Int J Comput Vis (2012) 100:203–215
DOI 10.1007/s11263-011-0461-z

Making a Shallow Network Deep: Conversion of a Boosting Classifier into a Decision Tree by Boolean Optimisation

Tae-Kyun Kim · Ignas Budvytis · Roberto Cipolla

Received: 17 December 2010 / Accepted: 11 May 2011 / Published online: 7 June 2011
© Springer Science+Business Media, LLC 2011

Abstract This paper presents a novel way to speed up the evaluation time of a boosting classifier. We make a shallow (flat) network deep (hierarchical) by growing a tree from decision regions of a given boosting classifier. The tree provides many short paths for speeding up while preserving the reasonably smooth decision regions of the boosting classifier for good generalisation. For converting a boosting classifier into a decision tree, we formulate a Boolean optimisation problem, which has been previously studied for circuit design but limited to a small number of binary variables. In this work, a novel optimisation method is proposed for, firstly, several tens of variables i.e. weak-learners of a boosting classifier, and then any larger number of weak-learners by using a two-stage cascade. Experiments on the synthetic and face image data sets show that the obtained tree achieves a significant speed up both over a standard boosting classifier and the Fast-exit—a previously described method for speeding-up boosting classification, at the same accuracy. The proposed method as a general meta-algorithm is also useful for a boosting cascade, where it speeds up individual stage classifiers by different gains. The proposed method is further demonstrated for fast-moving object tracking and segmentation problems.

T.-K. Kim (✉)
Department of Electrical and Electronic Engineering, Imperial College London, South Kensington Campus, London SW7 2AZ, UK
e-mail: tk.kim@imperial.ac.uk

I. Budvytis · R. Cipolla
Department of Engineering, University of Cambridge, Cambridge CB2 1PZ, UK

Keywords Boosting · Decision tree · Decision regions · Boolean optimisation · Boosting cascade · Face detection · Tracking · Segmentation

1 Introduction

Boosting is a popular method in object detection (Viola and Jones 2001, 2004), tracking (Grabner and Bischof 2006) and segmentation (Avidan 2006) problems, where a vast number of image sub-windows, across pixels and scales, need to be classified. Doing the tasks in a reasonable time demands very fast evaluation of a classifier per window. A boosting classifier makes a fast decision by aggregating simple weak-learners such as Haar-like features, whose computations are accelerated by an integral image. Despite its efficiency, it is often required to further reduce the classification time of a boosting classifier. A cascade of boosting classifiers, which could be seen as a degenerate tree (see Fig. 1(a)), effectively improves the classification speed by filtering out majority of negative class samples in its early stages (Viola and Jones 2001, 2004; Xiao et al. 2003). Designing a cascade, however, involves manual efforts for setting a number of parameters: the number of classifier stages, the number of weak-learners and the threshold per stage. See Sect. 7.1 for details.

In this work, we propose a novel way to accelerate the classification (or evaluation) time of a boosting classifier up to an order of magnitude without sacrificing its accuracy, not relying on a conventional cascade. The chance for improvement comes from the fact that a standard boosting classifier can be seen as a very shallow network, see Fig. 1(b), where each weak-learner is a decision-stump and all weak-learners are used to make a decision. The flat structure affords smooth decision regions which help avoid overfitting

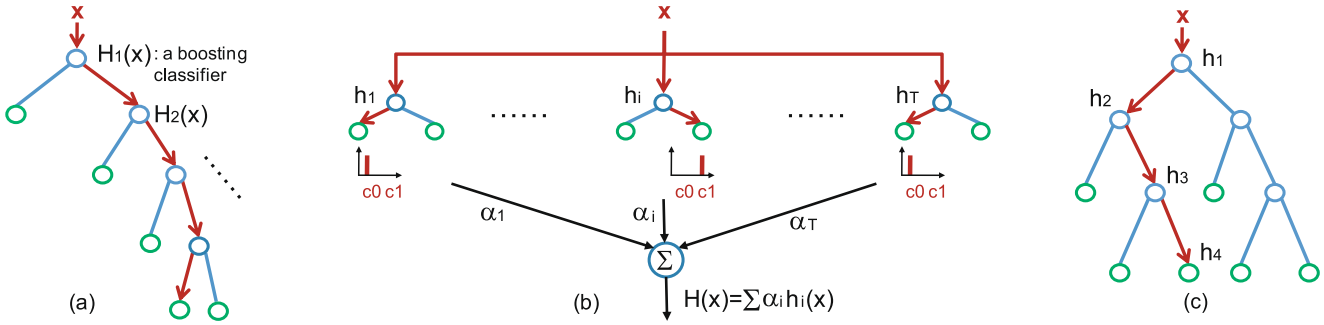

Fig. 1 Boosting as a tree. (**a**) A boosting cascade is seen as an imbalanced tree, where each node is a boosting classifier. (**b**) A boosting classifier has a very shallow and flat network where each node is a decision-stump i.e. weak-learner. (**c**) A conventional decision tree has multiple paths, each of which has the different number of decision-stumps i.e. path-length

to train data i.e. good generalisation, however, it is not optimal in classification time. The proposed method converts the shallow network (a given boosting classifier as input) to a deep hierarchical structure (a decision tree as output). The obtained tree speeds up a boosting classifier by having many short paths: easy data points are quickly classified by a small number of weak-learners on their traverses. Since it replicates the decision regions of a boosting classifier, the method alleviates a highly-overfit behaviour of conventional decision trees. We introduce a novel Boolean optimisation formulation and method. A boosting classifier splits a data space into 2^n primitive regions by n binary weak-learners. The decision regions of the boosting classifier are encoded by the boolean codes and class labels of the primitive regions. A decision tree is then grown using the region information gain. Further details are about a better way of packing the region information (Sect. 6) and the two stage cascade allowing the conversion with any number of weak-learners (Sect. 7). Without designing a multi-stage cascade (Viola and Jones (2001) used a 32 layer cascade) our method offers a convenient way of speeding up, while the method incorporated in such a multi-stage cascade could provide a further speed-up.

The paper is organised as follows: Sect. 2 reviews related work. Overview of the proposed method is given in Sect. 3 and the formulation as Boolean optimisation in Sect. 4. Sections 5, 6 and 7 present the proposed solutions. Experimental results are shown in Sect. 8 and the conclusion is drawn in Sect. 9.

2 Related Work

2.1 Accelerating Boosting Classifiers

A tree-structured system consisting of multiple boosting classifiers (Torralba et al. 2007; Wu and Nevatia 2007; Huang et al. 2005; Tu 2005; Li and Zhang 2004) has been studied for multi-pose or multi-category object detection problems. The common structure is a tree hierarchy where each path is a strong boosting classifier dedicated to a single object pose or category. Torralba et al. have proposed sharing weak-learners among multiple boosting classifiers (Torralba et al. 2007) for reducing the classification time. By placing the most common weak-learners at the top of the tree structure, the total number of weak-learners of multiple boosting classifiers to use is reduced accelerating classification. The method requires pre-defined sub-pose or category labels, which are often not straightforward to obtain. In light of this problem, there have been attempts (Wu and Nevatia 2007; Huang et al. 2005; Tu 2005) to simultaneously learn the sub-category labels and multiple boosting classifiers by clustering samples in a tree. Whereas all above methods are relevant to speed up a system of *multiple* boosting classifiers for multi-pose or multi-category problems, our work targets at a *single* boosting classifier. The proposed method as a meta-algorithm can also help accelerate a multiple boosting system.

The method called AdaTree (Grossmann 2004a, 2004b) or ADtree (Freund and Mason 1999), which is obtained by modifying boosting algorithms, yields a decision tree during learning. A conceptual difference lies in that the previous studies (Grossmann 2004a; Freund and Mason 1999) present a novel way of boosting learning that alters the decision regions of a standard boosting classifier. Our method takes a boosting classifier learnt in a standard way as input and replicates the decision regions (so the accuracy) of the input classifier but speeds it up. It is worth noting that, in the experiments of Grossmann (2004a), AdaTree exhibited significantly worse accuracy than Adaboost, suffering from lack of generalisation due to its tree nature.

More relevant to ours are the works by Sochman and Matas (2005) and Zhou (2005). For speeding up the classification time of a single boosting classifier, the shortest set of weak-learners for a given error rate has been obtained by the sequential probability ratio test in Sochman and Matas

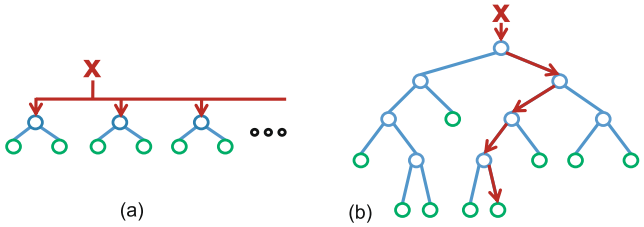

Fig. 2 Fast-exit vs super tree. Fast-exit methods have the structure of a single path of varying lengths (**a**), while our method yields the tree structure of multiple paths of different lengths (**b**)

(2005). It takes an early exit when the boosting sum reaches a certain value whose sign cannot be altered by the remaining weak-learners. Similarly, Fast-exit method has been proposed in Zhou (2005) (see Sect. 7.2). This line of methods uses a (so called) *single path of varying length* (see Fig. 2), where data points go through the weak-learners in the same order but exit at different points. In contrast, our method has a tree structure i.e. *multiple paths of different lengths*. The proposed method is more efficient in terms of speed by considering various combinations and orders of weak-learners (see Sect. 8). Zhou's first represented a boosting classifier by a Boolean table and implemented a binary decision tree (Zhou 2005). His solution, however, is a brute force search for all possible tree configurations, which has a high computational cost. It therefore affords to only about 5 and 10 weak-learners.

2.2 Boolean Expression Minimisation

Boolean expression minimisation is used to minimise the number of terms and binary variables in the Boolean expression. Algorithms for the minimisation have mainly been studied in the circuit design (Schwender 2007). Since circuits have strictly predefined specifications, exact minimisation was the goal of most studies. The complexity of a logic expression rises exponentially when the number of binary variables increases. Therefore, conventional minimisation methods are limited to a small number of binary variables, typically from a few to about 15 variables (Schwender 2007). Boolean minimisation has been also applied to size down a redundant decision tree, represented by a Boolean table (Chen 1994).

3 Conversion of a Boosting Classifier into a Tree

Both a boosting classifier and a decision tree are composed of weak-learners (or called decision-stumps/split-nodes). Whereas a boosting classifier places decision stumps in a flat structure, a decision tree has a deep and hierarchical structure (see Fig. 1(b, c)). The different structures lead to different behaviours: Boosting has a better generalisation via

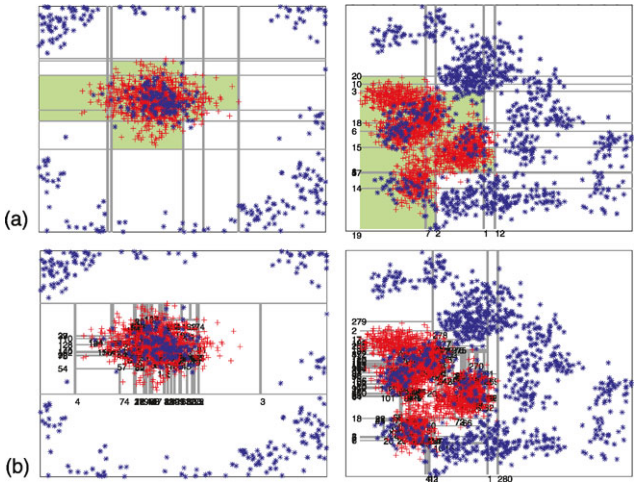

Fig. 3 (Color online) Decision regions (**a**) of a boosting classifier and (**b**) a conventional decision tree. The decision regions of the boosting classifier are smooth compared to those of the decision tree

reasonably smooth decision regions and decision trees use few weak learner tests to determine a class of a data point. Figures 3 and 4 illustrates both differences.

Figure 3 demonstrates the difference in smoothness of decision regions learnt by both methods. Here a part of negative-class (blue) data points are scattered in the middle of positive-class (red) samples. A conventional decision tree and a boosting classifier are learnt over the points using arbitrary lines in 2D as weak-learners. Whereas the decision tree forms complex decision regions trying classification of all training points, the boosting classifier exhibits reasonable smoothness in decision regions. In boosting the smoothness of decision regions is induced by choosing weak learners globally i.e. by looking at all the data points (with appropriate data weights) which provides an opportunity to account for noise. Where in standard decision tree methods only the first weak learner is chosen by looking to all the data points and other ones are chosen locally (by looking to data points of a subregion) which especially in presence of noise (as demonstrated in Fig. 3), yields to a severe overfitting. Overfitting in decision trees is tackled by either bagging (Quinlan 1996) or pruning (Esposito et al. 1997). However the first way introduces much larger classification cost (number of bagged classifiers). The second way is limited in accuracy for high dimensional data as demonstrated in Sect. 8.2.

We propose a method to grow a tree from the decision regions of a boosting classifier. As shown in Fig. 4, the tree obtained, called *super tree*, preserves the Boosting decision regions: it places a leaf node on every region that is important to form the identical decision boundary (i.e. accuracy). In the mean time, Super tree has many short paths that reduce the average number of weak-learners to use when classifying a data point. In the example, the super tree on average needs 3.8 weak-learners to perform classification whereas

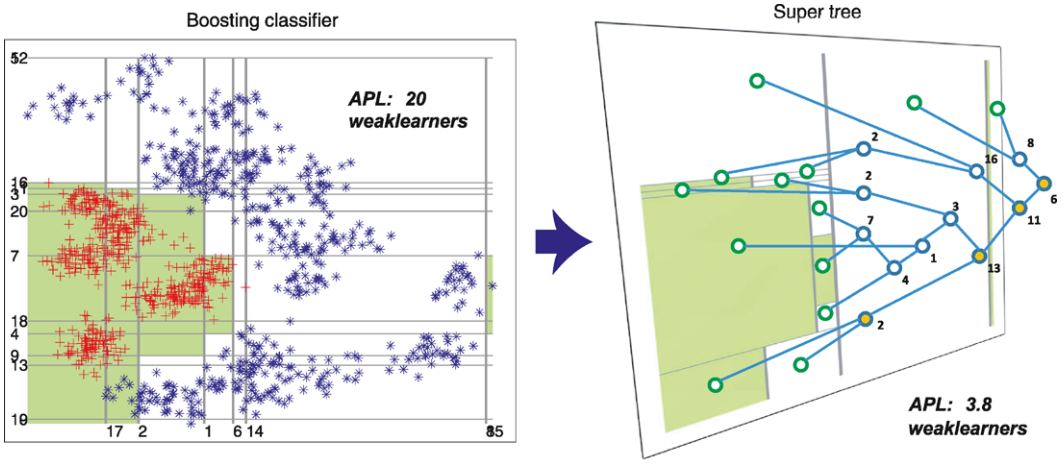

Fig. 4 Converting a boosting classifier into a tree for speeding up. The proposed conversion preserves the Boosting decision regions and speeds up 5 times by having many short paths

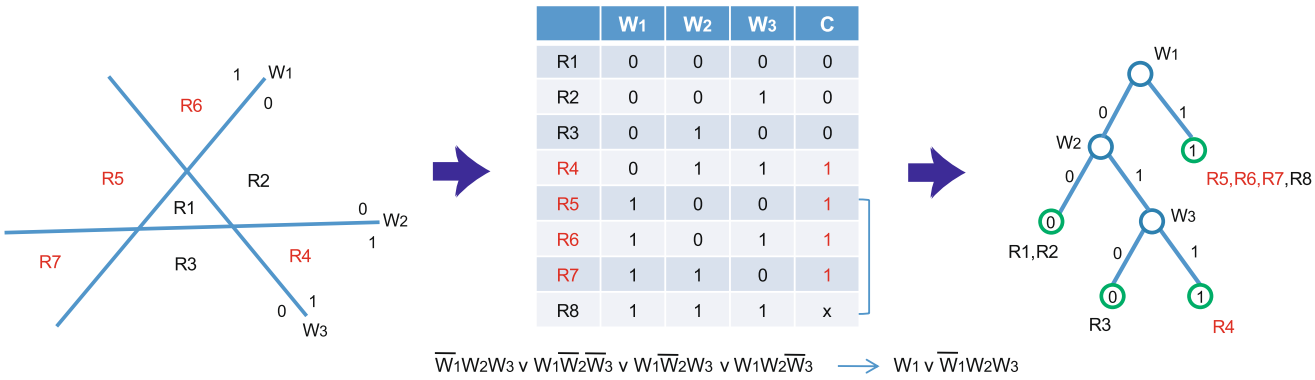

Fig. 5 Boolean expression minimisation for an optimally short tree. (**a**) A boosting classifier splits a space by binary weak learners (*left*). The regions are represented by the boolean table and the boolean expression is minimised (*middle*). An optimal short tree is built on the minimum expression (*right*)

the boosting classifier needs 20: all 20 weak-learners are used for every point.

4 Boolean Optimisation Formulation

There are various Boosting techniques such as LogitBoost (Friedman et al. 2000), ConfidenceBoost (Schapire and Singer 1998) etc. as well as AdaBoost (Freund and Schapire 1997) (arguably the most popular boosting algorithm), all of which is discrete and interpreted as the unified boosting framework by gradient descent, called AnyBoost (Mason et al. 2000). Such a boosting classifier is represented by the weighted sum of binary weak-learners as

$$H(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x}), \tag{1}$$

where α_i is the weight and h_i the i-th binary weak-learner in $\{-1, 1\}$. The boosting classifier splits a data space into 2^m primitive regions by m binary weak-learners. Regions $R_i, i = 1, \ldots, 2^m$ are expressed as boolean codes (i.e. each weak-learner h_i corresponds to a binary variable w_i). See Fig. 5 for an example, where the boolean table is comprised of 2^3 regions. The region class label c is determined by (1). Region R_8 in the example does not occupy the 2D input space and thus receives the *don't care* label marked "x" being ignored when representing decision regions. The region prior $p(R_i)$ is introduced for data distribution as $p(R_i) = M_i/M$ where M_i and M are the number of data points in the i-th region and in total. The decision regions of the boosting classifier are encoded by a set of regions represented as

$$\begin{cases} B(R_i): & \text{boolean expression} \\ c(R_i): & \text{region class label} \\ p(R_i): & \text{region prior} \end{cases} \tag{2}$$

With the region coding, an optimally short tree is defined in terms of average expected path length of data points as

$$\mathbf{T}^* = \min_{\mathbf{T}} \sum_i E(l_{\mathbf{T}}(R_i))\, p(R_i), \tag{3}$$

where $\mathbf{T}$ denotes all possible configurations of a decision tree. $E(l_{\mathbf{T}}(R_i))$ is the expected path length of the i-th region in $\mathbf{T}$. The path length is simply the number of weak-learners (or split-nodes) on the path to the i-th region. The decision tree should closely duplicate the decision regions of the boosting classifier as a constraint of the optimisation: the regions that do not share the same class label $c(R_i)$ must not be put in the same leaf-node of the tree. Any regions of *don't care* labels are allowed to be merged with other regions for the shortest path possible.

Preserving the decision regions of a boosting classifier yields reasonably smooth decision regions for good generalisation. The region smoothness can be defined by

$$\sum_{i,j} \frac{|c(R_i) - c(R_j)|}{H_d(B(R_i), B(R_j))}, \tag{4}$$

where H_d is the hamming distance of the region boolean codes and c is the region label. Neighbouring regions (i.e. the regions of small hamming distance) have coherent labels for the smoothness. The region smoothness formula is not explicitly used in the optimisation in Sect. 5 but explains how a boosting classifier differs from a tree in terms of the smoothness of decision regions.

Discussion on Boolean Expression Minimisation The boolean expression for the table in Fig. 5 can be minimised by optimally joining the regions that share the same class label or *don't care* label as

$$\begin{aligned} &\overline{\mathrm{w}}_1\mathrm{w}_2\mathrm{w}_3 \vee \mathrm{w}_1\overline{\mathrm{w}}_2\overline{\mathrm{w}}_3 \vee \mathrm{w}_1\overline{\mathrm{w}}_2\mathrm{w}_3 \vee \mathrm{w}_1\mathrm{w}_2\overline{\mathrm{w}}_3 \\ &\quad \longrightarrow \mathrm{w}_1 \vee \overline{\mathrm{w}}_1\mathrm{w}_2\mathrm{w}_3 \end{aligned} \tag{5}$$

where $\vee$ denotes OR operator. The minimised expression has a smaller number of terms. Only the two terms, w_1 and $\overline{\mathrm{w}}_1\mathrm{w}_2\mathrm{w}_3$ are remained representing the joint regions $R_5 - R_8$ and R_4 respectively. A short tree is then built from the minimised boolean expression by placing more frequent variables at the top of the tree (see Fig. 5(right)). The method for Boolean expression minimisation has been widely used for optimising redundant trees or similarly minimising circuits composed of "AND" and "OR" gates but it is NP-hard when it has a large number of variables i.e. weak-learners. Note that the existing methods are typically limited to 10 or 15 variables, which makes the application difficult for our problem that involves a large number of variables i.e. weak-learners. Furthermore, all regions are treated with equal importance in the kind of methods, while an optimally short tree is defined by considering the data distribution i.e. region prior in (3).

5 Growing a Super Tree

We propose a novel boolean optimisation method for obtaining a reasonably short tree for a large number of weak-learners of a boosting classifier. The classifier information is efficiently packed by using the region coding and a tree is grown by maximising the region information gain. First, a base algorithm is explained, then its limitations and an improved method are presented in the following section. We use the notations in Sect. 4 to describe the algorithms.

Regions of Data Points The number of primitive regions 2^m is intractable when m is large. Regions R_i that are occupied by any training data points are only taken as input s.t. $p(R_i) > 0$. The number of input regions is thus smaller than the number of data points. Regions with no data points are labeled *don't care*.

Tree Growing by the Region Information Gain Huffman coding (Cormen et al. 2001) is related to our optimisation. It minimises the weighted (by region prior in our problem) path length of code (region). The technique works by creating a binary tree of nodes by maximising the entropy-based information gain. We similarly grow a tree based on the region information gain for an optimally short tree. For a certain weak-learner w_j, $j = 1, \ldots, m$, the regions in the left split and the right split w.r.t. the weak-learner are readily given from the boolean expressions as

$$\begin{aligned} &\mathbf{R}_l = \{R_i | B(R_i) \wedge \overline{\mathrm{w}}_1 \cdots \mathrm{w}_j \cdots \overline{\mathrm{w}}_m) = 0\} \\ &\mathbf{R}_r = \mathbf{R}_n \setminus \mathbf{R}_l \end{aligned} \tag{6}$$

where $\mathbf{R}_n$ is the set of regions arriving at the node n and $\wedge$ is AND operator. At each node, it is found the weak-learner that maximises

$$\Delta I = -\frac{\sum_{\mathbf{R}_l} p}{\sum_{\mathbf{R}_n} p} E(\mathbf{R}_l) - \frac{\sum_{\mathbf{R}_r} p}{\sum_{\mathbf{R}_n} p} E(\mathbf{R}_r) \tag{7}$$

where p is the region prior and E is the entropy function of the region class distribution, which is

$$Q(c^*) = \sum_{\mathbf{R}_c^*} p, \quad \text{where } \mathbf{R}_c^* = \{R_i | c(R_i) = c^*\}. \tag{8}$$

The node splitting is continued until all regions in a node have the coherent region label. The key ideas in the method have two-folds: (1) growing a tree from the decision regions and (2) using the region prior (data distribution). Compared to conventional decision trees built on data points, the proposed tree is grown upon smooth decision regions, guaranteeing better generalisation. Using the region prior helps getting an optimally short tree in the sense of average path length of data points.

6 Tree Growing with the Extended Regions

The base algorithm in the previous section encounters performance degradation for a high dimensional input space. Only encoding the regions of data points does not reproduce the exactly same decision regions of a boosting classifier. Regions of no data points may be assigned different class labels from the original ones, since they are *don't cares* in the tree learning. When a test point falls into those regions, the boosting classifier and the tree would make different decisions. This degrades classification accuracy when data has a high dimension for a given number of training data. Regions along the decision boundary are important although they do not have an actual data point when training. Covering as much of the regions as possible ensures good performance. Adding up the primitive regions, however, becomes soon computationally prohibitive. To help close replication of the decision regions, we propose the extended regions and the accordingly modified region information gain.

Extended Regions The region transformation is proposed to cover the regions in a fairly sufficient and yet computationally tractable manner. It takes each primitive region of data point (R_i) multiple times (see Table 3(a) for this effect) and pushes it closer into the decision boundary by randomly flipping 1's to 0's (if the region class is positive) or 0's to 1's (if negative) until the boosting sum gets close to 0 but still keeps the same class. The extended region ER_i is then obtained by replacing all 0's in the boolean code of the pushed region with *don't care* variables.

A toy example is provided in Table 1. A positive class region coded as 101111 by weak-learner response has it's boosting sum equal to $1.0 - 0.8 + 0.7 + 0.6 + 0.4 + 0.2 = 2.1$. It is turned into boundary regions 001111 (boosting sum $-1.0 - 0.8 + 0.7 + 0.6 + 0.4 + 0.2 = 0.1$) and 101001 by accordingly flipping response of weak-learner w_1 (*boundary region* 1) and weak-learners w_4 and w_5 (*boundary region* 2). The resulting extended regions 1 and 2 then represent two overlapping however not self-contained sets of primitive regions: {001111, 011111, 101111, 111111} (*extended region* 1 in Table 1) and {101001, 101011, 101101, 101111, 111001, 111011, 111101, 111111} (*extended region* 2). Since the region space is big enough, it is unlikely to get identical extended regions or many regions with significant overlaps by the random drawing. The extended regions maintain the region class label $c(R_i)$ and prior $p(R_i)$.

Table 1 Extended region coding

	W1	W2	W3	W4	W5	W6	Sum	C
Weight	1.0	0.8	0.7	0.6	0.4	0.2	3.7	
Region	1	0	1	1	1	1	2.1	1
Boundary region 1	0	0	1	1	1	1	0.1	1
Extended region 1	x	x	1	1	1	1	0.1-3.7	1
Boundary region 2	1	0	1	0	0	1	0.1	1
Extended region 2	1	x	1	x	x	1	0.1-3.7	1

Modified Region Information Gain When splitting nodes (6) an extended region can be placed in both left and right splits due to the existence of *don't care* variables. The repetition of same extended regions at different nodes does not hinder from duplicating the decision regions but increases the average tree length. To compensate the repetition, the information gain is modified as

$$\Delta J = \left(\frac{|\mathbf{R}_l| + |\mathbf{R}_r|}{|\mathbf{R}_n|} \right)^t \Delta I \tag{9}$$

where ΔI is the information gain in (7), which takes a value in $[-\infty, 0]$. The first term equals to one for the primitive regions but is in the range of Viola and Jones (2001, 2004) for the extended regions. The modified gain penalises weak-learners that place extended regions in both splits. The weight factor t is set empirically (see Table 3(b)). See Fig. 6 for the pseudo-code of the proposed algorithm.

7 Two Stage Cascade

The proposed method scales up to meaningfully several tens of weak-learners (see Sect. 7.1) on a standard PC. For allowing any larger number of weak-learners, a two stage cascade is exploited. A conventional multi-stage boosting cascade and fast-exit method is briefly reviewed in Sect. 7.1 and Sect. 7.2 respectively, followed by the proposed two-stage cascade in Sect. 7.3.

7.1 Conventional Multi-Stage Cascade

A cascade of boosting classifiers is typically obtained by changing the number of weak-learners and the threshold of booting classifiers. The boosting classifiers of the smaller numbers (thus more efficient) of weak-learners are placed at early stages of a cascade, to reject as many of the negative class samples as possible while passing almost all positive class samples to next stages. Those at the early stages have a lower threshold, yielding higher detection rates and higher false positive rates. The detection rate and false positive rate of the entire cascade are

$$S = \prod_{i=1}^{K} s_i, \qquad E = \prod_{i=1}^{K} e_i,$$

where K is the number of classifiers, and s_i, e_i are the detection rate and false positive rate of the ith classifier respectively. Designing a cascade is a difficult optimisation problem on the number of classifiers or stages, the number

Fig. 6 Pseudocode of the algorithm

Algorithm: Growing a super tree

Input: a set of data point regions R or extended regions ER, encoded by $\{B, c, p\}$

Output: a decision tree

1. Start with a root node $n = 1$ containing the list of all regions $\mathbf{R}_n$.
2. For $i = 1, \ldots, m$
3. Spit the node: $(\mathbf{R}_l, \mathbf{R}_r) = \text{split}(\mathbf{R}_n, \text{w}_i)$ (by (6)).
4. Compute the gain: $\Delta I = \text{gain}(\mathbf{R}_l, \mathbf{R}_r)$ (by (7) or (9) for the extended region).
5. Find w_i^* that maximises the information gain.
6. If the gain is sufficient, save it as a split node. Else, save it as a leaf node.
7. Go to a child of split node and recurse the steps 2–6 setting $\mathbf{R}_n = \mathbf{R}_l$ or $\mathbf{R}_r$.

of features (or weak-learners) of each stage, the threshold of each stage, while minimizing the total number of features given a target value of S and E. The final detector of Viola and Jones (2001) driven by a trial and error process is a 32 layer cascade of classifiers which includes a total of 4297 features: the first classifier uses two features, the second classifier five features, and the next three layers are of 20-feature classifiers followed by two 50-feature classifiers followed by five 100-feature classifiers and then twenty 200-feature classifiers. Note that the super tree algorithm is applicable to each of the early stage boosting classifiers, which are more crucial to determine the speed-vs-accuracy trade-off, as they set less than a hundred weak-learners.

7.2 Fast-Exit

It applies the weak-learners in the order of weights α of the boosting classifier and lets a data point exit as soon as the boosting sum (1) reaches to the value whose sign cannot be altered by the remaining weak-learners. This method speeds up a boosting classifier while delivering the exactly same accuracy as the boosting classifier, regardless of the number of weak-learners used.

7.3 Cascade of Super Tree and Fast-Exit

Designing a cascade involves a number of parameters to set as explained in Sect. 7.1. The setting is more difficult with more stages. Our solution explained in the previous section can be seen as a convenient way of speeding up a boosting classifier up to several tens of weak-learners without need of a multi-stage cascade. We use a two stage cascade to cope with any larger number of weak-learners. We first designed a two-stage cascade of boosting classifiers in a conventional way, as described in Sect. 7.1, by varying the number of weak-learners (but limiting the number of weak-learners of the first stage to less than a hundred) and the thresholds. Then, the two stage boosting classifiers were replaced with the super-tree and the fast-exit method. As shown in Fig. 7, it places the super tree in the first stage and the fast-exit

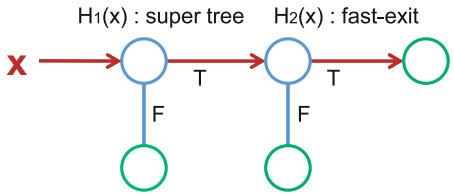

Fig. 7 Schematic diagram of the two stage cascade

method in the second stage. The fast-exit method, which yields the same accuracy as a boosting classifier of any large number of weak-learners, is required to meet the target accuracy of S and E of a cascade. The proposed cascade significantly speeds up a two-stage cascade of standard boosting classifiers and the same of the fast-exits at both stages, as well as a single boosting classifier (see Sect. 8.2).

8 Experiments

8.1 Classification of Synthetic 2D Data

We have made twelve 2D synthetic data sets. Data points of two classes were generated from Gaussian mixtures as shown in Fig. 8(left) and Fig. 9(left). The six test sets were created by randomly perturbing the train sets. The imbalanced data sets (Fig. 9) have a big dense chunk of negative points at a certain location. We have compared the two methods here: a boosting classifier (AnyBoost implementation (Mason et al. 2000)) and the proposed tree using the data point regions. Vertical and horizontal lines are the weak-learners of boosting. Figure 8(right) and Fig. 9(right) show the results. The left and right y-axis in the graph show the classification error rate and the average path length i.e. number of weak-learners used per point respectively. Note first that the both methods drop the error rate when the number of weak-learners is increased indicating good generalisation. The proposed method exhibited the same accuracy as the boosting classifier for all number of weak-learners. While the boosting classifier linearly increased the

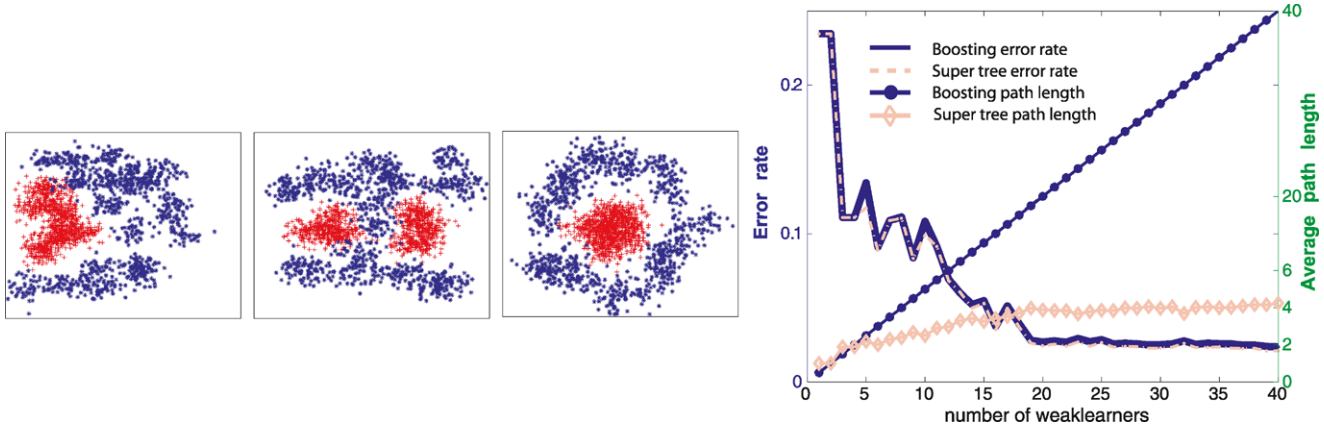

Fig. 8 Experimental results on the synthetic data 1. Examples of 2D synthetic data sets (*left*). Super tree obtains the same accuracy as the boosting classifier significantly shortening the average path length (*right*)

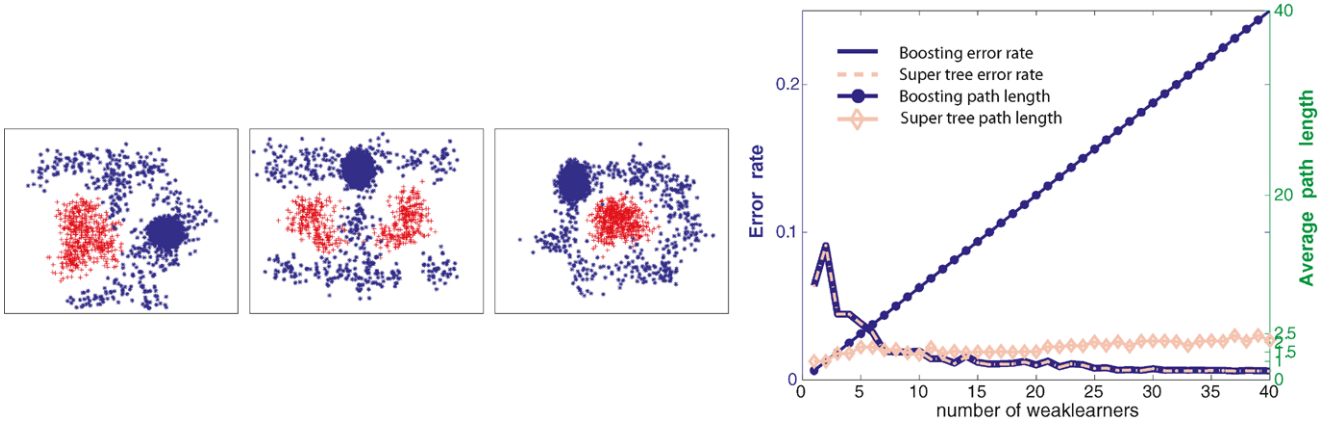

Fig. 9 Experimental results on the synthetic data 2. Examples of 2D synthetic data sets (*left*). The data sets have a big dense chunk of the negative-class samples at a location. Super tree yields the accuracy of the boosting classifier significantly shortening the average path length (*right*)

average path length for the number of weak-learners, the proposed method quickly converged significantly reducing down the average path length. At 40 weak-learners, the super tree speeds up the boosting classifier by 10 and 16 times in Fig. 8(right) and Fig. 9(right) respectively. As expected, the speed gain is bigger when the data distribution is imbalanced, since the proposed method achieves shorter paths effectively for more data point regions, whereas the path length is fixed for all regions in Boosting.

8.2 Face Experiment

For training, we used the MPEG-7 face data set (Kim et al. 2005) that has 11,845 face images collected from a few public face data sets such as Yale and XM2VTS, and non-public face data sets. BANCA face set (520 faces) and Caltech background image sets (900 images) were exploited for bootstrapping. The total number of negative-class images for training, which were either bootstrapped or randomly drawn, is 50,128. We used 21,780 Haar-like features on integral images as weak-learners. We have tested on the MIT+CMU frontal face test set (Rowley et al. 1998) which consists of 114 images with 507 labeled frontal faces. The 507 face and 57000 non-face image patches, which were randomly drawn in location and size (see the rightmost in Fig. 10), were cropped and resized into 24×24 images. Example images are shown in Fig. 10. The methods compared include a standard boosting classifier, Fast exit, Super tree, the two-stage cascade of Fast exits, and the two-stage cascade of Super tree and Fast-exit. For the super tree, we used the extended regions. Fixing the accuracy at 0 threshold, we have compared the average path lengths of the methods in Table 2. For all different numbers of weak-learners, the super tree significantly reduces the average path length of the boosting classifier and the fast exit. The two-stage cascade solution of 60 weak-learner super tree and 200 weak-learner fast exit speeded up the standard boosting by 6.6–12.7 times and even the two-stage cascade of 60 and 200 weak-learner fast exits by 2.5 times. Note that the super tree exploits various combinations of weak-learners (i.e. paths) for an opti-

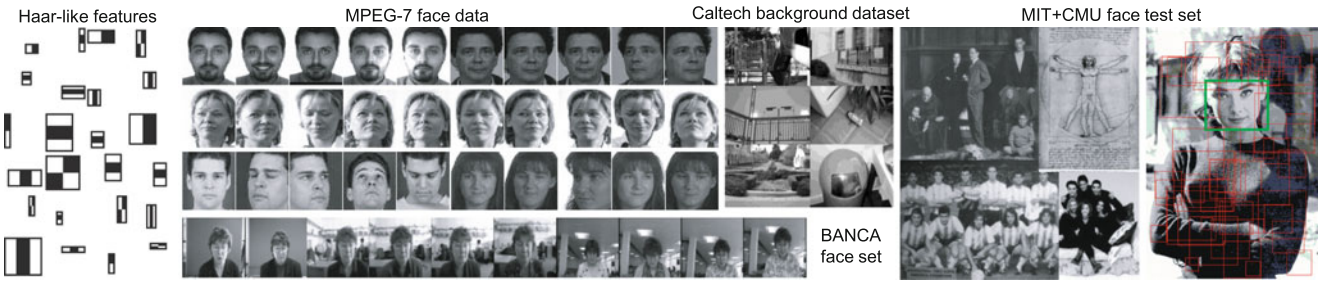

Fig. 10 (Color online) Example features (weak-learners) and face images used. *Green* and *red rectangles* in the rightmost image correspond respectively to positive and some negative samples harvested from the particular image

Table 2 Experimental results on the face images. The numbers in the brackets are for the two-stage cascades

No. of weak learners	Boosting			Fast exit (cascade)			Super tree (cascade)		
	False positives	False negatives	Average path length	False positives	False negatives	Average path length	False positives	False negatives	Average path length
20	501	120	20	501	120	11.70	476	122	*7.51*
40	264	126	40	264	126	23.26	231	127	*12.23*
60	222	143	60	222	143	37.24	212	142	*14.38*
100	148	146	100	148 (144)	146 (149)	69.28 (37.4)	(145)	(152)	(*15.1*)
200	120	143	200	120 (146)	143 (148)	146.19 (38.1)	(128)	(146)	(*15.8*)

Table 3 Performance of super tree for the different numbers of extended regions per region (**a**) and for varying power in the information gain (**b**)

No. of weak-learners		10	20	30	40	50	60
No. per region		1	1	2	10	40	50
FP/FN	Super tree	593/157	367/146	292/136	262/129	203/142	224/129
	Boosting	588/157	378/143	291/137	264/126	202/142	222/143

(a)

Power	0.5	1	3	5	10
Avg path length	16.4	12.3	11.9	14.5	15.8
FP/FN	246/121	247/123	237/124	235/120	251/132

(b)

FN: false negatives, FP: false positives

mal classification speed, whereas the fast exit takes the combinations always in the order of the weak-learner weights. One can also compare the results of Zhou (2005) with ours by the standard boosting and the fast-exit as proxies. Whereas the solution in Zhou (2005) didn't gain much over the boosting and the fast exit method, ours significantly improved the both. More importantly, the method (Zhou 2005) has been tested only for 5 or 10 weak learners whereas our method in a single stage is conveniently scalable up to several tens of weak learners.

Table 3 shows performance of the super tree for the two internal parameters: the number of extended regions per primitive region and the power in the information gain (9). To obtain the close accuracy to the boosting classifier, the required number of extended regions per region grew as the number of weak-learners of Boosting increased. For about the given number of training samples, using 200 extended regions and 100 weak-learners would start hitting theoretical memory boundaries. As shown in Table 3(b), the performance is not very sensitive to different power values in the range. The number of weak-learners and extended regions was set as 40. Power 1–5 gave the best performance. The values smaller than 0.5 increased the average path length and the values larger than 10 increased the error rate.

Super Tree vs Conventional Decision Trees The performance of a conventional decision tree is affected by two main factors: node splitting criterion and pruning. Table 4

Table 4 Pruned conventional decision trees

	Un-pruned			Pruning method														
				Reduced error pruning (REP)			Cost complexity pruning (CCP)			Pre-pruning								
										Impurity			No of points			Max depth		
	FP	FN	APL	FP	FN	APL	FP	FN	APL	FP	FN	APL	FP	FN	APL	FP	FN	APL
Imurity measure																		
Entropy	771	143	9.36	576	135	7.68	639	139	6.61	**518**	**136**	**4.72**	764	142	9.36	743	123	9.32
Gini	1146	146	22.4	726	157	17.82	894	149	10.43	**764**	**144**	**7.72**	1001	150	23.36	829	140	12.06
Misclassification	1163	234	50.84	1070	239	49.27	**1140**	**234**	**49.17**	1163	234	50.84	1163	234	50.8	1138	234	50.38

FN: false negatives, FP: false positives, APL: average path length

shows the performance of decision trees learned using splitting criterions based on 3 impurity measures: Entropy (information gain), Gini index and Misclassification versus different methods of pruning (Esposito et al. 1997): Reduced Error Pruning (REP), Cost-Complexity Pruning (CCP) and pruning based on gain in impurity, number of points in leaf nodes and maximum depth. In order to train and prune the trees (as in Esposito et al. (1997)) we randomly divided the initial training data set for tree growing (70%) and validation (30%). For trees learnt using different splitting criteria pruning both increased accuracy and decreased average path length (APL). Pruning had little effect on trees learned on the misclassification score as it created a very deep imbalanced tree and any pruning resulted in reduced accuracy on the validation set. Pruning based on the impurity seemed to produce the best values both in accuracy and average depth for trees learnt using Shannon's entropy and Gini index. However even the best value of 136 false negatives and 518 false positives for the tree learned using Shannon entropy is inferior to the performance of Super Tree of the similar path length—122 false negatives and 476 false positives. See Table 2.

Training Time We have so far discussed the evaluation time of a boosting classifier, not the training time. Additional training time for the conversion into Super Tree is relatively small compared to the typical Boosting training time. The conversion time for the Super Tree for e.g. 40 weak-learners took about an hour. For more than 60–80 weak-learners we use a two-stage cascade. The boosting training time could also be significantly reduced by e.g. Pham and Cham (2007).

Over a Multi-stage Boosting Cascade Although the comparison has been made on the two-stage cascades in the experiments, the proposed method can afford a speed up over a standard multi-stage boosting cascade by replacing each stage of a boosting classifier in the cascade with a Super Tree. If one of the boosting classifiers is too large (for example 100 or 200 weak learners) then it is replaced with Fast-exit method instead of Super tree. The speed gain would vary depending on the number of weal-learners and the data distribution i.e. the ratio of positive over negative data points in each stage boosting classifier. See Table 2 for the speed gains obtained on the different numbers of weak-learners but on the same data distribution as specified above.

8.3 Rapid-Moving Object Tracking

Tracking is often done by fast re-detection. It sets a search window based on the previous location and speed of a target object, and detects the object again within the search window. The tracker using the Super Tree as the detector (or classifier) performed better than the boosting tracker, owing to its higher speed: it was able to process more frames than the boosting tracker. Under rapid object movement, missing a frame means experiencing a drift. We have collected two sample sequences of 320×240 pixels at 30 frames/second, called Toy car and Face, and used a public sequence of Ross et al. (2008). Each sequence is partitioned into two parts: one for training and the other for testing. For training, the positive samples were collected using a guide rectangle and the negative samples using randomly drawn patches around the guide rectangle. The pool of 21,780 haar-like features was exploited for weak-learners. For the Toy car sequence, the super tree obtained from the boosting classifier of 60 weak-learners had 14 weak-learners as its average path length. The execution time of the trackers using two methods, which were implemented by Matlab mex functions, is: 0.0015 (for integral images), 0.0117 (for weak-learners) and 0.0018 (for the weighted sum by α) seconds in Boosting, and 0.0015 (integral images) and 0.0027 (weak-learners) seconds in Super tree. Thus, the execution rates of the two trackers including the image capture time were 20 frames per second for Boosting and 27 frames per second for Super tree. By varying detection density similar speed gain was achieved for other two sequences. Figure 11 shows that the super tree tracks well the target object in the three test sequences (Toy car, Ross et al. (2008), Face) while the boosting tracker lost it and exhibited much drifting at the initial frames when it moved

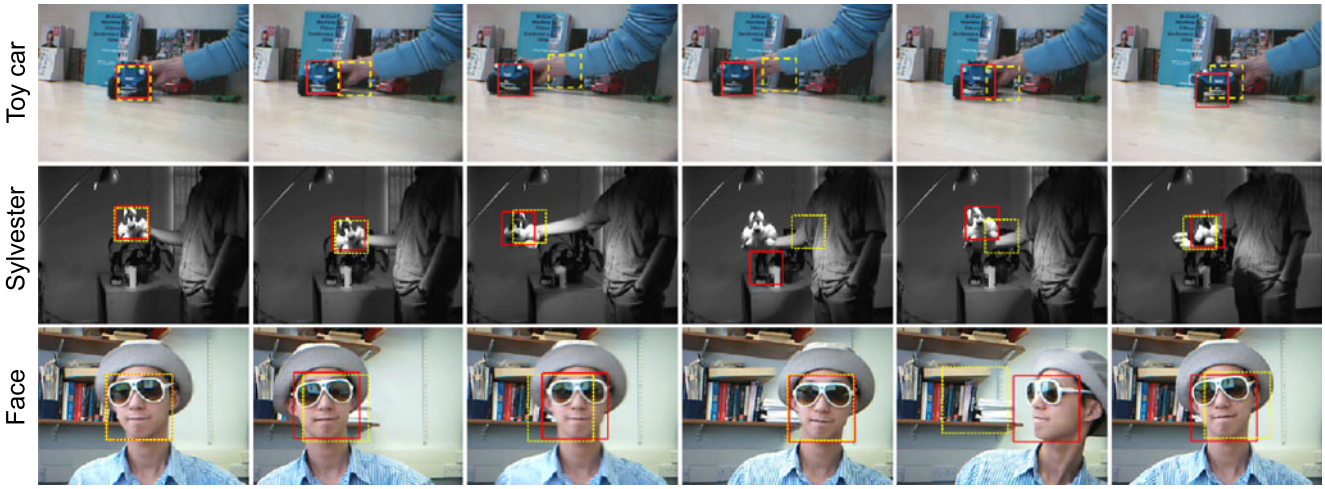

Fig. 11 (Color online) Performance of Boosting and Super tree trackers. Super tree tracker (*solid red line*) achieves a better tracking accuracy by faster classification than Boosting tracker (*dashed yellow line*). The super tree tracks well the target object while the boosting tracker lost the object and exhibited much drifting when it moved abruptly

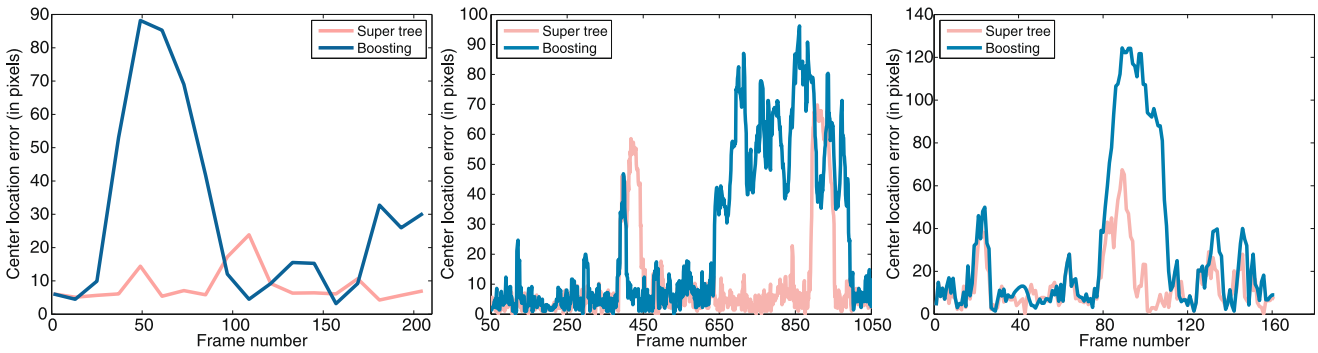

Fig. 12 Tracking errors. The center location errors (in pixels) of the two trackers are shown for three sequences: Toy car (*left*), Sylvester (*middle*), Face (*right*)

Table 5 Performance of Boosting and Super tree trackers

Experiment	No. of weak-learners	Boosting		Super tree	
		ACLE	Average path length	ACLE	Average path length
Toy car	60	28.6	60	**8.4**	*13.8*
Sylvester	40	24.6	40	**10.1**	*7.1*
Face	20	29.8	20	**13.9**	*6.0*

ACLE: average center location errors (in pixels)

abruptly. The average center location errors (in pixels) in Toy car experiment were 8.4 for the super tree and 28.6 for the boosting. See Table 5 for other sequences and Fig. 12 for the error graph. The benefit of using Super tree for rapid tracking would be bigger when a higher frame rate camera is available. A simple proof-of-concept experiment on tracking has only been performed in this study. More experimental analyses and online tree learning algorithms (Basak 2004; Yeh et al. 2007), which could further benefit the super tree tracker by adaptation and makes a comparative study with state-of-the-art trackers meaningful, are remained as future work.

8.4 Segmentation by Pixel-Wise Classification

The car driving sequences (Brostow et al. 2008) were exploited for the experiment. Boosting classifier and super tree were trained for the binary problem for the building class against non-building class. 1323 DCT features were drawn from 21×21 RGB image patch as weak-learners. The train set consisted of 7143 positive and 23217 negative pixels from 184 images of 11×15 pixel resolution. Randomisation in learning (similarly to Rahimi and Recht (2008)) reduced the train time of the boosting classifier. The test set contained 38445 points from 233 images. The correct recognition rate of Boosting of 40 weak-learners was 0.71 (as global accuracy). The super tree learnt by 10 extended

Table 6 Segmentation accuracies and average path lengths

No. of weak learners	Boosting		Fast exit (cascade)		Super tree (cascade)	
	Global accuracy (%)	Average path length	Global accuracy (%)	Average path length	Global accuracy (%)	Average path length
40	71	40	71	26.6	70	*15.2*
60	73.1	60	73.1	40.7	72.6	*17.6*
200	74.4	200	74.4 (74.6)	175.7 (108.4)	(74.8)	(*94.4*)

Fig. 13 (Color online) Segmentation results. Pixels classified into the building class by Super tree (or Boosting) are shown in *darker hue*

regions per region obtained the close accuracy as 0.70 using only 15 weak-learners on average. See Table 6 for the different numbers of weak-learners. The accuracy obtained seems comparable to Brostow et al. (2008). Note that as in the face experiment (Table 2) in order to apply Super tree for a boosting classifier of 200 weak-learners we used a two stage cascade consisting of 60 and 200 weak learners. The relative gain of Super tree in terms of average path length is considerably smaller in the segmentation experiment due to a much larger ratio (1:3 vs 1:110) of positive over negative data points. Figure 13 shows the segmentation results. The blocky effect was due to the low pixel image resolution used.

9 Conclusion

We have proposed a novel way to speed up a boosting classifier. The problem is formularised as boolean optimisation and a new optimisation method is proposed for a large number of weak-learners. The tree grown from the decision regions of a boosting classifier, called Super tree, provides many short paths and preserves the Boosting decision regions. The single super tree delivers the close accuracy to a boosting classifier with a great speed-up for up to several tens of weak-learners. The proposed two stage cascade allows any number of weak-learners. Experiments have shown that the tree obtained is reasonably short in terms of average path length outperforming a standard boosting classifier, Fast exit, their cascades. The method has been also demonstrated for rapid object tracking and segmentation problems.

References

Avidan, S. (2006). SpatialBoost: adding spatial reasoning to adaboost. In *Proc. ECCV*, Graz, Austria.

Basak, J. (2004). Online adaptive decision trees. *Journal of Neural Computation*, *16*, 1959–1981.

Brostow, G., Shotton, J., Fauqueur, J., & Cipolla, R. (2008). Segmentation and recognition using structure from motion point clouds. In *Proc. ECCV*, Marseilles.

Chen, J. (1994). Application of Boolean expression minimization to learning via hierarchical generalization. In *Proc. ACM symposium on applied computing* (pp. 303–307).

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2001). *Introduction to algorithms*. Cambridge: MIT Press and McGraw-Hill.

Esposito, F., Malerba, D., Semeraro, G., & Kay, J. (1997). A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*, 476–491.

Freund, Y., & Mason, L. (1999). The alternating decision tree learning algorithm. In *Proc. ICML*.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, *28*(2), 337–407.

Grabner, H., & Bischof, H. (2006). On-line boosting and vision. In *Proc. IEEE conf. CVPR* (pp. 260–267).

Grossmann, E. (2004a). AdaTree: boosting a weak classifier into a decision tree. In *IEEE workshop on learning in computer vision and pattern recognition* (pp. 105–105).

Grossmann, E. (2004b) *Adatree 2: boosting to build decision trees or Improving Adatree with soft splitting rules* (Technical report).

Huang, C., Ai, H., Li, Y., & Lao, S. (2005). Vector boosting for rotation invariant multi-view face detection. In *Proc. ICCV*.

Kim, T.-K., Kim, H., Hwang, W., & Kittler, J. (2005). Component-based LDA face description for image retrieval and MPEG-7 standardisation. *Image and Vision Computing*, *23*(7), 631–642.

Li, S. Z., & Zhang, Z. (2004). Floatboost learning and statistical face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(9), 1112–1123.

Mason, L., Baxter, J., Bartlett, P., & Frean, M. (2000). Boosting algorithms as gradient descent. In *Proc. advances in neural information processing systems* (pp. 512–518).

Pham, M., & Cham, T. (2007). Fast training and selection of Haar features using statistics in boosting-based face detection. In *Proc. ICCV*.

Quinlan, J. (1996). Bagging, boosting, and c4.5. In *Proc. national. conf. on artificial intelligence* (pp. 725–730).

Rahimi, A., & Recht, B. (2008). Random kitchen sinks: replacing optimization with randomization in learning. In *Proc. neural information processing systems*.

Ross, D., Lim, J., Lin, R., & Yang, M. (2008). Incremental learning for robust visual tracking. *International Journal of Computer Vision*, *77*(1), 125–141.

Rowley, H., Baluja, S., & Kanade, T. (1998). Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*, 22–38.

Schapire, R. E., & Singer, Y. (1998). Improved boosting algorithms using confidence-rated predictions. In *Proc. the eleventh annual conference on computational learning theory* (pp. 80–91).

Schwender, H. (2007). *Minimization of boolean expressions using matrix algebra* (Technical report). Collaborative Research Center SFB 475, University of Dortmund.

Sochman, J., & Matas, J. (2005). WaldBoost learning for time constrained sequential detection. In *Proc. CVPR*, San Diego, USA.

Torralba, A., Murphy, K. P., & Freeman, W. T. (2007). Sharing visual features for multiclass and multiview object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(5), 854–869.

Tu, Z. (2005). Probabilistic boosting-tree: learning discriminative models for classification, recognition, and clustering. In *Proc. ICCV*.

Viola, P., & Jones, M. (2001). Robust real-time object detection. In *2nd intl. workshop on statistical and computational theories of vision*.

Viola, P., & Jones, M. (2004). Robust real-time face detection. *International Journal of Computer Vision*, *57*(2), 137–154.

Wu, B., & Nevatia, R. (2007). Cluster boosted tree classifier for multi-view, multi-pose object detection. In *Proc. ICCV*.

Xiao, R., Zhu, L., & Zhang, H. (2003). Boosting chain learning for object detection. In *Proc. ICCV*.

Yeh, T., Lee, J., & Darrell, T. (2007). Adaptive vocabulary forests for dynamic indexing and category learning. In *Proc. ICCV*.

Zhou, S. (2005). A binary decision tree implementation of a boosted strong classifier. In *IEEE Workshop on analysis and modeling of faces and gestures* (pp. 198–212).