. The total probability that the correct class of this pattern will be correctly classified is thus $2^{(N-i)}(1/N)(0.5)^{N-i}(1-p)^{i-1} = (1/N)(1-p)^{i-1}$.

Next consider the cases when the correct class is at the edge of a list with N elements. For these cases, the total probability that the correct class will be correctly classified is obviously $2(1/N)(1-p)^{N-1}$.

Finally we sum up all above probabilities and we have the expected accuracy as:
$(\sum_{i=2}^{N-1}(1/N)(1-p)^{i-1}) + 2(1/N)(1-p)^{N-1} = (\sum_{j=1}^{N-1}(1/N)(1-p)^{j}) + (1/N)(1-p)^{N-1}$
$= (1/N)[(1-p)(1-(1-p)^{N-1})/p + (1-p)^{N-1}] = (1/N)[(1-p)/p + (1-p)^{N-1} - (1-p)^N/p]$
□

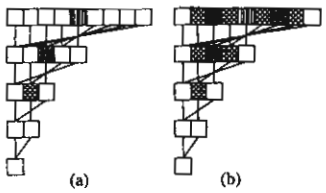


Fig. 8. The positions of classes that can be *bye-getting* elements.

Proof of Theorem 2 (Expected accuracy of the ADAG). Given N classes of examples, the height (the number of adaptive layers and the output layer) of the ADAG is obviously $\lceil log_2 N \rceil$. To be selected as the *winner* (as the output of the ADAG), some elements have to be compared with others for $\lceil log_2 N \rceil$ times, and there are some elements, called *bye-getting* elements that are compared with others for less than $\lceil log_2 N \rceil$ times. As the architecture of the ADAG always puts a bye-getting element (the

middle element) of the current list at the edge of the list of the next layer, any element can get at most one bye. Therefore, a bye-getting element will be compared with others for $\lceil log_2 N \rceil - 1$ times. There will be bye-getting elements only when the number of classes cannot be represented by 2^X, where X is a positive integer. These bye-getting elements will be at the middle of the initial list and of the current list representing each adaptive layer; e.g. the 5^{th}, 3^{rd} and 2^{nd} elements in the initial list, the list of the first adaptive layer, and the list of the second adaptive layer in Figure 8(a), respectively. A bye-getting element at the i^{th} adaptive layer can come from *two* elements in the $(i-1)^{st}$ layer, as shown in Figure 8(b). Therefore, 7 elements of the initial list in the figure can possibly be bye-getting ones. Let $F(N)$ be the number of all possible bye-getting elements of the list with N elements. It is obvious that with $F(2) = 0$, and

$$F(N) = (N \ mod \ 2) + 2 \cdot F(\lceil N/2 \rceil). \tag{6}$$

Next we will prove by induction on X that $F(N) = 2^{\lceil log_2(N) \rceil} - N$,when N is an integer between $2^X + 1$ and 2^{X+1},and X is a positive integer greater than or equal to 1.

First consider the base case of $X = 1$. In this case, N is equal to 3 or 4. It is obvious that $F(3) = 1$ which satisfies $F(3) = 2^{\lceil log_2(3) \rceil} - 3$, and $F(4) = 0$ satisfying $F(4) = 2^{\lceil log_2(4) \rceil} - 4$.

Next we prove general cases of $X \geq 2$ by induction: suppose $F(N)$ is equal to $2^{\lceil log_2(N) \rceil} - N$ when N is an integer between $2^{X-1} + 1$ and 2^X, and we will show that $F(M)$ is also equal to $2^{\lceil log_2(M) \rceil} - M$ when M is an integer between $2^X + 1$ and 2^{X+1}.

In case of $M = 2N$ (an even number) and $2^X + 2 \leq M \leq 2^{X+1}$, it is clear that $F(M) = 2F(N)$ according to Equation 6 (in case that M is an even number, a bye-getting elements of $F(N)$ can possibly come from two bye-getting elements of $F(M)$). Therefore, we will have the following.

$$\begin{aligned} F(M) &= 2F(N) = 2(2^{\lceil log_2 N \rceil} - N) = 2^{\lceil log_2 N \rceil + 1} - 2N = 2^{\lceil log_2 2N \rceil} - 2N \\ &= 2^{\lceil log_2 M \rceil} - M \end{aligned}$$

This proves the case of M is an even number.

Next in case of $M = 2N - 1$ (an odd number) and $2^X + 1 \leq M \leq 2^{X+1} - 1$, it is also clear that $F(M) = 2F(N) + 1$ according to Equation 1 (in case that M is an odd number, a bye-getting elements of $F(N)$ can possibly come from two bye-getting elements of $F(M)$, and there is one more bye-getting (the middle) elements of $F(M)$ that gets a bye of this round). Therefore, we will have the following.

$$\begin{aligned} F(M) &= 2F(N) + 1 = 2(2^{\lceil log_2 N \rceil} - N) + 1 = 2^{\lceil log_2 N \rceil + 1} - 2N + 1 \\ &= 2^{\lceil log_2 2N \rceil} - (2N - 1) \end{aligned}$$

As $2^X + 1 \leq 2N - 1 < 2N \leq 2^{X+1}$, we have $log_2(2^X + 1) \leq log_2(2N - 1) < log_2(2N) \leq log_2(2^{X+1})$, which means $\lceil log_2(2N - 1) \rceil = \lceil log_2(2N) \rceil = X + 1$. The above formula then becomes as follows.

$$F(M) = 2^{\lceil log_2(2N-1) \rceil} - (2N - 1) = 2^{\lceil log_2 M \rceil} - M$$

This proves the case of M is an odd number. The above then proves that $F(N)$ is equal to $2^{\lceil log_2(N) \rceil} - N$ when N is an integer between $2^X + 1$ and 2^{X+1}, where $X \geq 1$.

Having the value of $F(N)$ as above, we then can calculate the expected accuracy of the ADAG. As under a uniform distribution, the probability that the correct class is at any position of the initial list is $1/N$. Therefore, the expected accuracy is calculated by weighting the bye-getting correct elements with $F(N)/N$, and the non-bye-getting correct elements with $(N - F(N))/N$. Finally, we have the expected accuracy of the ADAG as follows.

$$\begin{aligned} &(N - F(N))/N \cdot (1-p)^{\lceil log_2 N \rceil} + F(N)/N \cdot (1-p)^{\lceil log_2 N \rceil - 1} \\ &= ((2N - 2^{\lceil log_2 N \rceil})/N) \cdot (1-p)^{\lceil log_2 N \rceil} + ((2^{\lceil log_2 N \rceil} - N)/N) \cdot (1-p)^{\lceil log_2 N \rceil - 1}. \end{aligned}$$

This proves the theorem. □

References

1. Abe, S., and Inoue, T. (2002) *Fuzzy support vector machines for multiclass problems*, The European Symposium on Artificial Neural Networks, 113-118.
2. Allwein, E., Schapire, R., and Singer, Y. (2000) *Reducing multiclass to binary: A unifying approach for margin classifiers*, International Conference on Machine Learning.
3. Bartlett, P. L., and Shawe-Taylor, J. (1999) *Generalization performance of support vector machines and other pattern classifiers*, Advances in Kernel Methods - Support Vector Learning, MIT Press, Cambridge, USA, 43-54.
4. Blake, C., Keogh, E., and Merz, C. (1998) *UCI repository of machine learning databases*, Department of Information and Computer Science, University of California, Irvine.
5. Burges, C. (1998) *A tutorial on support vector machines for pattern recognition*, Data Mining and Knowledge Discovery, 2(2):121-167.
6. Cook, W., and Rohe, A. (1997) *Computing minimum-weight perfect matchings*, Technical Report 97863, Forschungsinstitut für Diskrete Mathematik, Universität Bonn.
7. Crammer, K., and Singer, Y. (2001) *On the algorithmic implementation of multiclass kernel-based vector machines*, Journal of Machine Learning Research, 2:265-292.
8. Cristianini N., and Shawe-Taylor, J. (2000) *An introduction to support vector machines and other kernel-based learining methods*, Cambridge University Press.
9. Dietterich, T. G., and Bakiri, G. (1995) *Solving multiclass learning problems via error-correcting output codes*, Journal of Artificial Intelligence Research, 2:263-286.
10. Dong, X., Zhaohui, W., and Yunhe, P. (2001) *A new multi-class support vector machines*, IEEE International Conference on System, Man, and Cybernatics, 3:1673-1676.
11. Friedman, J. H. (1996) *Another approach to polychotomous classification*, Technical report, Stanford University, Department of Statistics.
12. Hsu, C., and Lin, C. (2002) *A comparison of methods for multiclass support vector machines*, IEEE Transactions on Neural Networks, 13:415-425.
13. Joachims. T. (1998) *Making large-scale SVM learning practical*, Advances in Kernel Methods - Support Vector Learning, MIT Press.
14. Joachims, T. (1999) SVM^{light}, http://ais.gmd.de/~thorsten/svm_light.
15. Kindermann, J., Leopold, E., and Paass, G. (2000) *Multi-class classification with error correcting codes*, Treffen der GI-Fachgruppe 1.1.3, Maschinelles Lernen, GMD Re. 114.
16. Knerr, S., Personnaz, L., and Dreyfus, G. (1990) *Single-layer learning revisited: A stepwise procedure for building and training a neural network*, In Fogelman-Soulie and Herault, editors, Neurocomputing: Algorithms, Architectures and Applications, NATO ASI Series. Springer.
17. Li, K., Huang, H., and Tian, S. (2002) *A novel multi-class SVM classifier based on DDAG*, The 1st International Conference on Machine Learning and Cybernatics, 1203-1207.
18. Mayoraz, E., and Alpaydm, E. (1998) *Support vector machines for multi-class classification*, IDIAP-RR 6, IDIAP.
19. Mitchell, T. (1997) *Machine Learning*, McGraw Hill.
20. Pichitdej, S. (2001) *Thai Printed Character Recognition Using a Neural Network Ensemble*, Degree of Master Computer Engineering in Department of Computer Engineering Faculty of Engineering Chulalongkorn University.

21. Platt, J., Cristianini, N., and Shawe-Taylor, J. (2000) *Large margin DAGs for multiclass classification*, Advance in Neural Information Processing System, 12, MIT Press.
22. Roth, V., and Tsuda, K. (2001) *Pairwise coupling for machine recognition of hand-printed japanese characters*, IEEE International Conference on Computer Society, 1:1120-1125.
23. Schölkopf, B. (1997) *Support vector learning*, Ph.D. Thesis, R.Oldenbourg Verlag Publications, Munich, Germany.
24. Schwenker, F. (2000) *Hierarchical support vector machines for multi-class pattern recognition*, The 4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies, 561-565.
25. Sekhar, C., Takeda, K., and Itakura, F. (2002) *Close-class-set discrimination method for large-class-set pattern recognition using support vector machines*, IEEE International Joint Conference on Neural Networks, 577-582.
26. Takahashi, F., and Abe, S. (2002) *Decision-tree-based multiclass support vector machines*, The 9th International Conference on Neural Information Processing, 3:1418-1422.
27. Thubthong, N. and Kijsîrikul, B. (2001) *Support vector machines for Thai phoneme recognition*, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 9(6):803-813.
28. Vapnik, V. (1998) *Statistical Learning Theory*, New York, Wiley.
29. Vapnik, V. (1999) *An overview of statistical learning theory*, IEEE Transactions on Neural Networks, 10:988-999.
30. Weston, J., and Watkins, C. (1998) *Multi-class support vector machines*, Technical Report CSD-TR-98-04, Department of Computer science, Royal Holloway, University of London.

Multiclass Classification of Support Vector Machines by Reordering Adaptive Directed Acyclic Graph*

Thimaporn Phetkaew[1], Boonserm Kijsirikul[2] and Wanchai Rivepiboon[3]

Department of Computer Engineering, Chulalongkorn University, Phayathai, Patumwan, Bangkok, 10330, Thailand

Thimaporn.P@student.chula.ac.th[1] and Boonserm.K[2],Wanchai.R[3]@chula.ac.th

Received 24 November 2003

Abstract The problem of extending binary support vector machines (SVMs) for multiclass classification is still an ongoing research issue. Ussivakul and Kijsirikul proposed the Adaptive Directed Acyclic Graph (ADAG) approach that provides accuracy comparable to that of Max Wins, which is probably the currently most accurate method for multiclass SVMs, and requires low computation. However, different sequences of binary classifiers in nodes in the ADAG may provide different accuracy. In this paper we present a new method for multiclass classification, Reordering Adaptive Directed Acyclic Graph (RADAG), which is the modification of the original ADAG method. We propose an algorithm to choose an optimal sequence of binary classifiers in nodes in the ADAG by considering the generalization error bounds of all classifiers. We apply minimum-weight perfect matching with the reordering algorithm in order to select binary classifiers which have small generalization errors to be used in data classification and in order to find the best sequence of binary classifiers in nodes in polynomial time. We then compare the performance of our method with previous methods including the ADAG and the Max Wins algorithm. Experiments denote that our method gives higher accuracy. Moreover it runs faster than Max Wins, especially when the number of classes and/or the number of dimensions are relatively large.

Keywords Support Vector Machines, Multiclass Classification, ADAG, RADAG

* This work was supported by The Thailand Research Fund.

1 Introduction

Support vector machines (SVMs) [8] were primarily designed for two-class classification problems with their outstanding performance in real world applications. However, extending SVMs for multiclass classification is still an ongoing research issue. Previous methods for solving the multiclass problem of SVMs are typically to consider the problem as the combination of two-class decision functions, e.g. one-against-one and one-against-the-rest [5]. The one-against-the-rest approach works by constructing a set of k binary classifiers for a k-class problem. The i^{th} classifier is trained with all of the examples in the i^{th} class with positive labels, and all other examples with negative labels. The final output is the class that corresponds to the classifier with the highest output value. Friedman [5] suggested the Max Wins algorithm in which each one-against-one classifier casts one vote for its preferred class, and the final result is the class with the most votes. The Max Wins algorithm offers faster training time compared to the one-against-the-rest method. The Decision Directed Acyclic Graph (DDAG) method proposed by Platt et al. reduces training and evaluation time, while maintaining accuracy compared to the Max Wins [6]. The comparison experiments by several methods on large problems in [5] show that the Max Wins algorithm and the DDAG may be more suitable for practical use. Ussivakul and Kijsirikul [7] proposed the Adaptive Directed Acyclic Graph (ADAG) method which is the modification of the DDAG. This method reduces the dependency of the sequence of binary classifiers in nodes in the structure as well as lowers the number of tests required to evaluate for the correct class. Their approach yields higher accuracy and reliability of classification, especially in such a case that the number of classes is relatively large.

In this paper we reveal that the ADAG still is dependent on the sequence of its nodes, although it is less dependent on the order of binary classes in the sequence than the DDAG; there are still differences in accuracy between different sequences. This led to the reliability of the algorithm. Here we propose a novel method that improves reliability by choosing an optimal sequence, which has less chance to predict the wrong class, and dynamically reordering the sequence during classification process according to each test data. We also reveal that the problem of selecting the appropriate sequence can be solved by minimum-weight perfect matching.

This paper is organized as follows. In the next section, we review SVMs and the formulation to solve multiclass problems, i.e., the DDAG and the ADAG. In Section 3, we introduce the modification of the ADAG to improve the performance by using the reordering algorithm with minimum-weigh perfect matching. The numerical experiments are illustrated in Section 4. Finally, the conclusions are given in Section 5.

2 SVM classification

This section describes the basic idea of SVMs [7] and two previous works on multiclass SVMs which are related to our proposed method, i.e., the DDAG [5,6] and the ADAG [7].

2.1 Support vector machines

The main idea of support vector machine classification is to construct a hyperplane to separate the two classes.

2.1.1 Linear support vector machines

Suppose we have a data set D of l samples in an n-dimensional space belonging to two different classes (+1 and −1):

$$D = \{(\mathbf{x}_k, y_k) \mid k \in \{1,..,l\}, \mathbf{x}_k \in \Re^n, y_k \in \{+1,-1\}\} \quad \text{.....(1)}$$

The hyperplane in the n dimensional space is determined by the pair $(\mathbf{w},b)$ where $\mathbf{w}$ is an n-dimensional vector orthogonal to the hyperplane and b is the offset constant. The hyperplane $(\mathbf{w}\cdot\mathbf{x}) + b$ separates the data if and only if

$$(\mathbf{w}\cdot\mathbf{x}_i)+b>0 \quad \text{if} \quad y_i=+1$$
$$(\mathbf{w}\cdot\mathbf{x}_i)+b<0 \quad \text{if} \quad y_i=-1. \quad \text{.....(2)}$$

If we additionally require that $\mathbf{w}$ and b be such that the point closest to the hyperplane has a distance of $1/|\mathbf{w}|$, then we have

$$(\mathbf{w}\cdot\mathbf{x}_i)+b\geq 1 \quad \text{if} \quad y_i=+1$$
$$(\mathbf{w}\cdot\mathbf{x}_i)+b\leq -1 \quad \text{if} \quad y_i=-1 \quad \text{.....(3)}$$

which is equivalent to

$$y_i[(\mathbf{w}\cdot\mathbf{x}_i)+b]\geq 1 \quad \forall i. \quad \text{.....(4)}$$

To find the optimal separating hyperplane, we have to find the hyperplane that maximizes the minimum distance between the hyperplane and any sample of training data. The distance between two closest samples from different classes is

$$d(\mathbf{w},b)=\min_{\{\mathbf{x}_i|y_i=1\}}\frac{(\mathbf{w}\cdot\mathbf{x}_i)+b}{|\mathbf{w}|}-\max_{\{\mathbf{x}_i|y_i=-1\}}\frac{(\mathbf{w}\cdot\mathbf{x}_i)+b}{|\mathbf{w}|}.$$
.....(5)

From (3), we can see that the appropriate minimum and maximum values are ±1. Therefore, we need to maximize

$$d(\mathbf{w},b)=\frac{1}{|\mathbf{w}|}-\frac{-1}{|\mathbf{w}|}=\frac{2}{|\mathbf{w}|}. \quad \text{.....(6)}$$

Thus, the problem is equivalent to:

- minimize $|\mathbf{w}|^2/2$
- subject to the constraints
  (1) $y_i[(\mathbf{w}\cdot\mathbf{x}_i)+b]\geq 1 \quad \forall i.$

For non-separable case, the training data cannot be separated by a hyperplane without error. The previous constraints then must be modified. A penalty term consisting of the sum of deviations ξ_i from the boundary is added to the minimization problem. Now, the problem is to

- minimize $\frac{|\mathbf{w}|^2}{2}+C\sum_{i=1}^{l}\xi_i$
- subject to the constraints
  (1) $y_i[(\mathbf{w}\cdot\mathbf{x}_i)+b]\geq 1-\xi_i$,
  (2) $\xi_i\geq 0 \quad \forall i.$

The penalty term for misclassifying training samples is weighted by a constant C. Selecting a large value of C puts a high price on deviations and increases computation by effecting a more exhaustive search for ways to minimize the number of misclassified samples.

By forming the Lagrangian and solving the dual problem, this problem can be translated into:

- minimize
  $$L(\mathbf{w},b,\alpha)=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j)$$
  .....(7)
- subject to the constraints:
  (1) $0\leq\alpha_i\leq C$, $\forall i$
  (2) $\sum_{i=1}^{l}\alpha_i y_i=0$

where α_i are called Lagrange multipliers. There is one Lagrange multiplier for each training sample. In the solution, those samples for which $\alpha_i > 0$ are called *support vectors*, and are ones such that the equality in (4) holds. All other training samples having $\alpha_i = 0$ could be removed from the training set without affecting the final hyperplane.

Let α^0, an l-dimensional vector denote the minimum of $L(\mathbf{w},b,\alpha)$. If $\alpha_i^0 > 0$ then $\mathbf{x}_i$ is a *support vector*. The optimal separating hyperplane $(\mathbf{w}^0, b^0)$ can be written in terms of α^0 and the training data, specifically in terms of the support vectors:

$$\mathbf{w}^0=\sum_{i=1}^{l}\alpha_i^0 y_i\mathbf{x}_i=\sum_{\text{support vectors}}\alpha_i^0 y_i\mathbf{x}_i. \quad \text{.....(8)}$$
$$b^0=1-\mathbf{w}^0\cdot\mathbf{x}_i \quad \text{for } \mathbf{x}_i \text{ with } y_i=1 \text{ and } 0<\alpha_i<C.$$
.....(9)

The optimal separating hyperplane classifies points according to the sign of $f(\mathbf{x})$,

$$f(\mathbf{x}) = \text{sign}\left(\mathbf{w}^0 \cdot \mathbf{x} + b^0\right) = \text{sign}\left(\sum_{\text{suport vectors}} \alpha_i^0 y_i (\mathbf{x}_i \cdot \mathbf{x}) + b^0\right) \quad \ldots.(10)$$

Support vector $\mathbf{x}_i$ with $\alpha_i^0 = C$ may or may not be misclassified. All other $\mathbf{x}_i$'s are correctly classified.

2.1.2 Non-linear support vector machines

The above algorithm is limited to linear separating hyperplanes. SVMs get around this problem by mapping the sample points into a higher dimensional space using a non-linear mapping chosen in advance. This is, we choose a map $\Phi : \Re^n \mapsto \mathrm{H}$ where the dimensionality of H is greater than n. We then seek a separating hyperplane in the higher dimensional space; this is equivalent to a non-linear separating surface in $\Re^n$.

The data only ever appears in our training problem (7) in the form of dot products, so in the higher dimensional space we are only dealing with the data in the form $\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$. If the dimensionality of H is very large, then this could be difficult, or very computationally expensive. However, if we have a kernel function such that $k(\mathbf{x}_i,\mathbf{x}_j) = \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$, then we can use this in place of $\mathbf{x}_i\cdot\mathbf{x}_j$ everywhere in the optimization problem, and never need to know explicitly what Φ is. Some widely used kernels are:

Polynomial degree d: $k(\mathbf{x},\mathbf{y}) = \left|\mathbf{x}\cdot\mathbf{y} + 1\right|^d$ ...(11)

Radial basis function: $k(\mathbf{x},\mathbf{y}) = e^{-\left|\mathbf{x}-\mathbf{y}\right|^2/c}$ ...(12)

2.2 DDAG

Platt et al. [6] presented a learning architecture, the Decision Directed Acyclic Graph (DDAG), which is used to combine many two-class classifiers into a multiclass classifier. For a k-class problem, its training phase is the same as the one-against-one method by solving $k(k\text{-}1)/2$ binary SVMs, one for each pair of classes. However, in the testing phase, it uses a rooted binary directed acyclic graph which has $k(k\text{-}1)/2$ internal nodes and k leaves (see Figure 1). Each node is a binary SVM of the i^{th} and j^{th} classes. Given a test sample x, starting at the root node, the binary decision function is evaluated. Then it moves to either left or right depending on the output value. Therefore, we go through a path before reaching a leaf node which indicates the predicted class.

There are some issues on the DDAG as pointed out by [7]. First, it gives outputs whose probabilities are not uniformly distributed, and thus its output depends on the sequence of binary classifiers in nodes, affecting reliability of the algorithm. In addition, the correct class placed in a node near the root node is clearly at disadvantage by comparison with the correct class near leaf nodes since it is exposed to higher risk of being incorrectly rejected. Second, the number of node evaluations for the correct class is unnecessary high. This results in higher cumulative error and lower the accuracy. The depth of the DDAG is k-1 and this means that the number of times the correct class has to be tested against other classes, on average, scales linearly with k.

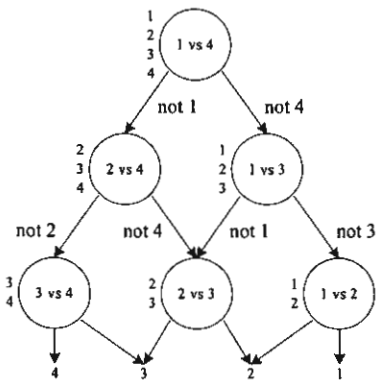


Figure 1: The DDAG finding the best class out of four classes

2.3 ADAG

Ussivakul and Kijsirikul [7] proposed an approach to alleviate the problem of the DDAG structure described above. An Adaptive DAG (ADAG) is a DAG with a reversed triangular structure. For a k-class problem, its training phase is the same as the DDAG method by solving $k(k\text{-}1)/2$ binary SVMs, one for each pair of classes. However, in the testing phase, the nodes are arranged in a reversed triangle with $k/2$ nodes (rounded up) at the top, $k/2^2$ nodes in the second layer and so on until the lowest layer of a final node. It has k-1 internal nodes, each of which is labeled with an element of Boolean function (see Figure 2). Given a test example x, starting at the top level, the binary decision function is evaluated. The node is then exited via the outgoing edge with a message of the preferred class. In each round, the number of candidate classes is reduced by half. Based on the preferred classes from its parent nodes, the binary function of the next-level node is chosen. The reduction process continues until reaching the final node at the lowest level. The value of the decision function is the value associated with the message from the final leaf node. Like the DDAG, the ADAG requires only k-1 decision nodes to be evaluated in order to derive an answer. Note that the correct class is evaluated against other classes for $\log_2 k$ times or less, considerably lower than the number of evaluations required by the DDAG, which scales linearly with k.

Using the reversed triangular structure, the ADAG reduces the number of times the correct class is tested against other classes, and thus reduces the cumulative errors. However, there are still differences in accuracy between different sequences of nodes. Next we will describe our method that improves the ADAG by finding a best sequence of nodes.

3 The proposed method

In this section, we introduce the modification of the ADAG to improve the performance of the original ADAG. This approach determines a best sequence of nodes in the ADAG by dynamically reordering the sequence during classification process according to each test data.

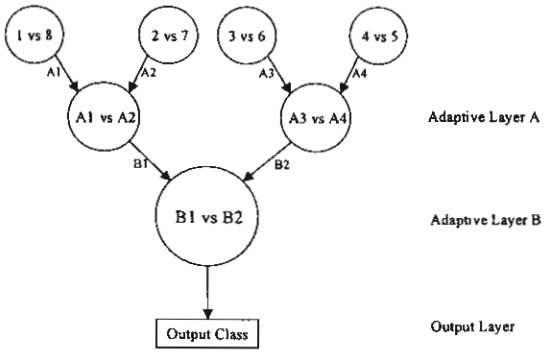


Figure 2: The structure of an Adaptive DAG for an 8-class problem

3.1 Generalization Performance of Support Vector Machines

The ability of a hypothesis to correctly classify data not in the training set is known as its generalization. Generalization analysis of pattern classifiers is concerned with determining the factors that affect the accuracy of a pattern classifier [1]. Generalization performance of Support Vector Machines can be approximated by bounding on the generalization error.

Define the class F of real-valued functions on the ball of radius R in $\Re^n$ as $F = \{\mathbf{x} \mapsto \mathbf{w} \cdot \mathbf{x} : \|\mathbf{w}\| \le 1, \|\mathbf{x}\| \le R\}$. There is a constant c such that, for all probability distributions, with probability at least 1-δ over m independently generated examples $\mathbf{z}$, if a classifier $h = \mathrm{sgn}(f) \in \mathrm{sgn}(F)$ has margin at least γ on all the examples in $\mathbf{z}$, then the error of h is no more than

$$\frac{c}{m}\left(\frac{R^2}{\gamma^2}\log^2 m + \log\left(\frac{1}{\delta}\right)\right) \quad \text{.....(13)}$$

Furthermore, with probability at least 1-δ, every classifier $h \in \mathrm{sgn}(F)$ has error no more than

$$\frac{k}{m}+\sqrt{\frac{c}{m}\left(\frac{R^2}{\gamma^2}\log^2 m+\log\left(\frac{1}{\delta}\right)\right)} \quad \text{.....(14)}$$

where k is the number of labeled examples in $\mathbf{z}$ with margin less than γ.

Below we show an example of the generalization error. The experiment is based on the English letter image recognition dataset from [2], which has 26 classes. Hence there are 325 classifiers. In this case, we construct classifiers by using the Polynomial kernel of degree 3. In Figure 3, the generalization errors of all classifiers expressed by Equation (14) are depicted. The generalization errors of all classifiers are varying.

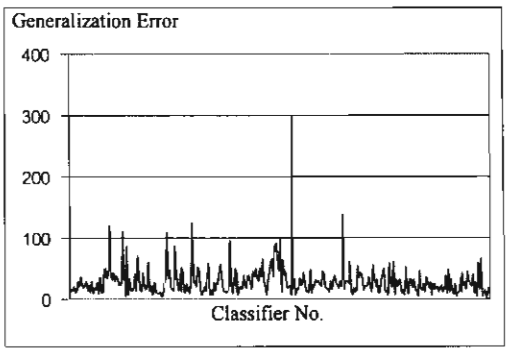


Figure 3: The generalization errors of 325 classifiers

3.2 Reordering ADAG

We propose a method, called Reordering Adaptive Directed Acyclic Graph (RADAG), to improve the accuracy of the original ADAG. For a k-class problem, the RADAG's training phase is the same as the ADAG method by solving $k(k\text{-}1)/2$ binary SVMs. However, the testing phase is organized as follows. The differences are the initialization of the binary classifiers in the top level and the order of sequence in each level (see Figure 4). In the first step, we use a reordering algorithm with minimum-weight perfect matching described in the next subsection to choose the optimal sequence to be the initial sequence. We use the sequence to evaluate every test example. In the second step, as in the ADAG, test points of the RADAG are evaluated against the decision nodes. In the third step, unlike the ADAG, the RADAG will reorder the sequence before processing in the next level by using the reordering algorithm with minimum-weight perfect matching. This sequence differs for each test example, and it depends on the results of nodes from the previous level. The second and the third steps are repeated until there is only one class remains.

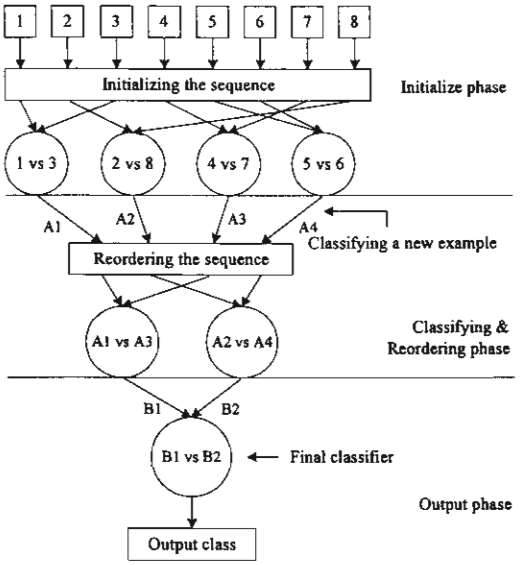


Figure 4: Classifying process of the RADAG

3.3 Reordering algorithm

For the reason described above, we consider the generalization errors in order to choose the optimal sequence from all possible $\frac{k!}{2^{\lfloor \frac{k}{2} \rfloor}}$ sequences with less chance to predict the wrong class. Among classifiers, $k/2$ classifiers which have small generalization errors will be considered to be used in data classification.

Let $G = (V, E)$ be a graph with node set V and edge set E. Each node in G denotes one class and each edge denotes one binary classifier

which has a generalization error expressed by Equation (14) (see Figure 5(a)). The output of the reordering algorithm for graph G is a subset of edges with the minimum sum of generalization errors of all edges and each node in G is met by exactly one edge in the subset (see Figure 5(b)). Given a real weight c_e being generalization error for each edge e of G, the problem of reordering algorithm can be solved by the minimum weight perfect matching that finds a perfect matching M of minimum weight $\Sigma(c_e : e \in M)$.

For $U \subseteq V$, let $E(U) = \{(i,j):(i,j)\in E,\ i\in U,\ j\in U\}$. $E(U)$ is the set of chosen classifiers. Let $\delta(i)$ denote the set of edges incident to node i; the set of classifiers with one class being i. The perfect matchings on a graph $G = (V, E)$ with $|V|$ even is given by

(a) $x \in R_+^m$

(b) $\sum_{e\in\delta(v)} x_e = 1$ for $v \in V$

(c) $\sum_{e\in E(U)} x_e \le \left\lfloor \frac{|U|}{2} \right\rfloor$ for all odd sets $U \subseteq V$ with $|U|\ge 3$

or by (a),(b) and

(d) $\sum_{e\in\delta(U)} x_e \ge 1$ for all odd sets $U \subseteq V$ with $|U|\ge 3$

.....(15)

where $|E| = m$, the number of classifiers, and $x_e = 1$ means that classifier e is chosen to be used in the sequence. Therefore, the reordering problem is to solve the following linear program:

$$\min \sum_{e\in E} c_e x_e \quad \text{.....(16)}$$

where x satisfies "(a),(b) and (c)" or "(a),(b) and (d)".

Currently, the minimum-weight perfect matching algorithm runs in time bounded by $O(n(m + n \log n))$ [3], where n is the number of nodes (classes) in the graph and m is the number of edges (binary classifiers). Hence the reordering algorithm can reorder the sequence in that polynomial time.

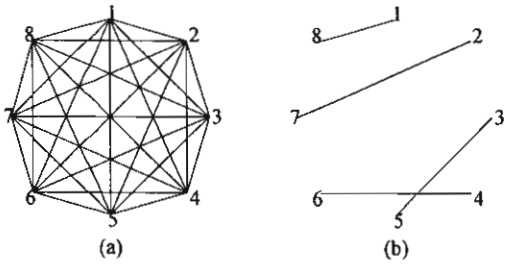


Figure 5: (a) A graph for an 8-class problem. (b) An example of the output of the reordering algorithm.

4 Numerical experiments

In this section, we present experimental results on several datasets from the UCI Repository of machine learning databases [2] including glass, satimage, segment, shuttle, vowel, soybean, letter and isolet (see Table 1). These datasets are different in the number of classes, the number of dimensions, and sizes. For the glass and segment problems, there is no provided test data so we used 5-fold cross validation. For the soybean problem, we discarded the last four classes because of missing values.

Table 1: Description of the datasets used in the experiments

Dataset	#training data	#test data	#class	#dimension
Glass	214	5-fold	6	9
Satimage	4,435	2,000	6	36
Segment	2,310	5-fold	7	18
Shuttle	43,500	14,500	7	9
Vowel	528	462	11	10
Soybean	290	340	15	35
Letter	15,963	4,037	26	16
Isolet	6,238	1,559	26	617

In these experiments we scaled both training data and test data to be in [-1,1] and employed Polynomial and RBF kernels. In the experiments, we compare three algorithms, i.e., the DDAG, the original ADAG, the RADAG, and the Max Wins algorithm. For the ADAG, we examined all

possible sequences for datasets having not more than 7 classes, whereas we randomly selected 50,000 sequences for datasets having more than 7 classes. Table 2 and 3 present the results of the comparison of these methods for Polynomial and RBF kernels, respectively. We present the optimal parameters (d and c in Equations (11) and (12)) of the kernels and the corresponding accuracies. The best accuracy among three methods is illustrated in bold-face. ***, ** and * in the tables means 99%, 95%, and 90% confidence interval for estimating the difference between accuracies of three algorithms and the RADAG using a one-tailed paired t-test.

The results show that our method yields highest accuracy in almost all of datasets. The results also show that our method performs statistically significantly better than the other methods in the glass problem in case of the RBF kernel and significantly better than the DDAG in the segment and letter problems in case of the Polynomial kernel. Another advantage of our method compared to the DDAG and the original ADAG is that our method always provides one best accuracy for each dataset using the reordering algorithm, whereas, depending on the sequence of classes, the DDAG and the original ADAG may give low accuracies. This shows the effectiveness of the RADAG.

Table 4 and 5 present the comparison of the computational time between the RADAG and the Max Wins for Polynomial and RBF kernels by using a 400 MHz Pentium II processor. There is no computational time of the glass dataset because it has too few test examples to measure the time. The results show that our method requires low computational time in all datasets, especially when the number of classes and/or the number of dimensions are relatively large. For a k class problem, the Max Wins requires $k(k\text{-}1)/2$ classifiers for the classification whereas the RADAG requires only k-1 classifiers. Hence the larger the number of classes the more running time the Max Wins requires than the RADAG. Moreover, the number of dimensions affects the running time of each classifier. For the RADAG, the number of classes affects the running time for reordering. However, it takes little time even when there are many classes.

5 Conclusion

In this paper, we have presented a new approach for multiclass SVMs, called Reordering Adaptive Directed Acyclic Graph (RADAG), which is the modification of the original ADAG. Our approach eliminates the dependency of the sequence of binary classifiers in nodes in the original ADAG by selecting an appropriate sequence from all possible sequences which consists of classifiers with small generalization error. By the use of minimum-weight perfect matching, the RADAG can reorder the sequence in polynomial time. The experimental results show that our new approach yields higher accuracy than the original ADAG and even Max Wins which is probably the currently most accurate method for multiclass SVMs. Moreover the running time used by the RADAG is less than Max Wins, especially when the number of classes and/or the number of dimensions are relatively large. Our future work is to test the method on datasets with a very large number of classes and dimensions.

Table 2: A comparison of the accuracy of classification using the Polynomial kernel

Dataset	d	DDAG	d	ADAG	d	Max Wins	d	RADAG
Glass	2	71.069	2	**71.135**	2	71.078	2	71.063
Satimage	6	88.408***	6	88.430	6	88.453	6	**88.900**
Segment	6	96.538	8	97.408	8	97.379	8	**97.489**
Shuttle	8	**99.924**	8	**99.924**	8	**99.924**	8	**99.924**
Vowel	3	64.237	3	64.293	3	64.329	2	**64.502**
Soybean	5	90.400	5	90.446	3	90.471	3	**91.176**
Letter	3	95.508*	3	95.984	3	**96.125**	4	96.111
Isolet	3	97.032	3	97.030	3	97.040	3	**97.049**

Table 3: A comparison of the accuracy of classification using the RBF kernel

Dataset	c	DDAG	c	ADAG	c	Max Wins	c	RADAG
Glass	0.08	72.850**	0.08	72.759**	0.09	73.238**	0.09	**74.319**
Satimage	3.0	91.971	3.0	91.968	3.0	**91.984**	3.0	91.950
Segment	0.7	97.276	0.7	97.282	0.7	**97.298**	0.7	97.273
Shuttle	3.0	**99.897**	3.0	**99.897**	3.0	**99.897**	3.0	**99.897**
Vowel	0.2	65.425	0.2	65.589	0.2	65.340	0.2	**67.100**
Soybean	0.07	90.353	0.08	90.412	0.08	90.468	0.07	**90.882**
Letter	3.0	97.901	3.0	97.909	3.0	97.918	3.0	**97.969**
Isolet	0.01	96.939	0.01	96.932	0.01	96.916	0.01	**96.985**

Table 4: A comparison of the computational time using the Polynomial kernel

Dataset	d	RADAG			Max Wins
		Classifying (seconds)	Reordering (seconds)	Total (seconds)	Classifying (seconds)
Satimage	6	1.90	0.50	2.40	9.47
Segment	8	0.11	0.08	0.19	0.41
Shuttle	8	1.75	3.38	5.13	5.15
Vowel	2	0.12	0.25	0.37	0.61
Soybean	3	0.30	0.25	0.55	1.86
Letter	4	8.48	4.20	12.68	125.58
Isolet	3	116.02	1.48	117.50	1,671.98

Table 5: A comparison of the computational time using the RBF kernel

Dataset	c	RADAG			Max Wins
		Classifying (seconds)	Reordering (seconds)	Total (seconds)	Classifying (seconds)
Satimage	3.0	11.75	0.51	12.26	37.13
Segment	0.7	0.24	0.10	0.34	0.82
Shuttle	3.0	3.36	0.63	3.99	9.27
Vowel	0.2	0.10	0.27	0.37	0.61
Soybean	0.07	0.32	0.45	0.77	2.20
Letter	3.0	62.27	3.69	65.96	802.85
Isolet	0.01	100.42	1.60	102.02	1,369.11

References

[1] P. L. Bartlett, J. Shawe-Taylor, "Generalization performance of support vector machines and other pattern classifiers", in Advances in Kernel Methods - Support Vector Learning, B. Schoelkopf, C. J. C. Burges, A. J. Smola (eds), pp. 43-54, MIT Press, Cambridge, USA, 1999.

[2] C. Blake, E. Keogh and C. Merz, UCI repository of machine learning databases, Department of Information and Computer Science, University of California, Irvine, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html

[3] W. Cook, and A. Rohe, "Computing minimum-weight perfect matchings", Technical Report 97863, Forschungsinstitut für Diskrete Mathematik, Universität Bonn, 1997.

[4] J. H. Friedman, "Another approach to Polychotomous classification", Technical report, Department of Statistics, Stanford University, 1996.

[5] C.-W. Hsu, and C.-J. Lin, "A comparison of methods for multiclass Support Vector Machines", IEEE Trans.on Neural Networks, Vol. 13, pp. 415-425, March, 2002.

[6] J. Platt, N. Cristianini, and J. Shawe-Taylor, "Large margin DAGs for multiclass classification", in Advances in Neural Information Processing Systems, MIT Press, Vol. 12, pp. 547-553, 2000.

[7] N. Ussivakul, and B. Kijsirikul, "Multiclass support vector machines using adaptive directed acyclic graph," in IEEE/INNS' Int. Joint Conf. On Neural Networks (IJCNN-2002), 2002.

[8] V. Vapnik, Statistical Learning Theory, New York, Wiley, 1998.

Learning Multiclass Support Vector Machines by Reordering Adaptive Directed Acyclic Graph

Thimaporn Phetkaew[1], Wanchai Rivepiboon[2] and Boonserm Kijsirikul[3]
Department of Computer Engineering
Chulalongkorn University
Phayathai, Patumwan, Bangkok, 10330, Thailand
E-mail: Thimaporn.P@student.chula.ac.th[1] and Wanchai.R[2], Boonserm.K[3]@chula.ac.th

Abstract

The problem of extending binary support vector machines (SVMs) for multiclass classification is still an ongoing research issue. Ussivakul and Kijsirikul proposed the Adaptive Directed Acyclic Graph (ADAG) approach that provides accuracy comparable to that of the standard algorithm—Max Wins and requires low computation. However, different sequences of nodes in the ADAG may provide different accuracy. In this paper we present a new method for multiclass classification, Reordering Adaptive Directed Acyclic Graph (RADAG), which is the modification of the original ADAG method. We propose an algorithm to choose an optimal sequence of binary classifiers in nodes in the ADAG by considering the generalization error bounds of all classifiers. We apply minimum-weight perfect matching with the reordering algorithm in order to select the best sequence of nodes in polynomial time. We then compare the performance of our method with previous methods including the ADAG and the Max Wins algorithm. Experiments denote that our method gives higher accuracy. Moreover it runs faster than Max Wins, especially when the number of classes and/or the number of dimensions are relatively large.

Key words: Multiclass Support Vector Machines, ADAG

Introduction

Support vector machines (SVMs) were primarily designed for two-class classification problems with its outstanding performance in real world applications. However, extending SVMs for multiclass classification is still an ongoing research issue. Previous methods for solving the multiclass problem of SVMs are typically to consider the problem as the combination of two-class decision functions, e.g. one-against-one and one-against-the-rest [5].

Friedman [4] suggested the Max Wins algorithm in which each one-against-one classifier casts one vote for its preferred class, and the final result is the class with the most votes. The Max Wins algorithm offers faster training time compared to the one-against-the-rest method. The decision Directed Acyclic Graph (DDAG) method proposed by Platt et al. reduces training and evaluation time, while maintaining accuracy compared to the Max Wins [6]. The comparison experiments in several methods on large problems in [5] show that the Max Wins algorithm and the DDAG may be more suitable for practical use. Ussivakul and Kijsirikul [7] proposed the Adaptive Directed Acyclic Graph (ADAG) method which is the modification of the DDAG. This method reduces the dependency of the sequence of binary classifiers in nodes in the structure as well as lowers the number of tests required to evaluate for the correct class. Their approach yields higher accuracy and reliability of classification, especially in such a case that the number of classes is relatively large.

In this paper we reveal that the ADAG still has the dependency on the sequence of its nodes, although it is less dependent on the order of binary classes in the sequence than the DDAG; there are still differences in accuracy between different sequences. This led to the reliability of the algorithm. Here we propose a novel method that improves reliability by choosing an optimal sequence, which has less chance to predict the wrong class, and dynamically reordering the sequence during classification process according to each test data. We also reveal that the problem of selecting the appropriate sequence can be solved by minimum-weight perfect matching.

This paper is organized as follows. In the next section, we review SVMs and the formulation to solve multiclass problems, i.e., the DDAG and the ADAG. In Section 3, we introduce the modification of the ADAG to improve the performance by using the reordering algorithm with minimum-weigh perfect matching. The numerical experiments are illustrated in Section 4. Finally, the conclusions are given in Section 5.

2. SVM classification

This section describes the basic idea of SVMs [7] and the formulation to solve multiclass problems.

2.1. Support vector machines

The main idea of support vector machine classification is to construct a hyperplane to separate the two classes.

2.1.1. Linear support vector machines

Suppose we have a data set D of l samples in an n-dimensional space belonging to two different classes (+1 and −1):

$$D = \{(\mathbf{x}_k, y_k) \mid k \in \{1,..,l\}, \mathbf{x}_k \in \Re^n, y_k \in \{+1,-1\}\} \quad (1)$$

The hyperplane in the n dimensional space is determined by the pair $(\mathbf{w},b)$ where $\mathbf{w}$ is an n-dimensional vector orthogonal to the hyperplane and b is the offset constant. The hyperplane $(\mathbf{w}\cdot\mathbf{x})+b$ separates the data if and only if

$$\begin{aligned} (\mathbf{w}\cdot\mathbf{x}_i)+b > 0 \quad &\text{if} \quad y_i = +1 \\ (\mathbf{w}\cdot\mathbf{x}_i)+b < 0 \quad &\text{if} \quad y_i = -1. \end{aligned} \quad (2)$$

If we additionally require that $\mathbf{w}$ and b be such that the point closest to the hyperplane has a distance of $1/|\mathbf{w}|$, then we have

$$\begin{aligned} (\mathbf{w}\cdot\mathbf{x}_i)+b \geq 1 \quad &\text{if} \quad y_i = +1 \\ (\mathbf{w}\cdot\mathbf{x}_i)+b \leq -1 \quad &\text{if} \quad y_i = -1 \end{aligned} \quad (3)$$

which is equivalent to

$$y_i[(\mathbf{w}\cdot\mathbf{x}_i)+b] \geq 1 \quad \forall i. \quad (4)$$

To find the optimal separating hyperplane, we have to find the hyperplane that maximizes the minimum distance between the hyperplane and any sample of training data. The distance between two closest samples from different classes is

$$d(\mathbf{w},b) = \min_{\{\mathbf{x}_i | y_i=1\}} \frac{(\mathbf{w}\cdot\mathbf{x}_i)+b}{|\mathbf{w}|} - \max_{\{\mathbf{x}_i | y_i=-1\}} \frac{(\mathbf{w}\cdot\mathbf{x}_i)+b}{|\mathbf{w}|}. \quad (5)$$

From (3), we can see that the appropriate minimum and maximum values are ±1. Therefore, we need to maximize

$$d(\mathbf{w},b) = \frac{1}{|\mathbf{w}|} - \frac{-1}{|\mathbf{w}|} = \frac{2}{|\mathbf{w}|}. \quad (6)$$

Thus, the problem is equivalent to:

- minimize $|\mathbf{w}|^2/2$
- subject to the constraints
  (1) $y_i[(\mathbf{w}\cdot\mathbf{x}_i)+b] \geq 1 \quad \forall i.$

For non-separable case, the training data cannot be separated by a hyperplane without error. The previous constraints then must be modified. A penalty term consisting of the sum of deviations ξ_i from the boundary is added to the minimization problem. Now, the problem is to

- minimize $\frac{|\mathbf{w}|^2}{2} + C\sum_{i=1}^{l}\xi_i$
- subject to the constraints
  (1) $y_i[(\mathbf{w}\cdot\mathbf{x}_i)+b] \geq 1-\xi_i,$
  (2) $\xi_i \geq 0 \quad \forall i.$

The penalty term for misclassifying training samples is weighted by a constant C. Selecting a large value of C puts a high price on deviations and increases computation by effecting a more exhaustive search for ways to minimize the number of misclassified samples.

By forming the Lagrangian and solving the dual problem, this problem can be translated into:

- minimize
$$L(\mathbf{w},b,\alpha) = \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j (\mathbf{x}_i\cdot\mathbf{x}_j) \quad (7)$$
- subject to the constraints:
  (1) $0 \leq \alpha_i \leq C, \forall i$
  (2) $\sum_{i=1}^{l}\alpha_i y_i = 0$

where α_i are called Lagrange multipliers. There is one Lagrange multiplier for each training sample. In the solution, those samples for which $\alpha_i > 0$ are called *support vectors*, and are ones such that the equality in (4) holds. All other training samples having $\alpha_i = 0$ could be removed from the training set without affecting the final hyperplane.

Let α^0, an l-dimensional vector denote the minimum of $L(\mathbf{w},b,\alpha)$. If $\alpha_i^0 > 0$ then $\mathbf{x}_i$ is a *support vector*. The optimal separating hyperplane $(\mathbf{w}^0, b^0)$ can be written in terms of α^0 and the training data, specifically in terms of the support vectors:

$$\mathbf{w}^0 = \sum_{i=1}^{l}\alpha_i^0 y_i \mathbf{x}_i = \sum_{\text{support vectors}}\alpha_i^0 y_i \mathbf{x}_i. \quad (8)$$

$$b^0 = 1 - \mathbf{w}^0\cdot\mathbf{x}_i \text{ for } \mathbf{x}_i \text{ with } y_i = 1 \text{ and } 0 < \alpha_i < C. \quad (9)$$

The optimal separating hyperplane classifies points according to the sign of $f(\mathbf{x})$,

$$\begin{aligned} f(\mathbf{x}) &= \text{sign}\left(\mathbf{w}^0\cdot\mathbf{x} + b^0\right) \\ &= \text{sign}\left(\sum_{\text{suport vectors}}\alpha_i^0 y_i(\mathbf{x}_i\cdot\mathbf{x}) + b^0\right). \end{aligned} \quad (10)$$

Support vector $\mathbf{x}_i$ with $\alpha_i^0 = C$ may or may not be misclassified. All other $\mathbf{x}_i$'s are correctly classified.

2.1.2. Non-linear support vector machines

The above algorithm is limited to linear separating hyperplanes. SVMs get around this problem by mapping the sample points into a higher dimensional space using a non-linear mapping chosen in advance. This is, we choose a map $\Phi : \Re^n \mapsto H$ where the dimensionality of H is greater than n. We then seek a separating hyperplane in the higher dimensional space; this is equivalent to a non-linear separating surface in $\Re^n$.

The data only ever appears in our training problem (7) in the form of dot products, so in the higher dimensional space we are only dealing with the data in the form $\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$. If the dimensionality of H is very large, then this could be difficult, or very computationally expensive. However, if we have a kernel function such that $k(\mathbf{x}_i,\mathbf{x}_j) = \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$, then we can use this in place of $\mathbf{x}_i\cdot\mathbf{x}_j$ everywhere in the optimization problem, and never need to know explicitly what Φ is. Some widely used kernels are:

$$\text{Polynomial degree } d\text{: } k(\mathbf{x},\mathbf{y}) = \left|\mathbf{x}\cdot\mathbf{y}+1\right|^d \qquad (11)$$

$$\text{Radial basis function: } k(\mathbf{x},\mathbf{y}) = e^{-\left|\mathbf{x}-\mathbf{y}\right|^2/c} \qquad (12)$$

2.2. DDAG

Platt et al. [6] presented a learning architecture, the Decision Directed Acyclic Graph (DDAG), which is used to combine many two-class classifiers into a multiclass classifier. For a k-class problem, its training phase is the same as the one-against-one method by solving $k(k\text{-}1)/2$ binary SVMs, one for each pair of classes. However, in the testing phase, it uses a rooted binary directed acyclic graph which has $k(k\text{-}1)/2$ internal nodes and k leaves (see Figure 1). Each node is a binary SVM of the i^{th} and j^{th} classes. Given a test sample x, starting at the root node, the binary decision function is evaluated. Then it moves to either left or right depending on the output value. Therefore, we go through a path before reaching a leaf node which indicates the predicted class.

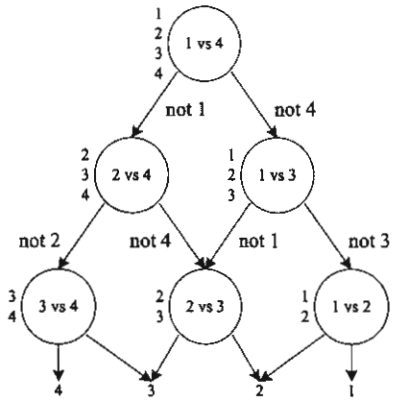


Figure 1. The DDAG finding the best class out of four classes

There are some issues on the DDAG as pointed out by [7]. First, it gives outputs whose probabilities are not uniformly distributed, and thus its output depends on the sequence of binary classifiers in nodes, affecting reliability of the algorithm. In addition, the correct class placed in a node near the root node is clearly at disadvantage by comparison with the correct class near leaf nodes since it is exposed to higher risk of being incorrectly rejected. Second, the number of node evaluations for the correct class is unnecessary high. This results in higher cumulative error and lower the accuracy. The depth of the DDAG is k-1 and this means that the number of times the correct class has to be tested against other classes, on average, scales linearly with k.

2.3. ADAG

Ussivakul and Kijsirikul [7] proposed an approach to alleviate the problem of the DDAG structure described above. An Adaptive DAG (ADAG) is a DAG with a reversed triangular structure. For a k-class problem, its training phase is the same as the DDAG method by solving $k(k\text{-}1)/2$ binary SVMs, one for each pair of classes. However, in the testing phase, the nodes are arranged in a reversed triangle with $k/2$ nodes (rounded up) at the top, $k/2^2$ nodes in the second layer and so on until the lowest layer of a final node. It has k-1 internal nodes, each of which is labeled with an element of Boolean function (see Figure 2). Given a test example x, starting at the top level, the binary decision function is evaluated. The node is then exited via the outgoing edge with a message of the preferred class. In each round, the number of candidate classes is reduced by half. Based on the preferred classes from its parent nodes, the binary function of the next-level node is chosen. The reduction process continues until reaching the final node at the lowest level. The value of the decision function is the value associated with the message from the final leaf node. Like the DDAG, the ADAG requires only k-1 decision nodes to be evaluated in order to derive an answer. Note that the correct class is evaluated against other classes for $\log_2 k$ times or less, considerably lower than the number of evaluations required by the DDAG, which scales linearly with k.

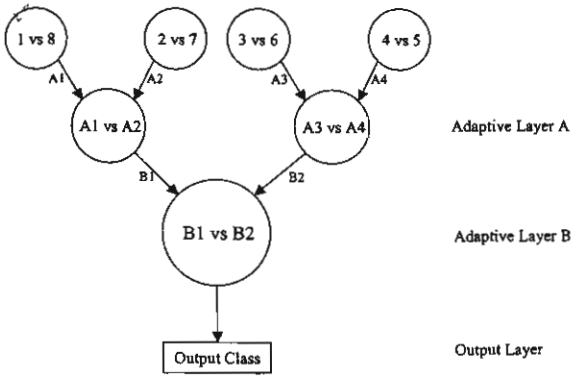


Figure 2. The structure of an Adaptive DAG for an 8-class problem

2.1.2. Non-linear support vector machines

The above algorithm is limited to linear separating hyperplanes. SVMs get around this problem by mapping the sample points into a higher dimensional space using a non-linear mapping chosen in advance. This is, we choose a map $\Phi : \Re^n \mapsto H$ where the dimensionality of H is greater than n. We then seek a separating hyperplane in the higher dimensional space; this is equivalent to a non-linear separating surface in $\Re^n$.

The data only ever appears in our training problem (7) in the form of dot products, so in the higher dimensional space we are only dealing with the data in the form $\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$. If the dimensionality of H is very large, then this could be difficult, or very computationally expensive. However, if we have a kernel function such that $k(\mathbf{x}_i,\mathbf{x}_j) = \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$, then we can use this in place of $\mathbf{x}_i\cdot\mathbf{x}_j$ everywhere in the optimization problem, and never need to know explicitly what Φ is. Some widely used kernels are:

$$\text{Polynomial degree } d\text{: } k(\mathbf{x},\mathbf{y}) = \left|\mathbf{x}\cdot\mathbf{y}+1\right|^d \qquad (11)$$

$$\text{Radial basis function: } k(\mathbf{x},\mathbf{y}) = e^{-\left|\mathbf{x}-\mathbf{y}\right|^2/c} \qquad (12)$$

2.2. DDAG

Platt et al. [6] presented a learning architecture, the Decision Directed Acyclic Graph (DDAG), which is used to combine many two-class classifiers into a multiclass classifier. For a k-class problem, its training phase is the same as the one-against-one method by solving $k(k\text{-}1)/2$ binary SVMs, one for each pair of classes. However, in the testing phase, it uses a rooted binary directed acyclic graph which has $k(k\text{-}1)/2$ internal nodes and k leaves (see Figure 1). Each node is a binary SVM of the i^{th} and j^{th} classes. Given a test sample x, starting at the root node, the binary decision function is evaluated. Then it moves to either left or right depending on the output value. Therefore, we go through a path before reaching a leaf node which indicates the predicted class.

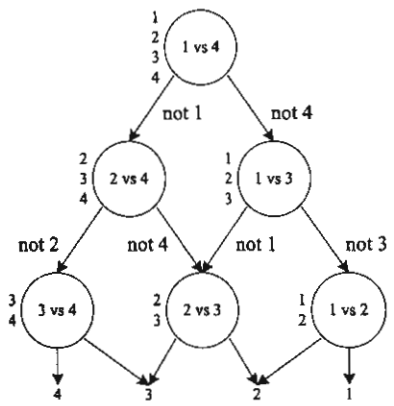


Figure 1. The DDAG finding the best class out of four classes

There are some issues on the DDAG as pointed out by [7]. First, it gives outputs whose probabilities are not uniformly distributed, and thus its output depends on the sequence of binary classifiers in nodes, affecting reliability of the algorithm. In addition, the correct class placed in a node near the root node is clearly at disadvantage by comparison with the correct class near leaf nodes since it is exposed to higher risk of being incorrectly rejected. Second, the number of node evaluations for the correct class is unnecessary high. This results in higher cumulative error and lower the accuracy. The depth of the DDAG is k-1 and this means that the number of times the correct class has to be tested against other classes, on average, scales linearly with k.

2.3. ADAG

Ussivakul and Kijsirikul [7] proposed an approach to alleviate the problem of the DDAG structure described above. An Adaptive DAG (ADAG) is a DAG with a reversed triangular structure. For a k-class problem, its training phase is the same as the DDAG method by solving $k(k\text{-}1)/2$ binary SVMs, one for each pair of classes. However, in the testing phase, the nodes are arranged in a reversed triangle with $k/2$ nodes (rounded up) at the top, $k/2^2$ nodes in the second layer and so on until the lowest layer of a final node. It has k-1 internal nodes, each of which is labeled with an element of Boolean function (see Figure 2). Given a test example x, starting at the top level, the binary decision function is evaluated. The node is then exited via the outgoing edge with a message of the preferred class. In each round, the number of candidate classes is reduced by half. Based on the preferred classes from its parent nodes, the binary function of the next-level node is chosen. The reduction process continues until reaching the final node at the lowest level. The value of the decision function is the value associated with the message from the final leaf node. Like the DDAG, the ADAG requires only k-1 decision nodes to be evaluated in order to derive an answer. Note that the correct class is evaluated against other classes for $\log_2 k$ times or less, considerably lower than the number of evaluations required by the DDAG, which scales linearly with k.

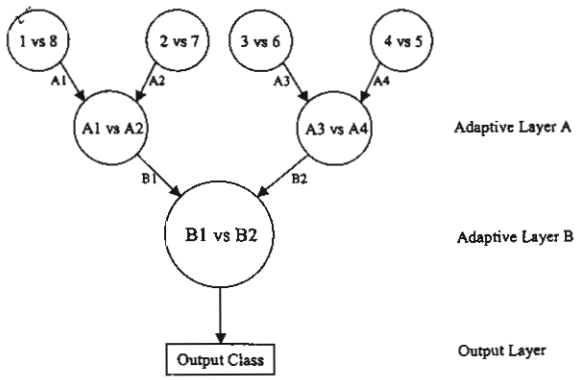


Figure 2. The structure of an Adaptive DAG for an 8-class problem

Using the reversed triangular structure, the ADAG reduces the number of times the correct class is tested against other classes, and thus reduces the cumulative errors. However, there are still differences in accuracy between different sequences of nodes. Next we will describe our method that improves the ADAG by finding a best sequence of nodes.

3. The proposed method

In this section, we introduce the modification of the ADAG to improve the performance of the original ADAG. This approach determines a best sequence of nodes in the ADAG by dynamically reordering the sequence during classification process according to each test data.

3.1. Generalization Performance of Support Vector Machines

Generalization analysis of pattern classifiers is concerned with determining the factors that affect the accuracy of a pattern classifier [1]. Generalization performance of Support Vector Machines can be approximated by bounding on the generalization error.

Define the class F of real-valued functions on the ball of radius R in $\Re^n$ as $F = \{\mathbf{x} \mapsto \mathbf{w} \cdot \mathbf{x} : \|\mathbf{w}\| \le 1, \|\mathbf{x}\| \le R\}$. There is a constant c such that, for all probability distributions, with probability at least $1-\delta$ over m independently generated examples $\mathbf{z}$, if a classifier $h = \mathrm{sgn}(f) \in \mathrm{sgn}(F)$ has margin at least γ on all the examples in $\mathbf{z}$, then the error of h is no more than

$$\frac{c}{m}\left(\frac{R^2}{\gamma^2}\log^2 m + \log\left(\frac{1}{\delta}\right)\right). \qquad (13)$$

Furthermore, with probability at least $1-\delta$, every classifier $h \in \mathrm{sgn}(F)$ has error no more than

$$\frac{k}{m} + \sqrt{\frac{c}{m}\left(\frac{R^2}{\gamma^2}\log^2 m + \log\left(\frac{1}{\delta}\right)\right)} \qquad (14)$$

where k is the number of labeled examples in $\mathbf{z}$ with margin less than γ.

Below we show an example of the generalization error of classifier. The experiment is based on the English letter image recognition dataset from [2], which has 26 classes. Hence there are 325 classifiers. In this case, the dataset is trained by using the Polynomial kernel of degree 3. In Figure 3, the generalization errors of all classifiers expressed by Equation (14) are depicted. The generalization errors of all classifiers are varying.

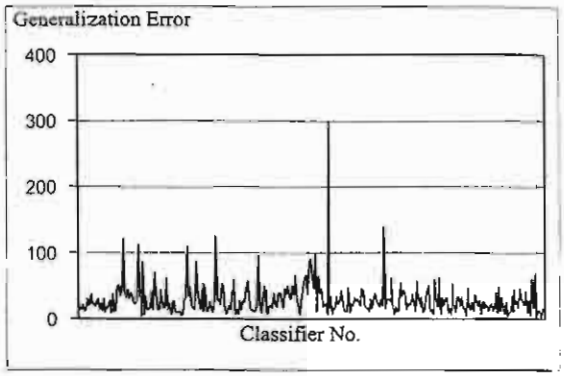


Figure 3. The generalization errors of 325 classifiers

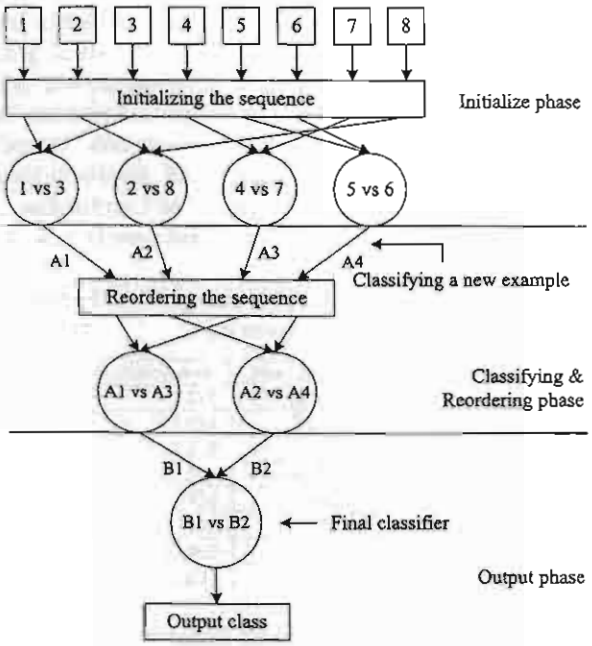


Figure 4. Classifying process of the RADAG

3.2. Reordering ADAG

We propose a method, called Reordering Adaptive Directed Acyclic Graph (RADAG), to improve the accuracy of the original ADAG. For a k-class problem, the RADAG's training phase is the same as the ADAG method by solving $k(k-1)/2$ binary SVMs. However, the testing phase is organized as follows. The differences are the initialization of the binary classifiers in the top level and the order of sequence in each level (see Figure 4). In the first step, we use a reordering algorithm with minimum-weight perfect matching described in the next subsection to choose the optimal sequence to be the initial sequence. We use the sequence to evaluate every test example. In the second step, as in the ADAG, test points of the RADAG are evaluated against the decision nodes. In the third step, unlike the ADAG, the RADAG will reorder the sequence before processing in the next level by using the reordering algorithm with minimum-weight

perfect matching. This sequence differs for each test example, and it depends on the results of nodes from the previous level. The second and the third steps are repeated until there is only one class remains.

3.3. Reordering algorithm

For the reason described above, we consider the generalization errors in order to choose the optimal sequence from all possible sequences with less chance to predict the wrong class. Among classifiers, $k/2$ classifiers which have small generalization errors will be considered to be used in data classification.

Let $G = (V, E)$ be a graph with node set V and edge set E. Each node in G denotes one class and each edge denotes one binary classifier which has a generalization error expressed by Equation (14) (see Figure 5(a)). The output of the reordering algorithm for graph G is a subset of edges with the minimum sum of generalization errors of all edges and each node in G is met by exactly one edge in the subset (see Figure 5(b)). Given a real weight c_e being generalization error for each edge e of G, the problem of reordering algorithm can be solved by the minimum weight perfect matching that finds a perfect matching M of minimum weight $\Sigma(c_e : e \in M)$.

For $U \subseteq V$, let $E(U) = \{(i,j):(i,j) \in E,\ i \in U, j \in U\}$. $E(U)$ is the set of chosen classifiers. Let $\delta(i)$ denote the set of edges incident to node i; the set of classifiers with one class being i. The perfect matchings on a graph $G = (V, E)$ with $|V|$ even is given by

(a) $x \in R_+^m$

(b) $\sum_{e \in \delta(v)} x_e = 1 \quad \text{for } v \in V$

(c) $\sum_{e \in E(U)} x_e \le \left\lfloor \frac{|U|}{2} \right\rfloor$ for all odd sets $U \subseteq V$ with $|U| \ge 3$

or by (a),(b) and

(d) $\sum_{e \in \delta(U)} x_e \ge 1$ for all odd sets $U \subseteq V$ with $|U| \ge 3$ (16)

where $|E| = m$, the number of classifiers, and $x_e = 1$ means that classifier e is chosen to be used in the sequence. Therefore, the reordering problem is to solve the following linear program:

$$\min \sum_{e \in E} c_e x_e \qquad (17)$$

where x satisfies "(a),(b) and (c)" or "(a),(b) and (d)".

Currently, the minimum-weight perfect matching algorithm runs in time bounded by $O(n(m + n \log n))$ [3], where n is the number of nodes (classes) in the graph and m is the number of edges (binary classifiers). Hence the reordering algorithm can reorder the sequence in that polynomial time.

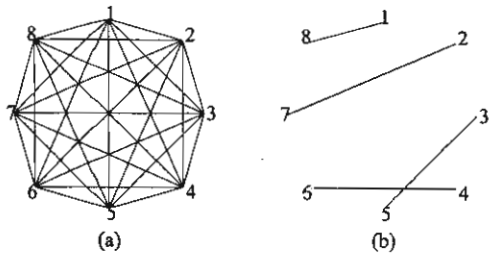


Figure 5. (a) A graph for an 8-class problem. (b) An example of the output of the reordering algorithm.

4. Numerical experiments

In this section, we present experimental results on several datasets from the UCI Repository of machine learning databases [2] including glass, satimage, segment, shuttle, vowel, soybean, letter and isolet (see Table 1). These datasets are different in the number of classes, the number of dimensions, and sizes. For the glass and segment problems, there is no test data so we used 5-fold cross validation. For the soybean problem, we discarded the last four classes because of missing values.

Table 1. Description of the datasets used in the experiments

Dataset	#training data	#test data	#class	#dimension
Glass	214	5-fold	6	9
Satimage	4,435	2,000	6	36
Segment	2,310	5-fold	7	18
Shuttle	43,500	14,500	7	9
Vowel	528	462	11	10
Soybean	290	340	15	35
Letter	15,963	4,037	26	16
Isolet	6,238	1,559	26	617

In these experiments we scaled both training data and test data to be in [-1,1] and employed Polynomial and RBF kernels. In the experiments, we compare three algorithms, i.e., the original ADAG, the RADAG, and the Max Wins algorithm. For the ADAG, we examined all possible sequences for datasets having not more than 7 classes, whereas we randomly selected 50,000 sequences for datasets having more than 7 classes. Table 2 and 3 present the results of comparing these methods for Polynomial and RBF kernels, respectively. We present the optimal parameters (d and c in Equations (11) and (12)) of the kernels and the corresponding accuracies. The best accuracy among three methods is illustrated in bold-face.

The results show that our method yields highest accuracy in almost all of datasets. Another advantage of our method compared to the DDAG and the original ADAG is that our method always provides one best accuracy for each dataset using the reordering algorithm, whereas, depending on the sequence of classes, the DDAG and the original ADAG may give low accuracies. This shows the effectiveness of the RADAG.

perfect matching. This sequence differs for each test example, and it depends on the results of nodes from the previous level. The second and the third steps are repeated until there is only one class remains.

3.3. Reordering algorithm

For the reason described above, we consider the generalization errors in order to choose the optimal sequence from all possible sequences with less chance to predict the wrong class. Among classifiers, $k/2$ classifiers which have small generalization errors will be considered to be used in data classification.

Let $G = (V, E)$ be a graph with node set V and edge set E. Each node in G denotes one class and each edge denotes one binary classifier which has a generalization error expressed by Equation (14) (see Figure 5(a)). The output of the reordering algorithm for graph G is a subset of edges with the minimum sum of generalization errors of all edges and each node in G is met by exactly one edge in the subset (see Figure 5(b)). Given a real weight c_e being generalization error for each edge e of G, the problem of reordering algorithm can be solved by the minimum weight perfect matching that finds a perfect matching M of minimum weight $\Sigma(c_e : e \in M)$.

For $U \subseteq V$, let $E(U) = \{(i,j):(i,j)\in E, i\in U, j\in U\}$. $E(U)$ is the set of chosen classifiers. Let $\delta(i)$ denote the set of edges incident to node i; the set of classifiers with one class being i. The perfect matchings on a graph $G = (V, E)$ with $|V|$ even is given by

(a) $x \in R_+^m$

(b) $\sum_{e\in\delta(v)} x_e = 1 \quad \text{for } v \in V$

(c) $\sum_{e\in E(U)} x_e \le \left\lfloor \frac{|U|}{2} \right\rfloor$ for all odd sets $U \subseteq V$ with $|U| \ge 3$

or by (a),(b) and

(d) $\sum_{e\in\delta(U)} x_e \ge 1$ for all odd sets $U \subseteq V$ with $|U| \ge 3$ (16)

where $|E| = m$, the number of classifiers, and $x_e = 1$ means that classifier e is chosen to be used in the sequence. Therefore, the reordering problem is to solve the following linear program:

$$\min \sum_{e\in E} c_e x_e \qquad (17)$$

where x satisfies "(a),(b) and (c)" or "(a),(b) and (d)".

Currently, the minimum-weight perfect matching algorithm runs in time bounded by $O(n(m + n \log n))$ [3], where n is the number of nodes (classes) in the graph and m is the number of edges (binary classifiers). Hence the reordering algorithm can reorder the sequence in that polynomial time.

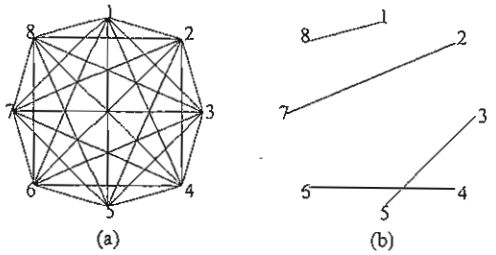


Figure 5. (a) A graph for an 8-class problem. (b) An example of the output of the reordering algorithm.

4. Numerical experiments

In this section, we present experimental results on several datasets from the UCI Repository of machine learning databases [2] including glass, satimage, segment, shuttle, vowel, soybean, letter and isolet (see Table 1). These datasets are different in the number of classes, the number of dimensions, and sizes. For the glass and segment problems, there is no test data so we used 5-fold cross validation. For the soybean problem, we discarded the last four classes because of missing values.

Table 1. Description of the datasets used in the experiments

Dataset	#training data	#test data	#class	#dimension
Glass	214	5-fold	6	9
Satimage	4,435	2,000	6	36
Segment	2,310	5-fold	7	18
Shuttle	43,500	14,500	7	9
Vowel	528	462	11	10
Soybean	290	340	15	35
Letter	15,963	4,037	26	16
Isolet	6,238	1,559	26	617

In these experiments we scaled both training data and test data to be in [-1,1] and employed Polynomial and RBF kernels. In the experiments, we compare three algorithms, i.e., the original ADAG, the RADAG, and the Max Wins algorithm. For the ADAG, we examined all possible sequences for datasets having not more than 7 classes, whereas we randomly selected 50,000 sequences for datasets having more than 7 classes. Table 2 and 3 present the results of comparing these methods for Polynomial and RBF kernels, respectively. We present the optimal parameters (d and c in Equations (11) and (12)) of the kernels and the corresponding accuracies. The best accuracy among three methods is illustrated in bold-face.

The results show that our method yields highest accuracy in almost all of datasets. Another advantage of our method compared to the DDAG and the original ADAG is that our method always provides one best accuracy for each dataset using the reordering algorithm, whereas, depending on the sequence of classes, the DDAG and the original ADAG may give low accuracies. This shows the effectiveness of the RADAG.

Table 2. A comparison of the accuracy of classification using the Polynomial kernel

Dataset	d	ADAG	d	Max Wins	d	RADAG
Glass	2	**71.135**	2	71.078	2	71.063
Satimage	6	88.430	6	88.453	6	**88.900**
Segment	8	97.408	8	97.379	8	**97.489**
Shuttle	8	**99.924**	8	**99.924**	8	**99.924**
Vowel	3	64.293	3	64.329	2	**64.502**
Soybean	5	90.446	3	90.471	3	**91.176**
Letter	3	95.984	3	**96.125**	4	96.111
Isolet	3	97.030	3	97.040	3	**97.049**

Table 3. A comparison of the accuracy of classification using the RBF kernel

Dataset	c	ADAG	c	Max Wins	c	RADAG
Glass	0.08	72.759	0.09	73.238	0.09	**74.319**
Satimage	3.0	91.968	3.0	**91.984**	3.0	91.950
Segment	0.7	97.282	0.7	**97.298**	0.7	97.273
Shuttle	3.0	**99.897**	3.0	**99.897**	3.0	**99.897**
Vowel	0.2	65.589	0.2	65.340	0.2	**67.100**
Soybean	0.08	90.412	0.08	90.468	0.07	**90.882**
Letter	3.0	97.909	3.0	97.918	3.0	**97.969**
Isolet	0.01	96.932	0.01	96.916	0.01	**96.985**

Table 4. A comparison of the computational time using the Polynomial kernel

Dataset	d	RADAG			Max Wins
		Classifying (seconds)	Reordering (seconds)	Total (seconds)	Classifying (seconds)
Satimage	6	1.90	0.50	2.40	9.47
Segment	8	0.11	0.08	0.19	0.41
Shuttle	8	1.75	3.38	5.13	5.15
Vowel	2	0.12	0.25	0.37	0.61
Soybean	3	0.30	0.25	0.55	1.86
Letter	4	8.48	4.20	12.68	125.58
Isolet	3	116.02	1.48	117.50	1671.98

Table 5. A comparison of the computational time using the RBF kernel

Dataset	c	RADAG			Max Wins
		Classifying (seconds)	Reordering (seconds)	Total (seconds)	Classifying (seconds)
Satimage	3.0	11.75	0.51	12.26	37.13
Segment	0.7	0.24	0.10	0.34	0.82
Shuttle	3.0	3.36	0.63	3.99	9.27
Vowel	0.2	0.10	0.27	0.37	0.61
Soybean	0.07	0.32	0.45	0.77	2.20
Letter	3.0	62.27	3.69	65.96	802.85
Isolet	0.01	100.42	1.60	102.02	1369.11

Table 4 and 5 present the comparison of the computational time between the RADAG and the Max Wins for Polynomial and RBF kernels by using a 400 MHz Pentium II processor. There is no computational time of the glass dataset because it has too little test examples to measure the time. The results show that our method requires low computational time in all datasets, especially when the number of classes and/or the number of dimensions are relatively large. For a k class problem, the Max Wins requires $k(k\text{-}1)/2$ classifiers for the classification whereas the RADAG requires only k-1 classifiers. Hence the larger the number of classes the more running time the Max Wins requires than the RADAG. Moreover, the number of dimensions affects the running time of each classifier. For the RADAG, the number of classes affects the running time for reordering. However, it takes a little time even when there are many classes.

5. Conclusions

In this paper, we have presented a new approach for multiclass SVMs, called Reordering Adaptive Directed Acyclic Graph (RADAG), which is the modification of the original ADAG. Our approach eliminates the dependency of the sequence of binary classifiers in nodes in the original ADAG by selecting an appropriate sequence from all possible sequences which consists of classifiers with small generalization error. By the use of minimum-weight perfect matching, the RADAG can reorder the sequence in polynomial time. The experimental results show that our new approach yields higher accuracy than the original ADAG and even Max Wins which is probably the currently most accurate method for multiclass SVMs. Moreover the running time used by the RADAG is less than Max Wins, especially when the number of classes and/or the number of dimensions are relatively large. Our future work is to test the method on datasets with a very large number of classes and dimensions.

6. Acknowledgment

This work was supported by The Thailand Research Fund.

7. References

[1] P. L. Bartlett, J. Shawe-Taylor, "Generalization performance of support vector machines and other pattern classifiers", in Advances in Kernel Methods - Support Vector Learning, B. Schoelkopf, C. J. C. Burges, A. J. Smola (eds), pp. 43-54, MIT Press, Cambridge, USA, 1999.

[2] C. Blake, E. Keogh and C. Merz, UCI repository of machine learning databases, Department of Information and Computer Science, University of California, Irvine, 1998. http://www.ics.uci.edu/~mlearn/MLRepository. html

[3] W. Cook, and A. Rohe, "Computing minimum-weight perfect matchings", Technical Report 97863, Forschungsinstitut für Diskrete Mathematik, Universität Bonn, 1997.

[4] J. H. Friedman, "Another approach to Polychotomous classification", Technical report, Department of Statistics, Stanford University, 1996.

[5] C.-W. Hsu, and C.-J. Lin, "A comparison of methods for multiclass Support Vector Machines", IEEE Trans.on Neural Networks, Vol. 13, pp. 415-425, March, 2002.

[6] J. Platt, N. Cristianini, and J. Shawe-Taylor, "Large margin DAGs for multiclass classification", in Advances in Neural Information Processing Systems, MIT Press, Vol. 12, pp. 547-553, 2000.

[7] N. Ussivakul, and B. Kijsirikul, "Multiclass support vector machines using adaptive directed acyclic graph," in IEEE/INNS' Int. Joint Conf. On Neural Networks (IJCNN-2002), 2002.

A NEW FRAMEWORK FOR LEARNING FIRST-ORDER REPRESENTATION

Thanupol Leardlumnouchai[1] and Boonserm Kijsirikul[1]

ABSTRACT: First-order logic rules are one of the most expressive and human readable representations for learning a hypothesis in form of a set of production rules (if-then rules). However, the first-order rules can be learned only by Inductive Logic Programming (ILP) systems, such as PROGOL, FOIL. Other machine learning techniques, such as Neural Networks, Bayesian Networks and Decision Tree Learning, cannot directly learn this kind of rules because these techniques could not select the appropriate values to substitute in variables of the first-order rules. In this paper, we propose the method that makes use of Multiple-Instance Learning (MIL) as a new framework for learning first-order representation. MIL is employed for determining the appropriate values for the substitution. Experimental results show that the proposed method effectively learns first-order representation and is comparable to ILP systems.

KEYWORDS: Inductive Logic Programming, First-Order Logic, Multiple-Instance Learning

1. INTRODUCTION

Knowledge based learning is one technique of machine learning (Mitchell 1997). Informally knowledge is a set of sentences that describe known logical facts. In learning, the knowledge is generally called background knowledge. Knowledge-based learning method receives background knowledge and positive and negative examples as inputs and outputs a learned hypothesis represented in form of a set of production rules (if-then rules). A prominent advantage of this representation is human readable. Also, there are two types of logic for representing learned rules. The first one is propositional logic which is simpler than the other one, first-order logic. Propositional rules have no variable in their rules. In contrast, first-order rules contain variables so they are more expressive than the propositional rules. Consider the difference between these two logical rules in **Figure 1**.

Propositional logic:	IF (Name1=Jannifer)∧(Name2=Andrew)∧(Father1 = Andrew)∧(Female1=True) THEN Daughter1,2 = True
First-order logic:	IF Father(y,x) Female(y) THEN Daughter(x,y)

Figure 1. The propositional logic and first-order logic rule.

The propositional rule is specific only for the first and the second persons being Jannifer and Andrew, respectively. So it is rarely useful for unseen pairs of people. While first-order rules have variables that make the rule much more expressive and general. However, only Inductive Logic Programming (ILP) systems such as PROGOL (Mitchell 1997), FOIL (Mitchell 1997) can learn the first-order rules. These first-order rule learners have ability to select the appropriate values to substitute in variables, while other machine learning method cannot, such as Neural Networks, Bayesian Networks and Decision Tree Learning. Therefore the advantages of these methods, such as robustness to noise, could not be applied in first-order rule learning to increase the efficiency in real world usage.

[1] *Department of Computer Engineering, Chulalongkorn University, Pathumwan, Bangkok, Thailand*

In this paper, we are interested in solving this challenge. We propose the method that can use other techniques to learn first-order rules by using a framework called Multiple-Instance Learning (MIL) (Huang, Chen et al. 2003). MIL is employed for determining the appropriate values for the variable substitutions. We evaluate the proposed method by learning first-order concept *mother(x,y)* with Backpropagation Neural Network (BNN) on MIL framework. The results show that our technique competently learns first-order representation.

2. FRAMEWORK

ILP is only one of machine learning which adapts the logical concept for hypothesis learning and represents the learned rules in first-order logic form. Other learning techniques cannot directly learn this kind of rules because of difficulty in selecting the appropriate values to substitute in variables of first-order rules. To illustrate this problem, consider the task of learning rules for classifying the rich person (*rich(x)*). Background knowledge, the positive and negative examples are given as follow.

- **background knowledge:** genius(Alan), diligent(Alan), genius(Bob), strong(Bob), dilligent(Chris), strong(Chris)
- **positive example:** rich(Alan)
- **negative example:** rich(Bob), rich(Chris)

Figure 2. Input for learning the concept *rich(x)*.

From these inputs, the ILP system could provide the learned rule as: *rich(x):- genius(x), diligent(x)*. The meaning of the above rule is if x is genius and diligent, then x will be a rich person. For comparing to ILP, we use the BNN as the representative of indirectly learning methods. BNN also learns the rich concept with the same inputs as ILP. First, we create the initial network that is equivalent to the background knowledge. The network has 3 input nodes, in this case. We then assign some small random value to initialize each weight. The network constructed from the background knowledge is shown in **Figure 3**.

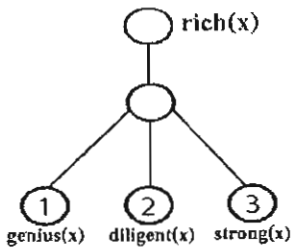


Figure 3. The constructed network.

Next, we feed an example to the network one by one. The input value for an input unit will be 1 if the literal of that unit is true in background knowledge when variables of the literal are substituted to some constants. Otherwise the input value for that unit is 0. For instance, if the fed example is *rich(Alan)*, then the value for each unit is 1, 1 and 0, respectively. Because literals *genius(Alan)* and *diligent(Alan)* are true in background knowledge, while *strong(Alan)* is not true. The target value for the output unit is 1 because *rich(Alan)* is a positive example. The target output is 0 in case of negative examples. Then we apply the Backpropagation algorithm to adjust the network weights to fit training examples. After training, the weight values of *genius(x)* and *diligent(x)* units are higher than the *strong(x)* unit. This is because the two previous literals are more significant than literal *strong(x)* and this is comparable to the induced rule from ILP.

However, when inputs are relational examples and background knowledge, neural network cannot easily learn as the previous example. As there are many constants that can be mapped to relational variables, the correct inputs for the neural network cannot be easily determined. Consider the following example.

- **background knowledge:** diligent(Alan), parent(Bob,Alan), genius(Bob), diligent(Bob), dilligent(Chris), strong(Chris)
- **positive example:** rich(Alan), rich(Bob)
- **negative example:** rich(Chris)

Figure 4. Inputs that contain relational predicate.

In this example, an additional relational literal *parent(Bob,Alan)* is given. This literal means Bob is Alan's parent. The construction of the network also adds nodes *parent(x,y)* and *parent(y,x)* in the input layer, so there are 5 input nodes in this case. When we feed positive example *rich(Alan)* to the network, the first three nodes which are *genius(x), diligent(x)* and *strong(x)* receive 0,1 and 0 as its input respectively. For node *parent(y,x),* the variable x is replaced by Alan. But we have to determine to which term (Alan, Bob or Chris) variable y should be replaced. If we select Bob for substitution, the truth value for this input unit will be 1. The other substitutions will give 0 for this unit. While the truth value for *parent(x,y)* is 0 for any substitution. This learning problem may occur when background knowledge contains relational data and the learner cannot determine the appropriate value for the variable substitution.

To solve this problem, we use the power of Multiple-Instance Learning (MIL) to provide input data for relational first-order rule learning. In MIL framework, the training set is composed of a set of bags, each of which is a collection of different number of instances. A bag is labeled as a negative bag if all the instances in it are negative. On the other hand, if a bag contains at least one positive instance then it is labeled as a positive bag. With this concept, we define a set of training examples as $\{B_1, B_2, \ldots B_n\}$, where n is a number of examples including positive and negative ones. A bag is labeled as a positive bag if an example is positive, and negative otherwise. Each bag contains m_i instances $\{B_{i1}, B_{i2}, \ldots, B_{imi}\}$ where each is one possible binding (substitution). Therefore the appropriate value selection would not be a problem because in one bag there are all cases of variable substitutions and the learning algorithm are designed for this kind of MIL problem. We can use these transformed data for learning a hypothesis. Consider an example of positive bag *rich(Alan)* as input data (see **Table 1**).

Table 1. Input data of the network for bag *rich(Alan)*.

Bag of example rich(Alan)	genius(x)	diligent(x)	strong(x)	parent(x,y)	parent(y,x)
Replace x by Alan, and y by Alan	0	1	0	0	0
Replace x by Alan, and y by Bob	0	1	0	0	1
Replace x by Alan, and y by Chris	0	1	0	0	0

As shown in **Table 1,** the bag *rich(Alan)* has 3 instances. Positive bag *rich(Bob)* and negative bag *rich(Chris)* also have 3 instances as same as bag *rich(Alan)*. For training, these 3 bags are fed to the network one by one. The network weights are adapted by the backpropagation algorithm for MIL (Zhou and Zhang 2002).

3. EXPERIMENT

In this section, we evaluate our proposed technique on learning first-order rules by BNN. The target concept that we try to learn is *mother(x,y)*. A family relationship as shown in **Figure 5** describes a set of training examples. Additionally, four predicates which are *father(x,y), husband(x,y), grandmother(x,y)* and *sister(x,y)* are given as background knowledge.

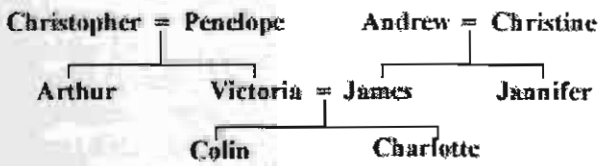


Figure 5. A family relationship where A=B means A marries B.

With these input data, we get 6 positive bags that are *mother(Penelope,Arthur), mother(Penelope,Victoria), mother(Christine,James), mother(Christine,Jannifer), mother(Victoria,Colin),* and *mother(Victoria,Charlotte).* Each positive bag is composed of 10 instances each of which is one case of relational variable *z* replacement. For negative bag, we take one person as one bag so there are 10 negative bags. Each bag has 10 cases of persons for substitution in variable *y* such as *mother(Christopher,Christopher), mother(Christopher,Penelope),* ... and each case contains 10 bindings. So each negative bag contains 90 instances, except for the 3 bags for Penelope, Christine and Victoria which contain only 70 instances as some of them are already used as positive bags. Moreover, each predicate from background knowledge is expanded to 6 literals which are for *(x,y), (x,z), (y,x), (y,z), (z,x),* and *(z,y).* Variable *z* is used for making a connection between literals. The network must be consistent with background knowledge, so we create a network with 24 input nodes and 1 output node. We set the number of hidden nodes to 1 node and 4 nodes for the first and second experiments, respectively. Then we train the network for 2000 epochs by using the backpropagation algorithm. The obtained networks are shown in **Figure 6**, with dark solid lines indicating the largest positive weights, and light lines indicating negligible weights. Both networks perfectly classified all of the training examples and the networks are almost equivalent. Furthermore from learned network weights, the network can be mapped to a rule mother(x,y) ← father(z,y), husband(z,x) (Towell and Shavlik 1993), which is the same as the rule learned by ILP.

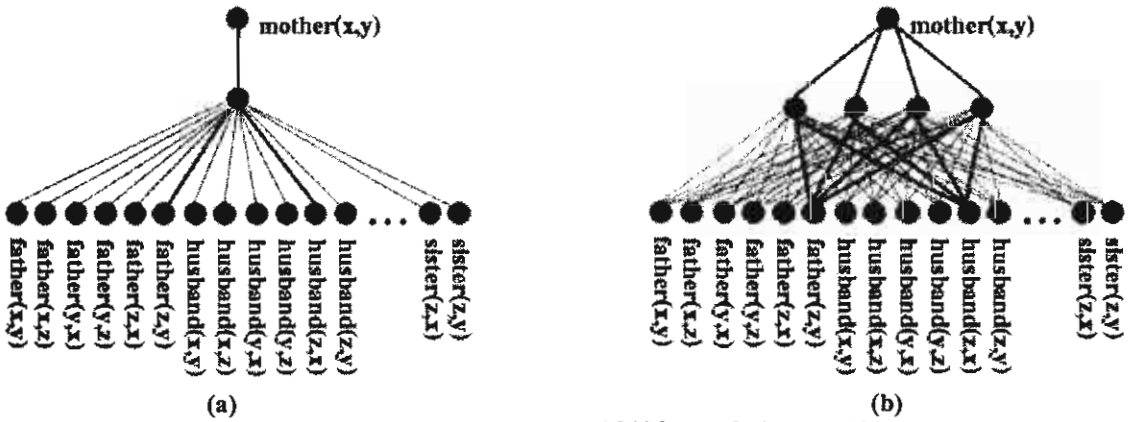


Figure 6. The complete trained networks with 1 hidden node (a) and 4 hidden nodes (b).

4. CONCLUSION

This paper presents a new framework for learning first-order rules. The objective is to solve the problem that other machine learning techniques except for ILP cannot select the appropriate values for variable substitution. So we applied MIL to provide certain input data from first-order logic input. The experimental results show that our proposed method is able to learn first-order representation. The refined network can be mapped to the rule which is comparable to one obtained by ILP. Consequently, as we can employ other techniques for learning first-order logic, the advantage of these methods can alleviate the weakness of ILP such as sensitivity to noise.

5. REFERENCES

Huang, X., S.-C. Chen, et al. (2003). An Open Multiple Instance Learning Framework and Its Application in Drug Activity Prediction Problems. Proceedings of the Third IEEE Symposium on BioInformatics and BioEngineering (BIBE'03), Bethesda, Maryland.

Mitchell, T. M. (1997). Machine Learning, The McGraw-Hill Companies Inc.

Towell, G. G. and J. W. Shavlik (1993). "The Extraction of Refined Rules from Knowledge-Based Neural Networks." Machine Learning Journal 13(1): 71-101.

Zhou, Z.-H. and M.-L. Zhang (2002). Neural Network for Multi-Instance Learning. Proceedings of the International Conference on Intelligent Information Technology, Beijing, China.

Multiclass Support Vector Machines Using Balanced Dichotomization

Boonserm Kijsirikul, Narong Boonsirisumpun, and Yachai Limpiyakorn

Department of Computer Engineering, Chulalongkorn University, Thailand
{Boonserm.K,Yachai.L}@chula.ac.th
Narong.Bo@student.chula.ac.th

The Support Vector Machine (SVM) has been introduced as a technique for solving a variety of learning and function estimation problems. The technique was originally designed for binary classification learning with its outstanding performance. However, many real world applications involve multiclass classification. Typical SVM solutions to N-class problems are to construct and combine several two-class classifiers into an N-class classifier such as the one-against-the-rest approach (1-v-r) and the one-against-one approach (1-v-1). The one-against-one methods solve $N(N-1)/2$ binary classifiers where each one is trained on data from two classes. There are different methods for the evaluation of the correct class after all $N(N-1)/2$ classifiers have been constructed. The Max Wins method takes the majority vote of a certain class as the final output [3]. A drawback of the 1-v-1 SVMs is their inefficiency of classifying data as the number of SVMs grows superlinearly with the number of classes. To improve the efficiency in classifying data, Platt et al. [5] proposed the Decision Directed Acyclic Graph (DDAG) with $N(N-1)/2$ internal nodes and N leaves. Only $N-1$ decision nodes will be evaluated in order to derive an answer, that is lower than $N(N-1)/2$ decisions required by Max Wins. To reduce the unnecessarily high number of node evaluations for the correct class, Kijsirikul, et al. [4] proposed the Adaptive Directed Acyclic Graph (ADAG) method, which is a modification of the DDAG. Like the DDAG, the ADAG requires $N-1$ decisions in order to derive an answer. However, using the reversed triangular structure reduces the number of evaluations the correct class is tested against other classes to $\lceil \log_2 N \rceil$ times or less, which is considerably lower than that of $N-1$ times required by the DDAG.

In this paper, we introduce a new method for constructing multiclass SVMs using binary classifiers, called *Balanced Dichotomization*. For an N-class problem, the system constructs $N(N-1)/2$ binary classifiers during its training phase like other one-against-one methods. Among those binary hyperplanes having been constructed, the system searches for the hyperplane at the most balanced position among all candidate classes, called *balanced dichotomization classifier* that separates the data classes into half-and-half on each side. Using a balanced dichotomization classifier can thus remove half of the candidate classes during each evaluation for the correct class, that is a higher number of elimination compared to other methods, such as the DDAG, the ADAG, which eliminate only one class using an ordinary binary classifier. As a result, the technique can optimally reduce the number of decisions in order to derive an answer to $\lceil \log_2 N \rceil$ times, rather than $N-1$ times in the DDAG and the ADAG.

The basic idea of the primary SVM classification is to find the optimal hyperplane separating the two classes of data as illustrated in Figure 2 (a). The hyperplane maximizes the margin between the data in class 1 and class 2. However, the hyperplane in Figure 2 (a) is not a balanced dichotomization classifier because when considering the positions of all candidate classes, it is not at the most balanced position as depicted in

C. Zhang, H.W. Guesgen, W.K. Yeap (Eds.): PRICAI 2004, LNAI 3157, pp. 973–974, 2004.

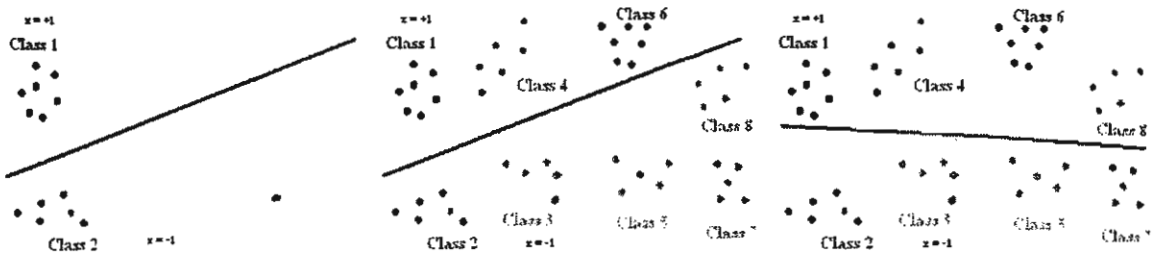


Fig. 2. (a) The optimal hyperplane for classes 1 and 2, (b) the hyperplane is not a balanced dichotomization classifier when considering other classes, and (c) an optimal balanced hyperplane.

Figure 2 (b). The hyperplane shown in Figure 2 (c) is an example of the balanced dichotomization hyperplane. It is posed at the optimal balanced position that separates candidate classes into half-and-half on each side.

Since Balanced Dichotomization requires considering positions of all candidate classes to arrive at a balanced hyperplane, there may be cases where a hyperplane in consideration is posed in between data of certain classes. To deal with these cases, two parameters are introduced in our approach, i.e. the optimal range of generalization error and the optimal pruning percentage. *Pruning percentage* is used as the threshold for the removal of data on either side of the hyperplane in consideration. The strategy of pruning is to achieve the balanced dichotomization that provides the minimum number of evaluations for the correct class while maintaining the accuracy within the range of generalization performance [1]. If the ratio between data of a class on one side and all data of the class is less than pruning percentage, the data on that side will be ignored. Moreover, using *the optimal range of generalization error*, only hyperplanes with the generalization error within the range will be considered.

We evaluate the performance of our method on sèveral datasets from the UCI Repository of machine learning databases [2]: Glass, Satimage, Segment, Shuttle, Vowel, Soybean, Letter, and Isolet. The experimental results show that Balanced Dichotomization runs faster and maintains accuracy comparable to Max Wins and better than the ADAG and the DDAG methods.

References

1. Bartlett, P. L. and Shawe-Taylor, J. (1999) *Generalization performance of support vector machines and other pattern classifiers*, In B.Schölkopf, C. Burges, & A. Smola (Eds.), Advances in Kernel Methods – Support Vector Learning, pp. 43-54, MIT Press, USA.
2. Blake, C., Keogh, E., and Merz, C. (1998) *UCI Repository of Machine Learning Databases*, Department of Information and Computer Science, University of California, Irvine. http://www.ics.uci.edu/~mlearn/MLSummary.html
3. Friedman, J. H. (1996) *Another Approach to Polychotomous classification*, Technical report, Department of Statistics, Stanford University.
4. Kijsirikul, B., Ussivakul, N., and Meknavin, S. (2002) *Adaptive Directed Acyclic Graphs for Multiclass Classification*, The Seventh Pacific Rim International Conference on Artificial Intelligence.
5. Platt, J., Cristianini, N. and Shawe-Taylor, J. (2000) *Large Margin DAGs for Multiclass Classification*, Advances in Neural Information Processing Systems, MIT Press, 12, 547-553.

First-Order Logical Neural Networks

Thanupol Lerdlamnaochai and Boonserm Kijsirikul
Department of Computer Engineering, Chulalongkorn University,
Pathumwan, Bangkok, 10330, Thailand
g46tll@cp.eng.chula.ac.th, boonserm.k@chula.ac.th

Abstract

Inductive Logic Programming (ILP) is a well known machine learning technique in learning concepts from relational data. Nevertheless, ILP systems are not robust enough to noisy or unseen data in real world domains. Furthermore, in multi-class problems, if the example is not matched with any learned rules, it cannot be classified. This paper presents a novel hybrid learning method to alleviate this restriction by enabling Neural Networks to handle first-order logic programs directly. The proposed method, called First-Order Logical Neural Network (FOLNN), is based on feedforward neural networks and integrates inductive learning from examples and background knowledge. We also propose a method for determining the appropriate variable substitution in FOLNN learning by using Multiple-Instance Learning (MIL). In the experiments, the proposed method has been evaluated on two first-order learning problems, i.e., the Finite Element Mesh Design and Mutagenesis and compared with the state-of-the-art, the PROGOL system. The experimental results show that the proposed method performs better than PROGOL.

1. Introduction

Inductive Logic Programming (ILP) [1, 2] is only one of machine learning techniques which adapts the first-order logical concepts for hypothesis learning. The advantages of ILP are the ability of employing background knowledge and the expressive representation of first-order logic. However, first-order rules learned by ILP have the restriction to handle imperfect data in real-world domains such as noisy unseen data. This problem noticeably occurs especially in multi-class classification. In multi-class classification, if an example is not covered any learned rule, it could not be classified. The simple solution is assigning the majority class recorded from training examples to the unable labeled test data [3]. Also, there is more efficient method to solve this problem by using the concept of intelligent hybrid systems [4].

Artificial Neural Networks (ANNs) [5] claim to avoid the restrictions of symbolic rule-based systems described above. Neural networks contain the ability of processing inconsistent and noisy data. Moreover, they compute the most reasonable output for each input. Neural networks, because of their potential for noise tolerance and multi-class classification, offer an attractiveness for combining with symbolic components. Although the ability of neural networks could alleviate the problem in symbolic rule-based systems, learned hypothesis from neural networks is not available in a form that is legible for humans. Therefore neural networks significantly require an interpretation by rule-based systems [4]. Several works show that the integration between robust neural networks and symbolic knowledge representation can improve classification accuracy such as Towell and Shavlik's KBANN [6], Mahoney and Mooney's RAPTURE [7], the works proposed by Rajesh Parekh and Vasant Honavar [8] and d'Avila Garcez et al. [9]. Nevertheless, these researches have been restricted to propositional theory refinement. Some models have been proposed for first-order theory. SHRUTI [10] employed a model making a restricted form of unification—actually this system only propagates bindings—. The work proposed by Botta et al. [11] created a network consisting of restricted form of learning first-order logic. Kijsirikul et al. [12] proposed a feature generation method and a partial matching technique for first-order logic but their method still uses an ILP system in its first-step learning and cannot select the appropriate values to substitute in variables.

In this paper, we are interested in direct learning of first-order logic programs by neural networks, called First-Order Logical Neural Network (FOLNN). FOLNN is a neural-symbolic learning system based on the feedforward neural network that integrates

0-7695-2291-2/05 $20.00 © 2005 IEEE

inductive learning from examples and background knowledge. We also propose the method that makes use of Multiple-Instance Learning (MIL) [13, 14] for determining the variable substitution in our model. Our proposed method has been evaluated on two standard first-order learning datasets i.e., the Finite Element Mesh Design [15] and Mutagenesis [16]. The results show that the proposed method provides more accurate result than the original ILP system.

The rest of this paper is organized as follows. Section 2 outlines the processes of first-order learning by FOLNN, splitting into three subsections. The experimental results on first-order problems are shown in Section 3. Finally the conclusions are given in Section 4.

2. First-Order Logical Neural Network (FOLNN)

Commonly the main reason for integrating robust neural networks and symbolic components is to reduce the weakness of the rule-based system. Combining these two techniques together is normally known as neural-symbolic learning system [9]. Our proposed method, FOLNN is also this type of learning system. FOLNN structure is based on the feedforward neural network and can receive examples and background knowledge in form of first-order logic programs as the inputs. FOLNN weight adaptation is based on the Backpropagation (BP) algorithm [17].

The following subsections explain the FOLNN algorithm composed of creating an initial network, feeding examples to the network and training the network.

2.1. Creating an initial network

In this subsection, we present the first step of the FOLNN algorithm, creating an initial network from background knowledge. A three layers feedforward network, composed of one input layer, one output layer and one hidden layer [18], is employed for FOLNN structure. We define the functionality of each layer as follows.

- Input layer: Input layer is the first layer that receives input data, computes received data, and then transmits processed data to the hidden layer. This layer represents the literals for describing the target rule. The number of units in this layer depends on the number of predicates in background knowledge. One predicate is represented by one unit in the input layer if that predicate contains only arity one. Otherwise, the number of units for a predicate equals to the number of all possible combinations of variables of that predicate.
- Hidden layer: This layer connects between the input layer and the output layer. The hidden layer helps the network to learn the complex classification. The number of units in this layer depends on the complication of the learning concept. The number of units in the hidden layer is determined from the experiments.
- Output layer: This layer is the last layer of the network producing the output for the classification. The target concept is represented in this layer so that the number of units in the output layer equals to the number of concepts to be learned or the number of classes.

An initial network is created by using the above definition. To illustrate the construction of the network, consider the task of learning rules for classifying the rich person (*rich(x)*). Background knowledge, the positive and negative examples are given as follows.

Figure 1. Inputs for learning the concept *rich(x)*.

As shown in Figure 1, background knowledge contains three predicates with arity one which are *genius(x)*, *diligent(x)* and *strong(x)* and one predicate having arity two which is *parent(x,y)*. Each predicate of arity one is represented by one unit in the input layer, so three units are created. Predicate *parent* is represented by two input units for literals *parent(x,y)* and *parent(y,x)*. Furthermore, the output layer, because of only one target concept *(rich(x))*, has only one unit. Therefore in this case, the constructed network will have five input units and one output unit. The created network from the inputs in Figure 1 is shown in Figure 2. In addition, all network weights are initialized to small random numbers.

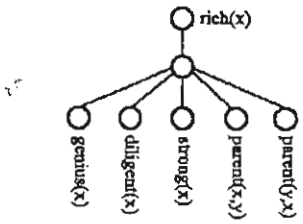


Figure 2. The created network with one hidden unit.

The completely constructed network then receives examples for refining the network. The process for feeding examples to the network is described in the next subsection.

2.2. Feeding examples to the network

In general, neural networks receive inputs in real value form. However, inputs of the ILP system (background knowledge and examples) are in logical form. So we change the logical inputs to the form that can be learned by neural networks. The examples are fed to the network one by one and independently transformed to the network input for each unit. The value for each unit is defined as follows.

$$X_{ij} = \begin{cases} 1 & \text{if } L_i\theta_j \text{ is true in background knowledge} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where X_{ij}, L_i, and θ_j are input value for input unit i when feeding example j, literal represented by input unit i, and variable binding with constants in example j, respectively.

The input value for an input unit will be 1 if there exists substitution that makes the truth value of the literal true in background knowledge. Otherwise the input value for that unit is 0. In addition, the target value for output unit is defined as follows.

$$T_{kj} = \begin{cases} 1 & \text{if } L_k\theta_j \text{ is positive example} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where T_{kj} and L_k are target value for output unit k when feeding example j, and literal represented by output unit k, respectively.

For instance, with the same inputs in Figure 1, if the fed example is *rich(Alan)* then the first three units i.e., *genius(x)*, *diligent(x)* and *strong(x)*, receive 0,1 and 0 as their inputs respectively, because literal *diligent(Alan)* is true in background knowledge, while literals *genius(Alan)* and *strong(Alan)* are false. The target value for the output unit is 1 because *rich(Alan)* is a positive example. However, the input value for literals *parent(x,y)* and *parent(y,x)* cannot be easily determined since there are many possible constants that can be mapped to relational variables. For unit *parent(y,x)*, the variable x is certainly replaced by *Alan*. However, it is quite ambiguous by which term (*Alan*, *Bob* or *Chris*) variable y should be replaced. If we select *Bob* for substitution, the truth value for this input unit will be 1. The other substitutions will give 0 for this unit (see Table 1). The truth value for *parent(x,y)* is 0 for any substitution. From the above example, the input value for unit *parent(y,x)* is not certain and cannot be easily determined for network training. This problem may occur when background knowledge contains relational data and the learner cannot determine the appropriate value for the variable substitution.

Table 1. Input value of unit *parent(y,x)* for each constant replacement.

Unit *parent (y,x)*	Input value
Replace x by *Alan*, and y by *Alan*	0
Replace x by *Alan*, and y by *Bob*	1
Replace x by *Alan*, and y by *Chris*	0

To solve this problem, we use the power of Multiple-Instance Learning (MIL) to provide input data for our network. In MIL framework [13, 14], the training set is composed of a set of bags, each of which is a collection of different number of instances. A bag is labeled as a negative bag if all the instances in it are negative. On the other hand, if a bag contains at least one positive instance then it is labeled as a positive bag. With this concept, we define FOLNN training data as a set of training examples $\{B_1, B_2, \ldots B_n\}$, where n is the number of examples including positive and negative ones. A bag is labeled as a positive bag if an example is positive, and negative otherwise (in multi-class classification, all bags are labeled as positive of their classes). The positive bag is given 1 as its target value and the negative bag is assigned 0, as defined in Equation (2). Each bag contains m_i instances $\{B_{i1}, B_{i2}, \ldots, B_{imi}\}$ where B_{ij} is one possible binding (substitution). This is a very important key because now we can use all cases of variable substitutions as one bag for learning; therefore the appropriate value selection would not be a problem. Consider an example of positive bag *rich(Alan)* as input data (see Table 2).

Table 2. Transformation of example *rich(Alan)* into input data of FOLNN

Positive bag of example *rich(Alan)*	*genius(x)*	*diligent(x)*	*strong(x)*	*parent(x,y)*	*parent(y,x)*
Replace x by *Alan*, and y by *Alan*	0	1	0	0	0
Replace x by *Alan*, and y by *Bob*	0	1	0	0	1
Replace x by *Alan*, and y by *Chris*	0	1	0	0	0

As shown in Table 2, the bag *rich(Alan)* has 3 instances each of which is one case of substitution. Also, positive bag *rich(Bob)* and negative bag *rich(Chris)* have 3 instances as same as positive bag *rich(Alan)*. For training, these 3 bags are fed to the network one by one and the network weights are adapted by the Backpropagation (BP) algorithm for MIL [19] as described in the next subsection.

2.3. Training the network

To train the network, training bags are fed to the network for adapting network weights. Weight adaptation is based on the BP algorithm and the activation function is Sigmoid function. Suppose the network has p input units, o output units, and one hidden layer. The global error function *(E)* of the network is defined as follows.

$$E = \sum_{i=1}^{n} E_i \qquad (3)$$

where E_i is the error on bag i. E_i is defined according to the type of the bag i as:

$$E_i = \begin{cases} \min\limits_{1 \le j \le m_i} \sum\limits_{k=1}^{o} E_{ijk} & \text{if } B_i = + \\ \max\limits_{1 \le j \le m_i} \sum\limits_{k=1}^{o} E_{ijk} & \text{if } B_i = - \end{cases} \qquad (4)$$

$$E_{ijk} = \begin{cases} 0 & \text{if } (B_i = +) \text{ and } (0.5 \le o_{ijk}), l_{ik} = 1 \\ 0 & \text{if } (B_i = -) \text{ and } (o_{ijk} < 0.5), \text{for all } k \\ \frac{1}{2}(l_{ik} - o_{ijk})^2 & \text{otherwise} \end{cases} \qquad (5)$$

where

- E_{ijk} is error of output unit k on instance j in bag example i
- B_i=+ is positive bag example
- B_i=– is negative bag example
- o_{ijk} is actual output of output unit k from bag example i, instance j, and
- l_{ik} is target output of output unit k from bag example i

With the defined error function above, the error BP algorithm is simply adapted for training FOLNN. In each training epoch, the training bags are fed to the network one by one. Then the error E_{ijk} is computed according to Equation (5). For a positive bag B_i, if E_{ijk} is 0 then all the rest of instances of this bag are disregarded, and the weights are not changed for this epoch. Otherwise the process continues and when all the instance of B_i are fed, E_i is computed by using Equation (4) and the weights in the network are changed according to the weight update rule of BP [17]. Then the next bag for training is fed to the network and the training process is repeated until the number of training iterations increases to some predefined threshold or the global error E in Equation (3) is decreased to some predefined threshold. After having been trained, the network can be used to classify unseen data.

3. Results

In the previous section, the three steps of learning FOLNN algorithm were described. In this section, we evaluate FOLNN by performing experiments on the finite element mesh design and the mutagenesis datasets, the well-known ILP problems. We also compare the results obtained by FOLNN with those obtained by an ILP system.

3.1. Datasets

3.1.1. Finite Element Mesh Design. The dataset for the finite element mesh design [15] consists of 5 structures and has 13 classes (13 possible number of partitions for an edge in a structure). Additionally, there are 278 examples each of which has the form *mesh(Edge,Number_of_elements)* where Edge is an edge label (unique for each edge) and *Number_of_elements* indicates the number of partitions. The background knowledge contains relations describing the types of an edge (e.g. *circuit, short*), boundary conditions (e.g. *free, fixed*), loadings (e.g. *not_loaded, one_side_loaded*) and the relations describing the structure of the object (e.g. *neighbour, opposite*). The goal of finite element mesh design is to learn general rules describing how many elements should be used to model each edge of a structure.

3.1.2. Mutagenesis. The dataset for the mutagenesis [16] consists of 188 molecules, of which 125 are mutagenic (active) and 63 are non-mutagenic (inactive). A molecule is described by listing its atoms as *atom(AtomID,Element,Type,Charge)* and the bonds between atoms as *bond(Atom1,Atom2,BondType)*. This problem is a two-class learning problem for predicting the mutagenicity of the molecules, whether a molecule is active or inactive in terms of mutagenicity.

3.2. Experiments

For the finite element mesh design dataset, we create the network containing 130 units in the input

layer (determined by predicates in background knowledge), 13 output units (as the number of classes) and one hidden layer with 80 hidden units (determined by the experiment). For the mutagenesis dataset, the constructed network has 235 input units, 100 hidden units and 2 output units. The weights of two networks are randomly initialized and then adapted by using the BP algorithm with sigmoid activation function. We performed three-fold cross validation [20] on each dataset. The dataset is partitioned into three roughly equal-sized subsets with roughly same proportion of each class as that of the original dataset. Each subset is used as a test set once, and the remaining subsets are used as the training set. The final result is the average result over three-fold data. For each fold, of both datasets, we trained FOLNN with learning rate 0.0001 and momentum 0.97.

Table 3. The percent accuracies of FOLNN and PROGOL on first-order datasets; FEM – Finite Element Mesh Design, MUTA – Mutagenesis.

Dataset	FOLNN	PROGOL
FEM	59.18	57.80
MUTA	88.27	84.58

The average results over three-fold data on FEM and MUTA datasets are summarized in Table 3. PROGOL [21], the state-of-the-art ILP system, has been used to compare the performance with our proposed method, FOLNN. The experimental results show that the accuracies of our proposed method, FOLNN are better than PROGOL in both datasets. The better results are according to the weakness of learned rules generated by PROGOL.

In addition to the results on the original dataset, to see how well our learner handles noisy data, we also evaluate FOLNN on noisy domain. The mutagenesis dataset is selected for this task. Using the three-fold data of the mutagenesis dataset in the last experiment, 10% and 15% class noise is randomly added into the training set, and no noise is added into the test set. In our case, adding x% of noise means that the class value is replaced with the wrong value in x out of 100 data by random selection. The accuracies of PROGOL and FOLNN on noisy data are shown in Table 4.

Table 4. Performance comparison on the noisy mutagenesis dataset.

Noise level in dataset	PROGOL 0% noise setting	PROGOL 10% noise setting	PROGOL 15% noise setting	FOLNN
10%	64.23	69.72	71.29	84.01
15%	60.56	61.54	65.31	81.28

Since PROGOL has an ability to handle noise in data as its option, "x% noise setting" in the table specifies that noise option of PROGOL is set to x%. As can be seen in the Table 4, our proposed algorithm still provides average accuracies higher than PROGOL. When 10% and 15% noise is added into the dataset, the PROGOL performance significantly drops due to its sensitivity to noise which is the main disadvantage of first-order rules directly induced by the ILP system. However, accuracy of our method decreased much slower and is much higher than that of PROGOL. FOLNN, because of the ability of noise tolerance by combining with neural networks, is more robust against noise than the original first-order rules. FOLNN prevents overfitting noisy data by employing neural networks to give higher weights to important features and give less attention to unimportant ones.

4. Conclusions

Learning first-order logic programs by using neural networks is still an open problem. This paper presents a novel hybrid connectionist symbolic system based on the feedforward neural network that incorporates inductive learning from examples and background knowledge, called FOLNN (First-Order Logical Neural Network). FOLNN alleviates the problem of first-order rules induced by the ILP system which are not robust enough to noisy or unseen data. The prominent advantage of FOLNN is that it can learn from inputs provided in form of first-order logic programs directly. Other learners cannot directly learn this kind of programs because they cannot select the appropriate values for variable substitution, but our method can solve this problem by applying the MIL concept to provide certain input data from first-order logic input.

The experimental results show that FOLNN presents the ability of noise tolerance and produces the better performance than PROGOL. This is because of the ability of neural networks that can select important attributes and then gives higher weights to these attributes and vice versa.

Although our main objective is to learn the first-order logic, FOLNN can be applied to other tasks such as learning from propositional datasets containing missing values in some attributes.

One interesting issue is knowledge extraction. Knowledge extraction from a trained network is one phase of the neural-symbolic learning system [9] and is of significant interest in data mining and knowledge discovery applications such as medical diagnosis. However, this phase is not included in this work and we have not yet explored rule extraction from trained networks. Nevertheless, we surmise that many researches [9, 22-24] can be adapted to extract rules from our networks.

5. Acknowledgement

This work was supported by the Thailand Research Fund.

6. References

[1] N. Lavrac and S. Dzeroski, *Inductive Logic Programming Techniques and Applications*, Ellis Horwood, New York, 1994.

[2] S.-H. Nienhuys-Cheng and R. d. Wolf, *Foundation of Inductive Logic Programming*, Springer-Verlag, New York, 1997.

[3] S. Dzeroski, S. Schulze-Kremer, K. R. Heidtke, K. Siems, and D. Wettschereck, "Applying ILP to Diterpene Structure Elucidation from 13C NMR Spectra", *Proceedings of the Sixth International Workshop on Inductive Logic Programming*, 1996.

[4] S. Wermter and R. Sun, "An Overview of Hybrid Neural Systems," *Hybrid Neural Systems, number 1778 in Lecture Notes in Artificial Intelligence*, S. Wermter and R. Sun, Eds., Springer, 2000, pp. 1-13.

[5] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[6] G. G. Towell and J. W. Shavlik, "Knowledge-based artificial neural networks", *Artificial Intelligence*, vol. 70(1-2), 1994, pp. 119-165.

[7] J. J. Mahoney and R. J. Mooney, "Combining connectionist and symbolic learning to refine certainty-factor rule-bases", *Connection Science*, vol. 5, 1993, pp. 339-364.

[8] R. Parekh and V. Honavar, "Constructive Theory Refinement in Knowledge Based Neural Networks", *Proceedings of the International Joint Conference on Neural Networks*, Anchorage, Alaska, 1998.

[9] A. S. d. A. Garcez, K. B. Broda, and D. M. Gabbay, *Neural-Symbolic Learning Systems*, Springer-Verlag, 2002.

[10] L. Shastri and V. Ajjanagadde, "From simple associations to systematic reasoning", *Behavioral and Brain Sciences*, vol. 16, 1993, pp. 417-494.

[11] M. Botta, A. Giordana, and R. Piola, "FONN: Combining First Order Logic with Connectionist Learning", *Proceedings of the 14th International Conference on Machine Learning*, Nashville, TN, 1997.

[12] B. Kijsirikul, S. Sinthupinyo, and K. Chongkasemwongse, "Approximate Match of Rules Using Backpropagation Neural Networks", *Machine Learning Journal*, vol. 44, pp. 273-299, 2001.

[13] Y. Chevaleyre and J.-D. Zucker, "A Framework for Learning Rules from Multiple Instance Data", *12th European Conference on Machine Learning*, Freiburg, Germany, 2001.

[14] X. Huang, S.-C. Chen, and M.-L. Shyu, "An Open Multiple Instance Learning Framework and Its Application in Drug Activity Prediction Problems", *Proceedings of the Third IEEE Symposium on BioInformatics and BioEngineering (BIBE'03)*. Bethesda, Maryland, 2003.

[15] B. Dolsak and S. Muggleton, "The Application of Inductive Logic Programming to Finite Element Mesh Design", *Inductive Logic Programming*, S. Muggleton. Ed., Academic Press, 1992, pp. 453--472.

[16] A. Srinivasan, S. H. Muggleton, M. J. E. Sternberg, and R. D. King, "Theories for mutagenicity: a study in first-order and feature-based induction", *Artificial Intelligence*, vol. 85, Elsevier Science Publishers Ltd., 1996, pp. 277-299.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation", *Parallel Distributed Processing*, vol. 1, D. E. Rumelhart and J. L. McClelland, Eds., The MIT Press, Cambridge, MA, 1986.

[18] S. Holldobler and Y. Kalinke, "Towards a massively parallel computational model for logic programming", *Proceedings of the ECAI94 Workshop on Combining Symbolic and Connectionist Processing, ECAI94*, 1994, pp. 68-77.

[19] Z.-H. Zhou and M.-L. Zhang, "Neural Network for Multi-Instance Learning", *Proceedings of the International Conference on Intelligent Information Technology*, Beijing, China, 2002.

[20] T. M. Mitchell, *Machine Learning*, The McGraw-Hill Companies Inc, New York, 1997.

[21] S. Roberts, *An Introduction to Progol. Technical Manual*, University of York, 1997.

[22] R. Andrew, J. Diederich, and A. B. Tickle, "Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks", *Knowledge-Based Systems*, vol. 8, 1995, pp. 373-389.

[23] M. W. Craven, "Extracting Comprehensible Models from Trained Neural Networks", *Department of Computer Science*: University of Wisconsin-Madison, 1996.

[24] G. G. Towell and J. W. Shavlik, "The Extraction of Refined Rules from Knowledge-Based Neural Networks", *Machine Learning Journal*, vol. 13, pp. 71-101, 1993.

การเรียนรู้เน็ตเวิร์กเบย์ลำดับที่หนึ่ง
Learning First-order Bayesian Networks

รัฐฉัตร ฉัตรพัฒนศิริ บุญเสริม กิจศิริกุล

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

จุฬาลงกรณ์มหาวิทยาลัย ถ.พญาไท ปทุมวัน กรุงเทพ 10330

Email:Ratthachat.C@Student.chula.ac.th, Boonserm.K@Chula.ac.th

บทคัดย่อ

ตัวจำแนกประเภทส่วนใหญ่มักมีข้อจำกัดที่สำคัญมากสองข้อคือ ตัวจำแนกประเภทมักแบ่งข้อมูลชนิดที่ข้อมูลแต่ละหน่วยขึ้นต่อกันได้ไม่ดี และข้อจำกัดข้อที่สองคือตัวจำแนกประเภทมักไม่สามารถแบ่งข้อมูลที่มีสัญญาณรบกวนได้อย่างมีประสิทธิภาพ "เน็ตเวิร์กเบย์ลำดับที่หนึ่ง" เป็นตัวจำแนกประเภทข้อมูลที่สามารถแก้ไขข้อจำกัดสองประการนี้ได้ อย่างไรก็ตามเนื่องจากเน็ตเวิร์กเบย์ลำดับที่หนึ่งมีความซับซ้อนมาก ระบบการเรียนรู้เพื่อสร้างเน็ตเวิร์กเบย์ลำดับที่หนึ่งจากข้อมูลดิบจึงพัฒนาได้ยากยิ่ง งานวิจัยชิ้นนี้เสนอระบบต้นแบบในการเรียนรู้เน็ตเวิร์กเบย์ลำดับที่หนึ่งจากข้อมูลดิบโดยการประยุกต์นำการโปรแกรมตรรกะเชิงอุปนัย และระบบเรียนรู้เน็ตเวิร์กเบย์เข้าไว้ด้วยกัน

Abstract

Most classifiers often struggle with two main problems when (1) there are some relations among the input data and, (2) the input data is imperfect or has some noise. A first-order bayesian network (FOBN) is a powerful classifier that can cope with those two problems. Because of its complication, however, it is very difficult to develop an efficient algorithm for constructing FOBNs. This paper proposes a new framework for constructing FOBNs by combining two algorithms, namely, inductive logic programming and a bayesian network learning algorithm.

1. บทนำ

ตัวจำแนกประเภท (classifier) ส่วนใหญ่มักมีข้อจำกัดที่สำคัญมากสองข้อ ข้อจำกัดแรกคือตัวจำแนกประเภทมักแบ่งข้อมูลชนิดที่ข้อมูลแต่ละหน่วยขึ้นต่อกันได้ไม่ดี และข้อจำกัดข้อที่สองคือตัวจำแนกประเภทมักไม่สามารถแบ่งข้อมูลที่มีสัญญาณรบกวนได้อย่างมีประสิทธิภาพ ในช่วงสิบปีที่ผ่านมานี้ได้มีการวิจัยอย่างจริงจังในการพัฒนาตัวจำแนกประเภทข้อมูลที่สามารถเอาชนะข้อจำกัดทั้งสองประเภทดังกล่าวซึ่งได้แก่*เน็ตเวิร์กเบย์ลำดับที่หนึ่ง (First-order Bayesian Network : FOBN)* [6,7] โดย FOBN ได้รวมข้อดีของตัวจำแนกประเภทข้อมูล 2 ชนิด คือ*ตรรกศาสตร์ลำดับที่หนึ่ง (First-order Logic : FOL)* [11] และตัวจำแนกประเภทข้อมูลชนิด*เน็ตเวิร์กเบย์ (Bayesian Network : BN)* [11,13] เพื่อความเข้าใจในขีดจำกัดของตัวจำแนกประเภท พิจารณาข้อมูลในตารางที่ 1

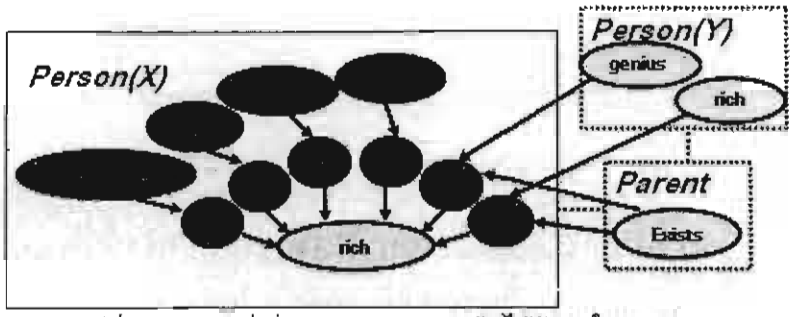
ตารางที่ 1 a. ฐานข้อมูลความรวยและคุณสมบัติของบุคคล

case	rich	genius	artist	diligent	save_money
b	+	1	0	0	0
c	+	0	1	0	1
d	+	1	1	1	1
e	+	0	0	0	0
f	-	0	1	0	0
g	-	0	0	1	0

b. ฐานข้อมูลพ่อแม่และลูก (สัมพันธ์กับตาราง a.)

parent	child
b	e
e	f
f	g

เราสามารถสร้างตัวจำแนกประเภทในรูปกฎตรรกศาสตร์ประพจน์(Propositional Logic) [11] ได้สองข้อคือ "`rich ← genius.`" และ "`rich ← artist,save_money.`" ซึ่งกฎสองข้อนี้อธิบายกรณีของนาย `e` ไม่ได้ (ในความเป็นจริง `e` รวยเพราะพ่อหรือแม่คือ `b` รวย) เนื่องจากตรรกศาสตร์ประพจน์มีความกำกวมในการแสดงกฎที่มีความสัมพันธ์ระหว่างข้อมูล เช่นกฎ "`rich ← rich.`" ไม่สามารถบอกได้ว่าความรวยของบุคคลใดส่งผลต่อบุคคลใด ในงานวิจัยนี้เราเรียกข้อมูลดิบประเภทที่ข้อมูลในแต่ละหน่วยสามารถมีผลกระทบต่อหน่วยอื่นได้ว่า*ข้อมูลเชิงสัมพันธ์ (relational data)* การมีความสัมพันธ์กันระหว่างข้อมูลทำให้การเรียนรู้ทำได้ยาก



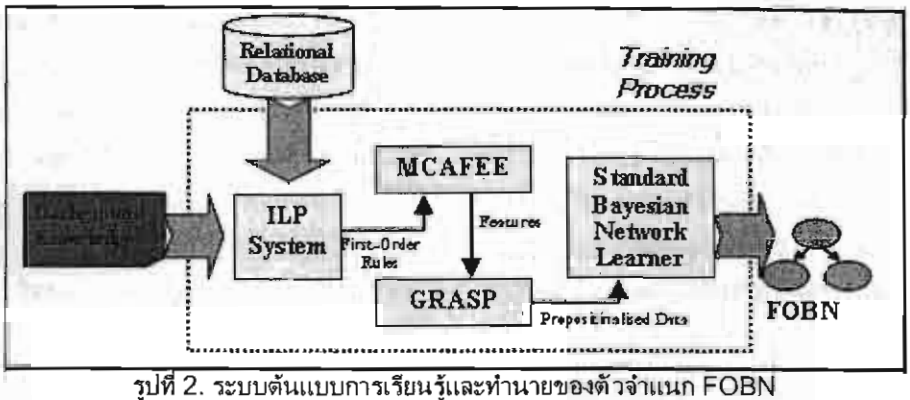
รูปที่ 1. FOBN ที่เพิ่มความสามารถจากต้นไม้ตัดสินใจและ FOL

ยิ่งขึ้นดังกรณีนาย e อย่างไรก็ตาม FOL ซึ่งถูกพัฒนาจากตรรกศาสตร์ประพจน์สามารถเขียนกฎสำหรับข้อมูลที่มีความสัมพันธ์ต่อกันได้อย่างชัดเจนโดยมีการนำตัวแปรทางตรรกศาสตร์มาใช้ดังนี้ "`rich(X) ← artist(X), save_money(X).`", "`rich(X) ← genius(X).`" และ "`rich(X) ← parent(Y,X),rich(Y),genius(Y).`" โดยในกรณีนี้ตัวแปรแต่ละตัวแสดงถึงบุคคลและกฎข้อสามครอบคลุมกรณีนาย e เนื่องจากเราสามารถสร้างกฎตรรกศาสตร์ประพจน์ด้วย FOL ได้เสมอทำให้การใช้ FOL เป็นระบบแบ่งประเภทข้อมูลช่วยลดขีดจำกัดของตัวจำแนกประเภทได้อย่างมาก สำหรับวิธีมาตรฐานในการเรียนรู้ FOL จากข้อมูลดิบนั้นถูกเรียกว่า*การโปรแกรมตรรกะเชิงอุปนัย (Inductive Logic Programming : ILP)* [10,11,12] อย่างไรก็ตามระบบ ILP ยังไม่สามารถจัดการข้อมูลที่ไม่สามารถแบ่งกลุ่มได้แน่นอนได้ดีนัก เมื่อจำเป็นต้องเรียนรู้กฎจากข้อมูลดิบที่ไม่ดีเหล่านี้ ระบบ ILP จะเผชิญกับปัญหาโอเวอร์ฟิต (overfitting problem) [10,11] ซึ่งจะทำให้ FOL ที่เรียนรู้ได้มีลักษณะเฉพาะเจาะจง (specific) กับข้อมูลอินพุตมากเกินไป (ทำงานได้ถูกต้องเฉพาะข้อมูลอินพุตเท่านั้น) พิจารณา FOL ทั้งสามข้อในตัวอย่างที่ผ่านมาพบว่าในบางกรณีกฎเหล่านี้มีความเฉพาะเจาะจงมากเกินไปเช่นกัน ตัวอย่างเช่นถ้าเพิ่มนาย a ลงไปในตารางที่ 1 โดยกำหนดคุณสมบัติให้นาย a ดังต่อไปนี้

`genius(a). save_money(a). parent(e,a).`

จะเห็นได้ว่าคุณสมบัติต่าง ๆ ของนาย a ไม่ตรงกับกฎข้อใดเลยทำให้เราสรุปจากกฎได้ว่านาย a เป็นบุคคลที่ไม่รวย อย่างไรก็ตามสังเกตว่าการสรุปแบบนี้เป็นการสรุปที่ไม่มีประสิทธิภาพนักเนื่องจากคุณสมบัติต่าง ๆ ของนาย a *ตรงบางส่วน (partially match) กับกฎ FOL ในตัวอย่างที่แล้วทั้งสามข้อ* คุณสมบัติการตรงบางส่วนของกฎนี้เองเป็นที่มาของการใช้ FOBN เป็นระบบจำแนกประเภทข้อมูล โดยเมื่อกำหนดคุณสมบัติต่าง ๆ ของนาย a ให้กับ FOBN แล้ว FOBN จะสามารถคำนวณหาค่าความจะเป็นภายหลัง (posterior probability) ที่นาย a จะรวยว่ามากน้อยเพียงใดด้วยเทคนิคการอนุมาน (inference) [13] ได้อย่างมีประสิทธิภาพ ทำให้การใช้ FOBN เป็นระบบจำแนกประเภทข้อมูลมีความยืดหยุ่นเป็นอย่างมากเนื่องจากรองรับทั้งคุณสมบัติความไม่แน่นอนของข้อมูลได้ด้วยการใช้ทฤษฎีความน่าจะเป็น (แทนที่จะสรุปว่า *ใช่* หรือ *ไม่ใช่* ซึ่งเป็นวิธีที่ระบบจำแนกข้อมูลทั่วไปใช้) นอกจากนี้ FOBN ยังมีคุณสมบัติรองรับความสัมพันธ์ระหว่างข้อมูลเช่นเดียวกับ FOL อีกด้วย รูปที่ 1 แสดง FOBN ที่ใช้แก้ปัญหาความรวยในตัวอย่างนี้

FOBN ประกอบไปด้วยสองส่วนหลักเช่นเดียวกับ BN [6,7,13] คือส่วน*โครงสร้าง (structure)* โดยหมายถึงโนดต่าง ๆ และเส้นเชื่อมแบบมีทิศทาง (directed link) ทั้งหมดที่เชื่อมโนดแต่ละโนดเข้าไว้ด้วยกัน โดยโนดแต่ละโนดแทนคุณสมบัติหรือคุณลักษณะ (attribute) การมีเส้นเชื่อมจากโนด A ไปยังโนด B หมายถึงโนด A ส่งผลกระทบโดยตรงต่อโนด B ส่วนประกอบส่วนที่สองของ FOBN คือ*ตารางความน่าจะเป็นมีเงื่อนไข (conditional probability table : CPT)* โดยใช้เพื่อคำนวณความน่าจะเป็นภายหลังเพื่อทำนายคุณสมบัติที่ต้องการ โดยทั่วไปการเรียนรู้ FOBN จึงประกอบไปด้วยสองส่วนหลักคือเรียนรู้โครงสร้างและ CPT อย่างไรก็ตามการเรียนรู้ FOBN ไม่สามารถทำได้โดยง่ายนักเนื่องจาก [1] และ [2] ได้พิสูจน์ว่าเพียงแค่การเรียนรู้ FOL หรือ BN เพียงลำพังก็ยังมีความซับซ้อนของอัลกอริทึมที่ใช้ในการเรียนรู้เป็นแบบ NP งานวิจัยนี้จึงได้เสนอแนวคิดใหม่โดยการนำระบบ ILP และ*ระบบเรียนรู้ BN (Bayesian Network Learner : BNL)* [3,13] สองระบบมาทำงานต่อเนื่องกันเพื่อลดความซับซ้อนของระบบเรียนรู้ FOBN ในการใช้ BNL เพื่อเรียนรู้ FOBN นั้น BNL ต้องการข้อมูลอินพุตประเภทฐานข้อมูลตารางเดี่ยวดังเช่น ตารางที่ 1 a. (ไม่รวมตารางที่ 1 b.) นั่นคืออินพุตประกอบไปด้วยตัวอย่าง *n* ตัวโดยตัวอย่างแต่ละตัวประกอบไปด้วยค่าของคุณสมบัติ *m* ค่า โดยตัวอย่างหนึ่งตัวแทนด้วยแถวนอนหนึ่งแถวในตารางอินพุต และหนึ่งหลักหรือแถวตั้ง (column) แทนคุณสมบัติหนึ่งตัว จากนั้นในขั้นตอน



รูปที่ 2. ระบบต้นแบบการเรียนรู้และทำนายของตัวจำแนก FOBN

การเรียนรู้ BNL จะสร้างเส้นเชื่อมที่เหมาะสมรวมทั้ง CPT ของโนดเหล่านั้นให้ ดังนั้นถ้าเราต้องการใช้ BNL เรียน FOBN ในลักษณะเดียวกับการเรียน BN เราจึงจำเป็นต้องสร้าง ฐานข้อมูลตารางเดี่ยวจากข้อมูลอินพุตประเภทข้อมูลเชิงสัมพันธ์ แต่ทว่าการสร้างฐานข้อมูลตารางเดี่ยวจากข้อมูลลักษณะนี้มีปัญหาตรงที่ไม่สามารถกำหนดคุณสมบัติที่ต้องการให้แน่ชัดได้เพราะคุณสมบัติที่เป็นไปได้ทั้งหมดในอินพุตประเภทนี้สามารถมีได้มากมายมหาศาล [4,9] ตัวอย่างเช่น อาจกำหนดให้ความรวยของพ่อแม่เป็นคุณสมบัติหนึ่ง ให้ความรวยของปู่ย่าเป็นคุณสมบัติหนึ่ง (ด้วยการเพิ่มตัวแปรเช่น `parent(Z,Y), parent(Y,X)` หมายถึง Z เป็นปู่ของ X) เห็นได้ว่าวิธีนี้สามารถเพิ่มคุณสมบัติของฐานข้อมูลตารางเดี่ยวที่ต้องการได้อย่างไม่มีที่สิ้นสุด อย่างไรก็ตาม*การแปลงหรือลดรูปปัญหาจากข้อมูลเชิงสัมพันธ์ไปเป็นฐานข้อมูลตารางเดี่ยว (propositionalization)* นั้นสามารถทำได้โดยการเลือกเฉพาะคุณสมบัติที่สำคัญหรือ*ลักษณะสำคัญ (feature)* ของข้อมูลมาเป็นแถวตั้งของฐานข้อมูลตารางเดี่ยวก็เพียงพอไม่จำเป็นต้องนำทุก ๆ คุณสมบัติที่เป็นไปได้มาสร้างเป็นแถวตั้งทั้งหมด อย่างไรก็ตามปัญหาก็คือเราไม่อาจทราบล่วงหน้าว่าต้องกำหนดความสัมพันธ์ลงไปลึกกี่ขั้นถึงจะได้ลักษณะสำคัญที่ต้องการ (ดังเช่นความสัมพันธ์แบบบรรพบุรุษในตัวอย่างที่แล้ว) ซึ่งปัญหานี้เป็นปัญหาเดียวกับการเรียนรู้ FOL ของ ILP นั่นเอง ดังนั้นงานวิจัยชิ้นนี้จึงเสนอการสร้างลักษณะสำคัญจำนวนหนึ่งที่สำคัญจริง โดยสร้างลักษณะสำคัญเหล่านั้นจาก FOL ที่ได้จาก ILP ดังเช่นจากตัวอย่างที่ผ่านมาเราสามารถบอกได้ว่าลักษณะสำคัญของข้อมูลในแง่ความสัมพันธ์ทางบรรพบุรุษจะจำกัดไม่เกินรุ่นพ่อแม่ (กฎข้อที่สาม)

ระบบต้นแบบในการเรียนรู้ FOBN ที่งานวิจัยนี้เสนอดูได้ที่รูปที่ 2 จากรูปที่ 2 อธิบายได้ดังนี้ ในขั้นแรกใช้ระบบ ILP สร้าง FOL ตามปกติ หลังจากนั้นนำ FOL ที่ได้มาสร้างลักษณะสำคัญ และนำลักษณะสำคัญเหล่านั้นมาสร้าง FOBN โดยใช้ BNL ต่อไป โดยงานวิจัยนี้เสนออัลกอริทึมสองชิ้นคือ *MCAFEE* เพื่อใช้ในการค้นหาลักษณะสำคัญต่าง ๆ จาก FOL เพื่อนำมาเป็นคุณสมบัติ (แถวตั้ง) ของฐานข้อมูลตารางเดี่ยวที่ต้องการ และอัลกอริทึมชิ้นที่สองที่เสนอ คือ *GRASP* เป็นอัลกอริทึมที่จะสร้างแถวนอนหรือตัวอย่าง (training examples) ในฐานข้อมูลตารางเดี่ยวที่ต้องการ ในงานวิจัยนี้จะเรียก FOBN ที่ถูกเรียนรู้เพื่องานทำนายข้อมูลโดยเฉพาะว่าตัว*จำแนก FOBN (FOBN classifier)* ซึ่งมีนิยามดังนี้

นิยามที่ 1 (FOBN Classifier). FOBN ใด ๆ จะถูกเรียกว่าเป็น ตัวจำแนก FOBN เมื่อมีคุณสมบัติสองข้อดังต่อไปนี้

(1) โนดของ FOBN ที่เราสนใจจะทำนายคุณสมบัติ (target node หรือ class node) จะต้องไม่มีโนดลูก (children node) ใด ๆ เลย

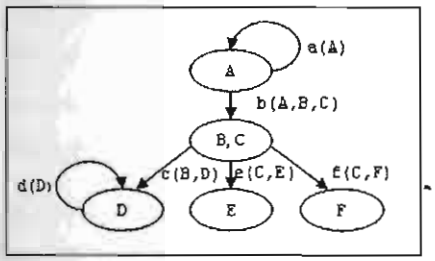
(2) โนดพ่อแม่ (parent node) ของโนดที่เราจะทำนายคุณสมบัติจะต้องเป็นลักษณะสำคัญต่าง ๆ เท่านั้น □

จุดมุ่งหมายของ (1) คือการป้องกันไม่ให้โนดที่เราต้องการทำนายไปทำนายคุณสมบัติของโนดอื่นซึ่งจะทำให้เราไม่ได้ตัวจำแนก FOBN ที่จะทำนายโนดที่ต้องการ ส่วน (2) มีจุดประสงค์ในการนำเทคนิคการตรงบางส่วนของกฎมาใช้ทำนายโนดที่ต้องการ (ดังได้แสดงข้อดีของวิธีนี้ไปแล้ว) รูปที่ 1 เป็นตัวจำแนก FOBN ที่ตรงตามนิยามที่ 1 โดยโนด `f1-f6` หมายถึงลักษณะสำคัญทั้ง 6 ตัว (ดูรายละเอียดในหัวข้อถัดไป)

2. อัลกอริทึมในการเรียนรู้ FOBN

MCAFEE (Minimal ChAin FEature Extraction) เป็นอัลกอริทึมที่ใช้ค้นหาลักษณะสำคัญจาก FOL ที่ได้จาก ILP โดยในที่นี้จะเรียกส่วนของ FOL ว่า*เชน (chain)* เราอาจเรียกลักษณะสำคัญที่งานวิจัยนี้ต้องการได้อีกอย่างว่า *เชนที่สำคัญ (significant chain)* โดยคุณสมบัติแรกของเชนที่สำคัญในงานวิจัยนี้คือต้องไม่ใช่ *เชนที่ไม่มีความหมาย (meaningless chain)* ในงานวิจัยนี้เชนที่ไม่มีความหมายคือการมีตัวแปรบางตัวในเชนถูกสร้างขึ้นมาโดยไม่ได้สัมพันธ์กับตัวแปรอื่นใน

เชน (ดู [8,9] เพิ่มเติม) เช่น พิจารณากฎ "`rich(X) :- dad(Y,X), rich(Y), good(W).`" สังเกตว่าตัวแปร `Y` ถูกสร้างขึ้นมาโดยสัมพันธ์กับตัวแปร `X` ทำให้สามารถทำความเข้าใจ `rich(Y)` ได้ ขณะที่ตัวแปร `W` ถูกสร้างขึ้นมาอย่างไม่สัมพันธ์กับตัวแปรอื่นทำให้ไม่สามารถตีความหมายได้ชัดเจน ฉะนั้นจากกฎข้อนี้เชนใด ๆ ที่มี `good(W)` อยู่จะไม่ถูกเลือกให้เป็นลักษณะสำคัญ วิธีในการสร้างเชนที่สำคัญจาก FOL ของ MCAFEE ได้แนวคิดจาก [8] โดยจะมองกฎ FOL เสมือนเป็นกราฟแบบระบุทิศทาง (directed graph) เช่นจากกฎ "`r(A) :- a(A),b(A,B,C),c(B,D),d(,D),e(C,E), f(C,F).`" สามารถสร้างกราฟแบบระบุทิศทางได้ดังรูปที่ 3



รูปที่ 3. กราฟแบบระบุทิศทาง

งานวิจัยนี้เรียกเชนที่มีความหมายอีกชื่อหนึ่งว่า *เชนที่ถูกต้อง (valid chain)* โดยนิยามได้ดังนี้

นิยามที่ 2 (valid chain). เมื่อมองกฎ FOL หนึ่ง ๆ เสมือนเป็นกราฟแบบระบุทิศทางแล้ว เส้นทาง (path) จากโนดรากไปยังโนดใด ๆ คือเชนที่ถูกต้องก็ต่อเมื่อเส้นทางนั้นมีคุณสมบัติใดคุณสมบัติหนึ่งต่อไปนี้ (1) เส้นทางนั้นมีวงวน (loop) เกิดขึ้น หรือ (2) เส้นทางนั้นมีโนดที่เป็นโนดใบ (leaf node) อยู่ในเส้นทาง □

โดยโนดใบในงานวิจัยนี้หมายถึงโนดที่ไม่มีเส้นเชื่อมออกจากตัวมันเอง เงื่อนไข (1) และ (2) ถูกสร้างขึ้นเพื่อต้องการให้แต่ละเชนที่ถูกต้องมีข้อมูลที่ครบถ้วนจากกฎ FOL เดิมให้มากที่สุดเท่าที่เป็นไปได้นั่นเอง ทั้งนี้เพื่อต้องการสร้างลักษณะที่สำคัญจริง ๆ จากกฎของ FOL (ดู [8] เพิ่มเติม) อย่างไรก็ตามในนิยามที่ 1 ได้กำหนดว่าลักษณะสำคัญทั้งหมดจะเป็นโนดพ่อแม่ของโนดที่ต้องการทำนาย ซึ่งหมายความว่าต้องมีการคำนวณค่าความน่าจะเป็นภายหลังทั้งหมดทุกรูปแบบที่เป็นไปได้ของลักษณะสำคัญหรือโนดพ่อแม่ทั้งหมดใน CPT ของโนดที่ต้องการทำนาย ดังนั้นถ้าเลือกทุก ๆ เชนที่ถูกต้องเป็นลักษณะสำคัญของโนดที่ต้องการทำนายอาจทำให้เกิดความซ้ำซ้อน (redundant) ได้โดยไม่จำเป็น ตัวอย่างเช่นจากรูปที่ 3 "`b(A,B,C), e(C,E), f(C,F)`" เป็นเชนที่ถูกต้องแต่ซ้ำซ้อนเนื่องจากสามารถเกิดจากการรวมกัน (combination) ของเชนที่ถูกต้องสองเชนคือ "`b(A,B,C), e(C,E)`" กับ "`b(A,B,C), f(C,F)`" ดังนั้นในการสร้างลักษณะสำคัญ MCAFEE จะสร้างเฉพาะเชนที่ถูกต้อง*และ*ไม่ซ้ำซ้อนเท่านั้น โดยจะเรียกเชนประเภทนี้ว่า *เชนที่ถูกต้องเล็กสุด (minimal valid chain)* และนิยามดังนี้

นิยามที่ 3 (minimal valid chain). เชนที่ถูกต้อง `r` ใด ๆ เล็กที่สุดก็ต่อเมื่อไม่มีเชนที่ถูกต้องอื่น ๆ `r'` ที่มีคุณสมบัติ $r' \subseteq r$

ตัวอย่างของเชนที่ถูกต้องเล็กสุดหรือลักษณะสำคัญจากรูปที่ 3 ที่สร้างโดย MCAFEE แสดงได้ในรูปที่ 4 และเชนที่ถูกต้องเล็กสุดทั้งหมดของ FOL สามกฎในตัวอย่างของหัวข้อที่ 1 แสดงในรูปที่ 5

```
f1(A) :- a(A).
f2(A,B,C,D) :- b(A,B,C),c(B,D),d(D).
f3(A,B,C,E) :- b(A,B,C),e(C,E).
f4(A,B,C,F) :- b(A,B,C),f(C,F).
```

รูปที่ 4. ลักษณะสำคัญจากรูปที่ 4 ที่สร้างโดย MCAFEE

```
f1(A) :- genius(A).
f2(A) :- diligent(A).
f3(A) :- artist(A).
f4(A) :- save_money(A).
f5(A,B) :- parent(B,A),rich(B).
f6(A,B) :- parent(B,A),genius(B).
```

รูปที่ 5. ลักษณะสำคัญจากตัวอย่างในหัวข้อที่ 1

อัลกอริทึม GRASP (GRound substitution AS example Propositionalization) เป็นอัลกอริทึมในการสร้างแถวนอนหรือตัวอย่างของฐานข้อมูลตารางเดี่ยว ทั้งนี้เนื่องจากบางกรณีตัวอย่างของข้อมูลที่มีความสัมพันธ์ต่อกันเมื่อแปลงรูปไปเป็นฐานข้อมูลตารางเดี่ยวตามแถวตั้งที่กำหนดแล้ว สามารถเปลี่ยนเป็นแถวใหม่มากกว่าหนึ่งได้เช่น พิจารณาตัวอย่างในหัวข้อที่ 1 อีกครั้ง สมมุติให้นาย `a` เป็นตัวอย่างหนึ่งในข้อมูลอินพุตเชิงสัมพันธ์และให้นาย `a` มีพ่อบุญธรรมอยู่อีกหนึ่งคน (นาย `h`) ซึ่งมีคุณสมบัติดังนี้ `parent(h,a). genius(h).` จากตัวอย่างนี้ เมื่อมีการแปลงข้อมูลให้ไปอยู่ในรูปฐานข้อมูลตารางเดี่ยวแล้วจะเปลี่ยนเป็นสองตัวอย่างใหม่ดังแสดงในตารางที่ 2 (ดู `f1-f6` ในรูปที่ 5)

ตารางที่ 2. ข้อมูลของนาย `a` (หัวข้อที่ 1) เมื่อแปลงให้อยู่ในรูปฐานข้อมูลตารางเดี่ยว

case	binding	f1	f2	f3	f4	f5	f6
a	A/a,B/e	1	0	0	1	1	0
	A/a,B/h	1	0	0	1	0	1

โดยกรณีแรกนาย a มีพ่อที่รวย (นาย e) ส่วนในกรณีที่สองนาย a มีพ่อที่ฉลาด (นาย h) แต่ว่าไม่ใช่คนเดียวกัน โดย [5] เสนอให้ใช้ *การแทนค่าพื้นฐาน (ground substitution : GS หรือ variable binding)* ทั้งหมดทุกกรณีเป็นตัวอย่างใหม่ทั้งหมดในฐานข้อมูลตารางเดี่ยว (โดยถือว่าตัวอย่างใหม่ทั้งหมดที่เกิดจากตัวอย่างเดิมนั้นไม่มีความข้องเกี่ยวต่อกันและกันเลย) ดังเช่น GS สองกรณีของนาย a ในตารางที่ 2 เกิดเป็นตัวอย่างใหม่สองตัว โดยวิธีนี้มีข้อดีสองข้อคือ ข้อแรกการแปลงข้อมูลประเภทนี้ทำให้เกิดข้อมูลใหม่เป็นฐานข้อมูลตารางเดี่ยวที่แท้จริงทำให้สามารถใช้ BNL เรียนรู้ได้ทันทีไม่ต้องปรับแต่งข้อมูลอีก และข้อดีข้อที่สองคือความสัมพันธ์ระหว่างโนดที่สร้างได้จากวิธีนี้จะมีความสมบูรณ์สูงกว่า (strong completeness) บางงานวิจัยเช่น [8,9] ที่เสนอให้นำเฉพาะ GS เดียวที่คิดว่าดีที่สุดมาเป็นตัวอย่างใหม่ (เนื่องจากวิธีเหล่านั้นไม่ใช้ข้อมูลทั้งหมดที่มีให้เกิดประโยชน์สูงสุด) พิจารณาตารางที่ 3

ตารางที่ 3. ตัวอย่างของความสมบูรณ์สูงกว่าใน GS

case	class	binding	f1	f2	f3	f4
E1	+	θ_{E11}	1	0	0	1
		θ_{E12}	0	1	0	0
E2	+	θ_{E21}	0	1	0	0
		θ_{E22}	1	0	0	1
E3	+	θ_{E31}	0	0	1	1
		θ_{E32}	1	0	1	0
E4	-	θ_{E41}	0	0	1	0
		θ_{E42}	1	0	0	0
E5	-	θ_{E51}	1	0	1	0
		θ_{E52}	1	0	0	0

จากตารางที่ 3 เห็นได้ว่าเนื่องจากความน่าจะเป็นภายหลัง $p(+|f2)$ และ $p(+|f4)$ มีค่า 1.0 ในขณะที่ $p(+)$ คำนวณได้ 0.6 ฉะนั้นโนด `class` จึงขึ้นกับ (depend on) โนด `f2` และ `f4` แต่ถ้าเลือกเพียงหนึ่ง GS ในตัวอย่างเดิมเป็นตัวอย่างใหม่จะทำให้คำนวณได้ว่าโนด `class` ขึ้นกับโนด `f2` หรือ `f4` โนดใดโนดหนึ่งเท่านั้นซึ่งไม่ครบถ้วน

อย่างไรก็ตามการเลือกทุก ๆ GS เป็นตัวอย่างใหม่อาจทำให้เกิดปัญหาสองอย่างคือ ปัญหาข้อแรกคือจำนวนตัวอย่างใหม่ที่ได้อาจมีมหาศาลทำให้หน่วยความจำในการเก็บ ฐานข้อมูลตารางเดี่ยวไม่เพียงพอ ปัญหาข้อที่สองคือวิธีนี้อาจทำให้เกิดความผิดพลาดในการเรียนรู้ได้เช่น กรณีในตารางที่ 4

จากตารางที่ 4 เห็นได้ว่าความน่าจะเป็นภายหลังของ $p(+|f4)$ คำนวณได้ 0.5 ในขณะที่ $p(+)$ คำนวณได้ 0.4 เนื่องจากค่าความน่าจะเป็นทั้งสองค่าใกล้เคียงกันอาจทำให้ BNL สรุปว่าโนด `class` เป็นอิสระต่อโนด `f4` ซึ่งไม่ถูกต้องเนื่อง

ตารางที่ 4. กรณีตัวอย่างเจ็ดตัวเดิมถูกเปลี่ยนเป็นสิบตัว

case	class	binding	f1	f2	f3	f4
E1	+	θ_{E11}	1	0	0	1
E2	+	θ_{E21}	1	0	0	1
E3	+	θ_{E31}	0	0	1	1
E4	+	θ_{E41}	0	0	0	1
E5	-	θ_{E51}	1	0	1	0
E6	-	θ_{E61}	1	0	1	0
E7	-	θ_{E71}	0	0	0	1
		θ_{E72}	0	0	0	1
		θ_{E73}	0	0	0	1
		θ_{E74}	0	0	0	1

จากในความเป็นจริงมีตัวอย่างลบเพียงตัวอย่างเดียว (`E7`) ที่ขัดแย้งกับตัวอย่างบวกสี่ตัวอย่าง (หรือ $p(+|f4) = 0.8$)

GRASP แก้ปัญหาทั้งสองข้อข้างต้นโดยการเลือกเพียงบาง GS เป็นตัวอย่างใหม่เท่านั้นโดยจะเลือกเฉพาะ *GS ที่ไม่ซ้ำ (non-duplicate) และมีความเฉพาะเจาะจงมากที่สุด (maximal specific binding : MSB)* โดย MSB มีนิยามดังนี้

นิยามที่ 4 (maximal specific binding). เมื่อกำหนด ω คือ GS ใด ๆ กำหนดให้ $f(\omega)$ คือเซตของลักษณะสำคัญทั้งหมดที่เป็นจริงของ ω พิจารณา GS θ ของแต่ละตัวอย่างเดิม e θ เป็น MSB ก็ต่อเมื่อไม่มี α ซึ่งเป็น GS อื่นของ e ที่ $f(\theta) \subseteq f(\alpha)$ □

จากนิยามที่ 4 พิจารณาตารางที่ 5 MSB ที่ไม่ซ้ำ (non-duplicate MSB) จากตารางมีสี่กรณีคือ θ_{E11} θ_{E13} θ_{E21} และ θ_{E22} สังเกตว่า θ_{E14} ก็เป็น MSB แต่ซ้ำกับ θ_{E11} เช่นเดียวกับ θ_{E23} ซึ่งซ้ำกับ θ_{E22} ฉะนั้นตัวอย่างใหม่ที่สร้างโดย GRASP มีเพียง GS สี่กรณีข้างต้นเท่านั้น

ตารางที่ 5. กรณีตัวอย่างสองตัวเดิมเปลี่ยนเป็นแปดตัวใหม่

case	class	θ	f1	f2	f3	f4
E1	+	θ_{E11}	1	1	0	1
		θ_{E12}	0	1	0	1
		θ_{E13}	0	1	1	0
		θ_{E14}	1	1	0	1
E2	-	θ_{E21}	1	1	0	1
		θ_{E22}	0	0	1	0
		θ_{E23}	0	0	1	0
		θ_{E24}	1	1	0	0

3. ผลการทดลองเบื้องต้น

การทดลองเบื้องต้นนี้ใช้ชุดข้อมูล (dataset) ของ*ความสามารถในการก่อกลายพันธุ์ของโมเลกุล (mutagenesis)* [8] ในการทดลองนี้ระบบต้นแบบของเราได้เลือกใช้ระบบ ILP คือ

PROGOL (เวอร์ชัน CProgol4.2) [12] ในการสร้าง FOL และใช้โปรแกรม WinMine [3] เป็น BNL ในขั้นตอนการทำนายของการทดลองครั้งนี้ใช้เทคนิค 3CV (3-fold cross-validation) [11] นอกจากนี้การทดลองนี้ยังได้เพิ่มสัญญาณรบกวนอย่างสุ่มจำนวน 10% และ 15% ในชุดข้อมูลนี้อีกด้วย เพื่อทดสอบประสิทธิภาพในการรับมือของสัญญาณรบกวนของ FOBN เนื่องจากระบบ PROGOL อนุญาตให้ผู้ใช้ปรับเปลี่ยนตัวเลือก (option) เพื่อรับมือกับสัญญาณรบกวนได้อยู่แล้ว การทดลองครั้งนี้จึงได้ลองปรับค่าตัวเลือกต่าง ๆ ของ PROGOL (ดังแสดงใน "x% noise setting") แล้วนำผลที่ได้มาเปรียบเทียบกับผลการทดลองจาก FOBN ผลการทดลองที่ได้แสดงในตารางที่ 6 เห็นได้ว่าเมื่อไม่มีสัญญาณรบกวนภายในชุดข้อมูล ผลการทำงานของ FOBN มีประสิทธิภาพใกล้เคียงกับ PROGOL แต่ว่าผลการทำงานของ FOBN ในชุดข้อมูลที่มีสัญญาณรบกวนให้ความถูกต้องสูงกว่า PROGOL อย่างมากในทุกกรณี ความถูกต้องที่สูงกว่านี้เมื่อวิเคราะห์แล้วเกิดจากสาเหตุใหญ่สองประการคือ 1) ลักษณะสำคัญที่ได้จาก MCAFEE ช่วยให้การใช้เทคนิคการตรงบางส่วนของกฎได้อย่างมีประสิทธิภาพ (ดูบทวิเคราะห์เพิ่มใน [8]) 2) การใช้ทฤษฎีความน่าจะเป็นของ FOBN ช่วยให้การใช้งานลักษณะสำคัญมีความยืดหยุ่นมากยิ่งขึ้น

4. สรุป

งานวิจัยนี้เสนอการเรียนรู้ตัวจำแนกข้อมูลชนิด FOBN ซึ่งเป็นตัวจำแนกประเภทข้อมูลที่มีความสามารถสูง เนื่องจากรวมความสามารถของ FOL และ BN ไว้ด้วยกันทำให้สามารถเรียนรู้จากข้อมูลเชิงสัมพันธ์และข้อมูลที่ไม่สามารถแบ่งกลุ่มได้แน่นอนได้ดี

กิตติกรรมประกาศ

ผู้เขียนขอขอบคุณดร.สุกรี สินธุภิญโญที่ได้แลกเปลี่ยนความคิดเห็นในงานวิจัยชิ้นนี้ ขอขอบคุณ Dr. David Maxwell Chickering และ Daniel Lowd สำหรับการช่วยเหลือในปัญหาต่าง ๆ ของโปรแกรม WinMine งานวิจัยชิ้นนี้ได้รับการสนับสนุนโดยสำนักงานกองทุนสนับสนุนการวิจัย (สกว.)

Reference

1. M. Botta and A. Giordana and L. Saitta and M. Sebag. Relational learning: Hard Problems and Phase transition. Selected papers from AIIA'99, Springer-Verlag, 2000.
2. D. M. Chickering. Learning Bayesian Networks is NP-Complete. In D. Fisher and H. J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics* V, 1996.
3. D. M. Chickering. The WinMine Toolkit. *Technical Report MSR-TR-2002-103*, Microsoft, 2002
4. L. De Raedt. Attribute value learning versus inductive logic programming: The missing links (extended abstract). In D. Page, editor, *Proc. of the 8th Int. Conference on Inductive Logic Programming*, pages 1-8. Springer-Verlag, 1998.
5. D. Fensel, M.Zickwolff, and M. Weise. Are substitutions the better examples ? In L. De Raedt, editor, *Proc. of the 5th International Workshop on ILP*, 1995.
6. L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning Probabilistic Relational Models. *Relational Data Mining*, S. Dzeroski and N. Lavrac, editors, 2001
7. K. Kersting, L. De Raedt. Basic Principles of Learning Bayesian Logic Programs. *Technical Report No. 174*, University of Freiburg, Germany, June 2002
8. B. Kijsirikul, S. Sinthupinyo, and K. Chongkasemwongse. Approximate Match of Rules Using Backpropagation Neural Networks. *Machine Learning Journal*, 2001
9. S. Kramer, N. Lavrac and P. Flach. Propositionalization Approaches to Relational Data Mining, in: Dzeroski S., Lavrac N, editors, *Relational Data Mining*, 2001.
10. N. Lavrac and S. Dzeroski. *Inductive Logic Programming : Techniques and Applications*. Ellis Horwood, 1994
11. T. Mitchel. *Machine Learning*. McGraw-Hill, 1997.
12. S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, Special issue on Inductive Logic Programming, 13(3-4):245–286, 1995.
13. J. Pearl. *Causality*. Addison Wesley, 2001.

ตารางที่ 6. เปอร์เซ็นต์ความถูกต้องในปัญหาความสามารถในการก่อกลายพันธุ์ของโมเลกุล

Noise Level in Dataset	PROGOL 0% noise setting	PROGOL 5% noise setting	PROGOL 10% noise setting	PROGOL 15% noise setting	PROGOL +FOBN
0%	84.58	82.99	77.14	77.14	84.34
10%	64.23	65.42	69.72	71.29	78.67
15%	60.56	59.02	61.54	65.31	74.33

การเรียนรู้โปรแกรมตรรกะโดยนิวรอลเน็ตเวิร์กอันดับที่หนึ่ง
Learning Logic Programs by First-Order Neural Networks

ธนุพล เลิศลำเนาชัย และ บุญเสริม กิจศิริกุล
ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
จุฬาลงกรณ์มหาวิทยาลัย ถนนพญาไท ปทุมวัน กรุงเทพฯ 10500
Email: g46tll@cp.eng.chula.ac.th, boonserm.k@chula.ac.th

Abstract

The advantages of Inductive Logic Programming (ILP) are the ability of employing background knowledge in form of first-order logic and its highly expressive representation. Nevertheless, an ILP system is not robust enough to noisy or unseen data. Moreover in multi-class classification, if the noisy example is not matched with any learned rules, it cannot be classified. In this paper we present a new learning method that alleviates this problem by enabling Neural Networks to handle first-order logic programs directly. The proposed method is called the First-Order Neural Network (FONN). FONNs can receive First-Order Logic programs as the input of the networks. Our proposed method has been evaluated on the Finite Element Mesh Design, a first-order learning dataset. The experimental results show that the proposed method provides more accurate result than the ILP system.

บทคัดย่อ

การโปรแกรมตรรกะเชิงอุปนัยหรือระบบไอแอลพี เป็นการเรียนรู้ของเครื่องรูปแบบหนึ่งซึ่งมีข้อดีที่สามารถนำความรู้ภูมิหลังมาใช้ในการเรียนรู้ได้และกฎที่ได้สามารถอธิบายความรู้ได้อย่างกว้างขวาง อย่างไรก็ตามระบบไอแอลพีเป็นระบบที่ไม่ทนทานต่อข้อมูลที่มีสัญญาณรบกวน เมื่อนำระบบไอแอลพีมาจำแนกข้อมูลลักษณะนี้อาจทำให้การจำแนกผิดพลาดได้ เนื่องจากไม่มีกฎที่ตรงกับตัวอย่างนั้น นอกจากนี้ในกรณีที่นำกฎไปใช้จำแนกตัวอย่างแบบหลายประเภทและไม่มีกฎที่ตรงกับตัวอย่างจะทำให้ตัวอย่างนั้นไม่สามารถถูกจำแนกได้ งานวิจัยนี้นำเสนอระบบการเรียนรู้วิธีใหม่ที่นำนิวรอลเน็ตเวิร์กมาประยุกต์เข้ากับระบบไอแอลพี ทำให้ระบบยืดหยุ่นขึ้น และสามารถรับอินพุตในรูปแบบของตรรกะอันดับที่หนึ่งมาทำการเรียนรู้ได้โดยตรง โดยเรียกระบบนี้ว่า นิวรอลเน็ตเวิร์กอันดับที่หนึ่ง และจากการทดสอบกับข้อมูลการวิเคราะห์ไฟไนต์เอลิเมนต์ พบว่าผลการทดลองที่ได้จากนิวรอลเน็ตเวิร์กอันดับที่หนึ่งมีเปอร์เซ็นต์ความถูกต้องสูงกว่าระบบไอแอลพี

Key-words: Inductive Logic Programming, First-Order Logic, Neural Networks

1. บทนำ

การเรียนรู้ของเครื่อง (Machine learning)[1] เป็นแนวคิดที่ต้องการสร้างโปรแกรมคอมพิวเตอร์ให้สามารถเรียนรู้จากตัวอย่างที่ผู้สอนได้จัดเตรียมไว้ให้ โดยโปรแกรมที่สร้างขึ้นนั้นต้องสามารถสร้างแนวคิด (concept)ที่สอดคล้องกับตัวอย่างที่ได้รับจากผู้สอนและสามารถนำแนวคิดที่ได้นั้นไปใช้ในการจำแนกตัวอย่างใหม่ในอนาคตได้อย่างถูกต้อง การเรียนรู้ของเครื่องมีหลายประเภทด้วยกัน เช่น การโปรแกรมตรรกะเชิงอุปนัย (Inductive Logic Programming: ILP)[1-3] นิวรอลเน็ตเวิร์ก (Neural Networks)[1, 4] เน็ตเวิร์กเบย์ (Bayesian Networks)[1] ต้นไม้ตัดสินใจ (Decision Tree)[1] เป็นต้น การเรียนรู้ของเครื่องแต่ละประเภทก็เหมาะที่จะนำไปใช้กับงานในลักษณะต่างๆกัน

การโปรแกรมตรรกะเชิงอุปนัยหรือไอแอลพีเป็นวิธีการเรียนรู้ของเครื่องประเภทหนึ่งซึ่งนำหลักการทางตรรกะ

มาประยุกต์ใช้ ทำให้มีข้อดีที่แตกต่างจากการเรียนรู้ของเครื่องแบบอื่นๆ เนื่องจากแนวคิดที่ได้จากการเรียนจะอยู่ในรูปแบบของตรรกะอันดับที่หนึ่ง (First-Order Logic) ซึ่งเป็นลักษณะของการอธิบายความรู้ของมนุษย์ ทำให้เข้าใจได้ง่ายกว่าแนวคิดที่ได้จากวิธีอื่น นอกจากนี้ตรรกะอันดับที่หนึ่ง ยังใช้อธิบายแนวคิดที่ซับซ้อนได้ดีกว่าตรรกศาสตร์ประพจน์ (Propositional Logic) อีกด้วย ปกติแล้วระบบไอแอลพีมักจะถูกนำมาใช้ในการจำแนกตัวอย่างที่มี 2 ประเภท (class) โดยไอแอลพีจะรับอินพุตเป็นความรู้ภูมิหลัง (Background Knowledge) ตัวอย่างบวก (positive example) และตัวอย่างลบ (negative example) แล้วจะพยายามสร้างกฎหรือแนวคิดที่ครอบคลุมตัวอย่างบวกแต่ไม่ครอบคลุมตัวอย่างลบ โดยเมื่อนำกฎที่ได้ไปใช้จำแนกตัวอย่างทดสอบ ตัวอย่างนั้นจะถูกจำแนกว่าเป็นตัวอย่างบวกถ้าตรงกับกฎ แต่ถ้าไม่ตรงก็จะจำแนกตัวอย่างนั้นเป็นตัวอย่างลบ อย่างไรก็ตามถ้าตัวอย่างที่นำมาจำแนกมีความผิดพลาด เช่น ถูกสัญญาณรบกวนทำให้ข้อมูลบางส่วนหายไป ก็จะทำให้การจำแนกข้อมูลมีความผิดพลาดไปด้วย นอกจากนี้ถ้าเรานำระบบไอแอลพีไปใช้ในการจำแนกตัวอย่างแบบหลายประเภท (Multiclass Classification) ตัวอย่างที่ทำการจำแนกต้องสอดคล้องกับกฎจึงจะสามารถจำแนกได้ว่าตัวอย่างนั้นอยู่ประเภทใด ซึ่งการที่ข้อมูลมีสัญญาณรบกวนอาจทำให้ตัวอย่างนั้นไม่สามารถถูกจำแนกได้ว่าอยู่ประเภทใดและทำให้ความแม่นยำในการจำแนกตัวอย่างลดลง

เนื่องด้วยข้อจำกัดที่ไม่สามารถจำแนกตัวอย่างในลักษณะนี้ได้ จึงมีงานวิจัยที่นำเสนอแนวทางเพื่อเพิ่มความแม่นยำของการจำแนกตัวอย่าง โดยParekhและHonavar[5] ได้ประยุกต์การเรียนรู้ทางตรรกะเข้ากับนิวรอลเน็ตเวิร์กทำให้ทนทานต่อข้อมูลที่มีสัญญาณรบกวนได้ดีขึ้น แต่ว่าตรรกะที่ใช้ยังเป็นลักษณะของตรรกศาสตร์ประพจน์อยู่ Kijsirikulและคณะ[6]ได้เสนอวิธีที่สามารถใช้ตรรกะอันดับที่หนึ่งร่วมกับนิวรอลเน็ตเวิร์กได้ ทำให้มีความยืดหยุ่นมากกว่าระบบไอแอลพีและทำให้ค่าความแม่นยำสูงขึ้น แต่ว่าวิธีที่ใช้นั้นในขั้นตอนแรกจำเป็นต้องใช้ระบบไอแอลพีมาทำการสร้างกฎขึ้นมาก่อน แล้วจึงนำลักษณะสำคัญของกฎที่ได้มาเรียนรู้ด้วยนิวรอลเน็ตเวิร์กอีกครั้งหนึ่ง วิธีนี้แม้ว่าจะทำให้ผลในการจำแนกตัวอย่างดีขึ้น แต่ว่ายังจำเป็นต้องพึ่งระบบไอแอลพีอยู่

ในงานวิจัยนี้จะนำเสนอการเรียนรู้แบบใหม่ที่ยืดหยุ่นขึ้น โดยนำนิวรอลเน็ตเวิร์กมาประยุกต์เข้ากับการโปรแกรมตรรกะเชิงอุปนัย แต่ว่าสามารถรับอินพุตในรูปแบบของตรรกะอันดับที่หนึ่งมาทำการเรียนรู้ได้โดยตรง ไม่ต้องใช้ระบบไอแอลพีมาช่วยในการเรียนรู้ วิธีการที่นำเสนอนี้สามารถจัดการกับข้อจำกัดของระบบไอแอลพีได้และให้ผลที่แม่นยำมากขึ้น โดยเรียกวิธีการที่พัฒนาขึ้นนี้ว่า นิวรอลเน็ตเวิร์กอันดับที่หนึ่ง (First-Order Neural Networks: FONNs)

2. นิวรอลเน็ตเวิร์กอันดับที่หนึ่ง

เนื้อหาในส่วนนี้จะอธิบายถึงการเรียนรู้ของนิวรอลเน็ตเวิร์กอันดับที่หนึ่ง โดยจะแบ่งอธิบายออกเป็นส่วนๆ คือ โครงสร้างของนิวรอลเน็ตเวิร์กอันดับที่หนึ่ง อินพุตของนิวรอลเน็ตเวิร์กอันดับที่หนึ่งโดยหลักการเรียนรู้แบบหลายตัวอย่าง (Multiple-Instance Learning: MIL)[7, 8] และในส่วนสุดท้ายเป็นวิธีการปรับค่าน้ำหนักของนิวรอลเน็ตเวิร์กอันดับที่หนึ่ง

2.1 โครงสร้างของนิวรอลเน็ตเวิร์กอันดับที่หนึ่ง

เนื่องจากนิวรอลเน็ตเวิร์กอันดับที่หนึ่งนั้นเป็นระบบการเรียนรู้ที่เราพัฒนามาจากนิวรอลเน็ตเวิร์ก ดังนั้นโครงสร้างก็จะมีพื้นฐานมาจากโครงสร้างของนิวรอลเน็ตเวิร์ก แต่ประยุกต์ให้รับตัวอย่างและความรู้ภูมิหลังในรูปแบบของตรรกะได้ โครงสร้างของนิวรอลเน็ตเวิร์กอันดับที่หนึ่งจะแบ่งออกเป็น 3 ชั้นด้วยกัน คือ ชั้นนำเข้า (input layer) ชั้นซ่อน (hidden layer) และชั้นผลลัพธ์ (output layer) ในแต่ละชั้นจะมีความหมายและหน้าที่ดังต่อไปนี้

1) ชั้นนำเข้า เป็นชั้นที่แสดงถึงสัญพจน์ (predicate) ต่างๆที่จะนำมาใช้เป็นตัวแทนของกฎหรือแนวคิดในการเรียน โดยจะเป็นชั้นแรกในการรับข้อมูลก่อนที่จะผ่านไปยังชั้นต่อๆ ไป จำนวนนิวรอนในชั้นนี้จะขึ้นอยู่กับจำนวนสัญพจน์ของความรู้ภูมิหลัง

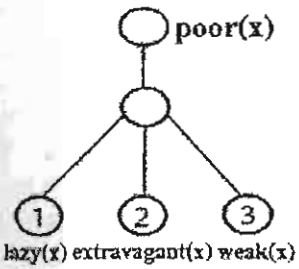
2) ชั้นซ่อน เป็นชั้นที่เชื่อมต่อระหว่างชั้นนำเข้าและชั้นผลลัพธ์ ช่วยเพิ่มความสามารถในการเรียนกฎที่มีความซับซ้อนได้ จำนวนนิวรอนในชั้นนี้มักจะกำหนดโดยดูจากความซับซ้อนของแนวคิดที่ต้องการเรียนรู้

3) ชั้นผลลัพธ์ เป็นชั้นที่แสดงถึงแนวคิดที่ต้องการให้ระบบทำการเรียนรู้ โดยชั้นนี้จะคำนวณหาผลลัพธ์สุดท้ายของเน็ตเวิร์ก จำนวนนิวรอนในชั้นผลลัพธ์จะขึ้นอยู่กับจำนวนแนวคิดทั้งหมดที่ต้องการเรียน

ตัวอย่าง เช่น การเรียนแนวคิดของ poor(x) ซึ่งมีอินพุตที่ใช้ในการเรียนเป็นดังรูปที่ 1

รูปที่ 1. อินพุตสำหรับการเรียนแนวคิด poor(x)

ตัวอย่างที่ได้รับประกอบไปด้วยตัวอย่างบวก 1 ตัว คือ poor(John) ซึ่ง John เป็นคนเกียจคร้าน (lazy(John))และขี้เหนียว (extravagant(John)) มีตัวอย่างลบ 2 ตัว คือ poor(Peter) และ poor(Bob) โดยที่ Peter เป็นคนเกียจคร้านและไม่แข็งแรง (weak(Peter)) Bob เป็นคนฟุ่มเฟือยและไม่แข็งแรง อินพุตที่ได้รับมีจำนวนสัญพจน์ที่ปรากฏในความรู้ภูมิหลัง 3 ตัวด้วยกัน คือ lazy(x), extravagant(x) และ weak(x)



รูปที่ 2. ลักษณะโครงสร้างของเน็ตเวิร์กโดยกำหนดให้ในชั้นซ่อนมี 1 นิวรอน

ดังนั้นในชั้นนำเข้าจะมีอยู่ 3 นิวรอน เพื่อแทนลักษณะทั้ง 3 สัญพจน์นั้น ส่วนจำนวนนิวรอนในชั้นผลลัพธ์จะมีเพียง 1 นิวรอนเท่านั้น เนื่องจากแนวคิดที่ต้องการเรียนมีเพียง 1 แนวคิดเท่านั้น คือ poor(x) ดังแสดงในรูปที่ 2

2.2 อินพุตของนิวรอลเน็ตเวิร์กอันดับที่หนึ่งโดยหลักการเรียนรู้แบบหลายตัวอย่างย่อย

2.2.1 อินพุตของนิวรอลเน็ตเวิร์กอันดับที่หนึ่ง

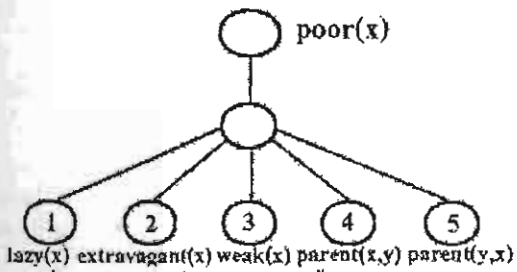
โดยปกติแล้วอินพุตของนิวรอลเน็ตเวิร์กจะอยู่ในรูปของจำนวนจริง แต่ว่าอินพุตของระบบไอแอลพีจะอยู่ในรูปของตรรกะ (ความรู้ภูมิหลังและตัวอย่าง) ดังนั้นจึงต้องเปลี่ยนอินพุตที่เป็นลักษณะของตรรกะไปเป็นอินพุตที่สามารถเรียนรู้ได้โดยนิวรอลเน็ตเวิร์กเสียก่อน การเปลี่ยนจะทำทีละ 1 ตัวอย่าง ทีละ 1 นิวรอน โดยตัวอย่าง 1 ตัวจะถูกเปลี่ยนให้เป็นค่า 1 เพื่อเป็นอินพุตของนิวรอน ถ้าแทนค่าตัวแปรในสัญพจน์ประจำนิวรอนด้วยค่าคงที่ที่ปรากฏในตัวอย่างแล้วมีค่าความจริงเป็นจริงในความรู้ภูมิหลัง แต่ถ้าไม่เป็นจริงในความรู้ภูมิหลังก็จะกำหนดให้อินพุตของนิวรอนมีค่าเป็น 0 ตัวอย่างเช่น จากตัวอย่างอินพุตในรูปที่ 1 เมื่อเปลี่ยนตัวอย่าง poor(John) ไปเป็นอินพุตของเน็ตเวิร์กแล้ว จะได้ค่าอินพุตของนิวรอน lazy(x), extravagant(x) และ weak(x) เป็น 1, 1 และ 0 ตามลำดับ เนื่องจากเมื่อแทนตัวแปร x ในสัญพจน์ของแต่ละนิวรอนด้วย John ซึ่งเป็นค่าคงที่ของตัวอย่าง poor(John) แล้วจะได้เป็น lazy(John), extravagant(John) และ weak(John) ซึ่ง lazy(John) และ extravagant(John) เป็นจริง ในความรู้ภูมิหลังจึงให้ค่าอินพุตของนิวรอนเป็น 1 ในขณะที่ weak(John) ไม่เป็นจริงในความรู้ภูมิหลัง จึงมีค่าเป็น 0 สำหรับตัวอย่าง poor(Peter) และ poor(Bob) จะได้ค่าอินพุตของนิวรอนเป็น 1,0,1 และ 0,1,1 ตามลำดับ

อย่างไรก็ตามเนื่องจากแนวคิดที่ได้จากระบบไอแอลพีอาจมีการสร้างตัวแปรใหม่ขึ้นในสัญพจน์โดยที่ตัวแปรนั้นไม่ปรากฏอยู่ในส่วนหัวของกฎ ตัวแปรใหม่ที่สร้างขึ้นนั้นเพื่อนำมาใช้อธิบายความสัมพันธ์ที่เกี่ยวข้อง (relation) กันของแต่ละสัญพจน์ อย่างเช่น poor(x) ← parent(y,x), poor(y), lazy(x) ซึ่งมีการสร้างตัวแปร y ขึ้นมาเพื่อแสดงความสัมพันธ์

ระหว่างสัญพจน์ parent(y,x) และ poor(y) แต่ว่าการสร้างตัวแปรขึ้นใหม่ในลักษณะนี้จะทำให้การสร้างอินพุตสำหรับนิวรอนเกิดปัญหาความไม่แน่นอนขึ้น เช่น ถ้าความรู้ภูมิหลังและตัวอย่างที่รับเข้ามามีลักษณะเป็นดังรูปที่ 3

รูปที่ 3. อินพุตที่เพิ่มสัญพจน์ parent(x,y) ในความรู้ภูมิหลัง

ในกรณีนี้โครงสร้างของเน็ตเวิร์กจะมีความซับซ้อนมากขึ้นเนื่องจากในความรู้ภูมิหลังมีสัญพจน์ parent(x,y) เพิ่มเข้ามา และเนื่องจากสัญพจน์นี้มีอาร์กิวเมนต์ 2 ตัวและเราไม่สามารถรู้ได้ว่าอาร์กิวเมนต์ 2 ตัวนี้ควรมีการวางตำแหน่งอย่างไร จึงได้ทำการสร้างนิวรอนขึ้นมาสำหรับทั้ง 2 กรณี คือ parent(x,y) และ parent(y,x) ทำให้จำนวนนิวรอนในชั้นนำเข้าเพิ่มขึ้นอีก 2 นิวรอน ดังรูปที่ 4 แต่ว่านิวรอนที่เพิ่มขึ้นมานี้มีตัวแปรใหม่อยู่ด้วย ทำให้มีปัญหาในการกำหนดค่าอินพุตให้กับนิวรอน เช่น ตัวอย่าง poor(John) เมื่อจะกำหนดค่าอินพุตให้กับนิวรอน parent(y,x) ก็จะแทนที่ตัวแปร x ด้วยค่าคงที่ John ส่วนตัวแปร y ซึ่งไม่มีการกำหนดค่ามาจากตัวอย่าง ทำให้ไม่สามารถระบุได้ว่าต้องแทนตัวแปร y ด้วยค่าคงที่ใด และการแทนด้วยค่าคงที่แต่ละแบบนั้นก็จะให้ค่าอินพุตของนิวรอนแตกต่างกันดังแสดงในตารางที่ 1



รูปที่ 4. เน็ตเวิร์กที่มีนิวรอนในชั้นนำเข้า 5 นิวรอน

ส่วนกรณีของ parent(x,y) ไม่เกิดปัญหาเนื่องจากเมื่อแทนค่าตัวแปร x ด้วย John แล้วไม่ว่าจะแทนค่าตัวแปร y ด้วยค่าคงที่ใด ก็ไม่เป็นจริงในความรู้ภูมิหลัง ทำให้ค่าอินพุตของนิวรอน parent(x,y) เป็น 0 ในทุกกรณี

ตารางที่ 1. ค่าอินพุตของนิวรอน parent(y,x) เมื่อแทนตัวแปรด้วยค่าคงที่ต่างๆ

parent(y,x)	ค่าอินพุตของนิวรอน
แทน x ด้วย John แทน y ด้วย John	0
แทน x ด้วย John แทน y ด้วย Peter	1
แทน x ด้วย John แทน y ด้วย Bob	0

จากตัวอย่างข้างต้นแสดงให้เห็นว่าการกำหนดอินพุตให้กับนิวรอนในกรณีที่สัญพจน์ประจำนิวรอนนั้นมีตัวแปรที่ใช้แสดงความสัมพันธ์กับสัญพจน์อื่น โดยตัวแปรนั้นปรากฏอยู่ในส่วนหัวของกฎด้วย ในบางกรณีจะทำให้สามารถกำหนดค่าที่แน่นอนให้กับนิวรอนได้ เนื่องจากไม่ทราบว่าควรแทนค่าให้กับตัวแปรที่แสดงความสัมพันธ์ด้วยค่าใด ได้นำหลักการของการเรียนรู้แบบหลายตัวอย่างย่อย (Multiple Instance Learning: MIL)[7, 8]มาประยุกต์เพื่อทำการสร้างอินพุตให้กับนิวรอลเน็ตเวิร์กอันดับที่หนึ่ง ดังกล่าวในหัวข้อ 2.2.2 ด้านล่างนี้

2.2.2 หลักการเรียนรู้แบบหลายตัวอย่างย่อย

การเรียนรู้แบบหลายตัวอย่างย่อยจะนิยามตัวอย่างการเรียนรู้เป็นถุง (bag) $\{B_1, B_2, \ldots, B_n\}$ โดยที่ n เป็นจำนวนของถุง แต่ละถุง (B_i) จะประกอบไปด้วยตัวอย่าง (instance) m_i ตัว $\{B_{i1}, B_{i2}, \ldots, B_{imi}\}$ โดยที่ถุงหนึ่งๆจะกำหนดให้เป็นถุงตัวอย่างบวก (positive bag) ก็ต่อเมื่อในนั้นมีตัวอย่างย่อยอย่างน้อย 1 ตัวที่เป็นบวก และจะกำหนดเป็นถุงตัวอย่างลบ (negative bag) ก็ต่อเมื่อตัวอย่างย่อยทุกตัวในถุงนั้นเป็นตัวอย่างลบทั้งหมด โดยที่ถุงตัวอย่างบวกจะกำหนดให้มีค่าเป้าหมาย (label) เป็น 1 ส่วนถุงตัวอย่างลบจะมีค่าเป็น 0

การแปลงตัวอย่างไปเป็นอินพุตให้กับเน็ตเวิร์ก จะแทนตัวอย่างบวก 1 ตัว แทนด้วยถุงตัวอย่างบวก 1 ถุง และแทนตัวอย่างลบ 1 ตัวด้วยถุงตัวอย่างลบ 1 ถุง (ในกรณีที่เป็นค

ย่างแบบหลายประเภทจะถือว่าทุกถุงเป็นถุงตัวอย่าง
ประเภทนั้นๆ) ในถุงจะประกอบไปด้วยตัวอย่างย่อยซึ่ง
มาจากการแทนค่าตัวแปรด้วยค่าคงที่แต่ละแบบ
จากตัวอย่างอินพุตตามรูปที่ 3 ในการเรียนแนวคิดของ
จะมีถุงตัวอย่างบวกทั้งหมด 2 ถุง คือ poor(John) และ poor(Peter) ถุงตัวอย่างลบ 1 ถุง คือ poor(Bob) และในแต่ละถุงจะประกอบไปด้วยตัวอย่างย่อยทั้งหมด 3 ตัว จากการแทนค่าตัวแปร y ด้วยค่า John, Peter และ Bob ดังแสดงตัวอย่างในตารางที่ 2

ตารางที่ 2. การแปลงตัวอย่างของ poor(John) ไปเป็นอินพุตให้กับนิวรอลเน็ตเวิร์กอันดับที่หนึ่ง

ถุงตัวอย่าง ของ poor(John)	lazy(x)	extravagant(x)	weak(x)	parent(x,y)	parent(y,x)
แทน x ด้วย John แทน y ด้วย John	1	0	0	0	0
แทน x ด้วย John แทน y ด้วย Peter	1	0	0	0	1
แทน x ด้วย John แทน y ด้วย Bob	1	0	0	0	0

จากนั้นจึงนำถุงตัวอย่างที่ได้ไปใส่เป็นอินพุต
เน็ตเวิร์กและทำการเรียนรู้โดยอาศัยอัลกอริทึมแบ็ค
พรอพาเกชันอัลกอริทึม (Backpropagation Algorithm)
การเรียนรู้แบบหลายตัวอย่างย่อย[9, 10]

การปรับค่าน้ำหนักของนิวรอลเน็ตเวิร์กอันดับที่หนึ่ง

การปรับค่าน้ำหนักของเส้นเชื่อมนั้นจะทำทั้งส่วนที่
ระหว่างชั้นนำเข้ากับชั้นซ่อน และระหว่างชั้นซ่อนกับชั้น
การปรับค่าน้ำหนักจะนำหลักการของแบ็กพรอพาเกชัน
มาใช้ โดยนิยามค่าความผิดพลาดรวม (Global Error:

$$E = \sum_{i=1}^{n} E_i \qquad (1)$$

เป็นค่าความผิดพลาดของถุงตัวอย่างแต่ละถุง ซึ่งนิยาม
ขึ้นกับประเภทของถุงตัวอย่างนั้นๆ คือ

$$E_i = \begin{cases} \min\limits_{1\le j\le m_i} E_{ij} & \text{if } B_i = + \\ \max\limits_{1\le j\le m_i} E_{ij} & \text{if } B_i = - \end{cases} \qquad (2)$$

ความผิดพลาดของตัวอย่างย่อยนิยามดังนี้

$$E_{ij} = \begin{cases} 0 & \text{if } (B_i = +) \text{ and } (0.5 \le o_{ij}) \\ 0 & \text{if } (B_i = -) \text{ and } (o_{ij} < 0.5) \\ \frac{1}{2}(o_{ij} - l_i)^2 & \text{otherwise} \end{cases} \qquad (3)$$

$B_i = +$ หมายถึง ถุงตัวอย่างบวก

$B_i = -$ หมายถึง ถุงตัวอย่างลบ

o_{ij} หมายถึง ผลลัพธ์ที่ได้จากถุงตัวอย่างที่ i ตัวอย่างย่อยที่ j (B_{ij})

l_i หมายถึง ค่าเป้าหมายของถุงตัวอย่างที่ i

ในการเรียนแต่ละรอบ ตัวอย่างที่ใช้ในการสอนจะถูกป้อนให้กับเน็ตเวิร์กทีละถุงทีละตัวอย่างย่อย จากนั้นจะทำการคำนวณหาค่าความผิดพลาดของแต่ละตัวอย่างย่อยในถุงนั้นๆ ตามสมการ (3) ถ้ากรณีที่ถุงตัวอย่างที่ป้อนให้กับเน็ตเวิร์กเป็นถุงตัวอย่างบวกและพบว่ามีตัวอย่างย่อยที่ให้ค่าความผิดพลาดเป็น 0 แล้ว ตัวอย่างย่อยที่เหลือไม่จำเป็นต้องป้อนให้กับเน็ตเวิร์กและไม่ต้องทำการปรับค่าน้ำหนักเส้นเชื่อมใดๆ เพราะถือว่าตัวอย่างย่อยตัวนั้นเป็นตัวอย่างที่ทำให้ถุงตัวอย่างเป็นบวกและเน็ตเวิร์กจำแนกได้ถูกต้องแล้ว แต่ถ้าเป็นกรณีอื่นๆค่าความผิดพลาดของถุงตัวอย่างก็จะเป็นไปตามสมการ (2) จากนั้นเมื่อป้อนครบทุกตัวอย่างย่อยในถุงแล้วก็จะนำค่าความผิดพลาดของถุงที่ได้ไปทำการปรับค่าน้ำหนักของเส้นเชื่อมตามวิธีของแบ็กพรอพาเกชัน แล้วจึงทำการป้อนถุงตัวอย่างใหม่เข้าไปและทำตามลำดับขั้นตอนเดิม โดยจะหยุดกระบวนการเรียนรู้เมื่อค่าความผิดพลาดรวมในสมการ (1) มีค่าลดลงจนถึงจุดที่กำหนดไว้หรือเรียนรู้จนครบจำนวนรอบที่ได้กำหนดไว้

3. การทดลองและผลการทดลอง

ในการทดลองเรานำชุดข้อมูล (Dataset) การวิเคราะห์ไฟไนต์เอลิเมนต์ (Finite Element Mesh Design: FEM)[11] มาใช้ทำการทดลอง ปัญหา FEM มีจุดมุ่งหมายเพื่อหากฎที่ใช้

ในการวิเคราะห์ไฟไนต์เอลิเมนต์ในโครงสร้างสำหรับงานทางด้านวิศวกรรม มีกลุ่มความรู้ภูมิหลังเป็นลักษณะต่างๆของโครงสร้าง ประกอบด้วยลักษณะของเส้นเชื่อม เช่น long และ circuit ฯลฯ เงื่อนไขขอบเขต เช่น free และ fixed ฯลฯ โลด เช่น not_loaded และ one_side_loaded ฯลฯ และตัวอย่าง 12 ประเภทด้วยกัน โดยจะแบ่งประเภทตามจำนวนองค์ประกอบ (Element) ที่เหมาะสมของโครงสร้างนั้น ตัวอย่างแต่ละตัวถูกจัดอยู่ในรูปแบบ mesh(Edge, Element) เมื่อ Edge คือชื่อโครงสร้างและ Element คือ จำนวนองค์ประกอบภายในโครงสร้างนั้น รวมจำนวนตัวอย่างทั้งหมด 278 ตัวอย่าง ในการทดลองนั้นใช้วิธี 4-fold cross validation [1] ทำโดยแบ่งข้อมูลทั้งหมดออกเป็น 4 ส่วนเท่าๆกัน ทำการทดลองทั้งหมด 4 ครั้ง ในแต่ละครั้งจะเลือกส่วนหนึ่งใดๆเป็นชุดทดสอบและส่วนที่เหลือ 3 ส่วนจะใช้เป็นชุดสอน

โครงสร้างของเน็ตเวิร์กที่ใช้จะมีจำนวนนิวรอนในชั้นนำเข้าทั้งหมด 130 นิวรอน กำหนดจากความรู้ภูมิหลังที่ได้รับเข้ามา นิวรอนในชั้นผลลัพธ์ 12 นิวรอน กำหนดจากประเภทของตัวอย่าง และใช้จำนวนนิวรอนในชั้นซ่อน 80 นิวรอน(จากการทดลอง) มีค่าอัตราการเรียนรู้เป็น 0.0001 และค่าโมเมนตัมเป็น 0.97 โดยกระบวนการเรียนรู้จะหยุดเมื่อครบ 6000 รอบ หรือค่าความผิดพลาดรวมมีค่าน้อยกว่า 0.05

ตารางที่ 3. ผลเปรียบเทียบเปอร์เซ็นต์ความถูกต้องในการทดสอบกับชุดข้อมูล FEM

อัลกอริทึม	เปอร์เซ็นต์ความถูกต้อง
FONN	59.18
PROGOL	57.80 [6]

ตารางที่ 3 แสดงผลการทดลองที่ได้ในการทดสอบกับชุดข้อมูล FEM เปรียบเทียบระหว่าง FONN (วิธีการที่นำเสนอ) กับ PROGOL ซึ่งเป็นระบบไอแอลพีที่มีประสิทธิภาพมากสุดระบบหนึ่งในปัจจุบัน จะเห็นว่า FONN สามารถเรียนแนวคิดจากชุดข้อมูลตรรกะอันดับที่หนึ่งได้และให้เปอร์เซ็นต์ความถูกต้องในการจำแนกสูงกว่า PROGOL[12]

4. สรุป

งานวิจัยนี้ได้นำเสนอนิวรอลเน็ตเวิร์กแบบไ
ประยุกต์เข้ากับแนวคิดของการเรียนรู้เชิงตรรกะ เรียกว่า นิ
เน็ตเวิร์กอันดับที่หนึ่ง เพื่อเป็นแนวทางหนึ่งในการนำ
จัดการกับข้อจำกัดของการโปรแกรมตรรกะเชิงอุปนัย
นิวรอลเน็ตเวิร์กอันดับที่หนึ่งมีข้อดีที่สามารถรับอิน
รูปแบบของตรรกะอันดับที่หนึ่งได้โดยตรงและมีข้อ
นิวรอลเน็ตเวิร์กซึ่งทนทานต่อข้อมูลที่มีสัญญาณรบกวนได้
ด้วย

5. รายการอ้างอิง

[1] T. M. Mitchell, *Machine Learning*: The McGraw
Companies Inc., 1997.

[2] N. Lavrac and S. Dzeroski, *Inductive L
Programming Techniques and Applications*:
Horwood, New York, 1994.

[3] S.-H. Nienhuys-Cheng and R. d. Wolf, *Foundati
Inductive Logic Programming*: Springer-Verlag
York, Inc., 1997.

[4] C. M. Bishop, *Neural Networks for Pa
Recognition*: Oxford University Press, 1995.

[5] R. Parekh and V. Honavar, "Constructive Th
Refinement in Knowledge Based Neural Networ
Proceedings of the International Joint Conferenc
Neural Networks, Anchorage, Alaska, 1998.

[6] B. Kijsirikul, S. Sinthupinyo, and
Chongkasemwongse, "Approximate Match of
Using Backpropagation Neural Networks," *Mac
Learning Journal*, vol. 44, pp. 273-299, 2001.

[7] Y. Chevaleyre and J.-D. Zucker, "A Framewor
Learning Rules from Multiple Instance Data,"
European Conference on Machine Learning, Frei
Germany, 2001.

[8] X. Huang, S.-C. Chen, and M.-L. Shyu, "An O
Multiple Instance Learning Framework and
Application in Drug Activity Prediction Proble
Proceedings of the Third IEEE Symposium
BioInformatics and BioEngineering (BIBE
Bethesda, Maryland, 2003.

[9] P. Wattuya, A. Rungsawang, and B. Kijsir
"Multiple-Instance Neural Network with Str
Boundary Criteria," The 7th National Comp
Science and Engineering Conference, Chon
Thailand, 2003.

[10] Z.-H. Zhou and M.-L. Zhang, "Neural Network
Multi-Instance Learning," Proceedings of
International Conference on Intelligent Informati
Technology, Beijing, China, 2002.

[11] B. Dolsak and S. Muggleton, "The Application
Inductive Logic Programming to Finite Element M
Design," in *Inductive Logic Programming*,
Muggleton, Ed.: Academic Press, 1992, pp. 453–47

[12] S. Roberts, *An Introduction to Progol. Techni
Manual*: University of York, 1997.

เทคนิคการแตกครึ่งตามสารสนเทศ สำหรับซัพพอร์ตเวกเตอร์แมชชีนแบบหลายประเภท

An Information-Based Dichotomization Technique for Multiclass Support Vector Machines

ทมศิริ สงศิริ และ บุญเสริม กิจศิริกุล
ภาควิชาวิศวกรรมคอมพิวเตอร์
วิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ถนนพญาไท ปทุมวัน กรุงเทพฯ 10330
boonsiri@hotmail.com and boonserm.k@chula.ac.th

ฐิมาพร เพชรแก้ว
สำนักวิชาสารสนเทศศาสตร์
มหาวิทยาลัยวลัยลักษณ์ เลขที่ 222 ต.ไทรบุรี
อ.ท่าศาลา จ.นครศรีธรรมราช 80160
Email: pthimapo@wu.ac.th

บทคัดย่อ

การแก้ปัญหาการจำแนกแบบหลายประเภทด้วยซัพพอร์ตเวกเตอร์แมชชีน โดยส่วนใหญ่จะพิจารณาเป็นปัญหาของการนำตัวจำแนกแบบสองประเภทหลายตัวมาใช้ร่วมกัน ซึ่งการดังกล่าวยังมีข้อจำกัดในเรื่องของความถูกต้องและระยะเวลาครั้งในการจำแนก บทความนี้นำเสนอวิธีการใหม่ในการจำแนกข้อมูลด้วยเทคนิคการแตกครึ่งตามสารสนเทศโดยสร้างต้นไม้สำหรับการจำแนกแบบไบนารีจากตัวจำแนกแบบสองประเภท ซึ่งแต่ละโนดของต้นไม้จะเป็นตัวจำแนกแบบสองประเภทที่มีค่าเอนโทรปีต่ำที่สุด วิธีนี้สามารถลดจำนวนตัวจำแนกแบบสองประเภทที่ใช้ในการจำแนกลงได้ในอัลกอริทึมของจำนวนประเภทซึ่งต่ำกว่าวิธีอื่น จากผลการทดลองแสดงให้เห็นว่าวิธีนี้สามารถลดจำนวนครั้งของการจำแนกลงได้ และยังคงให้ค่าความถูกต้องใกล้เคียงกับวิธีอื่น

คำสำคัญ: การแตกครึ่งตามสารสนเทศ, ซัพพอร์ตเวกเตอร์แมชชีนแบบหลายประเภท, เอนโทรปี

Abstract

Approaches for solving a multiclass classification problem by Support Vector Machines (SVMs) are typically to consider the problem as combination of two-class classification problems. Previous approaches have some limitations in classification accuracy and evaluation time. This paper proposes a novel method that employs information-based dichotomization for constructing a binary classification tree. Each node of the tree is a binary SVM with the minimum entropy. Our method can reduce the number of binary SVMs used in the classification to the logarithm of the number of classes which is lower than previous methods. The experimental results show that the proposed method takes lower evaluation time while it maintains accuracy compared to other methods.

Key Words: Information-Based Dichotomization, Multiclass Support Vector Machines, Entropy

1. บทนำ

ซัพพอร์ตเวกเตอร์แมชชีนเป็นเทคนิคการเรียนรู้ที่ใช้คุณสมบัติเชิงเรขาคณิตในการคำนวณหาระนาบหลายมิติ (Hyperplane) ที่ดีที่สุดในการแยกข้อมูลออกจากกัน โดยในช่วงแรกเริ่มนั้นเทคนิคนี้มีข้อจำกัดอยู่ที่สามารถจำแนกข้อมูลได้เพียงสองประเภทเท่านั้น แต่ในปัญหาการจำแนกส่วนใหญ่มักเป็นแบบหลายประเภท ต่อมาจึงมีผู้พัฒนาเทคนิคนี้ให้สามารถใช้ได้กับการจำแนกแบบหลายประเภท วิธีการส่วนใหญ่มักจะเป็นการนำฟังก์ชันที่จำแนกแบบสองประเภทหลายฟังก์ชันมาใช้ร่วมกัน เช่น การจำแนกแบบหนึ่งต่อที่เหลือ (One-against-the-rest) เป็นการเปรียบเทียบประเภทข้อมูลหนึ่งกับประเภทอื่นที่เหลือทั้งหมด และการจำแนกแบบ

หนึ่งต่อหนึ่ง (One-against-one) ซึ่งเป็นการเปรียบเทียบประเภทข้อมูลหนึ่งกับประเภทข้อมูลอื่นทีละประเภท [6]

อัลกอริทึมแมกซ์วินเป็นหนึ่งในวิธีการจำแนกแบบหนึ่งต่อหนึ่งซึ่งให้ค่าความถูกต้องสูงกว่าการจำแนกแบบหนึ่งต่อที่เหลือ แต่ใช้เวลาในการจำแนกนานกว่า [5] ต่อมา Phetkaew, et al. ได้เสนอวิธีอาร์เอดีเอจี (Reordering Adaptive Directed Acyclic Graph (RADAG)) [8] ซึ่งวิธีนี้จะมีการเลือกลำดับของโนดที่เหมาะสมซึ่งมีโอกาสที่จะจำแนกผิดพลาดน้อย โดยใช้อัลกอริทึมของการจับคู่สมบูรณ์แบบน้ำหนักน้อยสุด (Minimum-Weight Perfect Matching) [4] และมีการจัดเรียงโนดใหม่ในทุกชั้นของการจำแนก ทำให้มีค่าความผิดพลาดลดลงจากวิธีเอดีเอจี [9] อย่างไรก็ตามในวิธีที่กล่าวมาข้างต้นสำหรับการจำแนกข้อมูล k ประเภทใดๆ วิธีอาร์เอดีเอจีต้องใช้จำนวนครั้งของการจำแนกข้อมูลเป็น k-1 ครั้ง เนื่องจากในการจำแนกข้อมูลแต่ละครั้งสามารถตัดข้อมูลที่ไม่ถูกต้องออกไปได้เพียง 1 ประเภทต่อครั้งเท่านั้น ต่อมา Kijsirikul & Boonsirisumpun ได้เสนอการสร้างต้นไม้สำหรับการจำแนกแบบหลายประเภทด้วยวิธีการแตกครึ่งแบบสมดุล (Balanced Dichotomization Classification) [7] ที่ทำการจำแนกข้อมูลโดยค้นหาระนาบที่แบ่งข้อมูลในตำแหน่งที่สมดุลที่สุดของข้อมูลทั้งหมดในแต่ละรอบมาจำแนก เพื่อให้การจำแนกข้อมูลแต่ละครั้งสามารถตัดประเภทที่ไม่ถูกต้องออกไปได้มากกว่า 1 ประเภท ทั้งนี้ในการค้นหาระนาบนั้นจะเลือกระนาบที่แบ่งข้อมูลแล้วให้ค่าจำนวนประเภทซึ่งตกอยู่ในแต่ละด้านของระนาบมีค่าใกล้เคียงกันมากที่สุด ภายใต้สมมติฐานที่ว่าความน่าจะเป็นในการเกิดข้อมูลในแต่ละประเภทของข้อมูลสอนมีค่าใกล้เคียงกับข้อมูลทดสอบ วิธีนี้จะมีความเหมาะสมกรณีที่ข้อมูลแต่ละประเภทมีความน่าจะเป็นในการเกิดที่เท่ากัน โดยวิธีนี้จะพยายามสร้างต้นไม้ให้ทั้งสองกิ่งมีความสมดุลที่สุด ซึ่งส่งผลให้ค่าจำนวนครั้งในการจำแนกข้อมูลแต่ละประเภทจะมีค่าใกล้เคียงกัน ในความเป็นจริงข้อมูลที่นำมาจำแนกอาจไม่ได้มีความน่าจะเป็นในการเกิดเท่ากัน หากยึดวิธีสร้างต้นไม้ที่ให้จำนวนครั้งในการจำแนกข้อมูลแต่ละประเภทที่ใกล้เคียงกันโดยไม่พิจารณาความน่าจะเป็นในการเกิดข้อมูลแต่ละประเภทประกอบอาจไม่ได้ต้นไม้สำหรับจำแนกที่ให้จำนวนครั้งในการจำแนกเฉลี่ยต่ำที่สุด ซึ่งเป็นการสะสมความผิดพลาดในการจำแนกโดยไม่จำเป็น

บทความนี้จะนำเสนอวิธีการใหม่ในการจำแนกข้อมูลโดยสร้างต้นไม้สำหรับการจำแนกแบบหลายประเภทด้วยเทคนิคการแตกครึ่งตามสารสนเทศโดยการนำความน่าจะเป็นในการเกิดข้อมูลแต่ละประเภทมาใช้ในการค้นหาระนาบที่แบ่งข้อมูล ในทางทฤษฎีวิธีนี้จะสามารถลดจำนวนครั้งในการจำแนกลงเหลือประมาณ $\log k$ ครั้ง ซึ่งเป็นการลดเวลาของการจำแนกเฉลี่ยลงได้โดยเฉพาะอย่างยิ่งกรณีปัญหาการจำแนกที่มีจำนวนประเภทเป็นจำนวนมาก

2. ซัพพอร์ตเวกเตอร์แมชชีนและทฤษฎีสารสนเทศ (Support Vector Machines & Information Theory)

เทคนิคการแตกครึ่งตามสารสนเทศสำหรับซัพพอร์ตเวกเตอร์แมชชีนแบบหลายประเภทมีทฤษฎีที่เกี่ยวข้องดังนี้

2.1 แนวคิดพื้นฐานของซัพพอร์ตเวกเตอร์แมชชีน

แนวคิดหลักในการจำแนกข้อมูล ของซัพพอร์ตเวกเตอร์แมชชีนก็คือ การสร้างระนาบหลายมิติที่ใช้ในการแบ่งข้อมูลออกเป็นสองประเภท

2.1.1 ซัพพอร์ตเวกเตอร์แมชชีนแบบเชิงเส้น (Linear support vector machine)

สมมติมีเซตของข้อมูล D ที่ประกอบด้วยตัวอย่างจำนวน l ตัวในปริภูมิอันดับ n ที่มี 2 ประเภท (+1 และ -1)

$$D=\{(\mathbf{x}_k, y_k) \mid k\in\{1,..,l\}, \mathbf{x}_k\in\Re^n, y_k\in\{+1,-1\}\} \quad (1)$$

ระนาบหลายมิติในปริภูมิอันดับ n ถูกกำหนดโดย $(\mathbf{w},b)$ เมื่อ $\mathbf{w}$ คือเวกเตอร์ในปริภูมิอันดับ n ที่ตั้งฉากกับระนาบหลายมิติและ b คือค่าคงที่ระนาบหลายมิติ $(\mathbf{w}\cdot\mathbf{x})+b$ จะแบ่งข้อมูลได้ก็ต่อเมื่อ

$$\begin{aligned}(\mathbf{w}\cdot\mathbf{x}_i)+b>0 \quad &\text{if} \quad y_i=+1\\ (\mathbf{w}\cdot\mathbf{x}_i)+b<0 \quad &\text{if} \quad y_i=-1.\end{aligned} \quad (2)$$

ต้องการค่า w และ b ที่ทำให้จุดที่อยู่ใกล้ระนาบ
มากที่สุดมีระยะห่าง $1/|w|$ แล้วจะได้

$$(w \cdot x_i) + b \geq 1 \quad \text{if} \quad y_i = +1$$
$$(w \cdot x_i) + b \leq -1 \quad \text{if} \quad y_i = -1 \qquad (3)$$

หรือเขียนได้

$$y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1 \quad \forall i. \qquad (4)$$

ในการค้นหาระนาบหลายมิติที่ใช้แบ่งข้อมูลดีที่สุดจะต้องค้นหาระนาบหลายมิติที่มีระยะห่างระหว่างข้อมูลที่ใช้สอนกับระนาบหลายมิติที่น้อยที่สุดมีค่ามากที่สุด ระยะห่างระหว่างข้อมูลตัวอย่างสองตัวจากประเภทที่แตกต่างกันมีค่าเท่ากับ

$$d(\mathbf{w}, b) = \min_{\{x_i | y_i = +1\}} \frac{(\mathbf{w} \cdot \mathbf{x}_i) + b}{|\mathbf{w}|} - \max_{\{x_i | y_i = -1\}} \frac{(\mathbf{w} \cdot \mathbf{x}_i) + b}{|\mathbf{w}|}. \qquad (5)$$

จากสมการที่ (3) ค่าที่น้อยที่สุดและมากที่สุดที่เป็นไปได้คือ ± 1 ดังนั้นจำเป็นต้องเพิ่มค่าของฟังก์ชัน

$$d(\mathbf{w}, b) = \frac{1}{|\mathbf{w}|} - \frac{-1}{|\mathbf{w}|} = \frac{2}{|\mathbf{w}|}. \qquad (6)$$

ให้สูงที่สุด ดังนั้นปัญหานี้จึงเท่ากับ

- ลดค่าของ $|w|^2/2$ ให้ต่ำที่สุดโดยขึ้นกับเงื่อนไขต่อไปนี้

  (1) $y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1 \quad \forall i.$

สำหรับกรณีที่ไม่สามารถแบ่งข้อมูลได้โดยเงื่อนไขข้างต้น จะต้องปรับเงื่อนไขใหม่โดยเพิ่มพจน์ค่าปรับซึ่งประกอบด้วยผลรวมของความคลาดเคลื่อน ξ_i จากขอบเขตเข้าไปในเงื่อนไข ดังนั้นปัญหาตอนนี้คือ

- ลดค่าของ $\frac{|\mathbf{w}|^2}{2} + C\sum_{i=1}^{l} \xi_i$ ให้ต่ำที่สุด โดยขึ้นกับเงื่อนไขต่อไปนี้

  (1) $y_i[(\mathbf{w} \cdot \mathbf{x}_i) + b] \geq 1 - \xi_i,$

  (2) $\xi_i \geq 0 \quad \forall i.$

พจน์ค่าปรับสำหรับตัวอย่างข้อมูลสอนที่ทำนายผิดพลาดถูกเพิ่มน้ำหนักโดยค่าคงที่ C การเลือกค่า C สูงจะมีผลให้เพิ่มค่าของความคลาดเคลื่อน ξ_i และเพิ่มการคำนวณโดยทำให้การค้นหาหนทาง ที่จะลดจำนวนตัวอย่างข้อมูลสอนที่ทำนายผิดพลาดเพิ่มขึ้น นำลากรองเกรียนมาใช้กับปัญหานี้ สามารถแปลงปัญหาเป็น

- ลดค่าของฟังก์ชันนี้ให้ต่ำที่สุด

  $$L(\mathbf{w}, b, \alpha) = \sum_{i=1}^{l} a_i \quad \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \qquad (7)$$

  โดยขึ้นกับเงื่อนไขต่อไปนี้

  (1) $0 \leq \alpha_i \leq C, \forall i$

  (2) $\sum_{i=1}^{l} \alpha_i y_i = 0$

เมื่อ α_i เรียกว่าตัวคูณลากรอง ตัวอย่างข้อมูลสอนแต่ละตัวจะมีตัวคูณลากรองหนึ่งตัว ตัวอย่างที่มีค่า $\alpha_i > 0$ เรียกว่าซัพพอร์ตเวกเตอร์ และเป็นซัพพอร์ตเวกเตอร์ที่ทำให้ค่าทางขวามือของสมการที่ (4) เท่ากับ 1 ส่วนตัวอย่างอื่นๆที่มี $\alpha_i = 0$ สามารถตัดออกไปจากเซตของตัวอย่างข้อมูลสอนได้ โดยไม่มีผลกระทบใดๆ ต่อผลลัพธ์ของระนาบหลายมิติ

ให้ α^0 ซึ่งเป็นเวกเตอร์ในปริภูมิอันดับ 1 แทนค่าต่ำสุดของ $L(\mathbf{w},b,\alpha)$ ถ้า $\alpha_i^0 > 0$ แล้ว $\mathbf{x}_i$ เป็นซัพพอร์ตเวกเตอร์ของระนาบหลายมิติที่แบ่งแยกดีสุด $(\mathbf{w}^0, b^0)$ สามารถเขียนในเทอมของ α^0 และข้อมูลสอน โดยเฉพาะอย่างยิ่งในเทอมของซัพพอร์ตเวกเตอร์ได้ดังนี้

$$\mathbf{w}^0 = \sum_{i=1}^{l} \alpha_i^0 y_i \mathbf{x}_i = \sum_{\text{support vectors}} \alpha_i^0 y_i \mathbf{x}_i. \qquad (8)$$

$$b^0 = 1 - \mathbf{w}^0 \cdot \mathbf{x}_i \text{ for } \mathbf{x}_i \text{ with } y_i = 1 \text{ and } 0 < \alpha_i < C \qquad (9)$$

ระนาบหลายมิติที่แบ่งแยกดีสุดจะจำแนกจุดต่างๆ ตามเครื่องหมายของผลลัพธ์ของฟังก์ชัน $f(x)$

$$f(\mathbf{x}) = \text{sign}\left(\mathbf{w}^0 \cdot \mathbf{x} + b^0\right)$$
$$= \text{sign}\left(\sum_{\text{support vectors}} \alpha_i^0 y_i (\mathbf{x}_i \cdot \mathbf{x}) + b^0\right).$$

ซัพพอร์ตเวกเตอร์ $\mathbf{x}_i$ ที่มี $\alpha_i^0 = C$ อาจจะถูกจำแนกผิดพลาดหรือไม่ก็ได้ แต่ $\mathbf{x}_i$ ตัวอื่นๆ จะจำแนกได้อย่างถูกต้อง

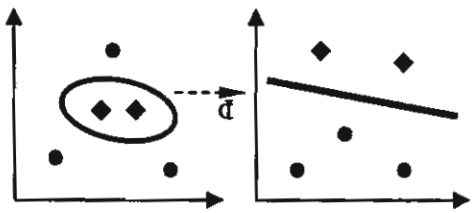
2.1.2 ซัพพอร์ตเวกเตอร์แมชชีนแบบไม่เชิงเส้น (Non-linear Support vector machine)

อัลกอริทึมที่ได้กล่าวมาแล้วใช้ได้กับข้อมูลที่สามารถแบ่งด้วยระนาบหลายมิติแบบเชิงเส้นเท่านั้น แต่กับข้อมูลที่ไม่อาจแบ่งแบบเชิงเส้นได้ ซัพพอร์ตเวกเตอร์แมชชีนแก้ปัญหานี้โดยแมปข้อมูลตัวอย่าง ไปยังปริภูมิอันดับสูง โดยเลือกใช้ฟังก์ชันการแมปที่ไม่เป็นเชิงเส้น นั่นคือ เราเลือกฟังก์ชันในการแมป $\Phi:\Re^n \mapsto \mathrm{H}$ เมื่อปริภูมิของ H มีอันดับสูงกว่าปริภูมิอันดับ n เราสามารถค้นหาระนาบที่แบ่งแยกดีสุดในปริภูมิอันดับสูงที่เทียบเท่ากับการแยกที่ไม่เป็นเชิงเส้นใน $\Re^n$

ตัวอย่างข้อมูลสอนที่ใช้ในสมการที่ (7) อยู่ในรูปของผลคูณเชิงสเกลาร์ระหว่างเวกเตอร์ ดังนั้นในปริภูมิอันดับสูงเราสามารถจัดการกับข้อมูลในรูปของ $\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$. ถ้าปริภูมิของ H มีอันดับสูงจะทำให้ยุ่งยาก หรือใช้การคำนวณมาก อย่างไรก็ตามถ้าเรามีฟังก์ชันเคอร์เนลที่สามารถคำนวณ $k(\mathbf{x}_i,\mathbf{x}_j) = \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$ แล้วเราสามารถใช้ฟังก์ชันนี้แทนที่ $\mathbf{x}_i\cdot\mathbf{x}_j$ ทุกๆที่ในการคำนวณ และไม่จำเป็นต้องรู้ฟังก์ชันที่ใช้แมป Φ จริงๆ ฟังก์ชันเคอร์เนลที่นิยมใช้ได้แก่

Polynomial degree d: $k(\mathbf{x},\mathbf{y})=\left|\mathbf{x}\cdot\mathbf{y}+1\right|^d$ (11)

Radial basis function: $k(\mathbf{x},\mathbf{y})= e^{-\|\mathbf{x}-\mathbf{y}\|^2/c}$ (12)



รูปที่1. แนวคิดการแมปข้อมูลไม่เชิงเส้นไปสู่ปริภูมิอันดับสูง

2.1.3 ประสิทธิภาพโดยนัยทั่วไปของซัพพอร์ต เวกเตอร์แมชชีน

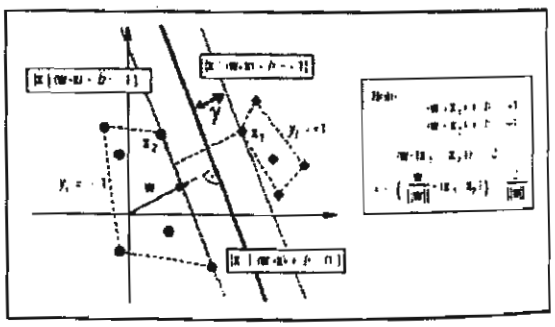
เราสามารถกำหนดขอบเขตความผิดพลาดของซัพพอร์ตเวกเตอร์ได้โดยกำหนดชนิดของฟังก์ชันเป็นค่าตัวเลขจำนวนจริงภายในลูกบอลที่มีรัศมี R ค่ารัศมีไม่เกิน $\Re^n$ เป็นฟังก์ชันดังนี้ $F=\{\mathbf{x}\mapsto \mathbf{w}\cdot\mathbf{x}:\|\mathbf{w}\|\le 1,\|\mathbf{x}\|\le R\}$ จะมีค่าคงที่ c ที่สำหรับการกระจายทุกประเภท ด้วยความน่าจะเป็นอย่างน้อย 1-δ บนตัวอย่าง z ที่เลือกมาอย่างอิสระจากกัน m ตัว ถ้าตัวจำแนก $h=\mathrm{sgn}(f)\in\mathrm{sgn}(F)$ มีค่าระยะห่างระหว่างระนาบหลายมิติ กับข้อมูลที่ใช้สอนอย่างน้อย γ (ดูรูปที่ 2) สำหรับตัวอย่างทุกตัวใน z แล้ว ความผิดพลาดของตัวจำแนก h จะไม่เกินค่า

$$\frac{c}{m}\left(\frac{R^2}{\gamma^2}\log^2 m+\log\left(\frac{1}{\delta}\right)\right). \qquad (13)$$

นอกเหนือจากนี้แล้วด้วยความน่าจะเป็นอย่างน้อย 1-δ ทุกตัวจำแนก $h\in\mathrm{sgn}(F)$ จะมีความผิดพลาดไม่เกิน

$$\frac{k}{m}+\sqrt{\frac{c}{m}\left(\frac{R^2}{\gamma^2}\log^2 m+\log\left(\frac{1}{\delta}\right)\right)} \qquad (14)$$

เมื่อ k คือจำนวนตัวอย่างข้อมูลใน z ที่มีระยะห่างระหว่างระนาบหลายมิติกับซัพพอร์ตเวกเตอร์อย่างน้อย γ



รูปที่ 2. ระยะห่างระหว่างระนาบหลายมิติกับข้อมูลที่ใช้สอน

ทฤษฎีสารสนเทศ (Information Theory)

สารสนเทศของข้อมูลขึ้นอยู่กับความน่าจะเป็นของ ซึ่งสามารถวัดอยู่ในรูปของบิตจากสูตร

$$\text{สารสนเทศของข้อมูล} = -\log_2 P \qquad (15)$$

เมื่อ P คือความน่าจะเป็นของข้อมูล

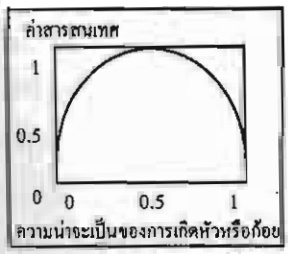
ให้ชุดของข้อมูล M ประกอบด้วยค่าที่เป็นไปได้คือ $(m_1, m_2, \ldots, m_n)$ และให้ความน่าจะเป็นที่จะเกิดค่า m_i มีค่าเท่ากับ $P(m_i)$ จะได้ว่าค่าสารสนเทศของ M หรือค่าเอนโทรปีของ M เขียนแทนด้วย $I(M)$ คำนวณได้จากสูตร

$$I(M) = \sum_{i=1}^{n} -P(m_i)\log_2 P(m_i) \qquad (16)$$

ความหมายของค่าสารสนเทศกรณีมีค่าน้อย หมายถึง ข้อมูลในชุดนั้นมีความแตกต่างกันน้อย หรือเกือบจะเป็นประเภทเดียวกัน แต่กรณีค่าสารสนเทศสูงจะบ่งบอกว่าข้อมูลในชุดนั้นมีความแตกต่างกันมากหรือประกอบด้วยตัวอย่างหลายประเภทที่มีจำนวนใกล้เคียงกัน ตัวอย่างเช่น การโยนหัวโยนก้อย ชุดข้อมูล M จะประกอบด้วยค่าที่เป็นไปได้ {หัว, ก้อย} จะได้ว่าค่าสารสนเทศของการโยนหัวก้อยเท่ากับ

$$-P(\text{หัว})\log_2(P(\text{หัว})) - P(\text{ก้อย})\log_2(P(\text{ก้อย}))$$

เมื่อพิจารณากรณีที่โยนออกหัวหมดหรือก้อยหมด ค่าสารสนเทศจะเป็น 0 และค่าสารสนเทศจะค่อยๆ เพิ่มขึ้นจนสูงที่สุด เมื่อความน่าจะเป็นของการเกิดหัวเท่ากับความน่าจะเป็นของการเกิดก้อยหรือสามารถแสดงได้ดังรูปที่ 3



รูปที่ 3. ความสัมพันธ์ระหว่างความน่าจะเป็นและค่าสารสนเทศ

3. งานวิจัยที่เกี่ยวข้อง

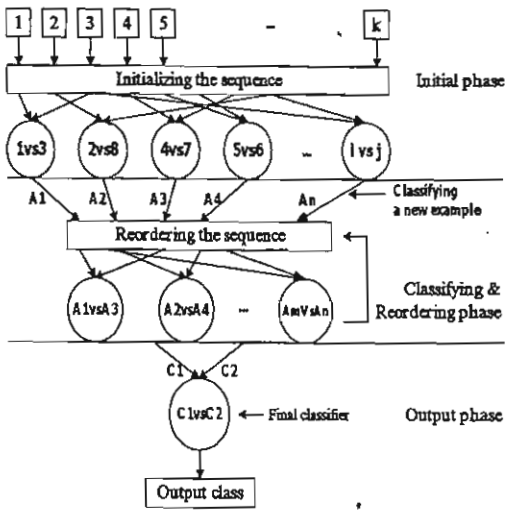
3.1 วิธีแมกซ์วิน (Max Wins)

อัลกอริทึมแมกซ์วินจะทำงานโดยอาศัยฟังก์ชันพื้นฐานของการจำแนกแบบหนึ่งต่อหนึ่ง สร้างตัวจำแนกข้อมูลทั้งหมด จำนวน $k(k\text{-}1)/2$ ตัว โดยที่ k คือจำนวนประเภทข้อมูลทั้งหมด ตัวจำแนกแต่ละตัวจะถูกสอนโดยข้อมูลจาก 2 ประเภท i และ j ใดๆ ตามเงื่อนไขดังนี้

$$\min_{w^{ij}, b^{ij}, \xi^{ij}} \quad \frac{1}{2}(w^{ij})^T w^{ij} + C\sum_t \xi_t^{ij}(w^{ij})^T$$
$$(w^{ij})^T \Phi(x_t) + b^{ij} \geq 1 - \xi_t^{ij}, \quad \text{if } y_t = i$$
$$(w^{ij})^T \Phi(x_t) + b^{ij} \leq -1 + \xi_t^{ij}, \quad \text{if } y_t = j \; \xi_t^{ij} \geq 0. \qquad (17)$$

แล้วใช้ฟังก์ชัน $sign((w^{ij})^T\Phi(x) + b^{ij})$ จำแนกให้คำตอบว่าตัวอย่าง x ใดๆ อยู่ในประเภท i หรือ j หลังจากได้คำตอบทั้งหมดแล้ว วิธีแมกซ์วินจะนำคำตอบที่ได้จากตัวจำแนกทั้ง $k(k\text{-}1)/2$ ตัว มาโหวตรวมกันโดยประเภทที่ได้ผลโหวตสูงที่สุด ก็จะเป็นคำตอบของแมกซ์วิน ส่วนในกรณีที่มีผลโหวตสูงสุดมากกว่า 1 ประเภท ก็จะทำการสุ่มคำตอบจากประเภทเหล่านั้น

3.2 วิธีอาร์เอดีเอจี (Reordering Adaptive Directed Acyclic Graph (RADAG))



รูปที่ 4. โครงสร้างการจำแนกข้อมูลของอาร์เอดีเอจี

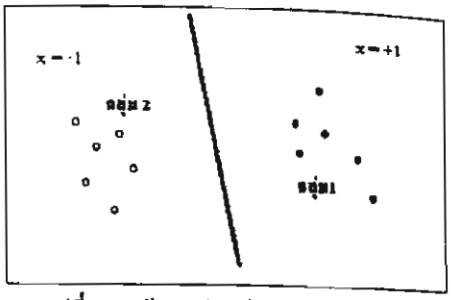
Phetkaew, et al. [8] นำเสนอวิธีการใหม่ เรียกว่า อาร์เอดีเอจี เพื่อปรับปรุงความถูกต้องของวิธีเอดีเอจีเดิม [9] โดยในโครงสร้างของการจำแนกนั้นจะเป็นรูปคล้ายสามเหลี่ยมคว่ำเหมือนวิธีเอดีเอจี แต่จะมีการเลือกลำดับที่ทำให้เกิดความผิดพลาดน้อยที่สุดของการจำแนกในแต่ละชั้นโดยวิธีการเลือกจะใช้อัลกอริทึมของการจับคู่สมบูรณ์แบบน้ำหนักน้อยสุด (Minimum-Weight Perfect Matching) [4] เพื่อจับคู่ให้ค่าขอบเขตความผิดพลาดรวมในแต่ละชั้นมีค่าน้อยที่สุด โดยค่าขอบเขตความผิดพลาดของฟังก์ชันแบบสองประเภทแต่ละตัวหาได้จากค่าประสิทธิภาพโดยนัยทั่วไปของซัพพอร์ตเวกเตอร์แมชชีน (Generalization Performance of Support Vector Machine) [2] และเมื่อมีการจำแนกผ่านไปสู่ชั้นใหม่ก็จะมีการเรียงลำดับการจำแนกใหม่ (Reordering) อีกจนเหลือประเภทเดียวในชั้นสุดท้ายซึ่งเป็นคำตอบของอาร์เอดีเอจี

3.3 วิธีจำแนกข้อมูลแบบแตกครึ่งแบบสมดุล (Balanced Dichotomization Classification)

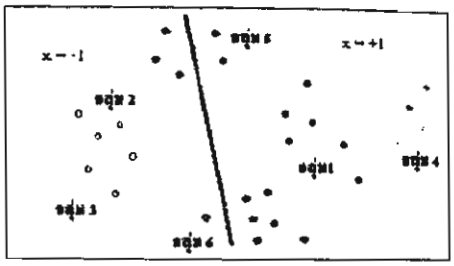
Kijsirikul & Boonsirisumpun [7] เสนอวิธีจำแนกข้อมูลแบบแตกครึ่งแบบสมดุล (Balanced Dichotomization) โดยมีแนวคิดในการจำแนกข้อมูลด้วยระนาบหลายมิติที่อยู่ในตำแหน่งกึ่งกลางที่สุดของข้อมูลมาจำแนก เพื่อลดจำนวนครั้งของการจำแนกลงโดยมีขั้นตอนในการจำแนกข้อมูลดังนี้

3.3.1 การค้นหาตำแหน่งของระนาบหลายมิติ

ในการสร้างฟังก์ชันของระนาบหลายมิติในการจำแนกแบบสองประเภทตามปกตินั้น เราจะได้ตำแหน่งของระนาบอยู่กึ่งกลางระหว่างสองประเภทข้อมูลที่เราใช้สอน โดยที่ตำแหน่งของข้อมูลทั้งสองประเภทจะอยู่คนละด้านของระนาบตัว อย่างเช่น ถ้าสร้างระนาบระหว่าง ประเภท 1 กับ ประเภท 2 ก็จะได้ระนาบหลายมิติอยู่กึ่งกลางระหว่าง 1 กับ 2 และได้ตำแหน่งของประเภท 1 อยู่ในด้าน $x = +1$ และตำแหน่งของประเภท 2 อยู่ในด้าน $x=-1$ เป็นต้น (ดูรูปที่ 5 (ก)) แต่จะไม่รู้ตำแหน่งเทียบกับประเภทข้อมูลอื่น ทำให้การจำแนกข้อมูล 1 ครั้งตัดประเภทที่ไม่ถูกต้องออกไปได้เพียง 1 ประเภท ในวิธีนี้จึงมีการค้นหาตำแหน่งของระนาบเทียบกับประเภทข้อมูลอื่นเพิ่มโดยนำฟังก์ชันของระนาบแบบสองประเภทที่มีอยู่แล้วไปคำนวณหาตำแหน่งของประเภทข้อมูลอื่นเพิ่มเติมก็จะได้ตำแหน่งของระนาบเมื่อเทียบกับข้อมูลทั้งหมด (ดูรูปที่ 5 (ข))



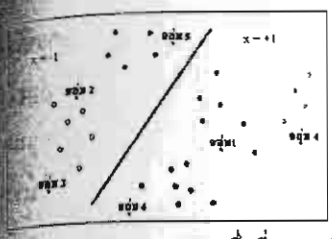
รูปที่ 5 (ก) ข้อมูลเรียนรู้ของระนาบตามปกติ



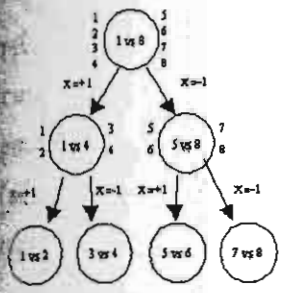
รูปที่ 5 (ข) ข้อมูลเรียนรู้เมื่อมีการค้นหาตำแหน่งเทียบกับประเภทข้อมูลอื่นเพิ่ม

3.3.2 การค้นหาระนาบแตกครึ่งแบบสมดุล

เมื่อทำการคำนวณเพื่อค้นหาตำแหน่งของระนาบหลายมิติเทียบกับประเภทข้อมูลอื่นตามขั้นตอนข้างต้นแล้ว ในการจำแนกข้อมูลแบบแตกครึ่งแบบสมดุลจะทำการค้นหาระนาบที่สามารถแบ่งข้อมูลในตำแหน่งสมดุลที่สุดของข้อมูลทั้งหมดมาจำแนกซึ่งจะเลือกระนาบที่แบ่งข้อมูลในแต่ละด้านของระนาบมีจำนวนประเภทที่ใกล้เคียงกัน เพื่อให้ในการจำแนกข้อมูลแต่ละครั้งสามารถตัดประเภทข้อมูลที่ไม่ถูกต้องออกไปได้มากที่สุด นั่นคือครึ่งหนึ่งของประเภทข้อมูลทั้งหมดในรอบนั้นนั่นเอง



รูปที่ 6 ตัวอย่างระนาบแตกครึ่งที่สมดุลที่สุด



รูปที่ 7 การจำแนกแบบแตกครึ่งแบบสมดุลที่ดีที่สุดสำหรับปัญหา 8 ประเภท

โครงสร้างของการจำแนกโดยใช้ระนาบแตกครึ่งที่สมดุลที่สุดนั้น ถ้าสามารถหาระนาบที่แบ่งข้อมูลเหลือครึ่งหนึ่งได้ทุกครั้งจะทำให้จำนวนครั้งของการจำแนกที่ดีที่สุดลดลงเหลือ $\log_2 k$ ครั้งในปัญหา k ประเภท ดังตัวอย่างในรูปที่ 7

ข้อจำกัดของการควบคุมความผิดพลาด วิธีการแตกครึ่งแบบสมดุลมีการกำหนดขอบเขตความผิดพลาดของการจำแนกข้อมูล โดยการกำหนดให้ระนาบหลายมิติที่จะนำมาหาระนาบสมดุลที่สุดต้องมีขอบเขตความผิดพลาดของช่วงค่าประสิทธิภาพโดยทั่วไปอยู่ในช่วงที่กำหนดในแต่ละชั้น เพื่อควบคุมค่าความถูกต้อง

4. เทคนิคการจำแนกข้อมูลแบบแตกครึ่งตามสารสนเทศ

ในหัวข้อนี้จะนำเสนอแนวคิดที่ทำให้การจำแนกข้อมูลแต่ละครั้งสามารถตัดประเภทข้อมูลที่ไม่ถูกต้องออกไปได้มากที่สุด โดยพิจารณาความน่าจะเป็นในการเกิดข้อมูลแต่ละประเภทประกอบเพื่อให้ได้จำนวนครั้งเฉลี่ยของการจำแนกลดลง

กรณีตัวอย่างจากปัญหาการเข้ารหัสข้อมูลของอักขระ 4 ตัว คือ A, B, C และ D โดยสมมติให้ความน่าจะเป็นของการเกิดข้อมูลแต่ละประเภทมีเท่ากัน จะได้ว่าจำนวนบิตที่น้อยที่สุดในการแทนอักขระเหล่านี้คือ 2 บิต โดยแทนแต่ละอักขระดังนี้

A	แทนด้วย	00
B	แทนด้วย	01
C	แทนด้วย	10
D	แทนด้วย	11

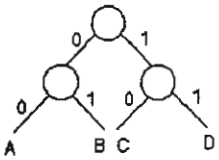
ตัวอย่าง หากต้องการเข้ารหัสข้อมูลของ ABACADAB จะได้ 0001001000110001 จำนวนบิตที่ใช้ คือ 16 เมื่อพิจารณาข้อมูลจากตัวอย่าง ตลอดจนข้อมูลที่เป็นข้อความทั่วไปในชีวิตประจำวันจะมีความน่าจะเป็นในการเกิดข้อมูลของแต่ละอักขระไม่เท่ากัน อาทิ อักขระ A, E, I, O และ U ซึ่งเป็นสระในภาษาอังกฤษจะมีความน่าจะเป็นในการเกิดมากกว่าอักขระอื่นๆ

จากตัวอย่างเดิมสมมติว่าความน่าจะเป็นในการเกิดแต่ละอักขระเป็นดังนี้

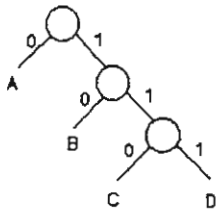
A	มีความน่าจะเป็นคือ 1/2 แทนรหัสเดิมด้วย	0
B	มีความน่าจะเป็นคือ 1/4 แทนรหัสเดิมด้วย	10
C	มีความน่าจะเป็นคือ 1/8 แทนรหัสเดิมด้วย	110
D	มีความน่าจะเป็นคือ 1/8 แทนรหัสเดิมด้วย	111

หากต้องการเข้ารหัสข้อมูลชุดเดิมคือ ABACADAB จะได้ 01001100111010 จำนวนบิตที่ใช้ คือ 14 จะเห็นว่าเมื่อนำความน่าจะเป็นในการเกิดของแต่ละอักขระ มาพิจารณาด้วย จะทำให้จำนวนบิตเฉลี่ยในการเข้ารหัสข้อมูลลดลง โดยอาศัยหลักพื้นฐานที่ว่าหากอักขระใดใช้บ่อย ก็ให้แทนด้วยจำนวนบิตน้อยๆ ส่วนอักขระใดไม่ค่อยใช้ก็ยอมให้แทนด้วยจำนวนบิตที่ยาวกว่าได้

เมื่อพิจารณาในกรณีแรกหากความน่าจะเป็นในการเกิดข้อมูลของแต่ละอักขระมีค่าเท่ากันก็เหมาะสมแล้วที่จะแทนอักขระแต่ละตัวด้วยรหัสไบนารีจำนวน 2 บิต เพื่อให้ง่ายแก่การเข้าใจจึงแสดงวิธีการแทนอักขระกรณีที่ความ น่าจะเป็น ในการเกิดข้อมูลของแต่ละอักขระเท่ากันและไม่เท่ากันด้วยรูปที่ 8 (ก) และ (ข) ตามลำดับ



รูปที่ 8 (ก) การแทนอักขระด้วยต้นไม้แบบไบนารี กรณีที่ความน่าจะเป็นในการเกิดของแต่ละอักขระเท่ากัน



รูปที่ 8 (ข) การแทนอักขระด้วยต้นไม้แบบไบนารี กรณีที่ความน่าจะเป็นในการเกิดของแต่ละอักขระไม่เท่ากัน

จากรูปที่ 8 (ก) และ (ข) ซึ่งแสดงการแทนอักขระด้วยต้นไม้แบบไบนารีจะเห็นว่าในรูปที่ 8 (ก) กรณีที่ความน่าจะเป็นในการเกิดของแต่ละอักขระเท่ากัน ต้นไม้แบบไบนารีที่สมดุลจะให้จำนวนเส้นเชื่อม (edge) จากรากไปถึงแต่ละใบนั้นเท่ากัน แต่ในรูปที่ 8 (ข) กรณีที่ความน่าจะเป็นในการเกิดของแต่ละอักขระไม่เท่ากันจะให้จำนวนเส้นเชื่อมจากรากไปถึงใบแต่ละใบด้วยจำนวนเส้นเชื่อมมากน้อยต่างกันขึ้นกับความน่าจะเป็นในการเกิดของอักขระนั้น เช่น อักขระ A มีความน่าจะเป็นสูงสุดจะให้จำนวนเส้นเชื่อมจากรากไปถึงใบต่ำที่สุด ส่วนอักขระ C และ D จะมีจำนวนเส้นเชื่อมจากรากไปถึงใบมากที่สุดเนื่องจากมีความน่าจะเป็นในการเกิดต่ำสุด

หากพิจารณาในแง่ของการค้นหาข้อมูลของรูปที่ 8 (ข) อักขระ A เกิดบ่อย จึงแทนด้วยจำนวนบิตต่ำ หรือกล่าวในเชิงการค้นหาข้อมูลคือ จำนวนครั้งในการค้นหาต่ำ ส่วนอักขระ C และ D มีโอกาสเกิดน้อย จึงแทนด้วยจำนวนบิตที่ยาวได้ หรือจำนวนครั้งในการค้นหาสูงได้ เมื่อพิจารณาการค้นหาโดยรวมจึงทำให้เวลาการค้นหาเฉลี่ยต่ำสุด

จากข้างต้นการสร้างต้นไม้สำหรับการจำแนกแบบหลายประเภทเทียบได้กับการเข้ารหัสข้อมูลโดยแต่ละโนดจะเป็นตัวจำแนกแบบสองประเภทที่เลือกมา ซึ่งจำนวนครั้งในการจำแนกข้อมูลเฉลี่ยย่อมขึ้นกับความน่าจะเป็นของการเกิดข้อมูลในแต่ละประเภทเช่นเดียวกับความน่าจะเป็นของการเกิดอักขระ

บทความนี้ได้เสนอวิธีการใหม่เรียกว่า เทคนิคการจำแนกข้อมูลแบบแตกครึ่งตามสารสนเทศ ซึ่งมีแนวคิดในการจำแนกข้อมูลด้วยระนาบหลายมิติที่แบ่งข้อมูลออกเป็นสองส่วนได้ดีที่สุดโดยพิจารณาจากความน่าจะเป็นในการเกิดข้อมูลแต่ละประเภท ซึ่งมีขั้นตอนในการจำแนกข้อมูลดังนี้

4.1 การค้นหาตำแหน่งของระนาบหลายมิติ

การสร้างฟังก์ชันของระนาบหลายมิติในการจำแนกแบบสองประเภทตามปกตินั้นจะได้ตำแหน่งของระนาบอยู่กึ่งกลางระหว่างข้อมูลสองประเภทที่เราใช้สอน โดยที่ตำแหน่งของข้อมูลทั้งสองประเภทจะอยู่คนละด้านของระนาบแต่จะไม่รู้ตำแหน่งเทียบกับประเภทข้อมูลอื่น จึงมีการค้นหาตำแหน่งของระนาบเทียบกับประเภทข้อมูลอื่นเพิ่มโดยนำฟังก์ชันของระนาบแบบสองประเภทที่มีอยู่แล้ว ไปคำนวณหาตำแหน่งของประเภทข้อมูลอื่นเพิ่มเติมก็จะได้ตำแหน่งของระนาบ เมื่อเทียบกับข้อมูลทั้งหมด เช่นเดียวกับในวิธีแตกครึ่งแบบสมดุล

4.2 การค้นหาระนาบแตกครึ่งตามสารสนเทศ

เมื่อทำการคำนวณเพื่อค้นหาตำแหน่งของระนาบหลายมิติเทียบกับประเภทข้อมูลอื่นตามขั้นตอนข้างต้นแล้วจะทำการค้นหาระนาบที่สามารถแบ่งข้อมูลได้ดีที่สุดโดยพิจารณาจากค่าเอนโทรปีของข้อมูลโดยเลือกระนาบที่ให้ค่าเอนโทรปีต่ำสุดซึ่งจะเป็นตัวบ่งชี้ว่าหากนำระนาบนั้นไปใช้ในการจำแนกข้อมูลแล้ว ข้อมูลที่ได้ในแต่ละด้านของระนาบจะมีจำนวนประเภทที่ปนกันอยู่น้อยที่สุดตามหลักของทฤษฎีสารสนเทศที่กล่าวข้างต้น เมื่อแต่ละด้านมีจำนวนประเภทของข้อมูลปนกันอยู่น้อยจึงสามารถตัดประเภทที่ไม่ใช่ออกไปได้ดีด้วย

ใช้ในการคำนวณค่าเอนโทรปีเป็นดังนี้

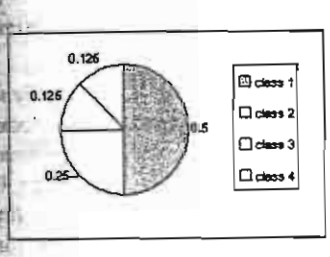
เอนโทรปีของตัวจำแนก $= \sum_{i=1}^{n} \frac{|t_i|}{|T|} I(t_i)$

$$\sum_{i=1}^{k} -P(m_i)\log_2 P(m_i) \qquad (18)$$

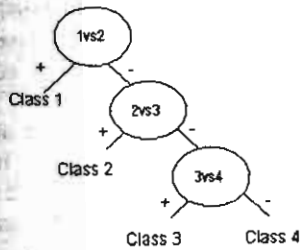
n เป็นจำนวนกิ่งของต้นไม้ในที่นี้จะมีค่าเป็น 2 ซึ่งได้แก่ กิ่งบวกและกิ่งลบ

k เป็นจำนวนประเภทของข้อมูลทั้งหมด

ตัวอย่าง กำหนดให้ชุดข้อมูลหนึ่งมีจำนวนประเภท(Class) 4 ประเภทและมีความน่าจะเป็นในการเกิดข้อมูลของแต่ละประเภทเป็นดังรูปที่ 9 (ก) และต้นไม้สำหรับการจำแนกหลายประเภทที่สร้างจากค่าเอนโทรปีของข้อมูลชุดตัวอย่างที่ดีที่สุดแสดงดังรูปที่ 9 (ข)



รูปที่ 9 (ก) ความน่าจะเป็นในการเกิดข้อมูลแต่ละประเภท



รูปที่ 9 (ข) ต้นไม้สำหรับการจำแนกแบบหลายประเภท

การคำนวณค่าจำนวนครั้งในการจำแนกที่คาดหวังสามารถคำนวณได้จากสูตร

$$-P(m_i)\log_2 P(m_i) \qquad (19)$$

สำหรับ ปัญหาการจำแนก k ประเภท และ m_i คือความน่าจะเป็นในการเกิดข้อมูลของประเภทที่ m_i

กรณีตัวอย่างจากรูปที่ 9 สามารถคำนวณค่าจำนวนครั้งในการจำแนกที่คาดหวังได้ดังนี้

$$\sum_{i=1}^{4} -P(m_i)\log_2 P(m_i)$$

$$=[-(0.5)*\log_2(0.5)]+[-(0.25)*\log_2(0.25)]+$$
$$[-(0.125)*\log_2(0.125)]+[-(0.125)*\log_2(0.125)]$$
$$= 1.75$$

5. ผลการทดลอง

สำหรับการทดลองได้ใช้ข้อมูลทดสอบจาก UCI Machine Learning Repository [3] จำนวน 4 ชุด ดังตารางที่ 1 ข้อมูลแต่ละชุดจะประกอบด้วย ข้อมูลสอนและข้อมูลทดสอบ ในขั้นตอนของการทดลองจะทำการแบ่งชุดข้อมูลสอนออกเป็น ข้อมูลสอนจริงและข้อมูลทดสอบความถูกต้อง(Validation data) เพื่อใช้ในการหาค่าพารามิเตอร์ที่เหมาะสม [7] ซึ่งประกอบด้วยค่า P คือค่าร้อยละในการตัดเล็ม ($0 \le P \le 10$) และ R คือ ค่าขอบเขตความผิดพลาด($x_{min} \le R \le x_{mean+sd}$ เมื่อ x_{min} คือ ค่าต่ำสุดของขอบเขตความผิดพลาดของตัวจำแนกทั้งหมด และ $x_{mean+sd}$ คือผลรวมของค่าเฉลี่ยและค่าเบี่ยงเบนมาตรฐานของขอบเขตความผิดพลาดของตัวจำแนกทั้งหมด) หลังจากนั้นจึงใช้ค่า P และ R ที่ได้เพื่อสร้างตัวจำแนกแบบหลายประเภทจากชุดข้อมูลสอนเดิมและใช้ชุดข้อมูลทดสอบเพื่อหาค่าความถูกต้อง และค่าจำนวนครั้งเฉลี่ยในการจำแนกข้อมูล ผลการทดลองแสดง ดังตารางที่ 2-3 (IBD คือ Information-Baesd dichotomization, BD คือ Balanced Dichotomization, Expected Value คือ ค่าจำนวนครั้งในการจำแนกที่คาดหวัง, ตัวอักษร c, d คือพารามิเตอร์ของฟังก์ชันเคอร์เนล และตัวอักษรเข้มในตารางแสดงค่าที่ดีที่สุดในแต่ละชุดข้อมูล)

ตารางที่ 1 : ลักษณะของข้อมูลที่นำมาทดสอบ

Dataset	#training set	#test set	#class	#feature
Satimage	4,435	2,000	6	36
Shuttle	43,500	14,500	7	9
Vowel	528	462	11	10
Soybean	290	340	15	35

ตารางที่ 2: เปรียบเทียบจำนวนครั้งของการจำแนก

Dataset	Polynomial Kernel					
	Expected Value of IBD	Output of IBD	Expected Value of BD ($\log_2 k$)	Output of BD	RADAG (k-1)	Maxwins (k(k-1)/2)
Satimage	2.474	4.999	2.585	4.847	5	15
Shuttle	0.965	5.359	2.807	5.378	6	21
Vowel	3.459	3.385	3.459	5.665	10	55
Soybean	3.617	6.971	3.907	6.783	14	105
Dataset	RBF Kernel					
	Expected Value of IBD	Output of IBD	Expected Value of BD ($\log_2 k$)	Output of BD	RADAG (k-1)	Maxwins (k(k-1)/2)
Satimage	2.474	4.734	2.585	4.577	5	15
Shuttle	0.965	4.982	2.807	5.758	6	21
Vowel	3.459	5.628	3.459	5.819	10	55
Soybean	3.617	5.400	3.907	6.591	14	105

ตารางที่ 3: เปรียบเทียบค่าความถูกต้อง

Dataset	Polynomial Kernel							
	d	IBD	d	BD	d	RADAG	d	Max Wins
Satimage	6	88.850	6	87.840	6	88.900	6	88.453
Shuttle	8	99.924	8	99.924	8	99.924	8	99.924
Vowel	3	64.935	2	65.022	3	64.502	3	64.329
Soybean	3	90.882	4	89.529	3	91.176	3	90.471
Dataset	RBF Kernel							
	c	IBD	c	BD	c	RADAG	c	Max Wins
Satimage	0.5	91.950	1.0	91.350	0.5	91.950	0.5	91.984
Shuttle	3.0	99.890	3.0	99.886	3.0	99.897	3.0	99.897
Vowel	0.3	66.450	0.2	62.900	0.2	67.100	0.2	65.340
Soybean	0.07	91.471	0.04	89.118	0.07	90.882	0.08	90.468

จากผลการทดลองการแตกครึ่งตามสารสนเทศให้ค่าความถูกต้องใกล้เคียงกับวิธีอื่น ส่วนจำนวนครั้งในการจำแนกเมื่อเปรียบเทียบกับอาร์เอดีเอจีและแมกซ์วิน พบว่าการแตกครึ่งตามสารสนเทศให้ค่าจำนวนครั้งในการจำแนกต่ำกว่าทุกกรณี และเมื่อเปรียบเทียบกับการแตกครึ่งแบบสมดุล พบว่าการแตกครึ่งตามสารสนเทศให้ค่าจำนวนครั้งในการจำแนกต่ำกว่าถึง 5 ใน 8 กรณี ส่วนอีก 3 ใน 8 กรณี แม้การแตกครึ่งตามสารสนเทศจะให้จำนวนครั้งในการจำแนกที่สูงกว่าแต่ก็ให้ค่าความถูกต้องที่สูงกว่าด้วย

ความสามารถในการลดจำนวนครั้งของการจำแนกจะขึ้นกับระนาบของตัวจำแนกแบบสองประเภทที่สร้างได้จากค่า P และ R ที่ดีที่สุดด้วย โดยหากระนาบที่ได้สามารถแบ่งประเภทของข้อมูลได้ดีนั่นคือ แต่ละด้านของระนาบมีประเภทของข้อมูลปนกันน้อยหรือถ้าหากด้านใดด้านหนึ่งของระนาบมีเพียงประเภทเดียว ค่าจำนวนครั้งเฉลี่ยของการจำแนกที่ได้จริงยิ่งให้ค่าใกล้เคียงกับค่าจำนวนครั้งในการจำแนกที่คาดหวังจากสูตร $\sum_{i=1}^{k} -P(m_i)\log_2 P(m_i)$ ในทางตรงกันข้ามหากระนาบที่ได้ระนาบใดที่แบ่งประเภทข้อมูลได้ดีพอก็ยิ่งส่งผลให้ตัวจำแนกหลายประเภทที่ได้มีจำนวนครั้งในการจำแนกเฉลี่ยสูงกว่าคาดคลื่อนจากค่าจำนวนครั้งในการจำแนกที่คาดหวังมากขึ้นเท่านั้น

6. สรุป

เทคนิคการแตกครึ่งตามสารสนเทศสามารถเพิ่มประสิทธิภาพการจำแนกประเภทของซัพพอร์ตเวกเตอร์แมชชีนแบบหลายประเภทด้านจำนวนครั้งของการจำแนก โดยผลดังกล่าวจะยิ่งปรากฏชัดในกรณีปัญหาที่มีจำนวนประเภทเป็นจำนวนมาก อีกทั้งยังคงให้ค่าความถูกต้องใกล้เคียงกับตัวจำแนกแบบหลายประเภทวิธีอื่น

7. เอกสารอ้างอิง

[1] บุญเสริม กิจศิริกุล., *เอกสารประกอบการเรียนวิชาปัญญาประดิษฐ์* ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย, 2546

[2] Bartlett, P. L. and Shawe-Taylor, J., *Generalization performance of support vector machines and other pattern classifiers*, in Advances in Kernel Methods Support Vector Learning, pp. 43-54, MIT Press, Cambridge, USA, 1999

[3] Blake, C., Keogh, E., and Merz, C, *UCI Repository of Machine Learning Databases*, Department of Information and Computer Science, University of California, Irvine, 1998. http://www.ics.uci.edu/~mlearn/MLSummary. html

[4] Cook, W. and Rohe, A., *Computing Minimum-Weight Perfect Matchings*, Technical Report 97863, Forschungsinstitut für Diskrete Mathematik, Universität Bonn, 1997.

[5] Friedman, J. H., *Another Approach to Polychotomous classification*, Technical report, Department of Statistics Stanford University, 1996.

[6] Hsu , C. and Lin, C., *A Comparison of Methods for Multiclass Support Vector Machines*, IEEE Transactions on Neural Networks, 13, 415-425, March, 2002.

[7] Kijsirikul, B., Boonsirisumpun, N. and Limpiyakorn, Y., *Multiclass Support Vector Machines Using Balanced Dichotomization*, The 8th Pacific Rim International Conference on Artificial Intelligence (PRICAI-2004), 2004.

[8] Phetkaew, T., Kijsirikul, B. and Rivepiboon, W., *Reordering Adaptive Directed Acyclic Graphs: An Improved Algorithm for Multiclass Support Vector Machines*, The 2003 IEEE/INNS International Joint Conference on Neural Networks, Portland, Oregon, July 20-24, 2003.

[9] Ussivakul, N. and Kijsirikul, B., *Adaptive DAG: Another Approach for Multiclass Classification*, International Conference on Intelligent Technologies, 2001.