Making a Shallow Network Deep: Growing a Tree from Decision Regions of a Boosting Classifier

Tae-Kyun Kim*, Ignas Budvytis*, Roberto Cipolla

CUED/F-INFENG/TR633

1 July 2009

* indicates equal contributions.

Department of Engineering
University of Cambridge
Trumpington Street
Cambridge CB2 1PZ, UK

{tkk22|ib255|cipolla}@eng.cam.ac.uk

Making a Shallow Network Deep: Growing a Tree from Decision Regions of a Boosting Classifier

Tae-Kyun Kim[1*], Ignas Budvytis[1*], Roberto Cipolla[1]

[1] : Department of Engineering, University of Cambridge, Cambridge, CB2 1PZ, UK.
[*] indicates equal contributions.

Abstract– This paper presents a novel way to speed up the classification time of a boosting classifier. Despite boosting already being known as a common technique in applications demanding very fast classification, its flat structure leaves a room to be improved in speed. We make the shallow (flat) network deep (hierarchical) by growing a tree from the decision regions of a boosting classifier. This provides many short paths for speeding up and preserves reasonably smooth decision regions for good generalisation. We express the conversion as a Boolean optimisation problem, which has been previously studied for circuit design. A novel method is proposed for a large number of binary variables, i.e. weak-learners of a boosting classifier. The learnt boosting classifier splits the data space into 2^n primitive regions by n binary weak-learners. The decision regions of boosting are encoded by the set of extended regions each of which aggregates many primitive regions of the same label. A decision tree is then grown using the modified information gain measure based on the coding of regions. Experiments on the synthetic and face image data sets have shown that the tree obtained is reasonably short in terms of average path length of data points, significantly speeding up a boosting classifier at similar accuracy. The proposed two stage cascade allows any number of weak-learners outperforming the corresponding boosting cascade in speed. The method is also demonstrated for rapid object tracking and segmentation problems.

Keywords– Boosting, Fast Classification, Decision Tree, Boolean Optimisation, Cascade, Object Detection, Rapid Object Tracking, Segmentation.

1 Introduction

Boosting classifiers have been widely adopted in visual recognition, in particular for object detection [1], tracking [2] and segmentation [3] problems, requiring very fast classification.

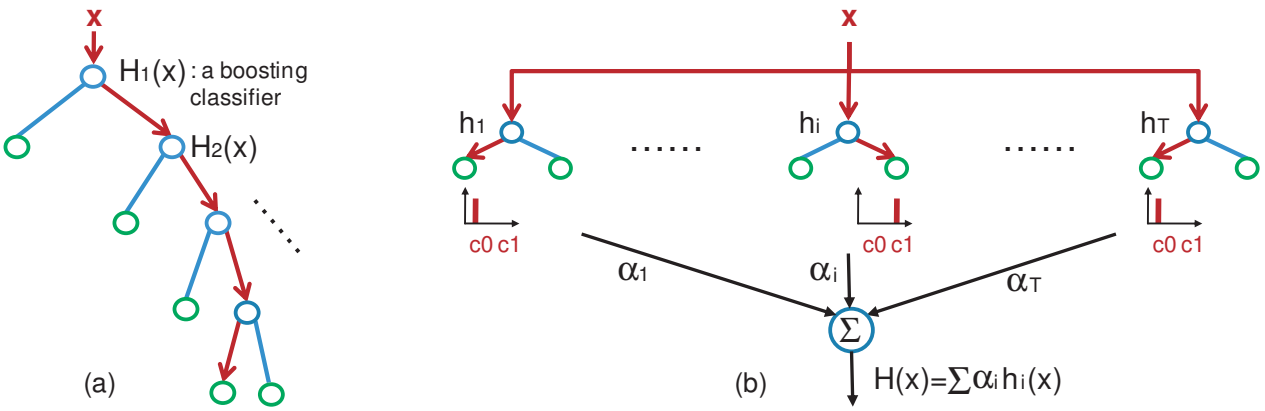

Figure 1: Boosting as a tree. *(a) Boosting cascade is seen as an imbalanced tree. Each node is a boosting classifier. (b) A boosting classifier has a very shallow and flat network where each node is a decision-stump i.e. weak-learner.*

Boosting forms a strong classifier by aggregating simple weak-learners which deliver both quick and accurate performance. Classification time is usually further reduced using a cascade of boosting classifiers (a so-called "coarse-to-fine" strategy [1]). The cascade could be seen as a degenerate tree as shown in Figure 1(a). In this tree, each node corresponds to a strong boosting classifier. First classifiers are set to have a smaller number of weak-learners in order to quickly filter out negative class samples. Subsequent classifiers have more weak-learners and therefore can make a finer decision. This deep structure effectively improves the speed of the single boosting classifier containing all weak-learners. However, designing a cascade, requires setting a number of parameters: the number of classifier stages, weak-learners and the threshold for each stage. Torralba et al have proposed sharing weak-learners among multiple boosting classifiers for efficient multi-object and multi-pose detection [4]. By placing the most common weak-learners of classifiers at the top of the hierarchical structure, whose path corresponds to a boosting classifier, the total number of weak-learners to use is reduced accelerating classification. The method is useful for multiple boosting classifiers.

In this work, we propose a novel way to reduce the classification time of a single boosting classifier without sacrificing its accuracy. Our algorithm is also applicable for multiple boosting classifiers as a general meta algorithm. The chance for improvement comes from the fact that a boosting classifier can be seen as a very shallow network as shown in Figure 1(b), where each weak-learner is a decision-stump. The flat structure ensures reasonably smooth decision regions for good generalisation. It is, however, not optimal in classification time since it exploits all weak-learners for any data point. Some data points are easier than others and thus can be classified by a smaller number of weak-learners. The proposed method converts the shallow network to a deep hierarchical structure, i.e. a decision tree. The obtained tree speeds up the boosting classifier by having many short paths and exhibits good accuracy by preserving the decision regions. We introduce a novel Boolean optimisation method for obtaining a reasonably short tree for a large number

of weak-learners of a boosting classifier. The two stage cascade built on the proposed tree allows the conversion with any number of weak-learners, significantly speeding up a conventional cascade as well as a single boosting classifier.

The remainder of the paper is organised as follows: Section 2 briefly reviews related works. Overview on the conversion is given in Section 3 and the problem formulation as Boolean optimisation in Section 4. Section 5 presents the proposed method. Experimental results are shown in Section 6 and the conclusion is drawn in Section 7.

2 Related work

Boolean expression minimisation. The aim of Boolean expression minimization is to minimize the number of terms and binary variables in the Boolean expression. Algorithms for the minimisation have mainly been studied in the circuit design [10]. Since circuits have strictly predefined specifications exact minimization was the goal of most studies. The complexity of a logic expression rises exponentionally when the number of binary variables increases. Therefore, exact minimisation is an NP-hard problem for a large number of variables. Conventional optimization methods are limited to a small number of binary variables, typically from a few to about 15 variables [10]. Boolean minimisation has been also applied to size down a redundant decision tree, represented by a Boolean table [11]. See Section 4.1 for an example.

Boosting and tree. There are few studies that connect a boosting classifier to a single decision tree because boosting has been typically thought as a regularisation method of multiple decision tree classifiers [9]. Grossmann has proposed the method grows a tree by the boosting mechanism [7]. The resulting tree, called AdaTree, consequently shows a reduced computation cost with respect to Adaboost without need of a cascade. It, however, has suffered from lack of generalisation due to its tree nature, showing much worse accuracy than Adaboost in the face detection experiment and the synthetic data experiment under noise. Zhou has first introduced an idea of representing a learnt boosting classifier by a Boolean table and implementing it as a binary decision tree [6]. His solution is a brute force search for all possible tree configurations, not involving a reasonable optimisation method. The method could afford to only about 5 and 10 weak-learners. The speed gain obtained seems not large over a standard boosting classifier and almost negligible over a straightforward way of improving the boosting classifier in speed, which is called Fast exit (See Section 5.2 for the fast exit method).

3 Conversion of a boosting classifier into a tree

A boosting classifier is closely related to a decision tree. Both a boosting classifier and a decision tree are composed of weak-learners (or equivalently decision-stumps/split-nodes). Whereas the boosting classifier puts decision stumps in a flat structure (See Figure 1(b)), the decision tree has a deep and hierarchical structure (See Figure 2(b)). The different

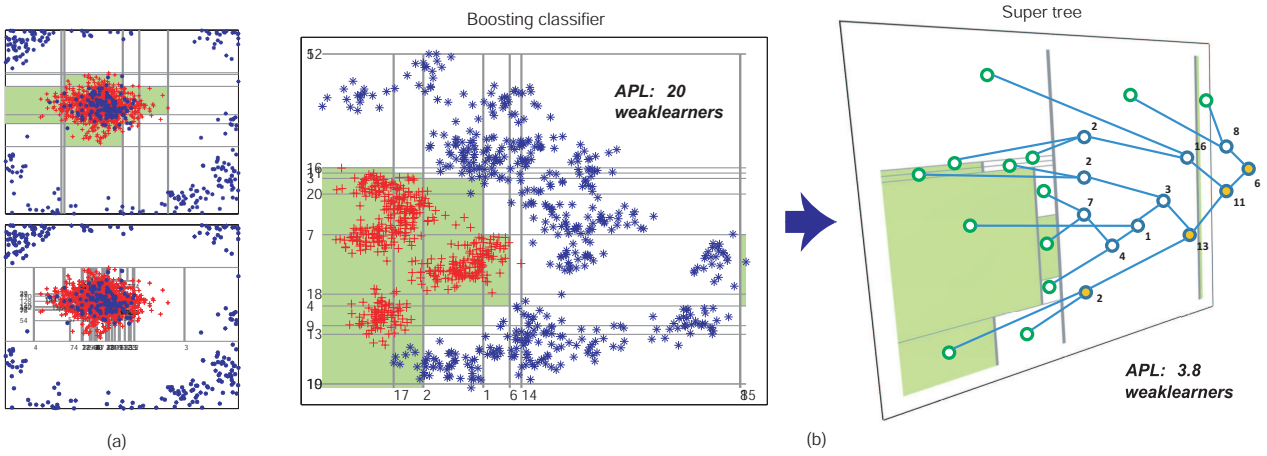

Figure 2: Converting a boosting classifier into a tree for speeding up. *(a) The decision region of boosting (top) is smooth compared to a conventional tree (bottom). (b) The proposed tree preserves the decision boundary of boosting and has many short paths speeding up 5 times.*

structures lead different behaviours: Boosting has a better generalisation but is slower than a tree. The decision regions of a boosting classifier are reasonably smooth, which yields good generalisation performance. Figure 2(a) exemplifies the decision regions of the two methods. Here part of negative (blue) data points are scattered in the middle of positive (red) samples. The boosting classifier shows a reasonable decision smoothness, while a conventional tree forms complex decision regions trying to perfect the classification of training points, which often causes overfitting. Tree pruning and regularisation of multiple trees have been an important topic to relieve the overfitting of a decision tree [9, 8]. We propose a method to grow a tree from the decision regions of a boosting classifier which would avoid overfitting. As shown in Figure 2(b), the tree obtained by the proposed method, called *super tree*, preserves the decision regions. (1) Super tree places a leaf node on every region that is important to form the identical decision boundary (i.e. accuracy). (2) Super tree has many short paths that reduces the average number of weak-learners to use when classifying a data point. In the example, super tree on average needs 3.8 weak-learners to perform classification whereas boosting classifier needs 20. Note that a booting classifier exploits all weak-learners learnt for every point.

4 Boolean optimisation formulation

A learnt boosting classifier is typically represented by the weighted sum of binary weak-learners as

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

where α_i is the weight and h_i the i-th binary weak-learner in $\{-1, 1\}$. The learnt boosting classifier splits a data space into 2^m primitive regions by m binary weak-learners. Regions

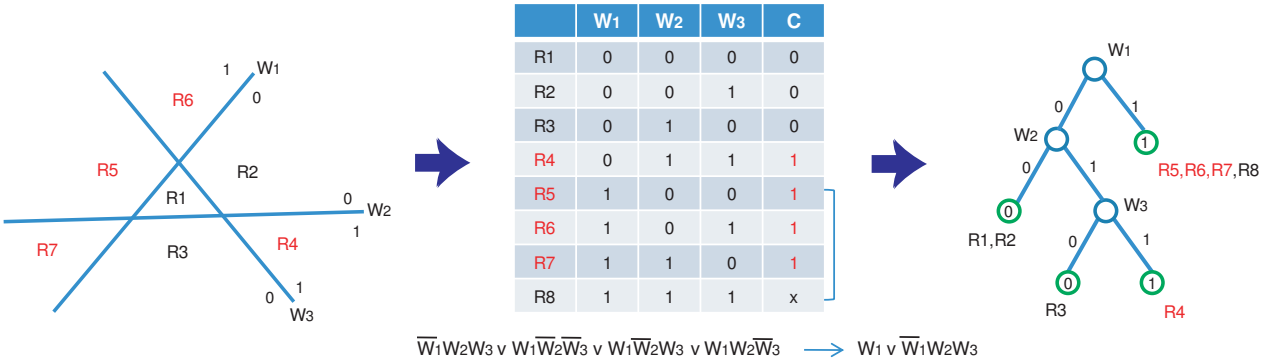

Figure 3: Boolean expression minimisation for an optimally short tree. *(a) A boosting classifier splits the space by binary weak learners (left). The regions are represented by the boolean table and the boolean expression is minimised (middle). An optimal short tree is built on the minimum expression (right).*

$R_i, i = 1, ..., 2^m$ are expressed as boolean codes (i.e. each weak-learner h_i corresponds to a binary variable w_i). See Figure 3 for an example, where the boolean table is comprised of 2^3 regions. The region class label c is determined by (1). Regions that have *don't care* labels are ignored in the optimisation. Region R_8 in the example does not occupy the 2D input space and is ignorable 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 conding, 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 of $\mathbf{T}$. 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 the boosting classifier yields reasonably smooth decision regions for good generalisation. The region smoothness is defined by

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

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) must have coherent labels for the smoothness.

4.1 Discussion on boolean expression minimisation

Boolean expression minimisation has been used for hierarchical generalisation in [11]. The expression for the table in Figure 3 can be minimised by optimally joining the regions that share the same class label or *don't care* label as

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

where $\vee$ denotes OR operator. The minimised expression has a smaller number of terms. Only the two terms corresponding to the joint regions $R_5 - R_8$ and R_4 respectively are left in the example. A short tree is then built from the minimised boolean expression by placing more frequent variables at the top of the tree (See Figure 3(right))[1]. The method for Boolean expression minimisation is close, but, however, not suited to 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. Note that we have formulated an optimally short tree by considering data distribution in Section 4.

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. We use the notations in Section 4 to describe the algorithm.

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 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. We have found that the Huffman coding [12] is closely 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 $w_j, j = 1, ..., m$, the regions in the left split and the right split w.r.t. the weak-learner

[1]The example tree has the weak-learners arranged by weights α_i in the hierarchy, which is not a general case. See Figure 2 for the other example.

are readily given from the boolean expressions as

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

where 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

$$\triangle \mathcal{I} = -\frac{\sum_{\mathcal{R}_l} p}{\sum_{\mathcal{R}_n} p} \mathcal{H}(\mathcal{R}_l) - \frac{\sum_{\mathcal{R}_r} p}{\sum_{\mathcal{R}_n} p} \mathcal{H}(\mathcal{R}_r) \tag{7}$$

where p is the region prior and $\mathcal{H}$ is the entropy function of the region class distribution, which is

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

The node splitting is continued until all regions in a node have the coherent region label.

The key idea in the method above has two-folds: 1) growing a tree from the decision regions and 2) using the region prior (data distribution). Compared to conventional trees built on data points, the proposed tree is grown upon smooth decision regions guaranteeing good generalisation. Using the region prior helps getting an optimally short tree in the sense of average path length of data points.

5.1 Extended regions

The base algorithm in the previous section seems sufficient for a low dimensional input space e.g. 2D. Encoding only the regions of data points results in strictly different decision regions from those 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. Serious degradation is encountered in accuracy when the dimension of data space is high for a given number of training data. Regions along the decision boundary are apparently important although they do not have an actual data point when training. Covering as much of the decision regions of the boosting classifier as possible ensures good performance. Adding up the primitive regions is, however, soon computationally prohibitive. To help close duplication 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 decision regions in a fairly sufficient and yet computationally tractable manner. It takes each primitive region of data point (R_i) multiple times (See Figure 10) 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. See Figure 4 for an example. 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 as e.g. $B(ER_i) = \mathrm{w}_1\mathrm{x}\mathrm{w}_3\mathrm{x}\mathrm{x}$. Each extended region thus contains

	W1	W2	W3	W4	W5	Sum	C
Weight	1.0	0.8	0.7	0.5	0.2	3.2	
Region	1	0	1	1	0	1.2	1
Boundary region	1	0	1	0	0	0.2	1
Extended region	1	x	1	x	x	0.2-3.2	1

Figure 4: Extended region coding.

many primitive regions of the same class label including the ones near to the decision boundary. 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 preserves the region class label $c(R_i)$ and prior $p(R_i)$.

Modified region information gain. When splitting nodes (See (6)) an extended region can be placed in both left and right splits because of the existence of *don't care* variables. This doesn't hurt the duplication of the decision regions but does increase the average tree length by the repetition of same extended regions at different nodes. To compensate the repetition, the information gain is modified as

$$\Delta\mathcal{J} = \left(\frac{|R_l| + |R_r|}{|R_n|}\right)^t \Delta\mathcal{I} \tag{9}$$

where $\Delta\mathcal{I}$ is the information gain described in (7), which takes a value in $[-\infty, 0]$. The first term always equals to one for the primitive regions but is in the range of $[1, 2]$ for the extended regions. The modified gain penalises weak-learners that put many extended regions in both splits. The weight factor t is set empirically (See Figure 11). See Figure 5 for the algorithm.

5.2 Two stage cascade

The proposed method works well up to several tens of weak-learners on a standard PC. The two stage cascade is proposed for allowing any number of weak-learners of a boosting classifier. It places the super tree at the first stage and the fast-exit method (described below) at the second stage. The proposed design significantly speeds up the corresponding two stage fast exit cascade, two stage boosting cascade, as well as a single boosting classifier (See Section 6.2). The super tree as a general meta algorithm is not limited to two stage cascade. It can be further speeded up by more stages as in a standard boosting cascade. In the other sense, the proposed method can be seen as a convenient way of obtaining the comparable speed-up to many stage cascade only by the single super tree or the proposed two stage cascade. Note that designing a cascade requires a lot of parameters to be set, which is obviously more difficult with more stages.

Algorithm: Growing a super tree

Input: a set of the data point regions R or the 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 R_n.
2.For i=1,...,m
3. Spit the node: $(R_l, R_r) = \text{split}(R_n, \mathrm{w}_i)$ (by (6)).
4. Compute the gain: $\Delta\mathcal{I} = \text{gain}(R_l, R_r)$ (by (7) or (9) for the extended region).
5.End
6.Find w_i^* that maximises the information gain.
7.If the gain is sufficient, save it as a split node. Else, save it as a leaf node.
8.Go to a child of split node and recurse the steps 2-7 setting $R_n = R_l$ or R_r.

Figure 5: Pseudocode of the algorithm

Fast-exit. It applies the weak-learners in the order of weights α and exits 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 for exactly the same accuracy regardless of the number of weak-learners.

6 Experiments

6.1 Classification of synthetic 2D data

We have made 2D synthetic data sets. Data points of two classes were generated from Gaussian mixtures as shown in Figure 6. The bottom row shows the imbalanced sets that have a big chunk of negative points at a certain location. The six test sets were created by randomly perturbing the train sets. We have compared the two methods here: a boosting classifier (AnyBoost implementation [5]) and the proposed tree using the data point regions. Vertical and horizontal lines are weak-learners of boosting. Figure 7(left) and (right) shows the results for the data sets in the top and bottom row respectively. 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. Note first that the both methods do drop the accuracy 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 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 around 10 times and 20 times for the balanced and imbalanced cases respectively. As expected, the speed gain is bigger when the data distribution is imbalanced, since the proposed method achieves shorter paths for more data point regions

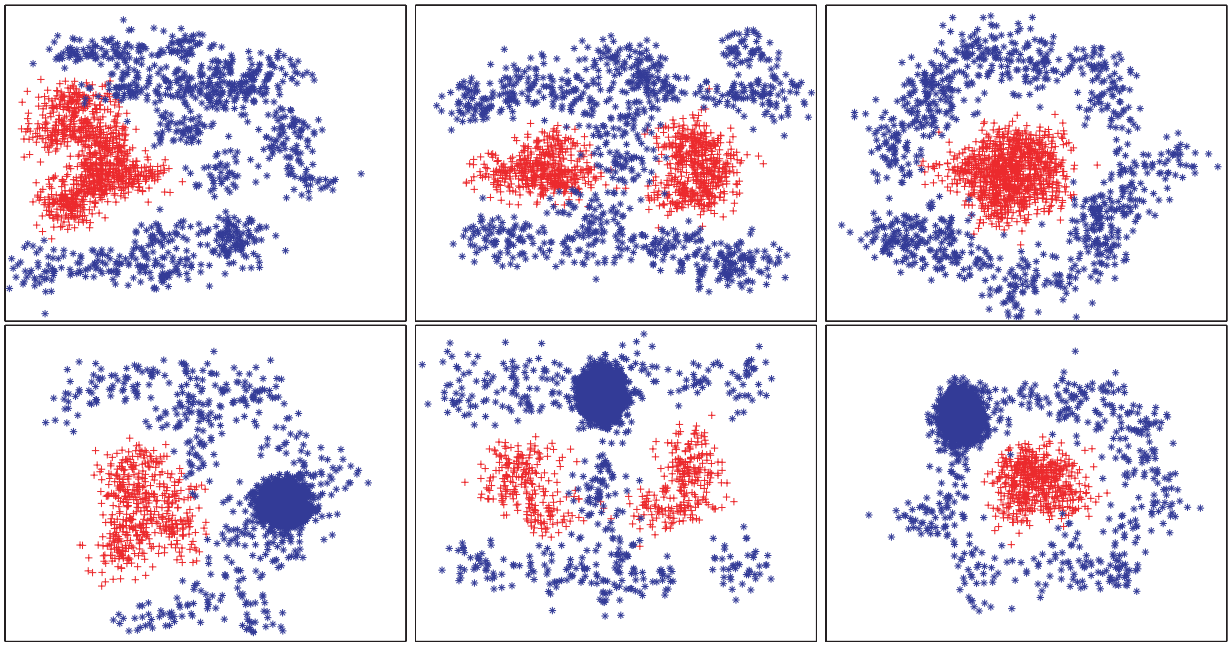

Figure 6: 2D synthetic data sets. *The bottom row shows the imbalanced data sets that have a big chunk of negative samples at a location.*

whereas the path length is fixed for all regions in Boosting.

6.2 Object detection

For training, we used the MPEG-7 face data set that has 11,845 face images [14]. BANCA face set (520 faces) and Caltech background image sets (900 images) were exploited for bootstrapping. The total number of negative 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 [13], which consists of 130 images with 507 labeled frontal faces. 507 face and 57000 random image patches were cropped and resized into 24x24 images. Example images are shown in Figure 8. The methods include a boosting classifier, Fast exit, Fast exit (two-stage cascade), Super tree and Super tree (two-stage cascade). 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 Figure 9. The super tree speeds up the boosting classifier by 3-4.3 times and the fast exit by 1.6-2.6 times. The two-stage cascade solution of 60 weak-learner super tree and 200 weak-learner fast exit outperformed 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.

Figure 10 and Figure 11 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 (2). 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

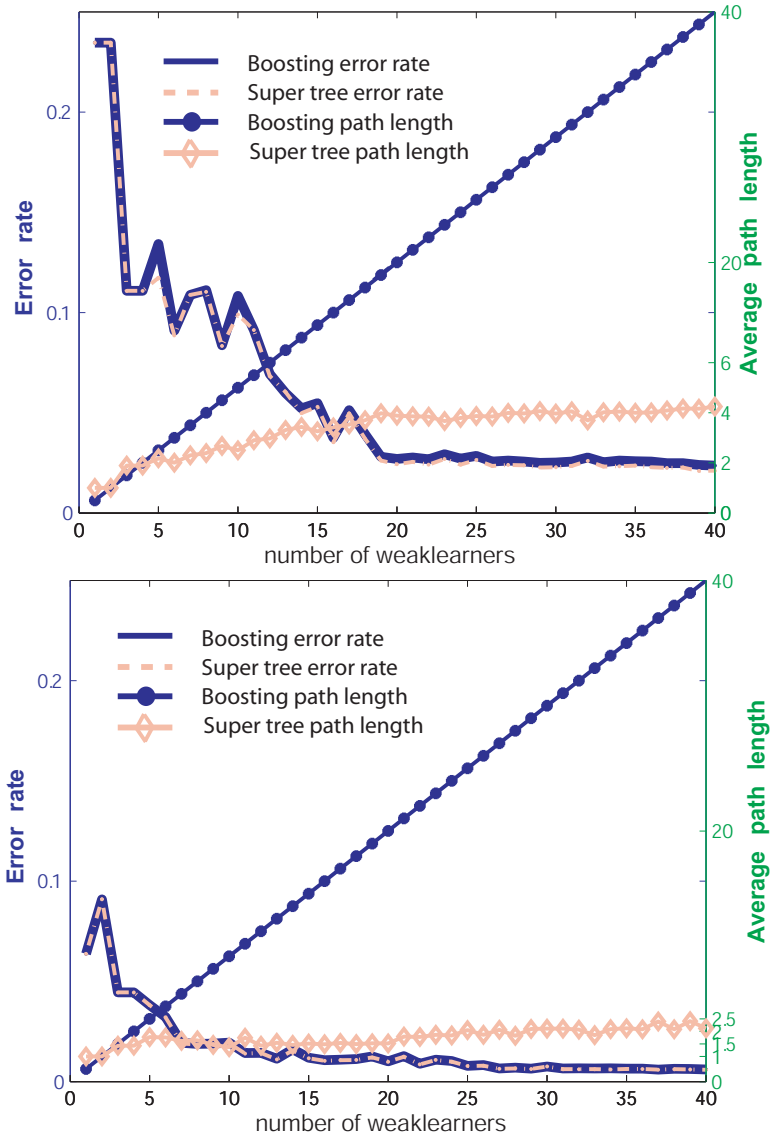

Figure 7: Experimental results on the synthetic data. *Super tree obtains the same accuracy as the booting classifier significantly shortening the average path length for the balanced data sets (left) and the imbalanced sets (right).*

increased. The exponential growth in the number of extended regions made our single tree solution limited to about 60-80 weak-learners. For about the given number of training samples, using 200 extended regions and 100 weak-learners would start hitting theoretical memory boundaries. The performance is not very sensitive for different parameter values in the range as shown in Figure 11. 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.

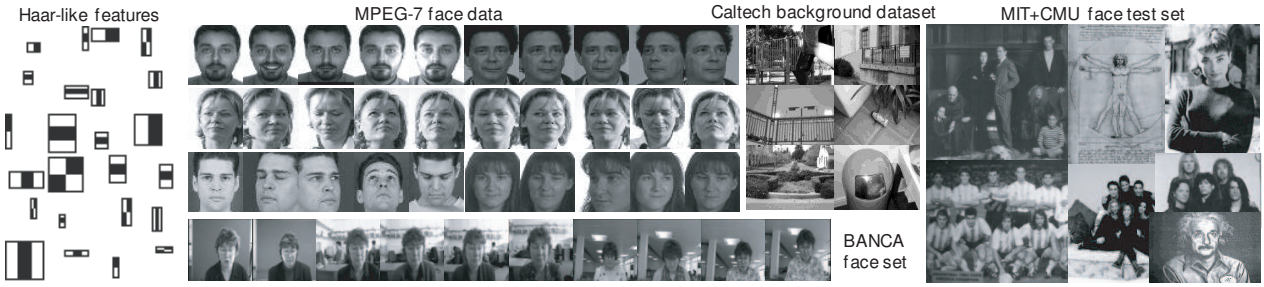

Figure 8: Example face images.

	Boosting			Fast exit			Fast exit (cascade)			Super tree			Super tree (cascade)		
No. of weak learners	False positives	False negatives	Average path length	False positives	False negatives	Average path length	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	146	69.28	144	149	37.4				145	152	15.1
200	120	143	200	120	143	146.19	146	148	38.1				128	146	15.8

Figure 9: Experimental results on the face image sets.

No. weak-learners		10	20	30	40	50	60
No. per region		1	1	2	10	40	50
False+es/ False–es	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

Figure 10: Performance of super tree for the different numbers of extended regions per region.

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

Figure 11: Performance of super tree for varying power in the information gain.

Comparison with Random Forest

The single conventional decision tree of all possible pruning [9] has been shown to be very poor. The accuracy of the single tree (false positives: 1995/false negatives: 120) is by far worse than that of the single super tree of 20 weak-learners (false positives: 476/false negatives: 122). The super tree was even shorter than the decision tree: the depth of the super tree and conventional tree was about 7.5 and 9 respectively. It has been shown that using more trees improves the accuracy of the decision tree but increases the classification

method	Super tree	Random Forest	Super tree	Random Forest
Avg path length	7.76	18.99	15.8	27.4
False +es/False -es	359/145	376/130	128/146	117/132

Figure 12: Comparison with Random forest. *Super tree is faster than RF by about 2 times at similar accuracy.*

time. We implemented Random Forest (RF) by various numbers of trees [8]. Figure 12 shows the average path length of super tree and RF by fixing the accuracy at 0 threshold. The super tree outperforms RF. It is faster than RF by 1.73-2.45 times. Constructing forest of super trees is an interesting open problem: it may improve the accuracy further taking the advantage of super tree in speed.

6.3 Rapid tracking

Super tree is an effective tracking solution for rapid moving objects. Super tree achieves a better tracking accuracy by faster classification than a boosting classifier. We have collected two sample sequences at 30 frames/second, one for training and the other for testing. The positive train 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. 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. Figure 13 shows that the super tree tracks well the fast moving object in the test sequence while the boosting tracker exhibits much drifting. The benefit of using Super tree for rapid tracking would be bigger when a higher frame rate camera is available. The execution rates for a 60 frame rate camera, based on the current speeds, are 31 frames (for boosting) and 48 frames (for super tree).

6.4 Class segmentation

We have tested the method for the segmentation by pixel classification. Video sequences collected from a camera mounted on a moving car were exploited for the experiment [15]. 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 21x21 RGB image patch as weak-learners. The train set consisted of 7143 positive and 23217 negative pixels from 184 images of 11x15 pixel resolution. Randomisation in learning (similarly to Random

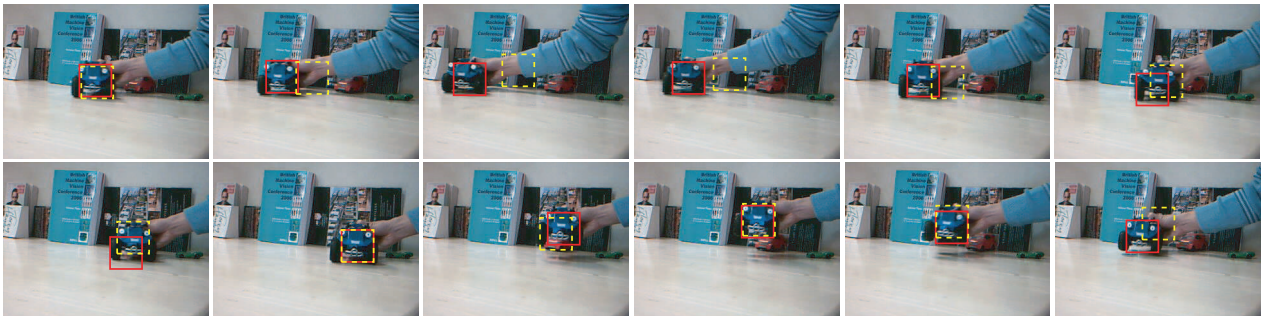

Figure 13: Performance of Boosting and Super tree trackers. *Super tree tracker (solid red line) achieves a better tracking accuracy by faster classification than Boosting tracker (dashed yellow line).*

Figure 14: Segmentation results.

forest) 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) or 0.736 (as average class accuracy). The super tree learnt by 10 extended regions per region obtained the close accuracy as 0.70 (as global accuracy) or 0.728 (as average class accuracy) using only 15 weak-learners on average. The accuracy obtained seems comparable to [15]. Figure 14 shows the segmentation results.

7 Conclusion

We have proposed a novel way to speed up a boosting classifier. The problem is formularised as boolean optimisation and a new method is proposed for boolean optimisation with a large number of binary variables, i.e. weak-learners. The tree grown from the decision regions of the boosting classifier, called Super tree, provides many short paths and preserves the decision regions of the boosting classifier. The single super tree delivers the close accuracy to the 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 cascade and Random forest. The method has been also demonstrated for rapid object tracking and segmentation problems.

Bibliography

[1] P. Viola and M. Jones, Robust real-time object detection, *Int'l J. Computer Vision*, 57(2):137–154, 2002.

[2] H. Grabner and H. Bischof, On-line boosting and vision, *Proc. IEEE Conf. CVPR*, pages 260–267, 2006.

[3] S. Avidan, SpatialBoost: Adding Spatial Reasoning to AdaBoost, *Proc. ECCV*, Graz, Austria, 2006.

[4] A. Torralba, K. P. Murphy and W. T. Freeman, Sharing visual features for multiclass and multiview object detection, *IEEE Trans. on PAMI*, 29(5):854–869, 2007.

[5] L. Mason, J. Baxter, P. Bartlett and M. Frean, Boosting algorithms as gradient descent, *Proc. Advances in Neural Information Processing Systems*, pages 512–518, 2000.

[6] S. Zhou, A binary decision tree implementation of a boosted strong classifier, *IEEE Workshop on Analysis and Modeling of Faces and Gestures*, pages 198–212, 2005.

[7] E. Grossmann, AdaTree: boosting a weak classifier into a decision tree, *IEEE Workshop on Learning in Computer Vision and Pattern Recognition*, pages 105–105, 2004.

[8] L. Breiman, Random forests, *Machine Learning*, 45:5–32, 2001.

[9] J. Quinlan, Bagging, boosting, and c4.5, *Proc. National. Conf. on Artificial Intelligence*, pages 725–730, 1996.

[10] H. Schwender, Minimization of Boolean Expressions Using Matrix Algebra, Technical report, Collaborative Research Center SFB 475, University of Dortmund, 2007.

[11] J. Chen, Application of Boolean expression minimization to learning via hierarchical generalization, *Proc. ACM symposium on Applied computing*, pages 303–307, 1994.

[12] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press and McGraw-Hill, 2001.

[13] H. Rowley, S. Baluja, and T. Kanade, Neural network-based face detection, *IEEE Trans. on PAMI*, 20:22–38, 1998.

[14] T-K. Kim, H. Kim, W. Hwang and J. Kittler, Component-based LDA Face Description for Image Retrieval and MPEG-7 Standardisation, *IVC*, 23(7):631–642, 2005.

[15] G. Brostow, J. Shotton, J. Fauqueur and R. Cipolla, Segmentation and Recognition using Structure from Motion Point Clouds, *Proc. ECCV*, Marseilles, 2008.