2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Sub-Classifier Construction for Error Correcting Output Code Using Minimum Weight Perfect Matching

Patoomsiri Songsiri [*1], Thimaporn Phetkaew [†2], Ryutaro Ichise [‡3], Boonserm Kijsirikul [*4]

[*] *Department of Computer Engineering, Chulalongkorn University, Bangkok, Thailand*
[†] *School of Informatics, Walailak University, Nakhon Si Thammarat, Thailand*
[‡] *National Institute of Informatics, Tokyo, Japan*
[1] patoomsiri.s@student.chula.ac.th
[2] pthimapo@wu.ac.th
[3] ichise@nii.ac.jp
[4] boonserm.k@chula.ac.th

***Abstract*—Multi-class classification is mandatory for real world problems and one of promising techniques for multi-class classification is Error Correcting Output Code. We propose a method for constructing the Error Correcting Output Code to obtain the suitable combination of positive and negative classes encoded to represent binary classifiers. The minimum weight perfect matching algorithm is applied to find the optimal pairs of subset of classes by using the generalization performance as a weighting criterion. Based on our method, each subset of classes with positive and negative labels is appropriately combined for learning the binary classifiers. Experimental results show that our technique gives significantly higher performance compared to traditional methods including One-Versus-All, the dense random code, and the sparse random code. Moreover, our method requires significantly smaller number of binary classifiers while maintaining accuracy compared to One-Versus-One.**

***Keywords*-multi-class classification; error correcting output code; minimum weight perfect matching; generalization performance**

I. Introduction

Error Correcting Output Code (ECOC) [1], [2] is one of the well-known techniques for solving multiclass classification. Based on this framework, an unknown-class instance will be classified by all binary classifiers corresponding to designed columns of the code matrix, and then the class with the closet codeword is assigned to the final output class. Each binary function including the large number of classes indicates the high capability as a shared classifier. Generally, when the number of classes increases, the complexity for creating the hyperplane also increases. The suitable combination of classes with the proper number of classes for constructing the model is still a challenge issue to obtain the effective classifier.

Several classic works are applied to the design of the code matrix such as One-Versus-One (OVO) [3], One-Versus-All (OVA) [4], dense random code, and sparse random code [1], [2], and for an N class problem, they provide the number of binary models of $N(N-1)/2$, N, $\lceil 10log_2N \rceil$, and $\lceil 15log_2N \rceil$, respectively. Some approaches using the genetic algorithm have been proposed [5], [6]. Moreover, a design of code matrix has also been proposed that applies a heuristic method based on a hierarchical partition of the class space to maximize a discriminative criterion [7]. However, according to the complexity of the problem that the solutions are searched from a large space including all possible $2^{N-1}-1$ columns [1] in case of dense code, and $\frac{3^N-2^{N+1}+1}{2}$ columns [8] in case of sparse code, design of code matrix is still ongoing research.

This research aims to find the suitable combination of classes for creating the binary models in the ECOC framework by providing both good classification accuracy and the small number of classifiers. Our method is based on the minimum weight perfect matching algorithm by using the relation between pair of subset of classes defined by the generalization performance as the criterion for constructing the code matrix. We study this multiclass classification based on Support Vector Machines [9], [10] as base learners. We also empirically evaluate our technique by comparing with the traditional methods on ten datasets from the UCI Machine Learning Repository [11].

This paper is organized as follows. Section II reviews the traditional ECOC frameworks. Section III presents our proposed method. Section IV performs experiments and explains the results and discussions. Section V concludes the research and directions for the future work.

II. Error Correcting Output Codes

Error Correcting Output Code (ECOC) was introduced by Dietterich and Bakiri [1] as a combining framework for multiclass classification. For a given code matrix with N rows and L columns, each element contains either '1', or '-1'. This code matrix is designed to represent a set of L binary learning problems for N classes. Each specific class is defined by the unique bit string called *codeword* and each sub-problem is indicated by the combination of positive and negative classes corresponding to the elements of the code matrix. Moreover, in order to allow the binary model learned without considering some particular classes, Allwein et al. [2] extended the coding method by adding the third symbol '0' as *" don't care bit"*.

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

Unlike the previous method, the number of classes for training a binary classifier can be varied from 2 to N classes.

Several classic coding designs have been proposed, e.g., One-Versus-All (OVA) [4], dense random codes, sparse random codes [2], and One-Versus-One (OVO) [3]; the first two techniques and the last two techniques are binary and ternary strategies, respectively. One-Versus-All codes including N columns were designed by setting the i^{th} class, and the remaining classes labeled with the positive and negative classes, respectively. Dense random codes and sparse random were introduced by randomizing sets of $\lceil 10log_2N \rceil$ and $\lceil 15log_2N \rceil$ binary functions, respectively. In case of One-Versus-One codes, they were designed to define each column by labeling '1' and '-1' to only two out of N classes, and therefore there are $N(N-1)/2$ possible columns.

For a decoding process, an instance with unknown-class will be classified by all L binary functions corresponding to designed columns of the code matrix. This output vector is compared to each row of the code matrix. The class corresponding the row of code matrix that provides highest similarity is assigned as the final output class. Several similarity measures have been proposed such as Hamming distance [1], Euclidean distance [12], extended strategies based on these two methods [12], and the loss-based technique [2].

III. Proposed Method

We aim to construct the code matrix in which each column is a suitable combination of positive and negative classes encoded to represent a binary model. As mentioned before, the best code matrix can be obtained by searching from all possible $\frac{3^N-2^{N+1}+1}{2}$ columns for an N-class problem, and thus this is comparatively difficult when the number of classes increases. Our objective is to construct the code matrix providing high quality of compression (the low number of binary classifiers) with high accuracy of classification. Although the highest compression of code can be possible by using $\lceil log_2N \rceil$ binary classifiers, compression without proper combination of classes may lead to suffer with the complexity of classifier construction. To design a code matrix, for any classes i and j, some binary classifier(s) has to be selected that constructs a pairing between a set of classes containing class i, and another set containing class j. We believe that the most important pairings affecting the classification accuracy are the pairs of classes with hard separation. These pairings cannot be avoided as they must be included in some combinations of classes to distinguish between each other. The number of classes combined in each classifier varies from 2 to N. If we do not construct a classifier to separate between a pair of difficult classes, to distinguish the two classes, we still have to build a classifier by using other classes together with these two classes that increases the complexity of classifier construction. For example, Fig. 1 shows that if we build the model with a linear function to distinguish classes 1 vs 2, which are hard-to-separate classes, by setting classes 1,3,4,7,8 as positive classes and classes 2,5,6 as negative classes, it will not be easy to learn a good hyperplane.

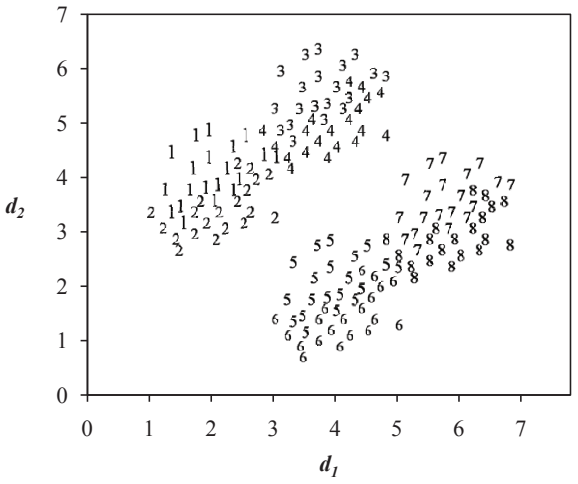

Fig. 1: An example of two-dimensional artificial data including eight classes.

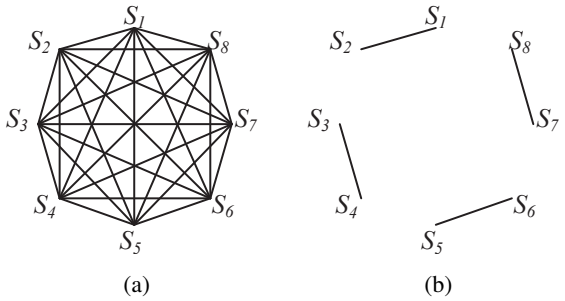

Fig. 2: An example of the output of the minimum weight perfecting matching algorithm: (a) all possible matchings of eight subsets of classes and (b) an output of the matching algorithm.

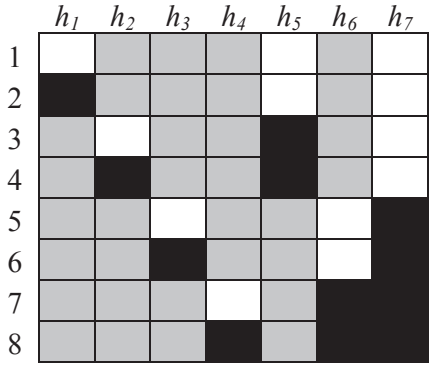

Fig. 3: An example of the code matrix constructed by the proposed method based on Linear Support Vector Machines with C = 1 for the artificial data from Fig. 1 (The white region, the dark region, and the gray region represent '1', '-1', and '0', respectively).

Algorithm 1 Optimal matching of subset of classes.

procedure OPTIMAL MATCHING OF SUBSET OF CLASSES.
  Initialize set of classes $C \leftarrow \{1, 2, 3, ..., N\}$, and set of subset of classes $S \leftarrow \{S_1, S_2, S_3, ..., S_N\}$ where $S_m \leftarrow \{m\}$ and $m \in C$.
  Construct binary models of all possible pairs of S_p and S_n as a subset of classes with the positive label and the negative label respectively, where $1 \leq p < n \leq |S|$.
  Calculate generalization performances of all possible pairs of S_p and S_n by using k-fold cross-validation.
  do
    Apply the minimum weight perfect matching [14] to find the optimal $\lfloor \frac{|S|}{2} \rfloor$ pairs of subset of classes from all possible pairs on S to obtain the closest pair of subset of classes based on the minimum generalization performance.
    Add all of optimal pairs as binary classifiers into the columns of the code matrix.
    Combine each pair of subsets of classes into the same subset as the new subset of classes.
    Replace the previous subsets of classes in S with all new subsets of classes.
    Construct binary models of all possible pairs of S_p and S_n.
    Calculate generalization performances of all possible pairs of S_p and S_n.
  while $|S| > 2$
  Add the last remaining two subsets as binary classifier into the last column of the code matrix.
  return *code matrix*.
end procedure

By the above reason, we carefully design the code matrix by considering the pairs of classes with hard separation as the first priority pairings and these pairings also indicate the high similarity between the classes. Intuitively, the pair of classes with difficult separation are allowed to combine with a low number of classes in order to avoid situation mentioned before, and we expect that the easier pairing can be combined with the large number of classes without affecting the classification ability much. Based on our idea, the classes with high similarity are grouped together as the same class label, and the classes with low similarity are separated. For example, consider two-dimensional artificial data shown in Fig. 1; classes 1,2,3,4 have high similarity and should be assigned with the same label (e.g. positive class), and classes 5,6,7,8 also have high similarity and should be assigned with the same label (e.g. negative class). Additionally, these two groups have low similarity by observation, and thus if we learn the binary model with a linear function, we will obtain absolutely separable hyperplane.

In order to obtain the code matrix satisfying the above requirements, we employ the minimum weight perfect matching algorithm [14] applied to find the optimal pairs of subset of classes by using the generalization performance [13] as a weighting criterion. Our method is called *optimal matching of subset of classes* algorithm described in Algorithm 1.

For solving the optimal matching problem, let $G = (V, E)$ be a graph with node set V and edge set E. Each node in G denotes one subset of classes and each edge indicates one binary classifier of which generalization performance is estimated from k-fold cross-validation (see Fig. 2(a)). The output of the matching algorithm for graph G is a subset of edges with the minimum sum of generalization performances of all edges and each node in G is met by exactly one edge in the subset (see Fig. 2(b)).

Given a real weight w_e that is the generalization performance for each edge e of G, the problem of matching algorithm can be solved using minimum weight perfect matching, which finds a perfect matching M of minimum weight $\sum(w_e : e \in M)$.

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 an even $|V|$ is given by

a) $x \in \{0, 1\}^m$

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 $|U| \geq 3$

where $|E| = m$, and $x_e = 1$ ($x_e = 0$) means that e is (is not) in the matching. Hence, the minimum weight of a perfect matching (mp) is at least as large as the following value.

$$mp = min \sum_{e \in E} w_e x_e \tag{1}$$

where x satisfies (a), (b), and (c). Therefore, the matching problem can be solved using the integer program in Eq. (1).

On each round of Algorithm 1, we consider a sub-problem including $|S|$ subsets of classes, and calculate optimal pairs of subsets of classes by using the minimum weight perfect matching algorithm. The cost function employed in the minimum weight perfect matching algorithm is the sum of generalization performance calculated from k-fold cross-validation [15]. The obtained subsets of classes are then used as a column of the code matrix. Consider data on Fig. 1 and the designed code matrix in Fig. 3, at the first round, each element of S contains only set of one class, and the size of S is 8. After applying the minimum weight perfect matching algorithm, we get four optimal pairings, i.e., classifier 1 vs 2, classifier 3 vs

4, classifier 5 vs 6, and classifier 7 vs 8. All of optimal four pairs as binary classifiers are added into the columns of the code matrix, i.e., classifiers h_1 to h_4 (see Fig. 3). The set S is re-assigned by the empty set. The classes from each pairing are combined together into the same subset and then are added into S as new members. Currently, the size of S is 4, so we continue the next round. After applying the minimum weight perfect matching algorithm, we get two optimal pairings, i.e., classifier 1,2 vs 3,4, and classifier 5,6 vs 7,8. These two optimal pairs as binary classifiers are added into the columns of the code matrix, i.e., classifiers h_5 to h_6. The set S is re-assigned by the empty set again, and then the classes from each pairing are combined together into the same subset and then are added into S as new members. Now, the size of S is 2, so we exit the loop. After that, the last remaining two subsets (the first subset including classes 1,2,3,4 and the second subset including classes 5,6,7,8) are added into the last column of the code matrix as binary classifier, i.e., classifier h_7.

In our methodology, a pair of classes that has been already paired will not be considered again. In the later round, the number of the members in S increases. For example at the last round, $S_1 \leftarrow \{1, 2, 3, 4\}$ and $S_2 \leftarrow \{5, 6, 7, 8\}$. It seems that this leads to complexity for calculating the hyperplane to separate between S_1 and S_2. However, the remaining classes that can be paired trend to allow easier separation due to the combination of classes in the same subset of classes having high similarity (in Fig. 1, classes 1,2,3,4 are very close, and classes 5,6,7,8 are also close), while the possible sixteen pairings including 1 vs 5, 1 vs 6, 1 vs 7, 1 vs 8, 2 vs 5, ..., 4 vs 8 contain classes with high dissimilarity. It illustrates that these sixteen pairings are encoded in only one binary classifier as One-Versus-One requires sixteen binary classifiers.

IV. Experiments

In this section, we design the experiment to evaluate the performance of the proposed method. We compare our method with the classic codes, i.e., One-Versus-All, the dense random code, the sparse random code [1], [2], and One-Versus-One [3]. This section is divided into two parts as experimental settings, and results & discussions.

A. Experimental Settings

We run experiments on ten datasets from the UCI Machine Learning Repository [11]. For the datasets containing both training data and test data, we added up both of them into one set, and used 5-fold cross validation for evaluating the classification accuracy.

In these experiments, we scaled data to be in [-1,1]. In the training phase, we used software package SVM^{light} version 6.02 [16], [17] to create the binary classifiers. The regularization parameter C of 1 was applied for model construction; this parameter is used to trade off between error of the SVM on training data and margin maximization. We employed the RBF kernel $K(x_i, x_j) \equiv e^{-\gamma||\mathbf{x_i}-\mathbf{x_j}||^2}$, and applied the degree $\gamma = 0.1$ to all datasets. For the dense random code and the sparse random code, ten runs of the experiment were conducted using the code matrices that are selected so that the minimum distances between pairs of distinct rows are maximized. These code matrices were constructed by using the ECOC library of Escalera team [18]. In case of sparse random codes, the probability 1/2 and 1/4 were applied to generate the bit of '0', and the other bits, i.e.. '1' and '-1', respectively. In the classification phase of the dense random code, the sparse random code, and the proposed method, all binary classifiers corresponding to the code matrices are used to classify instances in the validation set. We employed the attenuated euclidean [12] as distance measure by using $AED(x, y_i) = \sqrt{\sum_{j=1}^{L} |y_i^j||x^j|(x^j - y_i^j)^2}$ where x is a binary output vector, y_i is a code word belonging to a class i and L is the number of binary classifiers used in the code matrix. The class providing the closest distance will be assigned as the final output class.

B. Results & Discussions

We compared the proposed method with four traditional techniques, i.e., One-Versus-All, the dense random code, the sparse random code, and One-Versus-One as shown in Table II to Table V in which all datasets are sorted in ascending order by the number of classes. For each dataset, the best accuracy among these algorithms is illustrated in bold-face and the symbol '*' means that the method gives the higher accuracy at 95 % confidence interval with the non-parametric Wilcoxon signed rank test [19].

The experimental results in Table II show that the proposed method yields significantly higher accuracy compared to One-Versus-All in several datasets including Segment, Vowel, Abalone and Spectrometer. There is only one dataset, i.e., Arrhyth in which One-Versus-All provides statistically better result. In Table III, the dense random code was used as the baseline algorithm. The results indicate that the proposed method yields highest accuracy in almost all datasets, and, at the 95% confidence interval, the proposed technique performs statistically better than the dense random code in five datasets, i.e., Segment, Vowel, Libras Movement, Abalone, and Spectrometer. There is only one dataset, i.e., Arrhyth in which the proposed method does not achieve the better result. The experimental results in Table IV show that the proposed method gives higher accuracy compared to the sparse random code in almost all datasets. The results also show that, at the 95% confidence interval, the proposed technique performs statistically better than the sparse random code in four datasets, i.e., Segment, Vowel, Abalone, and Spectrometer, and there is no any dataset where our method gives significantly lower accuracy. Finally, the experimental results in Table V shows that the proposed method gives a little better results compared to One-Versus-One in several datasets, and there is no significant difference between these two methods at the 95% confidence interval.

Moreover, paired comparisons among all four traditional methods, and the proposed technique are summarized in Table VI. We show the win-tie-loss record (s) of the algorithm in the column against the algorithm in the row. A win-tie-

TABLE I: Description of the datasets used in the experiments.

Datasets	**#Classes**	**#Features**	**#Cases**
Segment	7	18	2,310
Arrhyth	9	255	438
Mfeat-factor	10	216	2,000
Optdigit	10	62	5,620
Vowel	11	10	990
Primary Tumor	13	15	315
Libras Movement	15	90	360
Soybean	15	35	630
Abalone	16	8	4,098
Spectrometer	21	101	475

TABLE II: A comparison of the classification accuracies by using One-Versus-All and the proposed method.

Datasets	One-Versus-All	Proposed method
Segment	91.039±1.078	**92.857±0.795***
Arrhyth	**65.525±1.872***	60.731±1.617
Mfeat-factor	**97.200±1.052**	97.050±0.942
Optdigit	**99.110±0.209**	99.093±0.374
Vowel	72.525±1.411	**80.404±3.018***
Primary Tumor	**46.667±3.478**	44.127±5.429
Libras Movement	**86.389±3.011**	84.722±3.106
Soybean	**93.810±1.627**	93.651±1.255
Abalone	8.102±1.899	**26.940±0.744***
Spectrometer	50.526±1.664	**57.474±5.398***

TABLE III: A comparison of the classification accuracies by using the dense random code and the proposed method.

Datasets	Dense random code	Proposed method
Segment	91.018±0.562	**92.857±0.795***
Arrhyth	**64.626±1.285***	60.731±1.617
Mfeat-factor	**97.073±1.140**	97.050±0.942
Optdigit	99.034±0.282	**99.093±0.374**
Vowel	59.632±0.980	**80.404±3.018***
Primary Tumor	43.931±2.835	**44.127±5.429**
Libras Movement	82.507±1.858	**84.722±3.106***
Soybean	93.273±2.092	**93.651±1.255**
Abalone	23.126±0.058	**26.940±0.744***
Spectrometer	50.428±2.102	**57.474±5.398***

TABLE IV: A comparison of the classification accuracies by using the sparse random code and the proposed method.

Datasets	Sparse random code	Proposed method
Segment	91.937±0.524	**92.857±0.795***
Arrhyth	**62.112±1.417**	60.731±1.617
Mfeat-factor	97.048±0.212	**97.050±0.188**
Optdigit	99.072±0.295	**99.093±0.374**
Vowel	61.986±0.951	**80.404±3.018***
Primary Tumor	**44.770±4.318**	44.127±5.429
Libras Movement	83.421±1.554	**84.722±3.106**
Soybean	93.115±2.107	**93.651±1.255**
Abalone	24.311±0.131	**26.940±0.744***
Spectrometer	52.670±2.165	**57.474±5.398***

TABLE V: A comparison of the classification accuracies by using One-Versus-One and the proposed method.

Datasets	One-Versus-One	Proposed method
Segment	92.742±0.868	**92.857±0.795**
Arrhyth	60.502±1.092	**60.731±1.617**
Mfeat-factor	**97.183±0.945**	97.050±0.942
Optdigit	**99.170±0.313**	99.093±0.374
Primary Tumor	**44.550±4.156**	44.127±5.429
Vowel	**80.909±1.291**	80.404±3.018
Libras Movement	84.352±1.584	**84.722±3.106**
Soybean	92.989±2.108	**93.651±1.255**
Abalone	26.326±0.314	**26.940±0.744**
Spectrometer	57.193±3.874	**57.474±5.398**

TABLE VI: Paired comparisons among the difference methods.

	Dense random code	Sparse random code	One-Versus-One	Proposed method
One-Versus-All	1-6-3	2-6-2	3-6-1	4-5-1
Dense random code		5-4-1	5-4-1	5-4-1
Sparse random code			4-5-1	4-6-0
One-Versus-One				0-10-0

TABLE VII: A comparison of the number of binary classifiers among the difference methods.

Datasets	#classes	One-Versus-All	Dense random code	Sparse random code	One-Versus-One	Proposed method
Segment	7	7	29	43	21	6
Arrhyth	9	9	32	48	36	8
Mfeat-factor	10	10	34	50	45	9
Optdigit	10	10	34	50	45	9
Vowel	11	11	35	52	55	10
Primary Tumor	13	13	38	56	78	12
Libras Movement	15	15	40	59	105	14
Soybean	15	15	40	59	105	14
Abalone	16	16	40	60	120	15
Spectrometer	21	21	44	66	210	20

loss record reports the number of datasets where the method in the column gives the results that are better, equal and worse, respectively than the method in the row at the 95% confidence interval. These results also show that our method is more effective than the previous works in term of classification accuracy.

We now discuss the characteristic of the obtained binary models by using our technique. The number of classes containing in the binary classifier varies from 2 to N for an N-class problem. Some binary classifiers include the large number of classes that seem to be complex to construct binary models with high accuracy. For example, in case of the Spectrometer dataset including 21 classes, twenty binary classifiers are obtained. The binary models with the lowest and the highest number of classes includes 2 and 21 classes, respectively. We believe that the complexity of the constructed hyperplane increases with the number of classes. Our method allows that hard-separation parings are firstly combined to construct the binary classifiers. Based on this strategy, the pair of classes with difficult separation are combined first with a low number of classes, and then we expect that easier pairing can be determined even with a large number of classes without affecting the classification ability much. The effectiveness in designing code matrix can be observed via the classification accuracy. Among all algorithms based on the Spectrometer dataset, our code matrix gives the higher classification accuracies compared to One-Versus-All, the dense random code, and the sparse random code, while providing a little better results compared to One-Versus-One. Though our proposed method is carefully designed to find the proper combination of classes, and to create the binary classifiers with high efficiency, the proper combination of classes are not possibly available in some cases because the proposed method may be forced to select inappropriate subset of classes.

In addition, we also compare the classification time (the number of binary classifiers employed) of the proposed method to the traditional works as shown in Table VII. The results illustrate that the proposed technique requires low running time compared to the dense random code, the sparse random code and One-Versus-One, in all datasets especially, when the number of classes is relatively large. The dense random code, the sparse random code, and One-Versus-One, require $\lceil 10log_2N \rceil$, $\lceil 15log_2N \rceil$, and $N(N-1)/2$ binary classifiers, respectively while our framework needs only $N-1$ classifiers that is close to the number of N classifiers required by One-Versus-All for N-class problems.

Our code matrix construction requires l rounds of calculation ($l = \lceil log_2N \rceil - 1$), and in each round the number of binary classifiers to be trained is proportional to $|S|$ that is the current number of subsets of classes. In the first round, the number of binary classifiers is $|S|(|S|-1)/2$ where $|S| = N$, and in the next round the number of subsets of classes is reduced to $\lceil \frac{N}{2} \rceil$. All possible pairs of these subsets of classes are used to learn binary models and the processes will be repeated until the last two subsets of classes remain (as described in Algorithm 1). Therefore, the number of binary classifiers that are trained during the construction of the code matrix is at most $\sum_{l=2}^{\lceil log_2N \rceil} 2^{l-1}(2^l - 1)$. Although this phase needs additional computation to estimate the weight values using k-fold cross validation, the task is conducted in the training phase, and not affecting the performance in the classification phase.

V. Conclusion

We propose an algorithm to find the suitable combination of classes for creating the binary models in the ECOC framework by using the generalization performance as a relation measure among subset of classes. This measure is applied to obtain the set of the closest pairs of subset of classes via the minimum weight perfect matching algorithm in order to generate the columns of the code matrix. The proposed method gives higher performance both in terms of accuracy and classification times compared to the traditional methods, i.e., the dense random code, the sparse random code, and the One-Versus-All.

Moreover, our approach requires significantly smaller number of binary classifiers while maintaining accuracy compared to One-versus-One. However, the expected matchings of subset of classes are not possibly available in some cases because relation of their subset of classes may force to combine inappropriate subset of classes and it may lead to misclassification. We will further analyze to address this situation in our future work.

VI. Acknowledgment

This research is supported by the Thailand Research Fund, Thailand.

References

[1] T.G.Dietterich, and G.Bakiri. "Solving multiclass learning problems via error-correcting output codes,"*Journal of Artificial Intelligence Research*, vol. 2, pp. 263-286, 1995.

[2] E.L.Allwein, R.E.Schapire, and Y.Singer. "Reducing multiclass to binary: a unifying approach for margin classifiers,"*Journal of Machine Learning Research*, vol.1, pp. 113-141, 2000.

[3] T.Hastie, and R.Tibshirani. "Classification by pairwise grouping,"*Neural Information Processing Systems*, vol.26, pp. 451-471,1998.

[4] V.Vapnik. *Statistical learning theory*, New York, Wiley, 1998.

[5] L.I.Kuncheva. "Using diversity measures for generating error-correcting output codes in classifier ensembles,"*Pattern Recognition Letters*, vol.26, pp. 83-90, 2005.

[6] A.C.Lorena, and A.C.P.L.F.Carvalho. "Evaluation functions for the evolutionary design of multiclass support vector machines,"*International Journal of Computational Intelligence and Applications*, vol.8, pp.53-68, 2009.

[7] O. Pujol et al., "Discriminant ECOC: A Heuristic Method for Application Dependent Design of Error Correcting Output Codes,"*IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.28, pp. 1007-1012, 2006.

[8] M.A.Bagheri, G.Montazer, and E.Kabir. "A subspace approach to error correcting output codes,"*Pattern Recognition Letters*, vol.34, pp.176-184, 2013.

[9] V.Vapnik. *The nature of statistical learning theory*, London, UK, Springer-Verlag, 1995.

[10] V.Vapnik. "An overview of statistical learning theory,"*IEEE Transactions on Neural Networks*, vol.10, pp.988-999, 1999.

[11] C.Blake, E.Keogh, and C.Merz. UCI repository of machine learning databases, Department of Information and Computer Science, University of California, Irvine, 1998.

[12] S.Escalera, O.Pujol, P.Radeva. "On the decoding process in ternary error correcting output codes,"*IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.32, pp.120-134, 2010.

[13] P.Bartlett, and J.Shawe-Taylor. "Generalization performance of support vector machines and other pattern classifiers,"*Advances in Kernel Methods - Support Vector Learning*, MIT Press, pp.43-54, 1998.

[14] W.Cook, and A.Rohe. "Computing minimum-weight perfect matchings,"*INFORMS Journal on Computing*, vol.11, pp.138-148, 1999.

[15] T.Mitchell. *Machine Learning*, McGraw Hill, 1997.

[16] T.Joachims. "Making large-scale SVM learning practical,"*Advances in Kernel Methods - Support Vector Learning*, MIT Press, pp.169-184, 1998.

[17] T.Joachims, SVM^{light}, http://ais.gmd.de/~thorsten/svm_light, 1999.

[18] M.A.Bagheri, Q.Gao, and S.Escalera, "Efficient pairwise classification using Local Cross Off strategy,"*Proceedings of the 25th Canadian Conference on Artificial Intelligence*, pp.25-36, 2012.

[19] J.Demšar, "Statistical Comparisons of Classifiers over Multiple Data Sets,"*Journal of Machine Learning Research 7*, pp.1-30, 2006.