Network Modeling Analysis in Health Informatics and Bioinformatics, December 2014, 3:53. The final publication is available at http://link.springer.com/article/10.1007/s13721-014-0053-2.

Greedy Hierarchical Binary Classifiers for Multi-class Classification of Biological Data

Salma Begum, Ramazan S. Aygun

Abstract Multi-class classification is an important and challenging problem for biological data classification. Typical methods for dealing with multi-class classification use a powerful single classifier such as neural networks to classify the data into one of many classes. Alternatively, the binary classifiers are used in one-versus-one (OVO) and one-versus-all (OVA) classifier schemes for multi-class classification. However, it is not clear whether OVO or OVA yield good performance results. In this paper, we propose a greedy method for developing a hierarchical classifier where each node corresponds to a binary classifier. The advantage of our greedy hierarchical classifier is that at the nodes any type of classifier can be used. In this paper, we analyze the performance of the proposed technique using neural networks and naive Bayesian classifiers and compare our results with OVO, OVA, and exhaustive methods. Our greedy technique provided better and more robust accuracy than others in general for biological data sets including 3 to 8 classes.

Keywords Hierarchical Binary Classifiers, Neural Networks, Error-Correcting Output Codes, Biological Data

Salma Begum
Computer Science Department
University of Alabama in Huntsville
Huntsville, Alabama
E-mail: sb0034@uah.edu

Ramazan S. Aygun
Computer Science Department
University of Alabama in Huntsville
Huntsville, Alabama
E-mail: raygun@cs.uah.edu

1 INTRODUCTION

Binary classification is the problem of classifying a data into two classes where one class typically represents belonging to the target class (true or belongs to) or one of the two classes whereas the other one corresponds to not belonging to the target class (false or does not belong to). A binary classifier is actually a model to separate data into two classes. Multi-class classification is the problem of classifying data into one of many classes. If the dataset contains multiple classes, generating a model that separates the data of different classes becomes difficult. Multi-class classification is an important and challenging problem for biological data classification [Gupta K et al. 2012]. Some of the examples of multi-class biological datasets include breast tissue, iris, yeast, thyroid diseases, and protein crystallization. The multi-class classification has been studied in (Sánchez-Maroño et al. 2010; Wang Y and Casasent D 2006 ; El-Alfy E 2010; Casasent D and Wang Y 2005 ; Jain P et al. 2008, Nagi S and Bhattacharyya D 2013).

Multi-class classification problem can be dealt with a) using a single multi-class classifier or b) merging the results of binary classifiers. Different feature selection methodologies are required to handle datasets with very high dimensionality [Hulse J et al. 2012]. A single multi-class classifier such as neural networks may not meet the performance or accuracy goals of the classification problem, since it may be difficult to come up a single model to categorize dataset. Alternatively, the results of binary classifiers are merged for multi-class classification problem. The results of binary classifiers are considered as a vote for the final classification or these binary classifiers may be used in a hierarchical fashion to yield the class of a data item. Although there have been many research studies that use the results of binary classifiers for multi-class classification problem, the research on hierarchical binary classifiers has been limited [Sánchez-Maroño N et al. 2010].

A hierarchical binary classifier classifies the data into two macro-classes at each node. A macro-class is a set of classes. Depending on the output of the classifier at a node, either the right child node or the left child node is taken for further classification and this evaluation continues until a node classifies the data into a single class (not into a macro-class). One of the decisions to be made for constructing hierarchical binary classifier is the choice of classifier to be used at the internal nodes. Although powerful binary classifiers such as support vector machines (Cortes C and Vapnik V 1995) have been used for hierarchical binary classifier, multi-class classifiers such as neural networks have also been used as a binary classifier at the internal nodes (El-Alfy E 2010). Casasent and Wang (Casasent D and Wang Y 2005) propose a balanced tree using SVM that has two equal-sized macro-classes to yield as a classifier at the internal nodes. However, splitting into two equal-sized macro-classes may not yield the best accuracy at each node. Tibshirani and Hastie (Tibshirani R and Hastie T 2007) propose SVM-based hierarchical margin trees for high-dimensional classification and test it on cancer microarray data. They propose splitting by using the largest margin at each time, but this yields the separation of the most different

class each time. Therefore, their greedy technique mostly separates one class from the rest at each node and this is not useful for biological subgrouping. To alleviate this problem, they propose complete linkage approach but it requires the computation of distances of items in different partitions. In addition, their approach is not generalizable to other classifiers. Hierarchical binary classifiers are also applied on other types of data such as hand written numerals using hierarchical GMDH-based neural networks (El-Alfy E 2010). The authors also consider the choice of features to be used at each node.

In this paper, we propose a greedy hierarchical binary classifier construction approach for multi-class classification problem which is an extension of our previous work (Begum and Aygun, 2012). In this work, we propose a fast technique to construct a greedy hierarchical binary classifier that allows using any type of classifier at the internal nodes. Two types of greedy techniques are proposed: a) top-down and b) bottom-up. The top down construction starts with a macro-class that contains all classifiers and tries to split into two sub-macro-classes. The bottom-up construction starts with best one-versus-one classifier and merges other classes that would yield high accuracy. The performance in terms of accuracy and the complexity of building a greedy-based hierarchical classifier is also explained. The performance of the proposed greedy hierarchical binary classifier is compared with two commonly used methods that utilize binary classifiers: one-versus-one and one-versus-all. In addition, to determine where the actual performance stands, all possible binary classifiers are generated exhaustively, and error-correcting output codes (ECOC) are used in labeling the class of a data item. Exhaustive method is usually avoided due to its significant cost for training a large number of classifiers (Sánchez-Maroño et al. 2010). When creating all binary classifiers in exhaustive method, we make statistical analysis of different binary classifiers which can be helpful in research work. Various techniques have been used to combine the results of many binary classifiers including the majority vote (Friedman J1996), error correcting output code (ECOC) model (Escalera S et al. 2010), the bradley–terry model (Hastie T and Tibshirani R 1998), and the directed acyclic graph model (Platt JC, Cristianini N and Shawe-Taylor J 2000). In this paper, we pick ECOC for merging results for the exhaustive, OVO, and OVA classification results.

This paper is organized as follows. The following section provides the background for ECOC and the basics of building a hierarchical binary classifier. Section 3 explains our greedy hierarchical binary classifiers including both top-down and bottom-up versions. Experiments and the evaluation of our method are discussed in Section 4. The last section concludes our paper.

2 BACKGROUND

Multiclass classification problem is to map the data samples into more than two classes. There are two main approaches for solving multiclass classification problems. The first approach deals directly with the multiclass problem and uses algorithms like Decision Trees (Breiman L et al. 1984; Quinlan J 1993), Neural Networks (Bishop CM 1995), k-Nearest Neighbor (Bay SD 1998) and naive Bayesian classifiers (Rish I 2001). The main problem with this approach is to determine features that will distinguish classes when the number of classes increases (Guyon I et al. 2006) As a result, this approach is likely to yield lower accuracy.

The second approach solves the multiclass problem by converting it into a set of binary classification problems using binary classifiers such as Support Vector Machines. Several methods have been proposed to decompose the multi-class problem into binary problems. The one-versus-all (OVA) and one-versus-one (OVO) are the two popular methods of decomposition. In OVA, $\boldsymbol{K}$ class problem is solved by $\boldsymbol{K}$ binary classifiers, where each classifier discriminates a given class from the other $\boldsymbol{K-1}$ classes (Duda R et al. 2000). In OVO, a binary classifier is built to distinguish a class from each other class. This requires building $\boldsymbol{K(K-1)/2}$ binary classifiers (Hastie T and Tibshirani R 1998). Dense (Allwein et al. 2002) and sparse random (Escalera et al. 2009) schemes are also introduced as a solution to decompose into binary classifiers. In dense scheme, the suggested number of classifiers to be learned is 10logK. In sparse method, 15logK classifiers are created. Another scheme known as exhaustive method generates all possible binary classifiers for a given multiclass problem (Sánchez-Maroño N et al. V 2010). A common criticism of these methods is that, they decompose the multiclass problem a priori, without considering the properties and characteristics of the data sets (Allwein et al. 2002).

Solving multiclass problem using binary classifiers also has several drawbacks. The main problem is to integrate the results of binary classifiers to classify data. Error-Correcting Output Codes (ECOC) is a general framework to integrate the results of binary classifiers to address the multiclass problem (Escalera S et al. 2010). It consists of two steps: encoding and decoding.

1. Encoding step

In the encoding stage, a codeword is assigned for each of the classes. If there are $\boldsymbol{n}$ possible binary classifiers for a $\boldsymbol{K}$- class problem, then a codeword of length $\boldsymbol{n}$ is obtained for each class where each position of the code corresponds to a response of a given binary classifier. Arranging the codewords as rows of a matrix, we define a ternary coding matrix $\boldsymbol{M}$, where $\boldsymbol{M}$ is a $K \times n$ matrix and $\boldsymbol{M_{ij}} \in \{-1, 0, +1\}$. In this matrix $\boldsymbol{M}$, +1 and -1 are defined by the class membership of the left and right part (class) of binary classifiers. For example, +1 for 1-2 classifier indicates that data belongs to class 1, and -1 indicates that data belongs to class 2. The value 0 is used to indicate that the class is not considered as a member of the binary classifier (Escalera S et al. 2008). Fig. 1 shows an example of encoded coding matrix $\boldsymbol{M}$ for 3-class problem.

2. Decoding step

In the decoding step, applying the $\boldsymbol{n}$ trained binary classifiers, a code is obtained for each data point in the test set. This code is compared to the base codewords of each class defined in the matrix $\boldsymbol{M}$, and the data point is assigned

to the class with the ”closest” codeword. The most frequently applied decoding designs are: hamming decoding, inverse hamming decoding, and euclidean decoding (Escalera S et al. 2010).

(a) OVO

	Classifiers		
Class Name	**1-2**	**1-3**	**2-3**
1	+1	+1	0
2	-1	0	+1
3	0	-1	-1

(b) OVA

	Classifiers		
Class Name	**1-23**	**2-13**	**3-12**
1	+1	-1	-1
2	-1	+1	-1
3	-1	-1	+1

(c) Exhaustive

	Classifiers					
Class Name	**1-2**	**1-3**	**2-3**	**1-23**	**2-13**	**3-12**
1	+1	+1	0	+1	-1	-1
2	-1	0	+1	-1	+1	-1
3	0	-1	-1	-1	-1	+1

Fig. 1 Encoded Coding Matrix for ECOC

The output space can also be divided in a hierarchical fashion where the classes are arranged into a tree where the path from the root node to a leaf node leads to a classification of a new pattern.

In Hierarchical Binary Classifiers (HBCs), each node of a tree is a binary classifier that uses ***K−1*** binary classifiers to classify a ***K***-class problem (Wang Y and Casasent D 2006). Fig. 2 is an example of HBC for a 5-class problem where each node in the tree is a Binary Classifier (BC). For testing a new pattern, a path is followed from the root to a leaf node, indicating the class label of an unknown sample. In the best case, it is possible to classify a sample at the top node and in the worst case, ***K−1*** binary classifiers may be required depending on the tree structure of HBC. Thus, the tree structure affects the number of classifiers to be used for

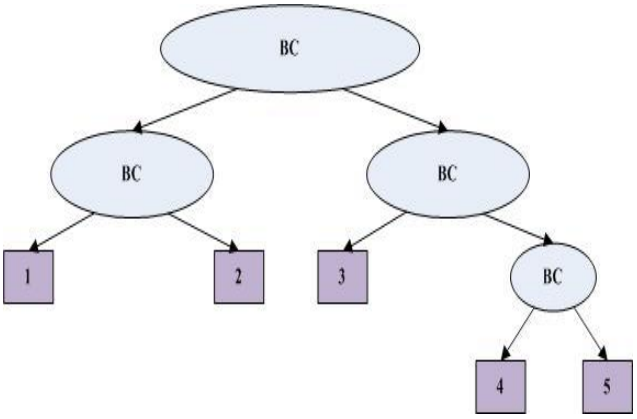


Fig. 2 Hierarchical Binary Classifier for 5 class problem

testing a data sample. Various hierarchical tree structures are possible for a K-class problem. In one-versus-all version, the trees are organized in a linked list fashion, whereas balanced hierarchical tree structure may reduce the number of classifiers to be used.

The tree structure may influence the classification accuracy of a test sample. Therefore, the hierarchical splitting (i.e., the macro-class selection) at each node in the hierarchy should not be done arbitrarily or by intuition. In literature, different clustering algorithms have been used for binary partition. A generalization of c-means clustering along with the ideas from simulated annealing can be used to obtain the binary partition of classes in the hierarchy (Kumar S et al. 2002). In (Vural V and Dy JG 2004), Vural and Dy suggested K-mean clustering method to define the binary partitions of the classes. Lorena and Carvalho proposed two general minimum spanning tree based algorithms to automatically produce the hierarchical tree structure using information collected from the multiclass data sets (Lorena A and Carvalho A 2008). In this way, hierarchical trees are organized in two different designs: bottom-up and top-down (Duda R et al. 2000). In this paper, these two approaches are used where macro-class selection is based on greedy technique of trained binary classifiers.

3 PROPOSED APPROACH

Our approach solves the multiclass classification problem by a hierarchical binary classifier. In this method, our major concern is to generate the best hierarchical tree in terms of accuracy. Since the number of all possible trees becomes high, we also propose two greedy techniques (top-down and bottom-up) to separate the classes at each node in the hierarchical tree structure. This greedy tree construction can be done using any classifier. In this work, we build the hierarchical greedy trees using neural and naïve Bayesian classifier. Our proposed method has the following major steps explained below.

1. Train binary classifiers

First, the dataset is divided into two sets: training and testing. For training, we created all possible binary classifiers for multiclass problem. This approach includes the schemes OVA and OVO. The number of possible combinations for ***K***-class problem can be obtained using the following formula:

$$f(K) = \sum_{i=1}^{\lfloor \frac{K}{2} \rfloor} \sum_{j=i}^{K-i} P_{i,j,K}$$

where $P_{i,j,K} = C(K,i) \times C(K-i,j)$

if $i = j$, $P_{i,j,K} = \frac{C(K,i) \times C(K-i,j)}{2}$ **(1)**

For example, using (1), the number of possible binary classifiers for 5-class problem is given in Table 1. The expression *f(K)* iterates for all possible macro-class splits. $P_{i,j,K}$ returns all possible number of classifiers when *K* macro-classes are split into *i* classes on one side and *j* classes on the other side. In Table 1, each row indicates a possible macro-class split and shows the number of classifiers for that split.

Table 1 : Possible binary classifiers for 5-class problem

Binary Classifier Name	Equation	Classifier Number
One-versus-one ($_{I}B_{I}$)	$\frac{C(5,1) \times C(4,1)}{2}$	10
One-versus-two ($_{I}B_{II}$)	$C(5,1) \times C(4,2)$	30
One-versus-three ($_{I}B_{III}$)	$C(5,1) \times C(4,3)$	20
One-versus-four ($_{I}B_{IV}$)	$C(5,1) \times C(4,4)$	5
Two-versus-two ($_{II}B_{II}$)	$\frac{C(5,2) \times C(3,2)}{2}$	15
Two-versus-three ($_{II}B_{III}$)	$C(5,2) \times C(3,3)$	10
Total		90

Table 2 depicts the number of possible classifiers for class 3-8 problems.

Table 2: Possible binary classifiers of different multiclass problems (Begum S and Aygun R 2012)

No. of Classes	No. of Binary Classifiers
3	6
4	25
5	90
6	301
7	966
8	3025

2. Generate all possible trees

To understand and analyze the nature of binary hierarchical trees we generated all possible trees using the following recursive equation,

$$f(K) = \sum_{i=1}^{\lfloor K/2 \rfloor} P_{i,K}$$

where $P_{i,K} = C(K,i) \times f(i) \times f(K-i)$

if $i = (K-i)$, $P_{i,K} = \frac{C(K,i) \times f(i) \times f(K-i)}{2}$

$f(1)=1$,

$f(2)=1$, **(2)**

and *K* is the number of classes.

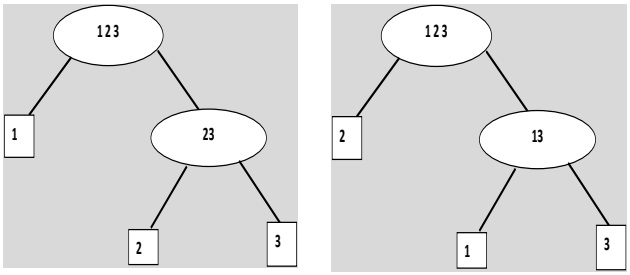
This is a recursive function and the number of possible trees becomes high with the increase in class number. According to (2), the number of possible hierarchical trees for 4-class problem is 15 (Begum S and Aygun R 2012). Table 3 shows the possible hierarchical binary trees for different number of classes.

Table 3: Possible hierarchical binary trees of different multiclass problems (Begum S and Aygun R 2012)

No. of Classes	No. of Trees
3	3
4	15
5	105
6	945
7	10395
8	135135

These trees are evaluated and the best tree is obtained by measuring the accuracy of binary classifiers used at each node in the hierarchical design. For example, to get the best tree for a 3-class problem, we need to evaluate all 3 hierarchical trees (Fig. 3). The comparative result of greedy approach with the best tree is also included in the experiments section.

(a)Tree 1 **(b) Tree 2**



(c) Tree 3

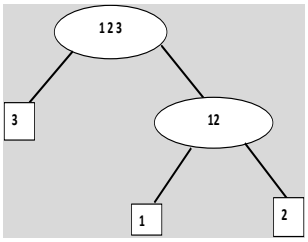


Fig. 3 All hierarchical binary trees for 3-class problem

3. Develop a greedy hierarchical classifier

Since training all the binary classifiers and building the hierarchical tree is computationally infeasible, we propose a fast algorithm to build the tree automatically while training. In this technique, the classes are separated into macro-classes at each node based on the accuracy of the neural binary classifier in the training phase. In our greedy hierarchical model, the tree has ***K-1*** binary classifiers and ***K*** leaf nodes for ***K***-class problem. Two different greedy tree construction algorithms are proposed here: top-down and bottom-up. Fig. 4 shows the pseudo-code of top-down and bottom-up hierarchical binary tree generation algorithm. The provided pseudo-code is an updated version of our implementation to increase clarity.

3.1 Top-down greedy tree construction

For top-down approach, for a ***K***-class problem at the top node it selects the best binary classifier $_i b_j$ that splits into two macro classes ***i*** and ***j*** where $i \cup j = S$ and ***S*** includes all classes**.** (Fig. 4 (a) Line 4 to 8). This step is followed recursively for all the macro-classes from top to bottom and a hierarchical binary tree is built with ***K*** leaf nodes where each leaf node corresponds to a given class (Fig. 4(a) Line 9 to 10). The iterative procedure of hierarchical tree generation for 4-class problem in top-down method is shown in Fig. 5

In Fig. 5, at the top node the best binary classifier $_2b_{134}$ ({2} in one macro-class, {1, 3, 4} in other macro-class) is selected from all classifiers in $_IB_{III}$, and $_{II}B_{II}$. Here $_IB_{III}$ represents the classifiers that split classes into two groups where the first group has one class and the other group has three classes. Since the left node has only one class, it then finds the best binary classifiers among the right macro classes. This recursive procedure stops when both the left and right node ends in one class.

3.2 Bottom-up greedy tree construction

For bottom-up approach a similar strategy is used starting from the bottom node. For a ***K***-class problem at the bottom node it selects the best binary classifier $_i b_j$ that splits into two macro classes ***i*** and ***j*** *where* $\|i\| = 1$ and $\|j\| = 1$, i.e., the macro classes include only one class. In other words, it chooses the best one-versus-one classifier (Fig. 4(b) Line 4 to 9). By merging these two (macro) classes into one macro class, in the next step, this procedure again chooses or merges two macro classes whose binary classifier would perform better than the others(Fig. 4(b) Line 10 to 17). This recursive merging procedure stops when it finds a binary classifiers $_i b_j$ that splits into two macro classes ***i*** and ***j*** where $i \cup j = S$ and S that includes all classes. This can also be stated as finding a binary classifier $_i b_j$ that merges macro classes i and j into a single macro class. The iterative procedure of hierarchical tree generation for 4-class problem in bottom-up method is shown in Fig. 6.

In Fig. 6, for bottom-up greedy tree structure at the bottom node the best binary classifier $_1b_3$ ({1} in one macro-class, {3} in other macro-class) is selected among all $_IB_I$, **(**one-versus-one) classifiers. It then merges the classifier and considering this as one macro class recursively finds the best one among all one–versus-one. This procedure stops when it finds the best classifiers with two macro classes and the union of macro classes includes all classes.

3.3 Comparison of top-down and bottom-up methods

Top-down approach tries to find the best binary split for a set of all classes at the root node. If there are ***K*** classes, all the binary splits where one side has ***1, 2,…, K/2*** classes need to be considered. This leads to $2^{K-1} - 1$ classifiers for the root node if K is odd. This leads to exponential number of classifier generations. On the other hand, since the decisions are initially made at the root node, selecting a good classifier for the root node is important Bottom-up approaches start building the tree from the leaf nodes. Initially, all binary one-versus-one classifiers need to be trained. This leads to $K(K-1)/2$ binary classifiers. In the worst case, the tree will look like a chain. In the worst case, $(K-1)^2$ classifiers will be trained. The total cost of training in the worst case is close to the summation of cost for OVO and OVA. The complexity for the bottom-up is at

a) Greedy-Top-Down Algorithm:

Greedy-Top-Down (S, K,node)
//IN: S=The set of all classes
//IN: K=The number of classes
//OUT: node contains the corresponding node in the tree
//BC=a binary classifier
//bsLeft=best left child of macroclasses
//bsRight=best right child of macroclasses
//max_{acc}=the best accuracy of classifiers so far

```
1.  begin
2.    max_acc=0
3.    if K>1 then
4.      for i=1 to K/2 do
5.        Q=(all i combinations of K classes)
6.        for each element in Q do
a.          Left=Q.current
b.          Right=S-Left
c.          BC=train(tset(Left),tset(Right))
d.          if BC.accuracy>max_acc then
e.            max_acc=BC.accuracy
f.            (bsLeft, bsRight) =(Left,Right)
g.            Node.classifier=BC
h.          endif
7.        endfor
8.      endfor
9.      Greedy-Top-Down(bsLeft,||bsLeft||,node.left)
10.     Greedy-Top-Down(bsRight,||bsRight||,node.right)
11.   endif
12. end
```

The function is called with the following parameters:
Greedy-Top-Down(S,K,root).
// train : This function is called with two parameters; left and right macro classes. It returns a binary classifier after training.
//tset: This functions returns the training set for a macro class

(b) Greedy –Bottom-Up Algorithm:

Greedy-Bottom-Up (S, K,node)
//IN: S=The set of all macroclasses
//IN: K=The number of classes
//OUT: node contains the corresponding node in the tree
//BC=a binary classifier
//MC_i=i^{th} macroclass in set S
/MC_i.node: the tree node for macroclass MC_i
//MC_i.node.classifier: the classifier for macroclass MC_i
//bsLeft=best left child of macroclasses
//bsRight=best right child of macroclasses
//max_{acc}=the best accuracy of classifiers so far

```
1.  begin
2.    max_acc=0
3.    if K>1 then
4.      for i=1 to K-1 do // create all OVO classifiers
5.        for j=i+1 to K do
6.          BC=train(tset(MC_i),tset(MC_j))
7.          if BC.accuracy>max_acc then
a.            max_acc=BC.accuracy
b.            (bsLeft, bsRight) =(MC_i,MC_j)
c.            bsClassifier=BC
d.          endif
8.        endfor
9.      endfor
10.     S=S-{bsLeft}// remove children
11.     S=S-{bsRight}
12.     MC_new=(bsLeft U bsRight) // new MC as union
13.     MC_new.node.classifier=bsClassifier
14.     MC_new.node.left=bsLeft.node
15.     MC_new.node.right=bsRight.node
16.     S=S U{MC_new}
17.     Greedy-Bottom-Up(S,K-1, MC_new.node)
18.   endif
19. end
```

The function is called with the following parameters:
Greedy-Bottom-Down (S,K,root)
// train : This function is called with two parameters; left and right macro classes. It returns a binary classifier after training
//tset: This functions returns the training set for a macro class

Fig. 4 Algorithm for greedy top-down and bottom-up tree structure

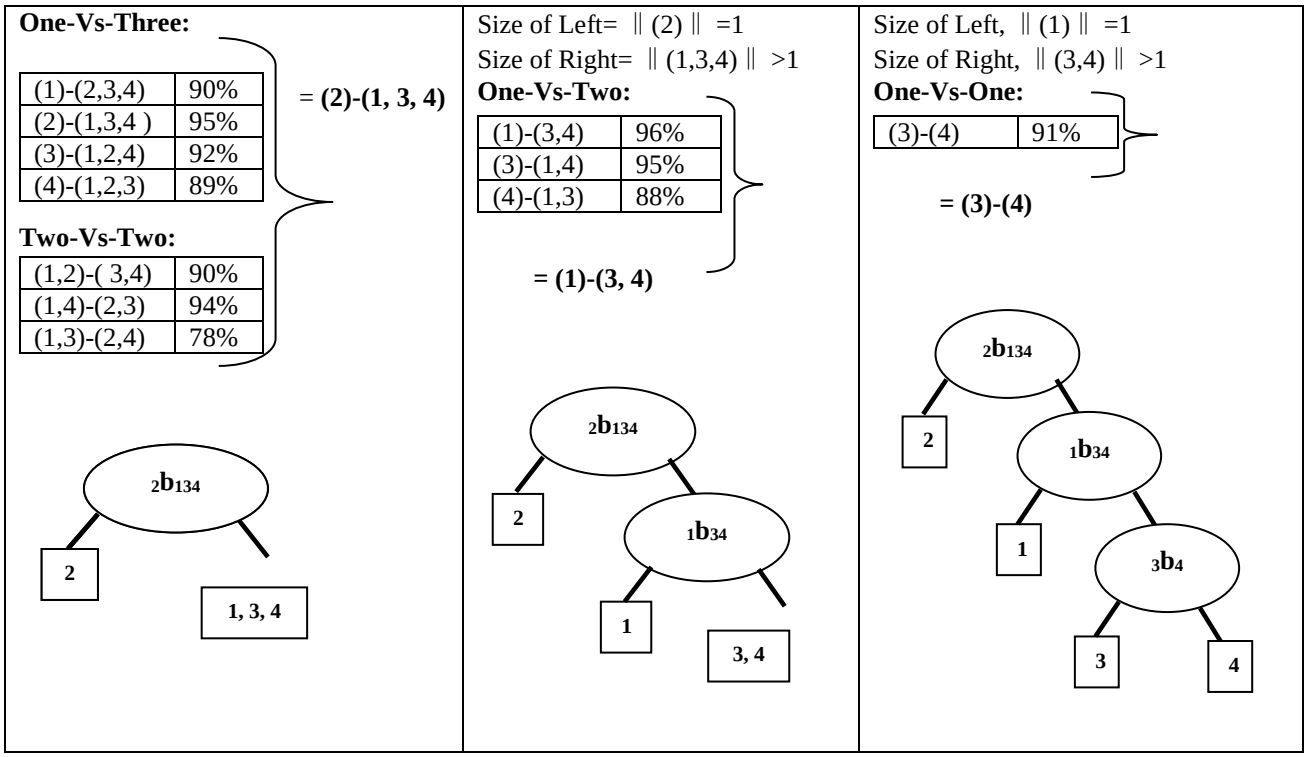


Fig. 5: Top-down greedy hierarchical tree construction for 4-class problem

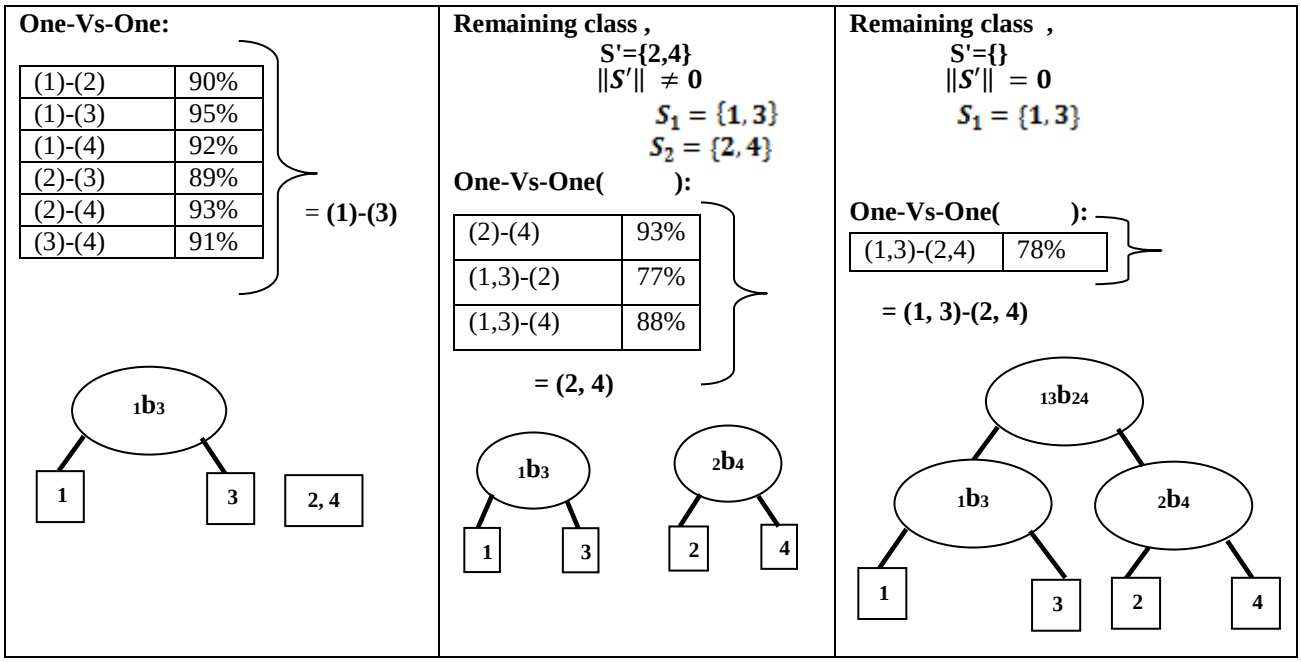


Fig. 6: Bottom-up greedy hierarchical tree construction for 4-class problem

an acceptable level. However, since the tree is built from the leaf node to the root node, when the method reaches to the top nodes, the accuracy for those binary classifiers might be low. The decisions at the top nodes are critical and misclassification at the top nodes cannot be fixed later even though there could be very good binary classifiers close to the leaf nodes.

3.4 Improved greedy bottom-up(IGBU) HBC testing

The bottom-up method is fast; however, if the binary classifier at the root node has a low accuracy, the performance of the hierarchical binary classifier is also low. This is a significant limitation. We overcome this problem by skipping the classifier at the root node when the accuracy of the classifier at the root is low (e.g., below 80%). Whenever a new data sample is provided, the sample is fed into both left and right classifiers of the root note. As a result, the tree will produce two class labels: one for the left sub-tree and the other one for the right sub-tree. For these two class labels, we apply one-versus-one classifier to determine which class the sample belongs to. For example, in Fig. 7, since the accuracy of the root binary classifier $_{13}b_{24}$ (13 in the left macro class and 24 in the right macro class) is less than 80%, rather than providing data at root node we provide a data sample to both sub trees. If the output of a test sample from the left binary tree $_{1}b_{3}$ is 1 and from the right binary tree $_{2}b_{4}$ is 4, we again test the sample with the one-versus-one classifier $_{1}b_{4}$ to get the desired label of test sample.
Since the accuracy of classifiers close to the leaf nodes is high, the overall accuracy is improved. We avoid the problem of misclassifying at the root node. Actually we are replacing the classifier at the root node with a one-versus-one classifier. This does not add any complexity to training since one-versus-one classifiers were already generated while building the bottom-up greedy tree.

4. HBC testing

After the construction of HBC, we test each sample to get the class label of sample. In general, for Hierarchical Binary Classifiers, we start testing the samples with the binary classifier at the root node of the tree. Each node actually corresponds to a binary classifier. Depending on the output of the binary classifier, the right branch or left branch is taken. Then we test the sample with the next classifier along the left or right path of the tree structure. This process is continuously followed until a leaf node of the tree where desired class of the sample is obtained.

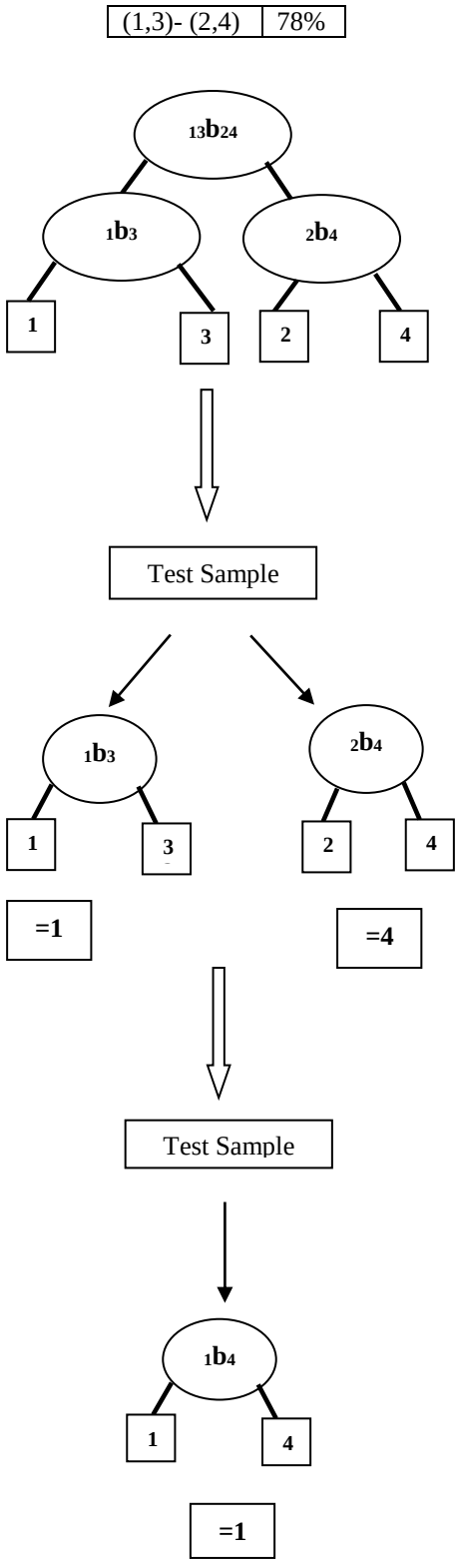


Fig. 7 Improve Greedy Bottom-Up (IGBU) HBC testing

5. Generate encoding codeword for ECOC framework

After decomposing into binary classification problem, we trained all the binary classifiers with the training datasets and store the corresponding accuracy. Since the number of samples in each class is not equally distributed, we also measure the number of misclassified images to evaluate the performance of each classifier. To combine the results of binary classifiers, a coding matrix $\boldsymbol{M}$ for these strategies (OVA, OVO, greedy, and exhaustive) is generated during the training stage (Fig. 1). In this matrix $\boldsymbol{M}$, each row represents a code for a class which will be compared in the decoding stage.

6. Apply hamming decoding strategy

Finally, to determine the final class label for each of OVA, OVO, greedy and exhaustive approach, hamming decoding technique is used. In this method, a coding matrix $\boldsymbol{M}'$ is obtained by testing the samples in the test set with all possible trained binary classifiers. In the matrix $\boldsymbol{M}'$, $\boldsymbol{i}^{\boldsymbol{th}}$ row represents a codeword for samples $\boldsymbol{i}$ in the test set and column $\boldsymbol{j}$ is the result class value of test samples. For example, if there are 200 samples in test dataset and 6 possible binary classifiers for a 3-class problem [Table 2], then 200x6 $\boldsymbol{M}'$ is generated in exhaustive approach. In the matrix $\boldsymbol{M}'$ **,** for example, 198th row represents the codeword for 198th data sample. This codeword is compared with each base codeword generated in the encoding step by finding the hamming distance (Escalera S et al. P 2010). The minimum distance codeword is considered to be the result class of the sample dataset. The equation for hamming distance is as follows:

$$\boldsymbol{HD}(\boldsymbol{x}, \boldsymbol{y}_i) = \sum_{j=1}^{n} \frac{[\mathbf{1} - \mathbf{sign}(x^j y^j{}_i)]}{\mathbf{2}} \qquad \textbf{(3)}$$

Here $\boldsymbol{x}$ is a test codeword from $\boldsymbol{M}'$ and is a base codeword from $\boldsymbol{M}$ corresponding to class $\boldsymbol{C_i}$.

4 EXPERIMENTAL RESULTS AND ANALYSIS

To solve the multiclass classification problem with different strategies and make a comparative study, we used various datasets (Asuncion A and Newman DJ 2007). 5 different sets of data of different classes were experimented using MATLAB (Demuth H and Baele M 1994). The number of samples and features of different biological data for both training and testing is shown in Table 4. Among the datasets, only *Protein Crystallization* and *Iris* are equally distributed, i.e., each class has the same number of samples.

Table 4: Experimented biological dataset (Begum S and Aygun R 2012)

No	Data set	Classes	Images	Features
1	Iris	3	150	4
2	Thyroid	3	2978	22
3	Protein Crystallization	5	100	45
4	Breast Tissue	6	106	9
5	Ecoli	8	336	7

We provide brief analysis and comparison in the following subsections.

4.1 Complexity of training and testing

In this section we compare the training and testing complexity of greedy methods with other methods for multiclass problem. Table 5 shows the number of classifiers required for different strategies.

Table 5: Number of classifiers in testing for different strategies to solve multiclass classification problem

Name	(3 class)	(5 class)	(6 Class)	(8 class)
Greedy (Worst case)	2	4	5	7
Greedy (Best case)	1	1	1	1
MLP	1	1	1	1
OVO	3	10	15	28
OVA	3	5	6	8
EX	6	90	301	3025

From Table 5 it is clear that, in exhaustive method (EX), the number of classifiers for testing phase increases dramatically with the increases in number of classes. Notice that, greedy strategy requires less number of classifiers than OVA, OVO and exhaustive approach. In the best case, greedy strategy (top-down and bottom-up) requires only one classifier to classify an unknown sample. The number of classifiers to be used for testing is significantly low with respect to OVO method. Though multi-layer perceptron (MLP) requires only one classifier solving multiclass problem, this strategy has quite lower performance than the greedy techniques discussed in section 4.3 (especially for breast and protein crystallization problems).

The MLP is used as the base classifier in this set of experiments. The complexity of MLP depends on the structure of the neural network, the number of samples, and the number of epochs. The number of epochs is the number of times that the samples are fed into the neural network. One of the advantages of high epochs is to remove the dependency of the weights of the neural network on the order of the samples. Our MLPs have two layers: hidden and output layers. The number of neurons in the output layer is equal to the number of classes. The number of neurons in the hidden layer is user defined. Every feature is linked to all neurons in the hidden layer; and every neuron in the hidden layer is linked to all neurons in the output layer. The weights of those links and the bias value for each neuron need to be updated for each sample. Figure 8 shows a sample MLP for 3 input features and 3 output classes. The hidden layer has 4 neurons, and the output layer has 3 neurons. Such a neural network has 3*4 (from features to the hidden layer) + 4*3 (from hidden layer to output layer) weights to be computed. In addition, 4+3 (the total number of neurons) bias values for each neuron needs to be computed. For this example, the total number of weights (or values) that need to be computed is 3*4+4*3+(4+3)=31.

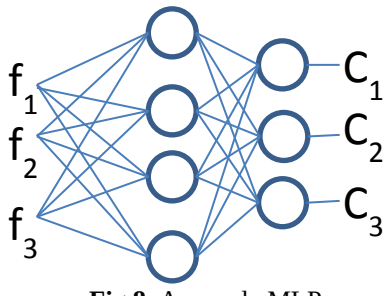


Fig 8. A sample MLP

This computation is performed per sample and repeated for the complete training set by a number of epochs. We have used 10 neurons for the hidden layer in our experiments. Assuming that the training set has *N* samples, *e* epochs, *f* features per sample, *K* classes, *10* neurons in the hidden layer, and K neurons in the output layer, the complexity of building MLP is

$$Cost(e, N, f, K) = e * N * w(f, K) \quad \textbf{(4)}$$

where

$$w(f, K) = (f * 10 + 10 * K + 10 + K) = (10f + 11K + 10)$$

and *w(f,K)* represents the number of weights to be computed.

The number of epochs is not provided as a parameter for the training. We rather stop when there is no improvement on the performance of the accuracy. So, the number of epochs is a variable number depending on how quickly the neural networks learn the model. For the MLP classifier, the number of epochs for one set of experiments for 3, 5, 6, and 8 classes was 45, 30, 47, and 59, respectively. After defining the cost of a general MLP, we may compute the cost of other strategies.

For comparison purposes, assume that the total number of training samples is N. Therefore, the cost of training a single MLP is *Cost(e,N,f,K)*. OVA method generates K number of classifiers and uses all the training set per classifier. The cost of OVA can be represented as

$$Cost_{OVA}(N, f, K) = \sum_{(i=1)}^{K} Cost(e_i, N, f, 2) \quad \textbf{(5)}$$

where e_i represents the number of epochs for the i^{th} classifier. If we assume the same number of epochs per classifier, we reach

$$Cost_{OVA}(N, f, K) = \sum_{(i=1)}^{K} Cost(e_i, N, f, 2) \quad \textbf{(6)}$$

$$= K * e_{OVA} * N * w(f,2)$$

$$= K * e_{OVA} * N * (10f + 32)$$

OVO method generates *K(K-1)/2* classifiers. Since OVO focuses on two classes, the training set only includes the samples for relevant classes. If each class has equal number of classifiers, the training set for OVO classifiers has *(N/K)*2* samples. The cost of OVO can be represented as

$$Cost_{OVO}(N, f, K) = \sum_{(i=1)}^{(K(K-1)/2)} Cost(e_i, (N/K) * 2, f, 2) \quad \textbf{(7)}$$

Again if we assume the same number of epochs per classifier,

(11)

$$Cost_{OVO}(N, f, K) = \sum_{(i=1)}^{(K(K-1)/2)} Cost(e_i, (N/K)*2, f, 2)$$

$$= (K(K-1))/2 * e_{OVO} * (N/K) * 2 * w(f,2)$$

$$= (K-1)* e_{OVO} * N * (10f + 32) \quad \textbf{(8)}$$

Our greedy techniques generate a variety of classifiers at different complexities. Let's start analyzing greedy top-down classifier. We should note that it is a recursive method. For the root node, $2^{K-1}-1$ classifiers are generated if K is odd, and $2^{K-1}-1+C(K,K/2)/2$ classifiers are generated if K is even. So, the number of classifiers for the root node, r_K, is defined as

$$r_K = \begin{cases} 2^{(K-1)} - 1 & \text{if } K \text{ is odd} \\ 2^{(K-1)} - 1 + C(K, K/2)/2 & \text{otherwise} \end{cases} \quad \textbf{(9)}$$

The cost of greedy top-down algorithm can be computed as

$$Cost_{GT}(N, f, K) = r_K * Cost(e, N, f, 2) + Cost_{GT}(N/K * M, f, B)$$

$$+ Cost_{GT}((K-M)N/K * M, f, K-B) \quad \textbf{(10)}$$

where $1 \le B \le \lfloor K/2 \rfloor$ and represents the best possible binary split for K classes. For example, if B=2, one branch has 2 classes and the other branch has K-2 classes after split.

For 3 classes, its cost is computed as

$$Cost_{GT}(N, f, 3) = r_3 * Cost_{OVA}(N, f, 3)$$

$$+ Cost_{OVO}(2N/3, f, 2)$$

$$= 3 * 3 * e_{OVA} * N * (10f + 32)$$

$$+ (2-1)* e_{OVO} * (2N/3)* (10f + 32)$$

$$= (9 * e_{OVA} + (2/3)* e_{OVO})* N * (10f + 32)$$

Let's analyze the greedy bottom algorithm for 3-class problem as follows:

$$Cost_{GB}(N, f, 3) = 3 * Cost_{OVO}(2N/3, f, 2) + Cost(N, f, 2)$$

$$= 3 * (2-1) * e_{OVO} * (2N/3) * (10f + 32)$$

$$+ e_{OVA} * N * (10f + 32)$$

$$= (2 * e_{OVO} + e_{OVA}) * N * (10f + 32) \quad (12)$$

The cost expression for the greedy bottom-up is not as straightforward as the greedy top-down. We should note that when we build one-versus-one classifier, it is possible to have classes that have varying samples. We represent it with $Cost_{OV2}$ as follows:

$$Cost_{OV2}(\{N_1, N_2\}, f) = Cost(e, (N_1 + N_2), f, 2) \quad \textbf{(13)}$$

where N_1 and N_2 correspond to the number of samples of the participating classes. We may compute the generic one-versus-one classifier using $Cost_{OV2}$ as follows:

$$Cost_{OVO}(N^* f, K) = \sum_{(i=1)}^{(K-1)} \sum_{(j=i+1)}^{K} Cost_{OV2}(\{N_i, N_j\}, f) \quad \textbf{(14)}$$

where N^* indicates that classes may have different number of samples.

We may represent the cost of Greedy bottom-up strategy as follows:

$$Cost_{GB}(N, f, K) = Cost_{OVO}(N^*, f, K) + Cost_{GB}(N, f, K-1) \quad \textbf{(15)}$$

Table 6 represents the number of binary classifiers for different approaches. Though the number of binary classifiers in training depends on the selection of best binary classifiers at each node of hierarchy, from our experimental result we can see that the number of classifiers for bottom-up greedy technique is lower than top-down.

Table 6: Number of classifiers in training for different strategies to solve multiclass classification problem

Name	(3 class)	(5 class)	(6 Class)	(8 class)
Greedy –Top-Down	4	~ 27	~44	~180
Greedy Bottom-up	4	~16	~25	~49
MLP	1	1	1	1
OVO	3	10	15	28
OVA	3	5	6	8
EX	6	90	301	3025

For example, for 8 class problem greedy bottom-up requires 49 binary classifiers in training to build the tree whereas it is 180 for greedy top-down [Table 6]. The number of classifiers for greedy top-down and bottom-up hierarchical trees depend on the architecture of selected best binary classifier in training. From Tables 5 and 6, we can see that , the number of classifiers in training and testing is same for MLP, OVO, OVA and also for EX approaches whereas its different for greedy approaches. This factor (number of classifiers) mainly determines the time to classify a sample. (Table 7 lists the training time of different strategies for different multiclass problems. We can see that MLP requires least training time among all strategies since it uses only one classifier during training. It is also noticeable that, training time for the exhaustive approach is significantly high for high number of classes. There are four factors that affect the complexity of the base classifier (MLP, in this case): the number of epochs, the number of training samples, the number of features, and the number of classes. Because of these varying factors, the running time may not increase consistently as the number of classes' increases.

The size of training set also has impact on training time. This is one of the the reasons why training time is almost the same for OVA and OVO method for 8-class problem. For 8-class problem, differences in the number of classifiers are 20, but OVA method uses the whole training dataset for each 8 classifiers. On the other hand, training set for OVO varies based on the distribution of sample in each class. Mostly, OVO uses the smallest training dataset. Greedy top-down method initially uses the whole dataset as this method uses one-vs-all binary classifiers at the root node. Training time falls down when this method goes downward and uses one-vs-one binary classifiers. For Greedy top-down, the number of classifiers at the root node is high since possible combination among all classes is high. In case of greedy bottom-up, training set is small at the bottom node as this method starts from bottom and all the classifiers are one-vs-one. Considering the size of training data set and required number of binary classifiers for training, our proposed greedy bottom-up approach is faster than greedy top-down for 3 to 8 class problems [Table 7]. Note that the number of epochs, the number of samples, the number of features, the number of classes, and the number of classifiers to be built influence the running time and running time may not always consistently increase..

Table 7: Training time (in minute) for different strategies to solve multiclass classification problem

Name	Iris	Protein Crystallization	Breast Tissue	Ecoli
	(3 Class)	(5 class)	(6 Class)	(8 class)
Greedy top-down	~.4	~3	~5	~8
Greedy bottom-up	~.4	~2.5	~4	~6
MLP	~.3	~.5	~.5	~.5
OVO	~1	~2.22	~4	~5
OVA	~.3	~1	~2.3	~3
EX	~2	~20	~120	~1080

4.2 Analysis of accuracy of binary classifiers

This section provides accuracy analysis for different neural binary classifiers for 5-class problem. For neural binary classifier, 80% of the data are used for training and 20% for validation/testing. First, for each dataset all possible binary classifiers based on neural networks are trained using (1). In order to compare statistical analysis of neural binary classifiers with different macro classes, we use a histogram plot of each multiclass problem. For example, Fig. 9 shows the histogram plot of different classifiers for 5-class problem (Protein Crystallization). It represents that, binary classifier one-versus-two has the highest number of binary classifiers and most of its classifiers have 90% to 100% accuracy. It is also noticeable from Table 8 that the performance of ${}_{I}\boldsymbol{B}_{III}$ (one-versus-three) binary classifier is the best binary classification (considering min, max and mode) for this dataset.

As different macro classes of binary classifiers may have different number of samples for each class, we also consider misclassified samples (MS) to measure the performance which is shown in Table 9. From these results it can be concluded that, only ${}_{I}\boldsymbol{B}_{I}$ (one-versus-one) and ${}_{I}\boldsymbol{B}_{II}$ (one-versus-two) binary classifiers are on the top positions of the rank. On the other hand, most of the ${}_{I}\boldsymbol{B}_{III}$ (one-versus-three) binary classifiers are in the bottom ten.

Table 8: Performance comparison of binary classifiers in accuracy for protein crystallization dataset

Top 10				Bottom 10		
Binary Classifier Name	***Accuracy***	***No. of Misclassified Samples***		***Binary Classifier Name***	***Accuracy***	***No. of Misclassified Samples***
$_1\mathbf{b}_2$	100	0		$_3\mathbf{b}_{1\,2\,4\,5}$	80	20
$_1\mathbf{b}_5$	100	0		$_4\mathbf{b}_{1\,3\,5}$	80	16
$_2\mathbf{b}_5$	100	0		$_4\mathbf{b}_{3\,5}$	76.66	14
$_2\mathbf{b}_{1\,3}$	100	0		$_1\mathbf{b}_{2\,3\,5}$	82.5	14
$_2\mathbf{b}_{3\,4}$	100	0		$_4\mathbf{b}_{2\,5}$	81.66	11
$_1\mathbf{b}_3$	97.5	1		$_4\mathbf{b}_{2\,3\,5}$	86.25	11
$_1\mathbf{b}_4$	97.5	1		$_3\mathbf{b}_5$	75	10
$_2\mathbf{b}_4$	97.5	1		$_4\mathbf{b}_{2\,3}$	85	9
$_1\mathbf{b}_{2\,3}$	98.33	1		$_1\mathbf{b}_{2\,3\,4}$	88.75	9
$_2\mathbf{b}_{3\,5}$	98.33	1		$_3\mathbf{b}_{1\,2\,4}$	88.75	9

Table 9: Performance results of binary classifiers for protein crystallization dataset

Binary Classifier Name	**Min**	**Max**	**Mode**
$_{\mathrm{I}}B_{\mathrm{I}}$ (One-Versus -One)	75	100	97.5
$_{\mathrm{I}}B_{\mathrm{II}}$(One-Versus -Two)	76.6	100	95
$_{\mathrm{I}}B_{\mathrm{III}}$(One-Versus-Three)	80	100	97.5
$_{\mathrm{I}}B_{\mathrm{IV}}$(One-Versus-Four)	80	98	95
$_{\mathrm{II}}B_{\mathrm{II}}$(Two-Versus-Three)	88.75	97.5	93.75
$_{\mathrm{II}}B_{\mathrm{III}}$(Two-Versus-Three)	90	95	95

4.3 HBCs with different classifiers

4.3.1 Using neural networks as binary classifiers

We have initially used neural networks for the internal nodes of the HBC. Table 10 shows the performance comparison of greedy strategy with OVO, OVA, exhaustive and multi-layer perceptron (MLP) network for different datasets of different classes.

To integrate the results of OVO, OVA and exhaustive approach, ECOC with hamming decoding method has been used. We also generate all possible binary hierarchical trees for *Thyroid*, *Iris* and *Protein Crystallization* using (2) and make comparison of the best and the worst tree with other strategies. Note that, for 3-classes the performance accuracy is almost the same for all strategies. From the third row in Table 10, we see that the best hierarchical binary tree outperforms greedy, MLP, OVO for 5-class problem. We provide the performance of the best HBC to check how good greedy algorithms are good at building HBCs. It can be also seen from Fig. 10 that, performance accuracy of greedy strategy is high for most multiclass problems comparing to MLP, OVO and OVA and this strategy is significant for 8-class problem (93% accuracy). When the number of classes becomes higher we can expect larger differences between greedy and other strategies (MLP, OVO and OVA).

Based on the performance of trained classifiers, both top-down and bottom-up greedy hierarchical trees are created for all datasets. As it is not possible to show all greedy hierarchical trees, we only compare top-down and bottom-up greedy structure for *Protein Crystallization* dataset [Table 11].

In this table, ***b*** is the binary classifier and ***MS*** is the misclassified samples at that level of the tree. Note that, both the hierarchical top-down and bottom-up trees start with one-versus-all and go downward in this way. It can be also seen that, the number of misclassified samples are less in top-down structure than bottom-up for this dataset.

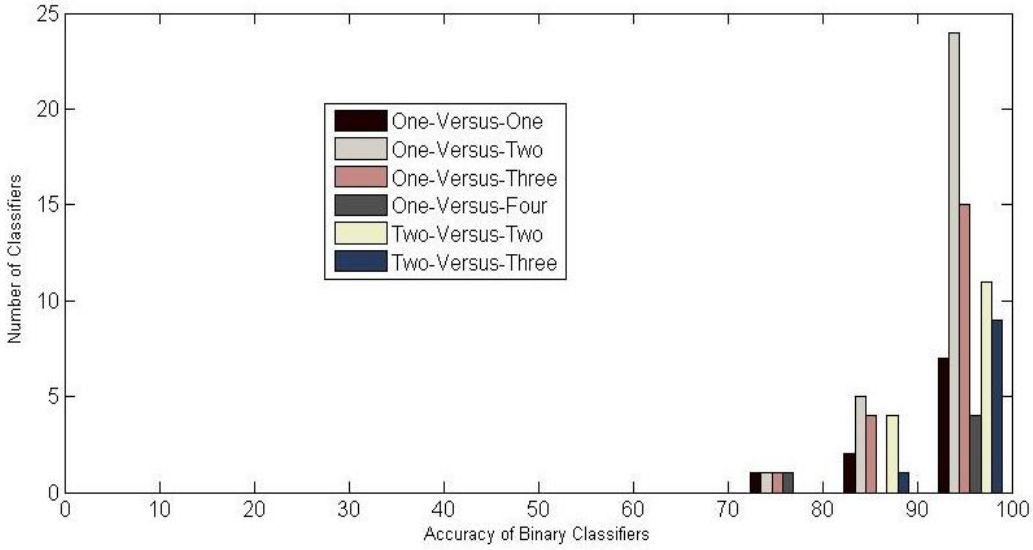


Fig. 9. Histogram plot of binary classifier of protein crystallization dataset

Table 10: Comparison of test results for different strategies to solve multiclass classification problem

No. of Classes	Best Hierarchical Binary Tree	Worst Hierarchical Binary Tree	Greedy Bottom-Up	Greedy Top-Down	MLP	OVO	OVA	Exhaustive Approach
3(Thyroid)	100	97.84	100	100	99.7	97	99.8	99.8
3 (Iris)	98.7	98.7	98.7	98.7	97.3	100	98.7	100
5(Protein Crystallization)	92	68	89	90	82	79	82	99
6(Breast Tissue)	×	×	89.6	89.6	78.301	68.9	92.5	100
8(Ecoli)	×	×	91.17	93.79	91.667	79.5	89.3	96.42

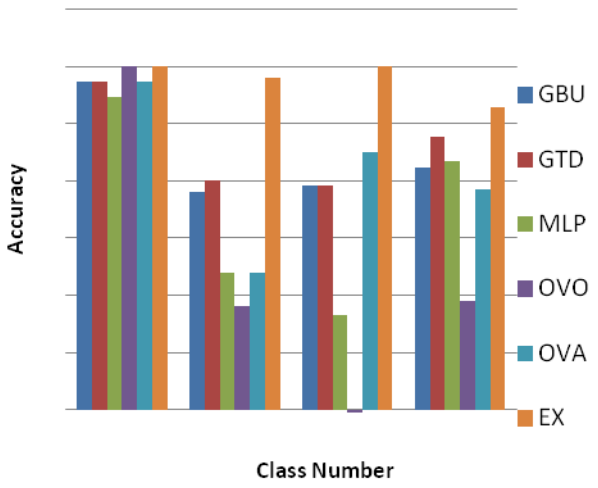


Fig. 10. Comparative results of GBU (Greedy Bottom-Up), GTD (Greedy Top-Down), MLP (Multi-Layer Perceptron), OVO (One-Versus-One), OVA (One-Versus-All) and EX (Exhaustive) approach

Table 11: Top- down and bottom-up tree structures for protein crystallization dataset (Begum S and Aygun R 2012)

Name	Greedy (Top-Down)	Greedy (Bottom-Up)
Tree Structure	2 **b** 1345 (MS-2) 5 **b** 134 (MS-3) 3 **b** 14 (MS-4) 1 **b** 4 (MS-1)	4 **b** 1235 (MS-6) 3 **b** 125 (MS-5) 5 **b** 12 (MS-0) 1 **b** 2 (MS-0)
Misclassified Samples	**10**	**11**

4.3.2 Using naïve Bayesian classifier as binary classifiers

In this work, we also build the greedy hierarchical tree using naïve Bayesian classifier for 5 to 8 class problem which is shown in Table 12. Though the bottom-up technique is fast enough, greedy top-down outperforms bottom-up considering accuracy. From Table 12 we can see that, the performance of greedy for Ex (exhaustive) using naïve Bayesian classifier is not high for all cases. Among OVO and OVA, OVO gets better accuracy and it's significant for 8–class problem. Since for greedy bottom-up hierarchical tree using naïve Bayesian classifier, the performance of root node is less than 75% for 5 to 8 class problem, we use the improved greedy bottom-up to improve the performance. The improved greedy bottom-up (IGBU) provides around 10% better accuracy than OVA using the naive Bayesian classifier. The results for IGBU are almost the same as the OVO. If we compare the results of the hierarchical classifier against OVO and OVA, our proposed technique provides more stable results than OVO and OVA. When using neural classifiers, OVO method generated very low accuracy results for 5- class, 6-class, and 8-class problems [Table 9]. On the other hand, when naive Bayesian classifiers are used, the accuracy of OVA was very low. However, our hierarchical classifier provided better results in general than OVO and OVA if MLP neural networks are used. While OVA method performs low using naïve Bayesian classifier, our improved hierarchical classifier performs as good as OVO.

Table 12: Comparison of test results for different strategies using naïve Bayesian classifier to solve multiclass classification problem

No. of Classes	Greedy Bottom-Up	Improved Greedy Bottom-Up	Greedy Top-Down	OVO	OVA	EX
5(Protein Crystallization)	73	88	89	87	69	86
6(Breast Tissue)	65.01	75	76	78.3	67.92	80.3
8(Ecoli)	81.84	89	89.5	90.17	83.61	87.5

5 CONCLUSION

The multi-class classification techniques such as one-versus-one and one-versus-all have been used in the literature assuming that they will outperform a single multi-class classifier. In this paper, we propose a greedy technique for building hierarchical binary classifiers for the multi-class classification problem. We compare our greedy techniques with OVO and OVA techniques.

We have tested and compared our method using 5 different biological datasets. In our analysis, we realize that greedy bottom-up produces less number of classifiers to be tested than the top-down greedy version. Although the number of classifiers is for bottom-up greedy classifier is close to the summation of the number of classifiers in OVO and OVA.

In terms of accuracy, greedy top-down usually outperformed the greedy bottom-up version, and in general these greedy classifiers outperformed OVO and OVA. If the hierarchical binary classifier is based on the MLP at the internal nodes both greedy techniques outperformed the OVO and OVA. When naïve Bayesian classifier at the internal nodes, the top-down version performs better than OVA but performs similarly as OVO. Our improved bottom-up greedy technique also performs similarly as the

top-down method. The exhaustive method should provide the optimal method. With the MLP classifier, the exhaustive method can produce high accuracies. However, with the naïve Bayesian classifier, the exhaustive method did not generate the best results.

In this paper, without adding significant cost with respect to OVO, we have shown that our greedy hierarchical classifiers can outperform widely used OVO and OVA techniques. Due to the design and use of the greedy logic, we minimize the number of classifiers to be built at the training stage. Since the number of classifiers for testing is at most equivalent to the number of classifiers in OVA, our greedy hierarchical binary classifiers can be used in many applications. As future work, our proposed method can be tested with other types of classifiers such as support vector machine at the internal nodes.

ACKNOWLEDGMENT

We would like to acknowledge Marc Pusey, Ph.D., of iXpressGenes, Inc. for providing the Protein Crystallization dataset and Madhav Sigdel for extracting features from this dataset.

References

Allwein, Schapire R and Singer Y(2002) Reducing Multiclass To Binary: A Unifying Approach For Margin Classifiers. Journal of Machine Learning Research, 1:113–141, doi: 10.1162/15324430152733133

Asuncion A and Newman DJ(2007) Uci Machine Learning Repository.University of California, Irvine, School of Information and Computer Sciences, http://mlearn.ics.uci.edu/MLRepository.html. Accessed: May 2012

Bay SD (1998) Combining Nearest Neighbor Classifiers Through Multiple Feature Subsets. In Proceedings of the 17th International Conference on Machine Learning, pp 37–45, Madison, WI

Begum S and Aygun R(2012) Analyzing the Performance of Hierarchical Binary Classifiers for Multi-class Classification Problem Using Biological Data. ICMLA 2, page 145-150. IEEE, doi: 10.1109/ICMLA.2012.165

Bishop CM(1995) Neural Networks for Pattern Recognition. Oxford University Press

Breiman L, Friedman J, Olshen R and Stone C(1984) Classification and Regression Trees. Chapman and Hall

Casasent D and Wang Y(2005) A Hierarchical Classifier Using New Support Vector Machine For Automatic Target Recognition. In IJCNN , IEEE, Volume 18, Issue 5-6, pp 541-548 , doi:10.1016/j.neunet.2005.06.033

Cortes C and Vapnik V(1995) Support-Vector Networks. Machine Learning, pp 273–297, doi:10.1007/bf00994018

Demuth H and Baele M (1994) Neural Network Toolbox. User's Guide. The MathWorks, Inc., Natick, MA

Duda R, Hart P and Stork D (2000) Pattern Classification. New York:Wiley- Interscience

El-Alfy E (2010) A Hierarchical GMDH-Based Polynomial Neural Network for Handwritten Numeral Recognition Using Topological Features. In IJCNN , IEEE, p. 1-7

Escalera S, Pujol O and Radeva P(2008) On the Decoding Process in Ternary Error-Correcting Output Codes. Pattern Analysis and Machine Intelligence, IEEE Transactions on, Vol. 32, No. 1. pp. 120-134, doi:10.1109/TPAMI.2008.266

Escalera S, Pujol O, and Radeva P(2009) Separability Of Ternary Codes For Sparse Designs Of Error Correcting Output Codes. Pattern Recognition Letters, 30:285–297, doi:10.1016/j.patrec.2008.10.002

Escalera S, Pujol O and Radeva P(2010) Error-Correcting Ouput Codes Library. In: J. Mach. Learn. Res., Vol. 11 Cambridge, MA, USA: MIT Press, p. 661—664

Friedman J(1996) Another Approach To Polychotomous Classification. Technical Report, Department of Statistics, Stanford University

Gupta K, Agarwal K, Prakash N, Singh B, Misra K (2012) Prediction of miRNA in HIV-1 genome and its targets through artificial neural network: a bioinformatics approach. Network Modeling Analysis in Health Informatics and Bioinformatics, Volume 1, Issue 4, pp 141-151, doi: 10.1007/s13721-012-0017-3

Hastie T and Tibshirani R(1998) Classification By Pairwise Coupling. Advances in neural information processing systems, vol. 10, MIT Press, pp. 507—513

Hulse J; Khoshgoftaar M,Napolitano,A and Wald, R (2012) Threshold-based feature selection techniques for high-dimensional bioinformatics data. Network Modeling Analysis in Health Informatics and Bioinformatics, Volume 1, Issue 1-2, pp 47-61, 10.1007/s13721-012-0006-6

Guyon I, Gunn S, Nikravesh M and Zadeh L (2006) An Enhanced Selective Naive Bayes Method with Optimal Discretization. Feature Extraction: Foundations And Applications, Chap. 25, pp. 499-507, Springer

Jain P, Wadhwa P, Aygun R, and Podila G(2008) Vector–G: Multi-Modular SVM-Based Heterotrimeric G-Protein Prediction. In Silico Biol. Vol. 8, Number 2, pp. 141-155

Kumar S, Gosh J and Crawford M (2002) Hierarchical Fusion Of Multiple Classifiers For Hyperspectral Data Analysis. Pattern Analysis and Applications 5 ,210–220, doi:10.1007/s100440200019

Lorena A and Carvalho A(2008) Tree Decomposition Of Multiclass Problems. Proceedings of the Brazilian Symposium on Neural Networks (SBRN), pp. 189–194, doi:10.1109/SBRN.2008.43

Nagi S, Bhattacharyya D (2013) Classification of microarray cancer data using ensemble approach, Network Modeling Analysis in Health Informatics and Bioinformatics, doi: 10.1007/s13721-013-0034-x

Quinlan J(1993) C4.5: Programs for Mahine Learning. Morgan Kaufmann

Platt JC, Cristianini N and Shawe-Taylor J(2000) Large Margin Dags For Multiclass Classification. Advances in Neural Information Processing Systems, MIT Press, pp. 547–553

Rish I(2001) An Empirical Study Of The Naive Bayes Classifier. In IJCAI Workshop on Empirical Methods in Artificial Intelligence

Sánchez-Maroño N, Alonso-Betanzos A, Garcia-Gonzalez P and Bolón-Canedo V (2010) Multiclass Classifiers Vs Multiple Binary Classifiers Using Filters For Feature Selection. IJCNN, IEEE , p. 1-8

Tibshirani R and Hastie T(2007) Margin Trees For High-Dimensional Classification. Journal of Machine Learning Research, volume 8 , pp. 637–652

Vural V and Dy JG(2004) A Hierarchical Method For Multi-Class Support Vector Machines. Proceedings of the 21st International Conference on Machine Learning, pp. 105, doi:10.1145/1015330.1015427

Wang Y and Casasent D(2006) Hierarchical K-Means Clustering Using New Support Vector Machines For Multi-Class Classification. In Proceedings of the international joint conf. on neural networks pp. 3457–3464