ORIGINAL RESEARCH

Partitioning trees: A global multiclass classification technique for SVMs

Ioannis Constantinidis*, Ioannis Andreadis

Democritus University of Thrace, Greece

Received: June 6, 2014 **Accepted:** June 23, 2014 **Online Published:** July 11, 2014
DOI: 10.5430/air.v3n2p41 **URL:** http://dx.doi.org/10.5430/air.v3n2p41

Abstract

Presented in this paper is a novel technique for multiclass classification in SVMs through combination of binary classifiers, namely that of Partitioning Trees (P-Trees). The technique aims at improving the Directed Acyclic Graphs (DAGs) both in terms of training as well as testing performance. It works by progressively constructing a decision graph, where each node is a binary classifier. Each trained node defines a dichotomy over the instance space which, in turn, is used to train subsequent nodes. In this way, every node trains against only a subset of the samples of its classes; namely the samples that reach the node through the decision graph in addition to a subsampled version of the ones that fail to reach it. Training sets reduce in size and decision surfaces become more compact, thus improving training and testing performance. Extensive experimental results demonstrate the effectiveness of the proposed technique in reducing the training and testing time in SVMs, while maintaining comparable generalization performance to the 1vs1 and DAGs techniques.

Key Words: Multiclass classification, Partitioning trees, Multi-class SVMs, Supervised learning, Global techniques

1 Introduction

There is an abundance of real life applications pertaining to the classification problem. In classification, the central task is the estimation of an optimal mapping that assigns a decision label (class) to each vector of measurements (feature vector).

Any supervised learning algorithm is at least capable of explicitly addressing a binary classification problem. Extending the applicability of these algorithms to multiclass classification problems is not always straight-forward, in as much as it might require structural changes that either invalidate some of the fundamental properties of the algorithms (e.g. convexity of the optimization problem) or create practical problems (e.g. super-linear increase of training time with respect to the number of classes). For example, in the case of Support Vector Machines (SVMs),[3] their ability to successfully address binary classification problems has been well documented.[15,32] Their extension to multiclass classification problems is well studied,[18,20,34] yet still considered an open issue.[27]

Effectively, there are currently two approaches in extending binary classifiers to the multiclass case. The first approach works by explicitly reformulating the underlying structure of the classifier, resulting in a unified multiclass optimization problem (embedded or single machine techniques),[28] while the second approach works by dividing a multiclass problem into multiple, independently trained, binary classification problems and properly combines them so as to form a multiclass classifier (combinational techniques).[8]

Combinational techniques are to date some of the most popular and successfully used methods,[11,28,31] as they share some relative advantages over embedded techniques, especially in the context of SVMs. They are global, for they can be readily applied to any binary classification algorithm. Training time scales well with respect to the number of classes in a multiclass problem. Additionally, their implementation is very simple—in contrast to embedded techniques—and efficient, as they allow the use of highly optimized versions of the underlying binary classifiers. Some popular combinational techniques are the 1-vs- All,[32] 1-

*Correspondence: Ioannis Constantinidis; Email: ioconst@gmail.com; Address: Democritus University of Thrace, Greece

vs-1[19] and DAGs technique,[27] all of which are briefly described in the following sections.

Conversely, embedded techniques can oftentimes provide superior generalization performance,[20, 28] at the expense of significantly increased implementation complexity or even training time, thus outweighing their relative advantages for many practical scenarios. This paper focuses on combinational methods for multiclass SVMs. We now review some recent advances in the field. Vural and Dy[33] introduce a new framework for multiclass SVMs which they call Divide-by-2. Initially all classes are grouped in two sets forming a binary classifier and subsequently, as the name implies, each group is split until each node becomes a decision node. This results in faster training than the 1-vs-All method and faster testing than both the 1-vs-All and 1-vs-1 methods. A similar approach is followed by Madzarov et al.,[22] with different heuristics in the grouping of classes.

Lei and Govindaraju[21] propose a method called Half-Against-Half for multiclass SVMs. They create a decision tree by training at each node half the classes against the remaining ones. They present criteria for the grouping of classes. Their results demonstrate faster testing than both the 1-vs-All and 1-vs-1 methods, for comparable generalization performance. Garcia Pedrajas and Ortiz Boyer[17] propose a method that combines the 1-vs-All and 1-vs-1 techniques in order to achieve superior classification accuracy, with a penalty in training time.

Tang and Mazzoni[31] propose a new framework called Reduced-Set SVMs in order to reduce the overall number of support vectors used in the testing stage, thus significantly reducing testing time. Chen et al.[7] provide a very interesting approach to the problem of reducing the testing time of multiclass SVMs, by use of an adaptive tree in order to select binary SVMs with the fewest average number of support vectors.

In this paper a new method for multiclass SVM classification is introduced, namely the Partitioning Trees (P-Trees). It is a modification of the DAGs technique that maintains and improves on the advantage of fast testing, while introducing significant reduction in training time. The proposed scheme is based on an iteration of training and testing that aims at reducing redundancies in the final decision graph. When a node is trained against two classes, the remaining classes are tested against it before any other node is trained. In this way, each trained node dichotomizes all remaining classes. In a subsequent choice of the classes against which another node is to be trained, instead of using the entire training set corresponding to a chosen class, the active set of each class is used.

The active set of a class is defined as the set containing all the class samples that reach the node through the graph, in addition to a properly subsampled version of the class samples that fail to do so. The training/testing iteration aims at reducing the redundancy of the training scheme, by excluding from training all samples that would never reach the node during training. The subsampling addition aims at reintroducing a rough picture of the information hidden from each node, without significantly affecting training performance.

Due to the super-linear scaling of many practical classification algorithms with respect to the training set size—including but not limited to SVMs[26]—the proposed technique can provide significant reduction in training time, while maintaining comparable generalization performance to that of DAGs. Empirical tests presented at the end of this paper show slightly above linear reduction of training time, with respect to the number of classes of the problem.

In what follows, Section 2 provides a general description of the multiclass classification problem and a broad categorization of various multiclass classification algorithms based on some distinctive attributes. In Section 3 we review some popular combinational techniques. Presented in Section 4 is the necessary theoretical framework upon which the proposed P-Trees technique is based. P-Trees are presented in detail in Section 5. Section 6 contains experimental results, while Section 7 contains concluding remarks.

2 Multiclass classification

Assume N training samples (x_i, t_i) are available, forming the training set $D = \{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$ where

$$(x_i, t_i) \subset \chi \times C \tag{1}$$

with $\chi \subset \Re^d$ the instance space of the d-dimensional input vectors x_i and $C \subset N$ the decision space of the class labels t_i. Sets C contains the decision labels, and without loss of generality we will assume that $C = \{1, 2, \ldots K\}$. Unless otherwise specified, an element of set C will be denoted by ω_i, with $i \in 1, 2, \ldots K$.

A set A will generally be denoted with capital calligraphic. A partition of a set A will be denoted by π_A, while all possible partitions of a set A will be denoted by PARTA.

Returning to the classification problem, $f_{\omega_a\omega_b}$ will denote a binary classifier trained against class labels ω_a and ω_a

$$f_{\omega_a\omega_b} : \{\chi^{\omega_a} \cup \chi^{\omega_b}\} \mapsto \{\omega_a, \omega_b\}, \omega_a, \omega_b \in C \tag{2}$$

where χ_{ω_i} the instance space of class ω_i

$$\chi_{\omega_i} = x : x \in \chi, p(\omega_i|x) > 0 \tag{3}$$

In the more general case of multiclass classification, f_{π_A} with $A \subset C$ will denote the classifier assigning a class label $\omega_i \in \pi_A$ for every input x

$$f_{\pi_A} : \bigcup_{\omega_k \in \pi_A} \chi_{\omega_k} \to \pi_A, A \subset C \tag{4}$$

Resampling and global techniques

Both resampling and global techniques can be viewed as special cases of combinational methods. Their difference lies on the way they combine the various decisions of the binary classifiers, in order to form the final decision.

Resampling methods learn how to optimally combine the various binary classifiers by an iterative process, which aims at gradually improving the accuracy of the available classifiers. Evidently, through this iterative process the techniques learn how to use and combine the information within each classifier. This in turn means that the available classifiers are considered weak, or are believed to have a significant amount of dependence (for example high correlation or mutual information). The ensemble learning techniques,[10] such as Bagging,[5] Random Forests,[4] Boosting[29] with AdaBoost[16,30] and its variants, are some of the most popular methods falling in this category.

On the other hand, global techniques follow a consistent way of combining the decisions of the various classifiers, independently of the form of the available classifiers at hand. The lack of adaptation in this case implies that the available classifiers are considered strong and relatively independent of one another.

In what follows, global techniques for multiclass classification will be examined. Henceforth, it will be implicitly assumed that the available classifiers are strong and relatively independent.

3 Popular global techniques

We now review some of the most popular global techniques for multiclass classification. We also briefly comment on their relative advantages and disadvantages, mostly focusing on training and testing time.

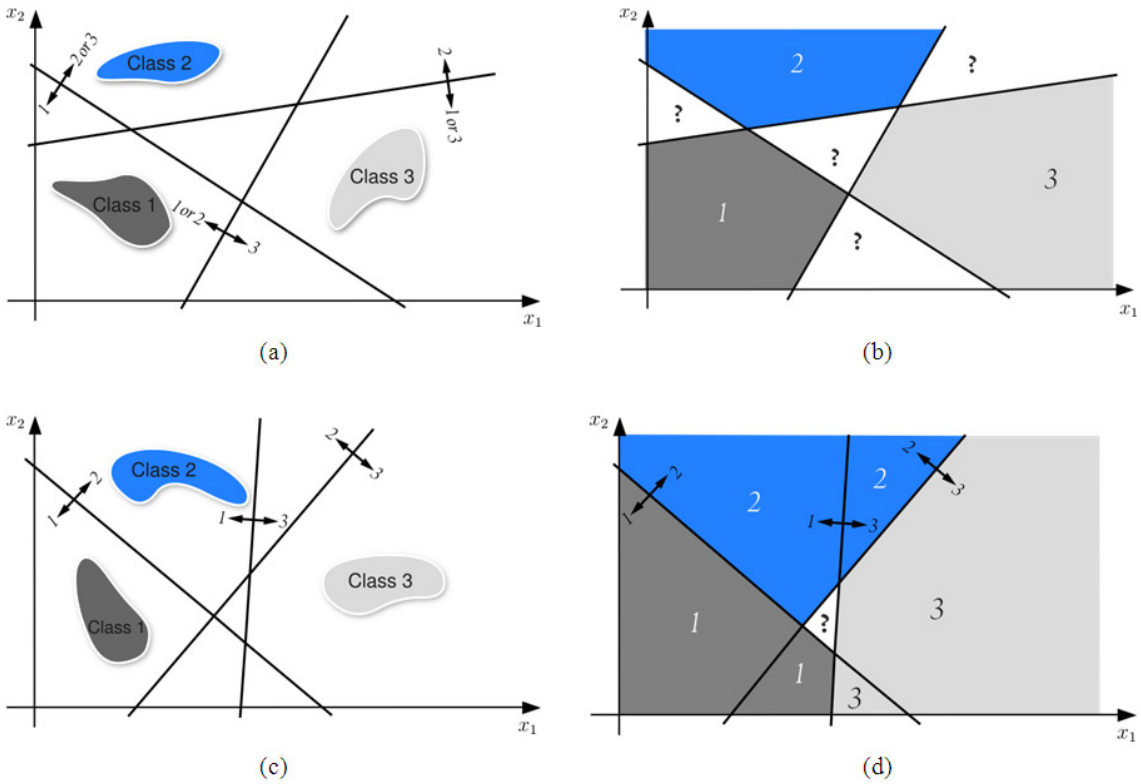

Figure 1: Ambiguous decision regions (denoted by ?) for majority voting. Trained classifiers and corresponding decision regions through majority voting. The 1-vs-All case is depicted in (a) and (b), while the 1-vs-1 case is depicted in (c) and (d).

3.1 1-vs-1

For a multiclass problem of K classes, the 1-vs-1 technique[19] trains $K(K-1)/2$ binary classifiers f_{π_k}, one for every unique pair of classes π_k, namely

$$\pi_k = \{i, j\}, \forall i \in C, j \in C \backslash i \tag{5}$$

Majority voting is typically employed for testing, though any other compatible technique is valid. In majority voting, for each feature vector x a K-dimensional vector v is constructed with each dimension v_i assuming a value equal to the sum of binary classifiers voting in favor of class i, namely

$$v_i = |V_i| \tag{6}$$

with

$$V_i = \{k : f_{\pi_k}(x) = i, i \in C\} \quad (7)$$

The final decision $\alpha \in C$ is then taken through majority voting

$$\alpha = \arg\max_i v_i \quad (8)$$

Majority voting may lead to ambiguous decision regions, whenever there are more than one classes sharing the same number of maximum votes, as depicted in Figure 1. Solutions as simple as random or fixed choices (e.g. the first from a predefined list of classes) between the classes bearing the maximum number of votes can effectively alleviate the problem.

In general, the $K(K-1)/2$ binary classifiers differ in training size. Assuming, for simplicity, that each classifier has an equal amount of training samples $2N/K$, the overall complexity of training all classifiers is $\frac{K(K-1)}{2}T(\frac{2N}{K})$, where $T(N)$ the amount of operations required to train a binary classifier with the employed algorithm, as a function of the training size N. The testing stage requires the examination of $K(K-1)/2$ binary classifiers. Each classifier is relatively simple, as it only separates two classes.

3.2 1-vs-All

The 1-vs-All technique is one of the first global techniques to appear in the literature.[32] For K classes, this method constructs an equal number of binary classifiers f_{π_k} with

$$\pi_k = \{\{k\}, C\backslash k\}, \forall k \in C \quad (9)$$

by training each class against all the rest. Testing can once again be performed through majority voting,[17] although there are many alternative choices that avoid the problem of ambiguous regions of majority voting (see Figure 1), such as the highest output value.[9, 13]

The 1-vs-All technique trains K classifiers of size N, giving an overall complexity of $KT(N)$ operations, for the training stage. At the testing stage, it requires the examination of K classifiers for each test vector x. Each classifier is typically complex, as it needs to befittingly separate each class from all the rest.

In practice, since for virtually all practical problems $K \ll N$, this method may lead to notably decreased training performance when compared to the 1-vs-1 method, provided it can be assured that the underlying training algorithms scales super-linearly with respect to the training size. However, there is a standing controversy as to whether this is true for large scale problems.[28]

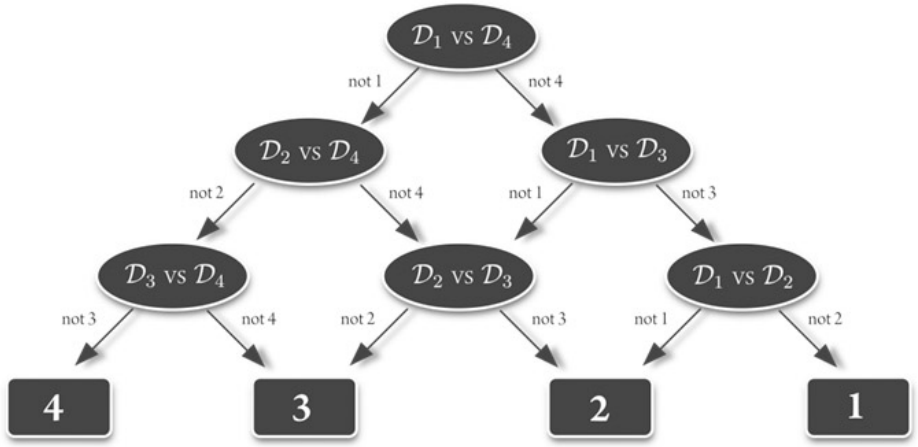

Figure 2: DAG example for 4 decision classes. Note that the positioning of the nodes corresponds to just one out of the many possible choices

3.3 DAGs

The DAGs (Directed Acyclic Graphs) technique proposed by Platt et al.[27] is an alternative way of testing for the 1-vs-1 technique. Just like the 1-vs-1 technique, it trains all $K(K-1)/2$ binary classifiers for each pair of classes. The difference lies in the way it combines them so as to form the final multiclass decision.

DAGs use complementary interpretation of the decision of each binary classifier. If a classifier f_{w_i,w_j} decides w_i then w_i is rejected. Through a series of successive disqualifications of candidate decision classes, the final decision is reached. The testing stage can then be interpreted in the following way:

(1) Form a list L containing any random permutation of the decision classes C.

(2) Choose the trained classifier corresponding to the classes of the first ω_s and last ω_e element of the list $(\omega_s, \omega_e \in L)$.

(3) Examine $f_{w_s,w_e}()$ for a test vector x and remove dis-

qualified class from the list.
(4) Repeat the previous steps until only one element remains in the list (decision class for x).
It is clear that DAGs only require K-1 examinations of binary classifiers to reach the final decision, in comparison to the $K(K-1)/2$ examinations required for 1-vs-1 testing with majority voting.
The above process is equivalent to a directed acyclic graph (see Figure 2). Note that access to any two elements in the list can be performed, in general, with more than one ways. That is, each node can have more than one parents.

4 Bayes optimal classifiers and partitioning

When the class-conditional probability density function $p(x|w_i)$ is analytically known, the Bayes optimal classifier can be derived. We prove in this section that the Bayes optimal classifier is left unaffected under both decision and instance space partitioning.
By introducing a loss function $\lambda()$ that quantitatively defines optimality in predictions, the problem of estimating the Bayes optimal classifier is well defined and admits a closed form solution. Assuming the loss function assigns different weights for the misclassification of each class ω_i, independently of the estimated class $\hat{\omega_j}$, namely

$$\lambda(\hat{\omega_j}|\omega_i) = \lambda(\omega_i), \forall \hat{\omega_j}\omega_i \in C, \hat{\omega_j} \neq \omega_i \tag{10}$$

the Bayes optimal classifier is the one minimizing the mean weighted loss

$$f^{id}(x) = \arg\min_{\omega_i \in C} \sum_{k \neq i} \lambda(\omega_k) p(\omega_k|x) \tag{11}$$

which is equivalent to

$$f^{id}(x) = \arg\max_{\omega_i \in C} \sum_{k \neq i} \lambda(\omega_i) p(\omega_i|x) \tag{12}$$

since

$$\arg\min_{\omega_i \in C} \sum_{k \neq i} \lambda(\omega_k) p(\omega_k|x) = \arg\min_{\omega_i \in C} \{-\lambda(\omega_i) p(\omega_i|x) + \sum_k \lambda(\omega_k) p(\omega_k|x)\} = \arg\max_{\omega_i \in C} \lambda(\omega_i) p(\omega_i|x)$$

4.1 Invariance under instance space partitioning

By examining the binary classifier in (12)
$f^{id}(x) = \arg\max_{\omega_i \in C} \sum_{k \neq i} \lambda(\omega_i) p(\omega_i|x)$
It is evident that the optimal decision is local with respect to x, since it is independent of any probability density function or loss function at other points in input space χ. Thus any partition of the instance space $\pi_\chi \in PART\{\chi\}$ breaks the problem of estimating the Bayes optimal classifier into $|\pi_x|$ independent sub-problems.

4.2 Invariance under decision space partitioning

Let π_C be any partition of the decision space C. Then if

$$B = \bigcup_{A \in \pi_c} \arg\max_{\omega_i \in A} \lambda(\omega_i) p(\omega_i|x) \tag{13}$$

it can be shown that

$$\arg\max_{\omega_i \in B} \lambda(\omega_i) p(\omega_i|x) = \arg\max_{\omega_i \in C} \lambda(\omega_i) p(\omega_i|x) \tag{14}$$

The right side of (14) is identical to the Bayes optimal classifier in (12). In this way, a Bayes optimal multiclass classifier (12) can be broken down into $|\pi_C|$ independent binary sub-problems, where π_C is any partition of the decision space C.

4.3 Combined instance-decision space invariance

Both (13) and (14) are local with respect to x. By combining the properties of instance and decision space partitioning invariance, combined partitioning can be achieved. In this way, any partitioning of the form $\pi_{\chi \times C}$ breaks the multiclass Bayes optimal classifier into $|\pi_{\chi \times C}|$ sub-problems, each of which can be solved independently.

5 Proposed technique - partitioning Trees

From the various global multiclass techniques introduced in Section 3, the DAGs appears to be an ideal choice, in terms of testing performance. In this section we introduce a new global technique for multiclass classification that provides significantly improved training and testing performance, compared to the DAGs technique.
For the remaining of this Section, the fundamental steps of the P-Tree classifier are described in subsection 5.1. Subsection 5.2 provides an algorithmic description of the training and testing process on a P-Tree, while subsection 5.3 provides a proof of the reduction of overall training time incurred by P-Trees, under very mild assumptions. Finally, a note on the parallelization of binary node training for P-Trees is provided in subsection 5.4.

5.1 P-Trees description

In this section we introduce the core version of the proposed technique, namely the Partitioning Trees (P-Trees). The idea is inspired by the DAGs technique, hence testing is performed in a decision graph.
One disadvantage of the DAGs technique is that, given it always tests samples in a graph-like structure, the training stage leads to significant redundancies. A node always uses

the entire training sets of the classes against which it is trained. The redundancy originates from the fact that for every node other than the root node, the previous nodes within its path will inevitably form a partition of instance space, as shown in Figure 3. This in turn means that a significant amount of the training samples of a class used within a node will never get to reach the node during testing. This redundancy causes a significant performance hit, especially for deep graphs (many decision classes), provided that the training time of many binary classifiers scales super-linearly with respect to the size of the training set.

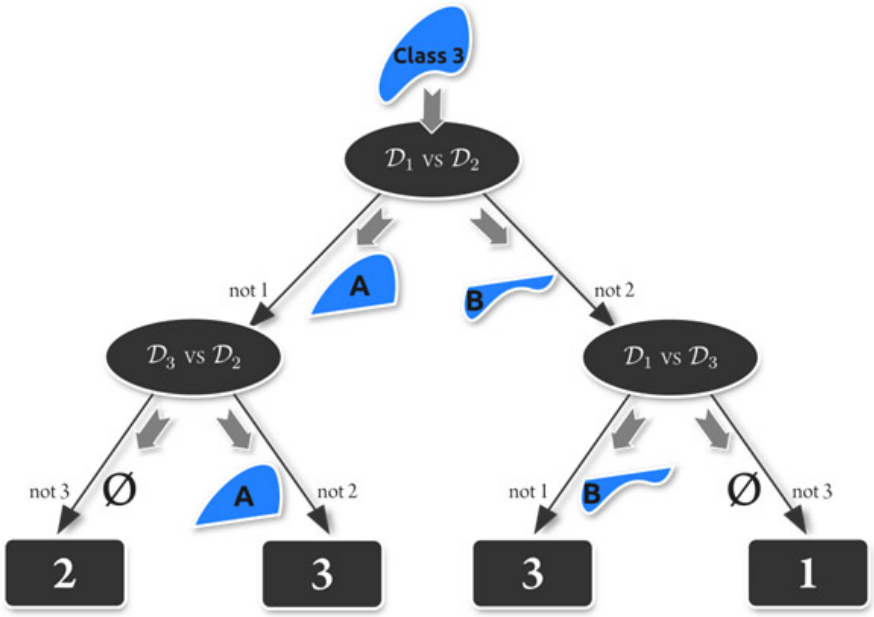

Figure 3: DAG training example. Samples not within the active sets of the trained classes of a node, $A_{\omega_A}(n)$ and $A_{\omega_B}(n)$, are used for its training

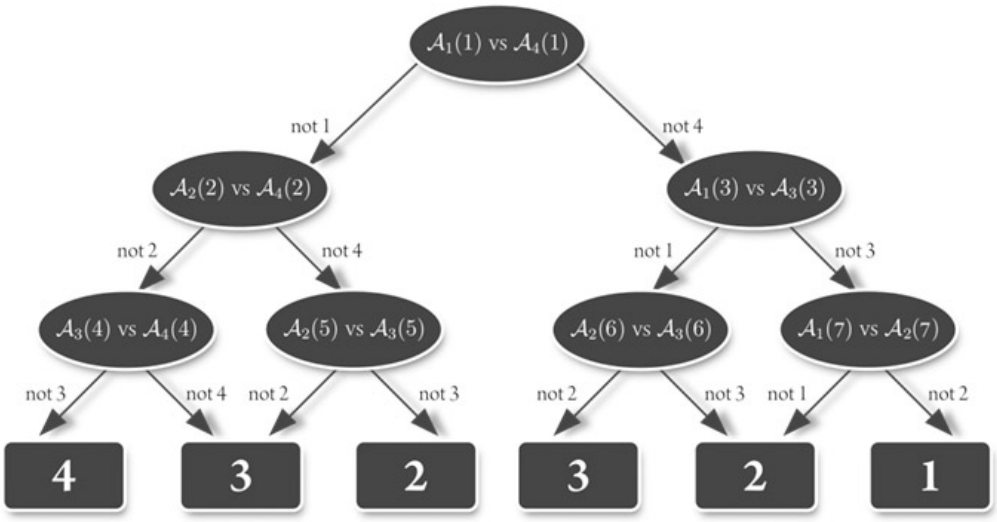

Figure 4: Example of a P-Tree. Training is only performed over the active sets of the training set of a class

It has been shown that partitioning of instance and decision space, in theory, leads to the same Bayes optimal classifiers. In this way, the proposed technique makes use of that property so as to iteratively use training and testing to avoid such redundancies.

Initially, the P-Trees method chooses two classes against which it trains the root node. After the completion of the training stage, it tests all remaining classes against this trained node. Each class is partitioned in two parts from the trained root node. Each partition is passed to either the left

or the right child of the root node. The training process proceeds in the same fashion. When a node is trained against two classes ω_i and ω_j, it only considers training samples within their active sets $A_{\omega_i}(n) \cup A_{\omega_j}(n)$, namely the subset of the training samples of each class that successfully reach the node, passing through the already trained nodes within its path (see Figure 4).

It should be clear by now that, in conjunction to the DAGs technique, P-Trees result in binary trees. This is true since a node is trained against the active set of its classes. Training the exact same pair of class labels in a different node is effectively a different classifier, in as much as the active sets corresponding to the same class labels differ. In this way, node merging is not applicable. This forms a restricting factor for the scaling of the proposed technique, with respect to the number of decision classes of a problem, that need be addressed for the algorithm to be applicable in problems with a relatively high number of classes. More details on that follow in section 5.1.3.

There are still parts of the P-Tree algorithm that need further clarification. The heuristic for the choice of the classes against which a node is trained needs to be specified. The way the extended partitions are created requires further details, while the problem of exponential scaling of tree nodes has to be properly addressed. A detailed description of each of these parts follows.

5.1.1 *Heuristic for choosing the training classes of a node*

With respect to the choice of the classes against which a node is trained, in the absence of any other universally applicable optimality criterion, the proposed technique always chooses to train against the pair of classes that have the minimum number of samples within their active sets. The expectation is that relatively large classes will be likely further subdivided from graph nodes, when their training is postponed further down the tree structure. This heuristic is best suited for minimizing training time. In case maximization of generalization performance is opted for, a better heuristic for the choice of the tree nodes could be one in the likes of that found in Bala et al.,[1] where information-theoretic measures of separation and dependence are employed, at a preprocessing step, in order to choose a tree structure that is more likely to provide better generalization performance.

5.1.2 *Extended partitions by reintroduction of rejected samples*

It has already been proven that partitioning of classes with respect to instance and decision space leads to the same Bayes optimal classifiers. However, this only holds in theory, where the class-conditional probability density function (pdf) is known beforehand. In practice, lack of knowledge of the pdf can lead to poor generalization performance.

This is further pronounced in the case of P-Trees, where the progressive partitioning of classes aggravates the problem. As every node partitions the classes passing through it, each child node only receives a fraction of a class, due to samples being rejected from the parent node. In practice though, the rejected part of each side of the sub-tree carries information relevant to the side of the tree from which it was rejected.

To avoid problems of this kind and to improve the generalization performance of P-Trees in pathological cases, an effective solution is to properly reintroduce small portions of the rejected samples to the training set of each node. This leads to a slight performance hit in terms of training time—easily controlled through reintroduction rate—but provides significantly improved generalization performance. Two simple techniques addressing this problem follow.

Random Subsampling: The easiest method of reintroducing discarded samples in order to improve generalization performance in P-Trees is through random subsampling of the rejected samples (see Figure 5). The subsampling rate in this case can control the training performance degradation. This technique is very easy to implement but requires relatively high subsampling rates (e.g. ≥ 0.3) in order to be effective against highly pathological cases. This in turn incurs unacceptable training performance degradation, the reduction of which is the primary goal of P-Trees. The use of stratified subsampling,[25] by subdivision of the rejected partition into discrete regions (strata) prior to subsampling, can further improve the effectiveness of this technique for fixed subsampling rates, especially in problems with highly inhomogeneous spatial distribution of samples.

Proximal Subsampling: This technique can be significantly more effective than random subsampling, even in highly pathological cases, in as much as it works with only a very limited number of selectively reintroduced samples. The idea is to randomly reintroduce only samples that are located in high proximity to the decision surface (separating hyperplane in SVMs) of the node that rejected them.

Samples evenly spread across the vicinity of the decision surface, at the side of the rejected area (see Figure 5), can be highly efficient on alleviating the degradation of the generalization performance of P-Trees, even for relatively small reintroduction rates (e.g. ≈ 0.1). Stratification can once again be employed to further improve the effectiveness of the technique in case of inhomogeneous sample distributions. Empirical tests at the end of this paper demonstrate the superiority of this technique in comparison to plain random subsampling.

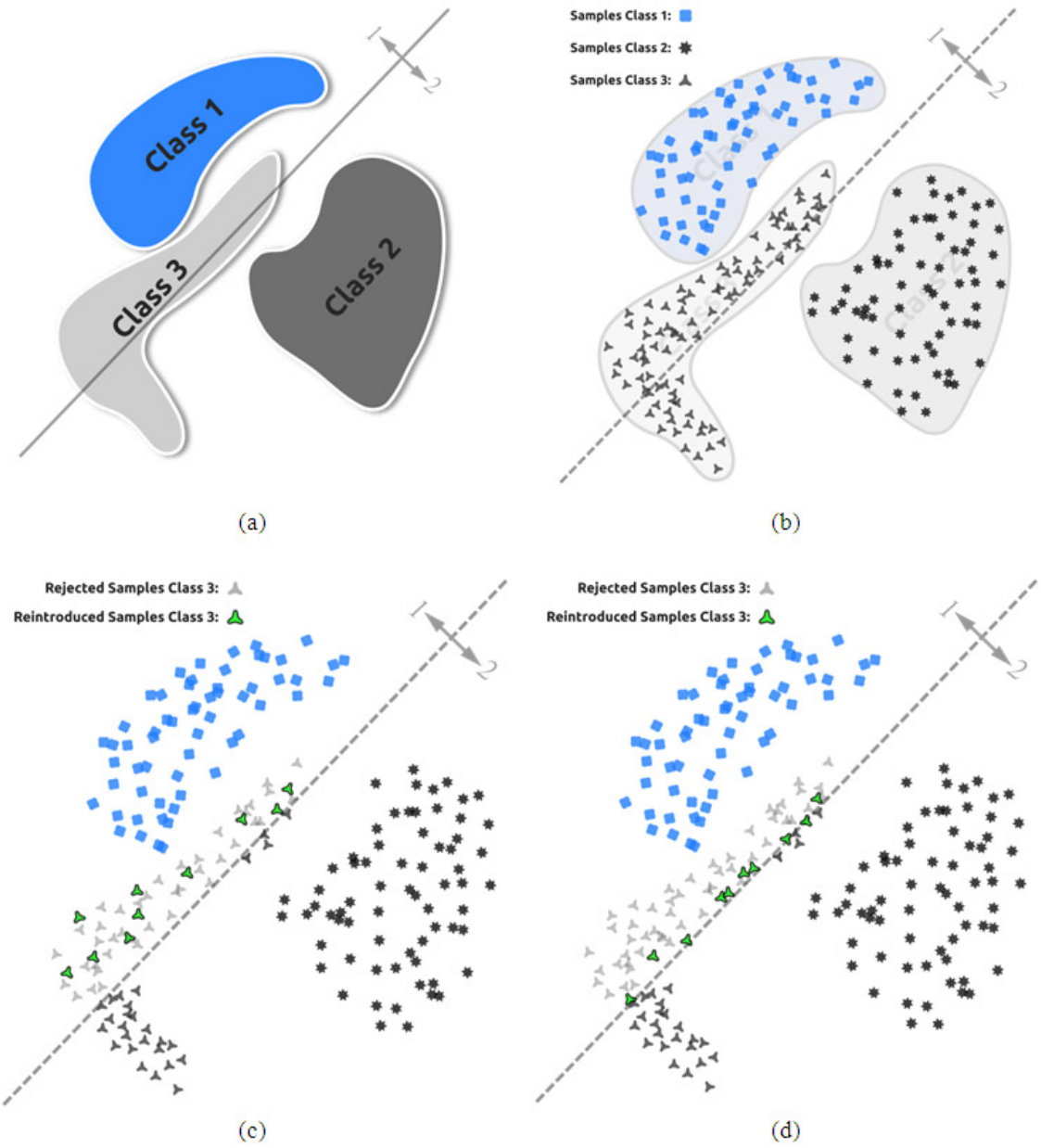

Figure 5: Comparison of random subsampling (c) and boundary proximal sampling (d). Fig. (a) shows the active sets of each class and the classifier f_{12} separating classes 1 and 2. Fig. (b) shows a typical training set, containing random samples from each class. It can be readily shown that proximal sampling is more effective than random subsampling, for the same reintroduction rate.

5.1.3 Exponential scaling of tree nodes

An important restricting factor of the proposed technique is that the upper bound on the number of nodes $n = 2^{K-1} - 1$ of the tree scales exponentially with respect to the number of decision classes K. In contrast, the number of nodes $n = K(K-1)/2$ in a DAG only has quadratic scaling with respect to K. For a small number of classes this does not constitute a limiting factor. Soon enough though, the exponential growth creates a problem with respect to the storage of the data structure representing the trained tree; for K=31 classes, an excess of 10^9 nodes are required.

A solution to this problem is straight-forward. Excessive partitioning will evidently lead to over-fitting. Adequate training time reduction can be easily achieved from a small number of class partitions. In this way, the P-Trees algorithm is applied for a fixed number of levels (e.g. 5 levels). From that point onwards, training continues in the regular fashion of the DAGs algorithm (quadratic scaling of nodes thereafter).

More analytically, P-Trees training is limited to a fixed num-

ISSN 1927-6974 E-ISSN 1927-6982

ber of levels W. From that point onwards, for each non-decision node of the 2^{W-1} nodes of the last level, a separate DAG is trained, only using the active sets of the respective node. This choice results at a worst case scenario of 2^{W-1} nodes for the P-Tree until level W, in addition to $2^(W-1)(K-1)(K-2)/2$ for each of the 2^{W-1} DAGs. Thus, once again, the overall training results in quadratic scaling with respect to K, albeit this time multiplied by approximately 2^{W-1}. Finally, with respect to testing, the worst case scenario is evaluating K-1 nodes per test sample, just like in the DAGs case; W for the P-Tree plus $K-W-1$ for the DAG with $K-W$ classes at most.

5.2 Algorithmic description and pertinent structures

We now provide an algorithmic description of the proposed method. Before we introduce the algorithms, we provide some basic definitions needed to understand their various steps.

The proposed technique trains binary trees. All nodes of a tree are binary classifiers trained against only two classes, except for the leaf nodes which are simple decision nodes. In what follows, decision nodes will not be considered. A P-Tree trained against K classes Comprises $2^{K-1}-1$ nodes (excluding leaf nodes).

A unique number n will be designated to each node, namely

$$n \in \{1, \ldots, 2^{k-1}-1\} \tag{15}$$

starting with 1 for the root node and continuing in increasing order, by levels and from left to right (see Figure 6).

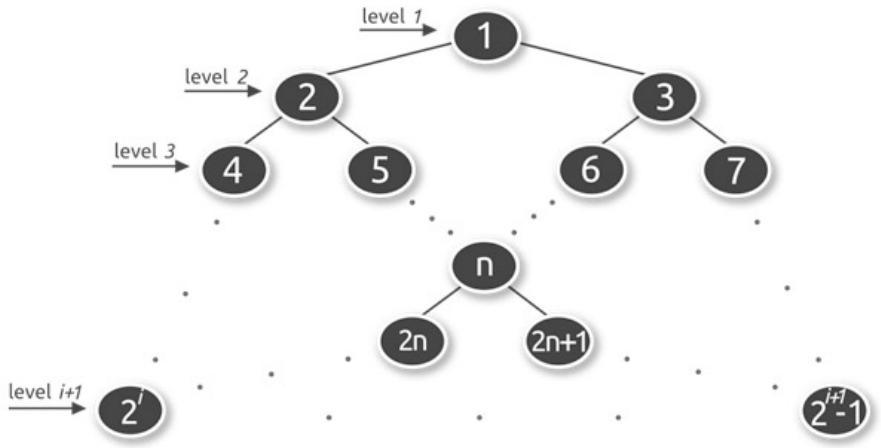

Figure 6: Numbering convention of tree nodes and tree levels.

The parent of a node n will be denoted as par(n), while chL(n) and chR(n) will denote the left and right child of a node respectively. For the chosen numbering convention of nodes it can be shown that

$$par(n) = [\frac{n}{2}] \tag{16}$$

$$chL(n) = 2n \tag{17}$$

$$chR(n) = 2n+1 \tag{18}$$

With isleft(n) the indicator function of left child validity will be denoted

$$isleft(n) = \begin{cases} 1 & n = chL(par(n)) \\ -1 & n = chR(par(n)) \end{cases} \tag{19}$$

For any trained node n corresponds a binary classifier $f_{(n)}(x)$, trained against classes $\omega_L(n)$ and $\omega_R(n)$. Each node creates a dichotomy over instance space χ, leading samples x for which $f_{(n)}(x) > 0$ to the right sub-tree of the node, and the rest to the left sub-tree. $\Re_L(n)$ and $\Re_R(n)$ for a node n will denote the class index, from the ones the node is trained against, that the node rejects from its left and right tree, respectively.
D_{ω_i} denotes the training samples $(x_j, t_j) \in D$ of class ω_i

$$D_{\omega_i} = \{(x_j, t_j) : (x_j, t_j) \in D, t_j \in \omega_i \tag{20}$$

$\chi(n)$ will denote the partition of the instance space χ reaching node n, by successfully traversing all nodes in its path. The active set $A_{\omega_i}(n)$ of a class ω_i can then be defined as the set of samples from D_{ω_i} that reach node n, namely

$$A_{\omega_i}(n) = \{(x_j, t_j) : (x_j, t_j) \in A_{\omega_i}, x_j \in \chi(n)\} \tag{21}$$

$\varepsilon(n)$ will denote the extended sets, namely the active sets in addition to the reintroduced rejected samples. Assuming that reint() returns the reintroduced sample set, the extended sets are defined as

$$\varepsilon_{\omega_i}(n) = \{A_{\omega_i}(n) \cup reint(A_{\omega i}(par(n)) \backslash A_{\omega_i}(n))\} \quad (22)$$

Finally $\Re(n)$ will denote the rejection set of a node, i.e. the class indices that have been rejected throughout the node's path

$$R(n) = x = \begin{cases} R(par(n)) \cup R_L(par(n)) & n = chL(par(n)) \\ R(par(n)) \cup R_R(par(n)) & n = chR(par(n)) \end{cases} \quad (23)$$

with the convention that $R(1) = \phi$, while $R^c(n)$ will denote its complement with respect to the decision classes

$$R^c(n) = \pi_C \backslash R(n) \quad (24)$$

We are now in position of introducing an algorithmic description of both the training (see Figure 7) and testing process (see Figure 8). Algorithm 2 (See figure 8)is largely self-explanatory. In Algoritm 1(see Figure 7), line 8 iterates over all levels of a P-Tree, while line 9 iterates over all nodes within each level. The active, extended and rejection sets are updated for every node within lines 14-28. The smallest classes from within the extended sets of a node are computed in lines 6 and 40, for the root node and every other P-Tree node respectively. A binary classifier is trained on the selected classes of a node in line 41, while a DAG is trained within lines 35-39 using the extended sets of all remaining (non-empty) classes in case the maximum number of allowed levels W is reached for a non-decision node.

5.3 Training complexity reduction

In this section we prove guaranteed tightening of computational bounds for the proposed technique. The benefits reported hereafter are in the mean sense.

Suppose the binary classification algorithm used to train a binary classifier $f_{ij}(x)$ for each node n_{ij} of the tree requires an overall number of computations upper-bounded by $T(N)$, as a function of the number of training samples N. Empirical bounds for various classification algorithms are typically provided in polynomial form

$$T(N) = cN^{\gamma} \quad (25)$$

where c and γ constants, though the following results hold for any non-decreasing bounding function.

Practical binary classification algorithms scale super-linearly (γ>1 in the polynomial case) with respect to the number of training samples. In the linear case (γ=1), for a classification problem of size N that can be partitioned into two independent sub-problems of size A and N-A respectively, it holds that

$$T(N) = T(N - A) + T(A), \forall 0 \leq A < N \quad (26)$$

In the super-linear case on the other hand, it holds that

$$T(N) \geq T(N - A) + T(A), \forall 0 \leq A < N \quad (27)$$

Hence it can be readily shown, through repetitive application of (27), that for practical training algorithms any partitioning π_D of the initial problem with training set D into independent sub-problems with respective training sets $D_i \in \pi_D$ always leads to tightening of computational bounds

$$T(N) \geq \sum_{D_i \in \pi_D} T(|D_i|), \forall \pi_D \in PART(D) \quad (28)$$

This is only true whenever no rejected samples reintroduction scheme is employed, as the above proof assumes each partition has no intersection with another partition of the same class. A proof of such a claim in case of samples reintroduction requires exact knowledge of the complexity function $T(N)$.

Comparing P-Trees with DAGs, it is self-evident that for every node of a DAG there are a number of nodes in the corresponding P-Tree containing a partition of the samples of the DAG's node into many nodes. Since each node of the P-Tree is trained independently of the others, from (28) the tightening of the computational bounds is proven.

As an example, assume for simplicity that for a multiclass problem of K classes, each class has the same number of training samples N/K and that every node of the tree dichotomizes each class (other than the ones against which it is trained) into two equal parts. In this case, the overall number of computations required by a P-Tree is bounded by

$$T_{tot}(N) = T(\frac{2N}{K}) + 2T(\frac{2N}{2K}) + \ldots + 2^{k-2}T(\frac{2N}{2^{k-2}K}) = \sum_{i=0}^{K-2} 2^i T(\frac{2N}{2^i K}) \tag{29}$$

1: **input** $\mathcal{D}, K, W, \omega_1, \ldots, \omega_K, \mathcal{D}_{\omega_1}, \ldots, \mathcal{D}_{\omega_K}$
2: $\mathcal{R}(1) \leftarrow \emptyset$
3: **for** $i \leftarrow 1$ **to** K **do**
4: $\quad \mathcal{A}_{\omega_i}(1) \leftarrow \mathcal{D}_{\omega_i}$
5: **end for**
6: $\{\omega_L(1), \omega_R(1)\} \leftarrow \text{argmin}_{\{A,B\}} |\mathcal{A}_A(1) \cup \mathcal{A}_B(1)|$
7: $f_{(1)}(\mathbf{x}) \leftarrow trainBinary\left(\mathcal{A}_{\omega_L(1)}(1), \mathcal{A}_{\omega_R(1)}(1)\right)$
8: **for** $i \leftarrow 2$ **to min** $\{W, K-1\}$ **do**
9: $\quad$ **for** $n \leftarrow 2^{i-1}$ **to** $2^i - 1$ **do**
10: $\quad\quad par \leftarrow \left\lfloor \frac{n}{2} \right\rfloor$
11: $\quad\quad sgn \leftarrow isleft(n)$
12: $\quad\quad node[n].isDecision \leftarrow false$
13: $\quad\quad node[n].isDAG \leftarrow false$
14: $\quad\quad$ **if** $n \bmod 2 == 0$ **then**
15: $\quad\quad\quad \mathcal{R}(n) \leftarrow \mathcal{R}(par) \cup \omega_R(par)$
16: $\quad\quad$ **else**
17: $\quad\quad\quad \mathcal{R}(n) \leftarrow \mathcal{R}(par) \cup \omega_L(par)$
18: $\quad\quad$ **end if**
19: $\quad\quad$ **if** $\mathcal{R}^c(n) == \emptyset$ **then**
20: $\quad\quad\quad$ **continue**
21: $\quad\quad$ **end if**
22: $\quad\quad$ **for all** $k \in \mathcal{R}^c(n)$ **do**
23: $\quad\quad\quad \mathcal{A}_{\omega_k}(n) \leftarrow \left\{(\mathbf{x}_l, t_l) \in \mathcal{A}_{\omega_k}(par) : f_{(par)}(sgn \cdot \mathbf{x}_l) > 0\right\}$
24: $\quad\quad\quad \mathcal{E}_{\omega_k}(n) \leftarrow \left\{\mathcal{A}_{\omega_i}(n) \cup reint\left(\mathcal{A}_{\omega_k}(par) \backslash \mathcal{A}_{\omega_k}(n)\right)\right\}$
25: $\quad\quad\quad$ **if** $\mathcal{E}_{\omega_k}(n) == \emptyset$ **then**
26: $\quad\quad\quad\quad \mathcal{R}(n) \leftarrow \mathcal{R}(n) \cup \omega_k$
27: $\quad\quad\quad$ **end if**
28: $\quad\quad$ **end for**
29: $\quad\quad$ **if** $|\mathcal{R}^c(n)| == 1$ **then**
30: $\quad\quad\quad node[n].isDecision \leftarrow true$
31: $\quad\quad\quad node[n].decision \leftarrow \{a : a \in \mathcal{R}^c(n)\}$
32: $\quad\quad\quad \mathcal{R}^c(n) \leftarrow \emptyset$
33: $\quad\quad\quad$ **continue**
34: $\quad\quad$ **end if**
35: $\quad\quad$ **if** $node[n].isDecision == false$ **and** $i == W$ **then**
36: $\quad\quad\quad node[n].isDAG \leftarrow true$
37: $\quad\quad\quad node[n].DAG \leftarrow trainDAG(\{\mathcal{E}_{\omega_k}(n) : k \in \mathcal{R}^c(n)\})$
38: $\quad\quad\quad$ **continue**
39: $\quad\quad$ **end if**
40: $\quad\quad \{\omega_L(n), \omega_R(n)\} \leftarrow \text{argmin}_{\{A,B\}: A,B \in \mathcal{R}^c(n), A \neq B} |\mathcal{E}_A(n) \cup \mathcal{E}_B(n)|$
41: $\quad\quad f_{(n)}(\mathbf{x}) \leftarrow trainBinary\left(\mathcal{E}_{\omega_L(n)}(n), \mathcal{E}_{\omega_R(n)}(n)\right)$
42: $\quad$ **end for**
43: **end for**

Figure 7: Training P-Tree

```
input x, K, M = 2^K − 1, node[], f_(1)(),..., f_(M)()
n ← 1
while true do
    if node[n].isDecision then
        classEst ← node[n].decision
        break
    end if
    if node[n].isDAG then
        classEst ← testDAG(node[n].DAG, x)
        break
    end if
    result ← f_(n)(x)
    if result ≥ 0 then
        n ← 2n
    else
        n ← 2n + 1
    end if
end while
return classEst
```

Figure 8: Testing P-Tree

Let us further assume that the binary training algorithm chosen is an SVM. It has been empirically observed[26] that an SVM classifier scales super-linearly with respect to the training samples N according to the approximate polynomial relation

$$T(N) = cN^{\gamma} \simeq cN^2 \tag{30}$$

So for P-Trees with SVMs as binary classifiers, under the assumptions of equation (29) we have

$$T_{tot}(N) = \sum_{i=0}^{K-2} 2^i \frac{4c^2N^2}{2^{2i}K^2} = \frac{4c^2N^2}{K^2} \sum_{i=0}^{K-2} 2^{-i} = \frac{8c^2N^2}{K^2}(1 - \frac{1}{2^{k-1}-1}) < \frac{8c^2N^2}{K^2} \tag{31}$$

The corresponding bound in the DAGs case, under the same assumptions, is

$$T_{tot}(N) = (\frac{K(K-1)}{2})\frac{4c^2N^2}{K^2} \simeq 2c^2N^2 \tag{32}$$

A comparison between (31) and (32) shows that, in the SVMs case, for $K \geq 2$ the P-Trees scale quadratically better than DAGs with respect to the available number of classes K. This is true though under the assumption that every class has the exact same number of samples and each trained node perfectly splits each class in two parts. In practice, our experimental results show slightly above linear scaling with respect to the number of classes K.

5.4 Parallelization of nodes training

Training child nodes in P-Trees requires the computation of their respective active sets, which in turn requires the completion of the training of their parents. This type of sequential training hinders the ability to perform training execution parallelism at a high level (many nodes at a time), especially at the initial training steps; the number of trained parents is small when close to the root node.

Thus, execution parallelism, if opted for, need be performed at a lower level. This is hardly ever a problem, as the underlying binary classification algorithm typically lends itself for ample low level parallelism (many samples at a time), thus obviating the need to resort to node level parallel execution. This type of parallelism better fits the objectives of combinational techniques, where the implementation of the multiclass extension scheme is kept relatively simple and the performance enhancements are off-loaded to the employed binary classification algorithm.

6 Empirical comparisons

In this section we demonstrate empirical results that justify the effectiveness of the proposed technique both in terms of training and testing time, as well as generalization performance.

The experimental tests were conducted upon multiclass datasets from the UCI[2] and StatLog[24] repositories. More specifically, from the UCI repository we used the datasets iris, glass, vowel and wine. From the StatLog repository, we used datasets dna, letter, satimage, segment, shuttle and vehicle. Table 1 contains the specifics of each dataset.

Table 1: Dataset Details

Dataset	Classes	Features	Training Instances	Testing Instances
Shuttle	7	9	43500	14500
Letter	26	16	15000	5000
SatImage	6	36	4435	2000
Dna	3	180	2000	1186
Segment	7	19	2310	0
Vehicle	4	18	846	0
Vowel	11	10	528	0
Glass	6	13	214	0
Wine	3	13	178	0
Iris	3	4	150	0

For datasets containing a separate testing set, it was merged with the training set and a uniform approach was later followed for every dataset. The binary classifiers used were SVMs with RBF kernels. The libSVM library[6] was used to implement the SVM classifiers. On training the SVMs, the termination condition was the fulfillment of the KKT conditions with accuracy better than 10^{-3}.

The C and γ parameters were chosen from within the sets[18]

$$C = [2^{-2}, 2^{-1}, \ldots, 2^{12}] \tag{33}$$

and

$$\gamma = [2^{-10}, 2^{-9}, \ldots, 2^{4}] \tag{34}$$

The best possible (C,γ) combination was chosen through 5-fold cross validation.[12] Since the pseudo-random choice of both folds and training-testing sets significantly affects the classification accuracy, the above process was repeated 20 times for each method, with a different seed for the random number generator each time. In what follows, the average values over 20 iterations are presented.

All empirical tests were independently conducted for the 1-vs-1, DAGs, as well as two variants of the P-Trees technique; one using random subsampling and reintroduction rate equal to 0.3 and one using boundary proximal subsampling and a 0.15 rate. With respect to the P-Trees, to avoid exponential growth of tree nodes in problems with a large number of classes, class partitioning was limited to 5 levels within the tree structure.

In Table 2 the generalization performance of the various techniques is assessed. The reported mean generalization performance along with its standard deviation correspond to 70% training and 30% testing. A first observation is that partitioning seems to be creating notable generalization performance degradation in datasets like vowel, glass and letter, when random subsampling is used. All these datasets share some characteristics that lend them problematic with respect to partitioning. They have a relatively high number of classes, which translates into many successive partitions, and they have a relatively small number of samples per class at high dimensional space (very sparse samples).

This means that sample rejection conceals significant extrapolating information. It is worth noting how effective proximal subsampling is at alleviating the problem, considering that it only uses half the reintroduction rate used in random subsampling. Other than that, the performance of P-Trees with proximal subsampling appears to be comparable to that of the DAGs and 1-vs-1 technique. This hypothesis has been verified by use of the McNemar test[14,23] for significance level 0.05.

Table 3 demonstrates the mean training time, in seconds, for the 20 repetitions of each technique. Note that the proposed method consistently provides significant reduction in training time. The rate of reduction strongly depends on the available number of classes and appears to scale slightly super-linearly with respect to the number of classes K. Boundary proximal subsampling also leads to improved training time over random subsampling, due to the use of smaller sample reintroduction rate.

Table 2: Generalization performance

Dataset	Generalization Accuracy			
	1-vs-1	DAGs	P-Trees	
			Random	Proximal
Shuttle	99.8±0	99.8±0	99.8±0	99.8±0.1
Letter	97.3±0.2	97.2±0.2	95.9±0.2	96.5±0.3
SatImage	91.5±0.6	91.6±0.6	91.4±0.7	91.2±0.7
Dna	59.4±0.8	59.4±0.8	60.3±0.7	60.3±0.7
Segment	96.2±0.7	96.4±0.6	96±0.7	95.9±0.7
Vehicle	71±3.4	71±3.3	71.5±3.1	71±3.1
Vowel	98.5±0.8	98.5±0.8	96.4±1.1	97.3±1.1
Glass	69±3.2	69.4±2.9	67.7±3.9	68.4±5.1
Wine	91.9±3.5	92.1±3.5	93.8±2.7	93.6±3
Iris	95.6±2.1	95.5±2.1	95.7±2.2	95.7±2.2

Table 3: Training complexity

Dataset	Mean Training Time (sec.)			
	1-vs-1	DAGs	P-Trees	
	Training	Training	Random training	Proximal training
Shuttle	8.0	8.0	3.7	2.4
Letter	12.7	12.7	8.5	5.9
SatImage	2.3	2.3	1.1	0.8
Dna	3.2	3.2	2.2	2.0
Segment	0.16	0.16	0.09	0.067
Vehicle	0.063	0.063	0.041	0.039
Vowel	0.085	0.085	0.077	0.063
Glass	0.004	0.004	0.002	0.002
Wine	0.005	0.005	0.003	0.003
Iris	0.001	0.001	$9.2{\cdot}10^{-4}$	$6.7{\cdot}10^{-4}$

Table 4: Testing complexity

Dataset	Mean Training Time (sec.)						
	1-vs-1	DAGs		P-Trees			
	Total			Random		Proximal	
		Total	Mean	Total	Mean	Total	Mean
Shuttle	1079	1079	432	1935	277	1592	242
Letter	8483	8483	5028	12309	2634	10907	2121
SatImage	2718	2718	1721	3035	1239	2859	1111
Dna	2127	2127	2127	2127	1979	2127	1958
Segment	657	657	560	1057	456	837	418
Vehicle	541	541	523	552	468	548	446
Vowel	570	570	484	663	320	632	315
Glass	128	128	103	137	67.9	131	69.7
Wine	122	122	119	120	107	119	106
Iris	44.4	44.4	37.9	42.1	34.4	40.8	33.8

Notable reduction has also been recorded in terms of testing time. In the SVM case, testing time is directly proportional to the chosen number of Support Vectors (SVs). In Table 4 the total number of SVs as well as the mean number of SVs used per testing are presented. Note that in the 1-vs-1 case, total and mean used number of SVs are equivalent. It can be seen that the mean number of used SVs per test is significantly reduced in the case of DAGs and P-Trees. In most cases, P-Trees are performing much better than DAGs. This can be attributed to the fact that the exclusion of rejected points from the tree nodes makes a decision surface more compact (smaller area of interest). In this way, less

SVs are needed to express this decision hyper-surface. It is also worth noting that proximal subsampling appears to be marginally better than random subsampling in terms of testing performance, for the exact same reason.

Finally, in Table 5 a comparison between the tested methods is made, in terms of the total number of trained graph nodes as well as the mean number of nodes evaluated per sample, during testing. As seen in this table, although the total number of trained nodes is significantly increased in the P-Trees case, especially for problems with a large number of classes (e.g. letter), the mean number of evaluated nodes is almost always identical to that of the DAGs case. In addition to that, each P-Tree node typically contains notably less SVs and is faster to evaluate.

Table 5: Evaluation Complexity

Dataset	Total Trained and Mean Evaluated Nodes							
	1-vs-1		DAGs		P-Trees			
					Random		Proximal	
	Total Trained	Mean Eval.	Total Trained	Mean Eval.	Total Trained	Mean Eval.	Total Trained	Mean Eval.
Shuttle	21	21	21	6	126.8	6	127	6
Letter	325	325	325	25	12287	25	12287	25
SatImage	15	15	15	5	63	5	63	5
Dna	3	3	3	2	7	2	7	2
Segment	21	21	21	6	126.9	6	127	6
Vehicle	6	6	6	3	15	3	15	3
Vowel	55	55	55	10	724.7	10	767	10
Glass	15	15	15	5	61	4.9	63	5
Wine	3	3	3	2	7	2	7	2
Iris	3	3	3	2	7	2	7	2

7 Conclusions

A new multiclass classification technique has been presented, that demonstrates significant reduction in terms of training and testing time, when compared to DAGs. At the same time it maintains comparable generalization performance with currently popular techniques. The benefits from the use of the proposed technique become increasingly prevalent with the increase of the available number of classes.

One drawback of the proposed technique is the relatively increased implementation complexity and the inability to efficiently provide node-level training parallelism. Perhaps a more important limitation is the exponential scaling of the maximum number of nodes with respect to the number of classes, in juxtaposition to the quadratic scaling of DAGs.

Nevertheless, the exponential scaling of the nodes with respect to the number of classes can become limiting only in problems with a large number of decision classes. In such circumstances, the problem can be efficiently addressed by simply halting class partitioning as soon as the total number of nodes exceeds a predefined threshold. Effectively, from that point onwards, the training procedure returns to that of the DAGs, with quadratic scaling.

References

[1] Manju Bala and R. K. Agrawal (2011). Optimal decision tree based multi-class support vector machine. Informatica, (pp. 197-209).

[2] C. L. Blake and C. J. Merz (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/mlearn/MLRepository.html

[3] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. (1992). A training algorithm for optimal margin classifiers. Proceedings of the fifth annual workshop on Computational learning theory, (pp. 144-152), NY, USA, ACM. http://dx.doi.org/10.1145/130385.130401

[4] Leo Breiman (2001). Random Forests. Machine Learning, (pp. 5-32). http://dx.doi.org/10.1023/A:1010933404324

[5] Leo Breiman and Leo Breiman (1996). Bagging predictors. In Machine Learning, (pp. 123-140). http://dx.doi.org/10.1007/BF00058655

[6] Chih-chung Chang and Chih-Jen Lin (2001). Libsvm: a library for support vector machines.

[7] T. Lei, R. Guo, X. Wang et al. Application of grey system theory in agriculture, Henan Science and Technology Press. http://dx.doi.org/10.1016/j.neucom.2009.03.013

[8] Koby Crammer and Yoram Singer (2001). On the algorithmic implementation of multiclass kernel-based vector machines. Journal of Machine Learning Research, (pp. 265-292).

[9] Thomas G. Dietterich and Ghulum Bakiri (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. AAAI Press, (pp. 572-577).

[10] Thomas G. Dietterich and Doug Fisher (2000). An experimental comparison of three methods for constructing ensembles of decision trees. Machine Learning, (pp. 139-157). http://dx.doi.org/10.1023/A:1007607513941

[11] Kai-bo Duan and Sathiya S. Keerthi (2005). Which is the best multiclass SVM method? An empirical study. (pp. 278-285).

[12] Bradley Efron and Gail Gong (1983). A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation. The American Statistician, (pp. 36-48).

[13] Sergio Escalera, David M. J. Tax, Oriol Pujol, Petia Radeva, and Robert P. W. Duin (2008). Subclass problem-dependent design for error-correcting output codes. IEEE TPAMI, (pp. 1041-1054). http://dx.doi.org/10.1109/TPAMI.2008.38

[14] B.S. Everitt (1993). The analysis of contingency tables (2nd Ed.)

[15] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio (2000). Regularization Networks and Support Vector Machines. Advances in Computational Mathematics, (pp. 1-50). http://dx.doi.org/10.1023/A:1018946025316

[16] Yoav Freund and Robert E. Schapire (1996). Experiments with a new boosting algorithm. In proceedings of the 13th International Conference on Machine Learning, (pp. 148-156).

[17] N. Garcia-Pedrajas and D. Ortiz-Boyer (2006). Improving multiclass pattern recognition by the combination of two strategies. IEEE TPAMI, (pp. 1001-1006). http://dx.doi.org/10.1109/TPAMI.2006.123

[18] Chih-Wei Hsu and Chih-Jen Lin (2002). A comparison of methods for multiclass support vector machines. IEEE Transactions on Neural Networks, (pp. 415-425). http://dx.doi.org/10.1109/72.991427

[19] Ulrich H.-G. Krebel (1999). Pairwise classification and support vector machines. (pp. 255-268).

[20] Y. Lee, Y. Lin, and G. Wahba (2001). Multicategory support vector machines.

[21] H. Lei and V. Govindaraju (2005). Half-against-half multi-class support vector machines.

[22] Gjorgji Madzarov, Dejan Gjorgjevikj, and Ivan Chorbev (2009). A multi-class svm classifier utilizing binary decision tree. Informatica, (pp. 233-241).

[23] Quinn McNemar (1947). Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages. Psychometrika, (pp. 153-157). http://dx.doi.org/10.1007/BF02295996

[24] D. Michie, D. J. Spiegelhalter, and C.C. Taylor (1994). Machine learning, neural and statistical classification. ftp://ncc.up.pt/pub/statlog/.

[25] Jerzy Neyman (1934). On the two different aspects of the representative method: The method of stratified sampling and the method of purposive selection. Journal of the Royal Statistical Society, (pp. 558-625). http://dx.doi.org/10.2307/2342192

[26] John C. Platt (1999). Fast training of support vector machines using sequential minimal optimization. (pp. 185-208).

[27] John C. Platt, Nello Cristianini, and John Shawe-Taylor (2000). Large margin dags for multiclass classification. In Advances in Neural Information Processing Systems 12, (pp. 547-553).

[28] Ryan Rifkin and Aldebaro Klautau (2004). In defense of one-vs-all classification. Journal of Machine Learning Research, (pp. 101-141).

[29] Robert E. Schapire (1990). The strength of weak learnability. In Machine Learning.

[30] Robert E. Schapire (1999). A brief introduction to boosting. In Journal of Japanese Society for Artificial Intelligence, (pp. 1401-1406).

[31] Benyang Tang and Dominic Mazzoni (2006). Multiclass reduced-set support vector machines. (pp. 921-928).

[32] Vladimir N. Vapnik (1998). Statistical Learning Theory. Wiley-Interscience.

[33] Volkan Vural and Jennifer G. Dy (2004). A hierarchical method for multi-class support vector machines. (pp. 105-113).

[34] J. Weston and C. Watkins (1998). Multi-class support vector machines.