Journal of Classification 22:17-48 (2005)
DOI: 10.1007/s00357-005-0004-9

A Hierarchical Methodology for Class Detection Problems with Skewed Priors

Christopher K. Eveland
Equinox Corporation, Baltimore MD

Diego A. Socolinsky
Equinox Corporation, Baltimore MD

Carey E. Priebe
Johns Hopkins University, Baltimore MD

David J. Marchette
Naval Surface Warfare Center, Dahlgren VA

Abstract: We describe a novel extension to the Class-Cover-Catch-Digraph (CCCD) classifier, specifically tuned to detection problems. These are two-class classification problems where the natural priors on the classes are skewed by several orders of magnitude. The emphasis of the proposed techniques is in computationally efficient classification for real-time applications. Our principal contribution consists of two boosted classifiers built upon the CCCD structure, one in the form of a sequential decision process and the other in the form of a tree. Both of these classifiers achieve performances comparable to that of the original CCCD classifiers, but at drastically reduced computational expense. An analysis of classification performance and computational cost is performed using data from a face detection application. Comparisons are provided with Support Vector Machines (SVM) and reduced SVMs. These comparisons show that while some SVMs may achieve higher classification performance, their computational burden can be so high as to make them unusable in real-time applications. On the other hand, the proposed classifiers combine high detection performance with extremely fast classification.

Keywords: Classification; Boosting; Prototype selection; Face detection.

CKE and DAS partially suported by Defense Advanced Research Projects Agency Grant F49620-01-C-0008. CEP partially supported by Office of Naval Research Grant N00014-01-1-0011. CEP and DAS partially supported by Defense Advanced Research Projects Agency Grant F49620-01-1-0395. DJM partially supported by the Office of Naval Research In-House Laboratory Independent Research Program.

Authors' Addresses: C.K. Eveland and D.A. Socolinsky, Equinox Corporation, 207 East Redwood St., Suite 205, Baltimore, MD 21202, USA, e-mail: diego@equinoxsensors.com; C.E. Priebe, Department of Mathematical Sciences, Johns Hopkins University, Baltimore, MD 21218-2682, USA, e-mail: cep@jhu.edu; D.J. Marchette, Naval Surface Warfare Center, B10, Dahlgren, VA 22448, USA, e-mail: david.marchette@navy.mil

1. Introduction

One-class classification problems are abundant in real-world applications. They often arise in the context of *detection*, where a system is tasked with determining whether or not a given stimulus is part of a certain type. A common example of such a system comes from target detection, either from imagery or acoustic data. In this case, a continuous unlabeled stream of data is processed in order to segment portions of it that are consistent with a predetermined definition of what a target is. Targets can be vehicles or people in the military context, abnormal tissue in the medical context, human speech in the acoustic context, etc. A common feature of these target detection applications is that the relative frequency of occurrence of targets is overwhelmingly lower than that of non-targets. For example, when searching for an armored vehicle in a radar image, most of the scene is occupied by things other than the desired target, such as trees, buildings, ground, etc. We may think of this as a classification problem with two classes, one of which is very common while the other is exceedingly rare. Alternatively, we may approach this as the problem of modeling the support of a single distribution (that of the target class).

Regardless of the approach we choose for this problem, it is important to understand that the relative abundance of the target and non-target classes in the real world must be taken into consideration when evaluating classifier performance. Relative priors for the two classes in typical detection problems can be off by six or more orders of magnitude. Under these circumstances, and considering that we often have a limited training and evaluation set of data, a naive performance evaluation will indicate that simply classifying every observation as non-target is the optimal strategy. Proper penalty terms on the type-I and type-II errors are required in order to obtain meaningful results.

A number of techniques for modeling the support of a distribution can be found in the literature (Duda and Hart 1973; Duda, Hart and Stork 2000; Jain, Duin and Mao 2000; Hastie, Tibshirani, and Friedman 2001; McLachlan and Peel 2000; Ripley 1996; Scott 1992; Silverman 1986). All of these degrade sharply as the dimensionality of the data increases, and most of them are not applicable for relatively high dimensions, say above ten. We are particularly interested in detection problems, where the data dimensionality is routinely on the order of several hundred. Additionally, in such problems the volume of data to be processed is normally very large, therefore the run-time speed of the classifier/detector is important. Our goal in this paper is to demonstrate a set of related strategies for approaching the one-class (detection) problem with low computational cost. Emphasis will be on obtaining the highest possible operational speed, and at every point choices will be made that favor speed or simplicity instead of performance or theoretical correctness. These choices

will, of course, be justified at the very least heuristically. The reader should keep in mind, however, that the principle of computational efficiency is a major driving force for the techniques proposed herein.

Throughout the paper we will use, as an example, the problem of detecting human faces in grayscale images. The data, in this case, will consist of 21×21 pixel image chips, interpreted as 441-dimensional vectors by raster scanning. The target class consists of those vectors corresponding to a face centered in the image chip with eyes at specified locations. The non-target class consists of all other possible 441-dimensional vectors, with the distribution of naturally occurring non-face images. This is a well studied problem, which exhibits the key aspects of the skewed-priors one-class problem in high dimensions. In addition, a successful face detector must often operate in real-time on live video streams, so the role of computational complexity is very relevant. In our example application, as in most other detection problems, we have a limited amount of training/testing data for the target class and an essentially unlimited amount of training/testing data for non-targets. This is a typical scenario, since data from the target class usually must be manually processed prior to use, and is cumbersome or expensive to acquire. Non-target data, on the other hand, is plentiful and normally easily collected. Specific details of the application of our methods to face detection are reported in Socolinsky, Neuheisel, Priebe, Marchette, and DeVinney (2003). In the current article, we concentrate on pattern recognition aspects of the problem, and use the face detection dataset as a unifying thread.

The present work builds upon our previous research, specifically leveraging the Class Cover Catch Digraph classifier, developed in Priebe, Marchette, DeVinney, and Socolinsky (2003); DeVinney, Priebe, Marchette, and Socolinsky (2002); Socolinsky et al. (2003). Much of what follows can be adapted to other classifiers, and we attempt to make a note of that whenever relevant. An extension of this work using support vector machines is under way, and will be reported on in a later publication. A number of of authors have proposed related methods in various contexts (Viola and Jones 2001a,b; Fleuret and Geman 2001).

This paper is organized as follows: In Section 2 we describe the Class Cover Catch Digraph classifier. In Section 3 we develop a structure for boosting the CCCD classifier, which is then optimized for data with unequal priors in Section 4. A data-adaptive technique is explored in Section 5 to further improve performance. Finally, in Sections 6 and 7 we present results and conclusions.

2. The Class Cover Catch Digraph Classifier

This section provides a brief introduction to the Class-Cover Catch Digraph (CCCD) classifier. A more detailed description and analysis can be found in Priebe et al. (2003). In particular, this work uses a derivative of the random-walk CCCD classifier introduced in DeVinney et al. (2002).

The CCCD classifier is a nearest-prototype classifier with respect to a non-linear dissimilarity function. For simplicity, consider as training data two sets of class-conditional $\mathbb{R}^n$-valued observations $\mathcal{X}_0$ and $\mathcal{X}_1$, and a dissimilarity measure $\rho : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$, satisfying $0 = \rho(x,x) < \rho(x,y) < \infty$, for $x \neq y \in \mathbb{R}^n$. The goal of classifier design is to construct a function $g_{\mathcal{X}_0,\mathcal{X}_1} : \mathbb{R}^n \to \{0,1\}$ such that for a given unlabeled observation $x \in \mathbb{R}^n$ with unknown class label in $y \in \{0,1\}$, the probability of misclassification $P\left[g_{\mathcal{X}_0,\mathcal{X}_1}(x) \neq y\right]$ is close to Bayes optimal (Fukunaga 1990; Kulkarni, Lugosi, and Venkatesh 1998).

For a set of prototypes $C_i = \{c_{i,1}, \ldots, c_{i,k}\} \subset \mathcal{X}_i$ and scaling factors $R_i = \{r_1, \ldots, r_k\} \subset \mathbb{R}_+$, $i = 0, 1$, the CCCD *cover-dissimilarity measure* is defined by

$$d(x, C_i) = \min_k \frac{\rho(x, c_{i,k})}{r_k}. \tag{1}$$

Given sets C_i, and R_i as above, the CCCD classifier is defined in terms of the cover-dissimilarity measure (1) by

$$g_{\mathcal{X}_0,\mathcal{X}_1}(x) = \arg\min_i d(x, C_i). \tag{2}$$

The choice of prototypes C_i and scaling factors R_i, determines the classifier map. Below, we give a short account of the method for choosing these parameters. A more thorough account can be found in Priebe et al. (2003); DeVinney et al. (2002); Socolinsky et al. (2003), along with performance analyses for applications other than face detection.

For each point $x_{i,j} \in \mathcal{X}_i$, we consider the random walk defined as follows

$$R_{x_{i,j}}(r) = |\{x \in \mathcal{X}_i : \rho(x_{i,j}, x) < r\}| - |\{x \in \mathcal{X}_{(1-i)} : \rho(x_{i,j}, x) < r\}|, \tag{3}$$

for $r \in \mathbb{R}_+$, $i = 0, 1$. This random walk can be thought of as a ball growing around the point $x_{i,j}$. As it reaches each observation the walk takes a step either up or down, depending on the class of the observation. This is illustrated in Figure 1, where the random walk is depicted as the ball grows. The horizontal steps correspond to the distance taken to reach the next observation. A large value of $R_{x_{i,j}}$ indicates a high local density of same-class points around $x_{i,j}$, relative to the local density of other-class points (see DeVinney et al. (2002) for the case of unequal training priors). In fact, the value of the random walk at

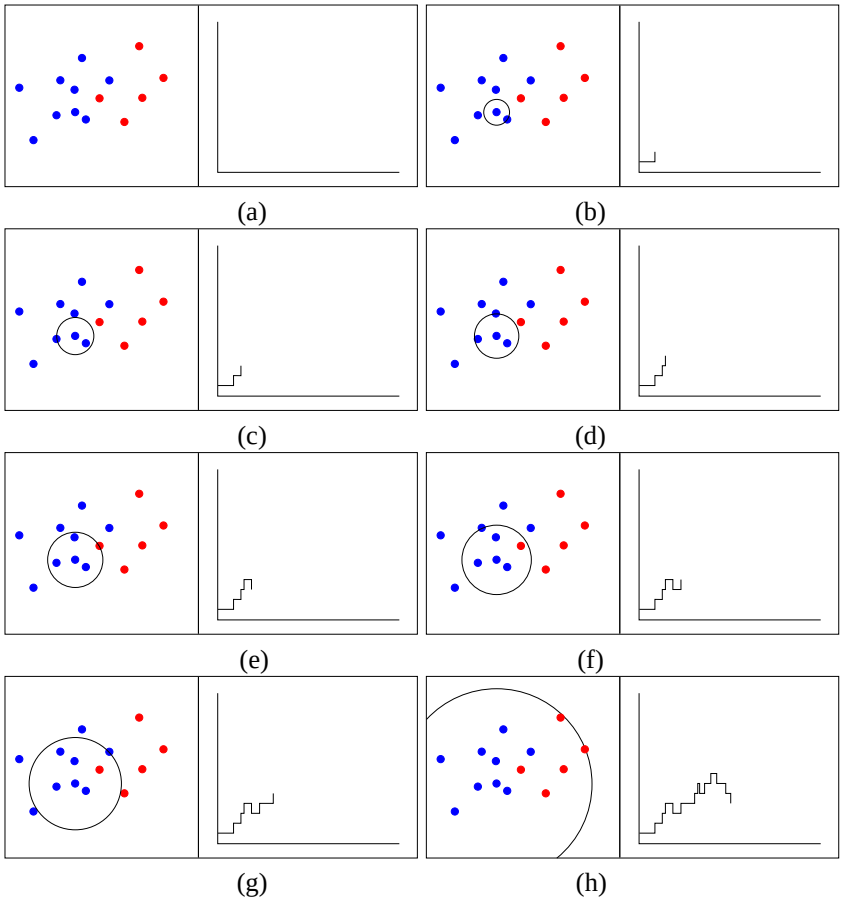

Figure 1. Example of random walk construction for a two-dimensional problem.

any given $r \in \mathbb{R}_+$ can be taken as a measure of relative deviation between the local densities of the training data. A Kolmogorov-Smirnov type test is applied in DeVinney et al. (2002) to obtain a distinguished choice of r for each training observation, given by

$$r^*_{x_{i,j}} = \arg\max_r R_{x_{i,j}}(r) - P(r), \tag{4}$$

where $P(r)$ is an increasing penalty function that biases the choice toward smaller values of r. This is done in order to encourage more local estimation of the classifier parameters, and is achieved in practice by the use of a linear penalty function.

Once a distinguished scaling factor $r^*_{x_{i,j}}$ has been chosen for each training observation, it remains to find the choice of class-conditional prototypes

C_i that will fully determine the CCCD classifier. The procedure in the standard CCCD classifier is somewhat different than the one used in this paper (described in detail below), but we include a summary of it for completeness. Ideally, the choice of prototypes would be that which maximizes classifier performance. However, the combinatorial explosion involved in checking all possible prototype sets precludes this direct approach. The choice of prototypes for each class proceeds in a greedy fashion, using a surrogate criterion for the classifier performance

$$T_{x_{i,j}} = R_{x_{i,j}}(r^*_{x_{i,j}}) - P(r^*_{x_{i,j}}). \tag{5}$$

For a given class, the first prototype $c_{i,1}$ is chosen to be that with the highest value of T. All training observations x for which $\rho(x, c_{i,1}) < r^*_{c_{i,1}}$ are then deleted from the training set, all scaling factors r^* are recomputed for the remaining training observations, and the next prototype is chosen using the surrogate criterion T as before. This process continues until all but a predetermined proportion of the class-i training data has been deleted.

3. A Sequentially Boosted CCCD Classifier

Although some degree of boosting is inherent to the CCCD classifier (by means of censoring the training data in the greedy prototype selection process), increased performance can be achieved by more explicit boosting during training. Additionally, it is necessary to incorporate the fact that in our skewed-priors scenario, the relative likelihood of observing a non-target (class 1) is several orders of magnitude larger than that of observing a target (class 0). Therefore, classifier design should be geared toward rejecting the background class, both for accuracy and performance reasons (Fleuret and Geman 2001). In our case, we will achieve this by boosting on only the background class and structuring the classifier as a sequential testing procedure.

3.1 Structure of the Class Cover

By virtue of the nature of the surrogate criterion (5), the CCCD classifier (2) for a set of prototypes $C_i = \{c_{i,1}, \ldots, c_{i,k}\} \subset \mathcal{X}_i$ and scaling factors $R_i = \{r_1, \ldots, r_k\} \subset \mathbb{R}_+$ is dominated by the influence of the first few elements of each set. In particular, the first prototype and scaling factor are the most important in determining the classification map. Figure 2 shows the number of training observations in each of the balls corresponding to succesively chosen prototypes for a typical two-class CCCD classifier. Clearly, most of the training data is represented by the first few prototypes in each class.

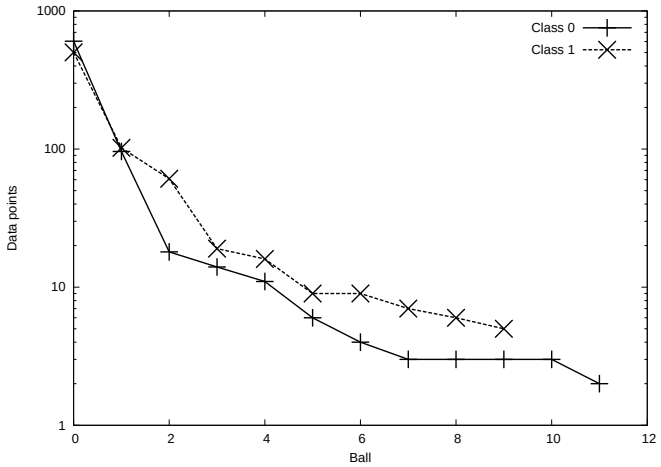

Figure 2. Number of data points per ball for a typical two-class CCCD classifier.

3.2 Biasing Classifier Performance

It is often necessary to bias the performance of a classifier to meet required tolerances on type-I and type-II errors. This is often done, in the Neyman-Pearson spirit, by varying a threshold. This is a quick and easy method, and we will take advantage of it below. We would like to point out that the appropriate way to bias performance for a CCCD classifier is to alter the random walk (3). We see that in (3), equal absolute value is assigned to observations of either class; that is, the penalty for covering a training observation of the opposite class is equal in magnitude as the reward for covering a like-class observation. Simply rewriting (3) as

$$R_{x_{i,j}}(r) = \alpha|\{x \in \mathcal{X}_i : \rho(x_{i,j}, x) < r, \}| - |\{x \in \mathcal{X}_{(1-i)} : \rho(x_{i,j}, x) < r, \}|, \tag{6}$$

with $\alpha > 0$ we obtain a classifier with a variable ratio of type-I to type-II error.

Note that the previous procedure requires that we re-train a classifier each time we wish to change the bias, which can be a time-consuming procedure. Since we desire the ability to easily change the bias, we forego the previous construction in favor of a simpler procedure. We easily bias the classification performance of any CCCD classifier in favor of lower type-I or type-II error by modifying the scaling factors as

$$\tilde{R_0} = t\, R_0, \qquad \tilde{R_1} = t^{-1}\, R_1, \tag{7}$$

for $0 < t < \infty$. Values of t in $(0, 1)$ favor lower class-1 error at the expense of higher error rate on class-0, and vice-versa. Geometrically, the value of t has the effect of shrinking or growing the estimated support of each class.

3.3 Sequentially Boosted Classifier

A sequentially boosted CCCD classifier is essentially a chain of CCCD classifiers trained in a specific fashion, detailed below, and applied in order to an unlabeled observation. The process is best explained by first considering the detection stage. In the original CCCD classifier, classification of an unlabeled observation is based on the class label of the nearest set of prototypes, where this is defined as the distance to the nearest prototype in each set. For simplicity, let us assume that the target class has a geometrically simple support, and therefore can be represented by a small number of prototypes, whereas the non-target class is more complex, and thus necessitates more prototypes. Computing the distance from the unlabeled observation to the target class prototype set is computationally cheap, since there are few prototypes. Unless implemented on a parallel computing architecture, computation of the distance from the unlabeled observation to the non-target prototypes proceeds sequentially and in order. The first simple observation is that if for any non-target prototype the distance is lower than the minimum distance to the target prototypes, then we need not compute the remaining distances, as the classifier output will be non-target, regardless of those distances. Therefore, the classification process can be shortened by bypassing the remaining computations. The second remark is equally pedestrian but has farther-reaching implications. We simply note that if an observation is closer to the first non-target prototype than to any of the target prototypes, then that changes our prior on the distribution of that observation. That is, we know more about the type of observation we are dealing with as we sequentially compute its distance to consecutive non-target prototypes.

The first remark above leads to a sequential testing structure for the ultimate classifier, where comparison with successive non-target prototypes must be favorable in order for the testing process to continue. If at any time, we find that the unlabeled observation is closer to a non-target prototype than to all target prototypes, we simply exit the decision process. In practice, this has the advantage of quickly classifying all “easy” observations. We should note that in real-world detection applications, the vast majority of observations are easy. This is simply because the hard observations are those which lie near the discriminant boundary between the two classes and for most problems this is an almost everywhere regular submanifold of space with codimension one. Therefore, the majority of possible non-target observations lie far away from this boundary and can be correctly classified with ease.

The second remark leads us to the boosted nature of the proposed classifiers. In the standard CCCD classifier (DeVinney et al. 2002), each class cover is computed separately using the greedy dominating set procedure in Section 2. This is done in the interest of training speed, since the dominating set problem is exponential in the size of the graph. The most time consuming step in the greedy algorithm is the computation of distances between training observations needed to construct the catch digraph, which is quadratic in the number of training observations. In the sequentially boosted classifier, we explicitly use the fact that unlabeled observations that have passed n tests against non-target prototypes are known to be drawn from a different distribution than those that have passed only $n-1$ such tests. We do this by filtering the training data with partially trained classifiers, as described below.

3.4 Training a Sequentially Boosted Classifier

Training of a sequentially boosted CCCD classifier is separated into *stages* and *sub-stages*. In what follows, we associate prototypes and their corresponding scaling factors, and we refer to them simply as prototypes. A stage is characterized by a fixed set of class-0 prototypes, while a sub-stage corresponds to a single class-1 prototype (see Figure 3). The first prototype in each class is selected by the same procedure as in Section 2 (if more class-0 prototypes are desired, they are all chosen in this step, but we describe the process for a 1-prototype stage, for simplicity). At this point, we have a simple CCCD classifier, with one prototype per class. Using the biasing procedure of Equation (7) and a test set of class-0 observations, we find the lowest value of t (or a suitable approximation) that yields an empirical error rate below a predetermined fixed tolerance. The two prototypes along with the resulting scaling factors constitute the first sub-stage of the sequential CCCD stage. In order to compute the second sub-stage, we apply the first sub-stage classifier (see Algorithm 2) to a set of class-1 data and collect the misclassified observations, which become the class-1 training observations for the second sub-stage. Using once again the procedure in Section 2, a single class-1 prototype is selected, and through the bias procedure in (7), the scaling factors for the single (fixed) class-0 prototype and the newly chosen class-1 prototype are computed. This process is repeated as many times as necessary to obtain the desired number of sub-stages. In Figure 4 we see an example of how the training data is filtered by this process for the face-detection example. On the left we see a uniformly sampled selection of the non-face training data, while on the right we have a sample of the non-face data that remains after training one stage of a boosted classifier. Note how the images on the right hand side have more face-like structure, such as dark circles located in the eye region, or horizontal striations that resemble eyebrows and

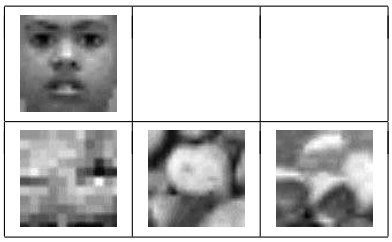

Figure 3. Schematic representation of a sample CCCD tree stage. In this case, there is a single target-class prototype and three sub-stages identified by distinct non-target prototypes.

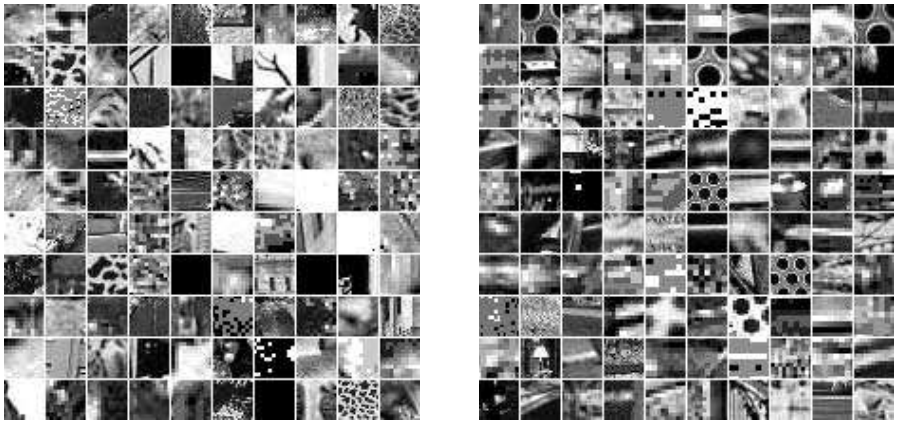

Figure 4. Sample non-face training data before and after the first full boosting stage.

mouths.

The number of sub-stages within a given stage is empirically determined; we continue to add sub-stages to a stage as long as the correct classification rate on the non-target class increases by a significant (predetermined) percentage over that of the previous sub-stage. At that point, we start training an entirely new classifier stage, using the original class-0 training data, and the class-1 training data that is misclassified by all previous classifier stages. While speed considerations may dictate (see Socolinsky et al. (2003)) that the first stage of the classifier have a single class-0 prototype, subsequent stages are allowed to have multiple such prototypes. In fact, it is natural to allow later stages to use more target-class prototypes than earlier ones, thus allowing the classifier to more closely model the effective support of the distribution. Rather than hand-select the number of target-class prototypes used for a given stage, we use the following simple method. We pick the first k prototypes for the target class, where k is the maximum desired number of prototypes. Then using the first

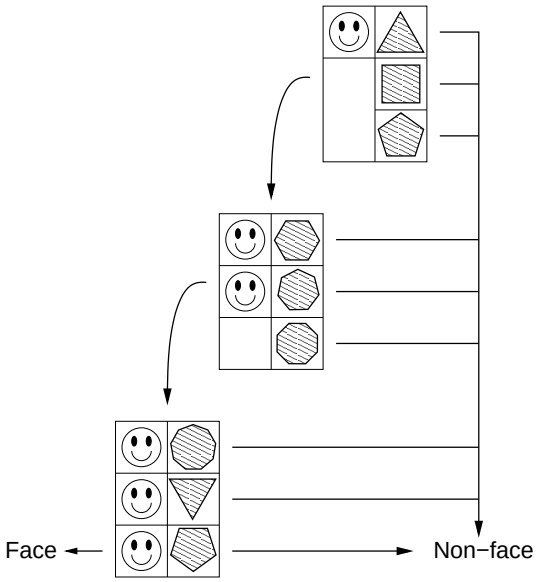

Figure 5. Sequential CCCD structure with three stages of three sub-stages each. Target prototypes are smiley faces and non-target prototypes are represented by polygonal shapes. This is a graphical representation of Algorithm 2.

sub-stage non-target prototype, we evaluate the emprirical performance of the k classifiers using 1 through k target prototypes, and select the one with highest accuracy. The resulting structure is illustrated in Figure 5, where we see a classifier with three stages, each of which contains three sub-stages. Algorithm 1 shows the steps in the boosted training algorithm. Here, the indexes i and j correspond to stages and sub-stages, respectively. Each stage i is defined by its f_i target-class prototypes $C_0^i = \{c_{0,1}^i, \ldots, c_{0,f_i}^i\}$ and scaling factors R_0^i, as well as the n_i non-target prototypes and scaling factors $C_1^i = \{c_{1,1}^i, \ldots, c_{1,n_i}^i\}$ and R_1^i corresponding to each sub-stage. A large (on the order of a million or more) set of non-target observations $\mathcal{T}_1^0$ is used for the boosting process, and a separate set $\mathcal{T}_1$ of target-class samples is used to evaluate the empirical classifier performance on the target class.

Figure 6 illustrates the result of sequentially boosted training for a simple problem in two-dimensions. The target class is supported on the union of two overlapping squares in the unit square, with the density being uniform with equal intensity on both components of their symmetric difference, and uniform with twice the intensity on their intersection. The background class is distributed uniformly on the unit square. One-thousand training observations of each class are used. Each row of Figure 6 corresponds to a stage of a sequentially

Algorithm 1: Boosted training algorithm.

Let $\mathcal{T}_0$ and $\mathcal{T}_1^0$ be $i.i.d.$ sets of class-0 and class-1 data respectively.
Let β be the required number of stages.
Let α be the threshold on the incremental classification

$j \leftarrow k \leftarrow 1$; $\mathcal{X}_1^0 \leftarrow \mathcal{T}_1^0$
for $i \leftarrow 1 \ldots \beta$ **do**
  Select $C_0^i = \{c_{0,1}^i, \ldots, c_{0,f_i}^i\}$ and R_0^i as in Section 2
  repeat
    Select $c_{1,j}^i$ and $r_{1,j}^i$ as in Section 2, using $\mathcal{X}_1^{k-1}$ as training data
    Adjust the scaling factors as in Equation 7 to enforce the required class-0 empirical performance bound
    Let $\mathcal{T}_1^k \subseteq \mathcal{T}_1^0$ be the class-1 observations incorrectly classified by the current classifier;
    Let $\mathcal{X}_1^k$ be the misclassified class-1 training observations
    $k \leftarrow k + 1$
  until $|\mathcal{T}_1^k|/|\mathcal{T}_1^{k-1}| \geq \alpha$
end for

Algorithm 2: Sequentially boosted classification.

Let x be the unlabeled observation.
Let β be the number of stages.
Let $C_j^i = \{c_{j,1}^i, \ldots, c_{j,k_j^i}^i\}$ for all $j \in \{0,1\}$, $i \in \{1, \ldots, \beta\}$ be the sets of target ($j = 0$) and non-target ($j = 1$) class prototypes for stage i.
Let $R_j^i = \{r_{j,1}^i, \ldots, r_{j,k_j^i}^i\}$ be the scaling factors for C_j^i.

for $i \leftarrow 1, \ldots, \beta$ **do**
  $m_0 \leftarrow \min_{k=1}^{k_0^i} \rho(x, c_{0,k}^i)$
  for $j \leftarrow 1, \ldots, k_1^i$ **do**
    if $\rho(x, c_{1,j}^i) < m_0$ **then**
      Classify as non-target and **exit**
    end if
  end for
end for
Classify as target

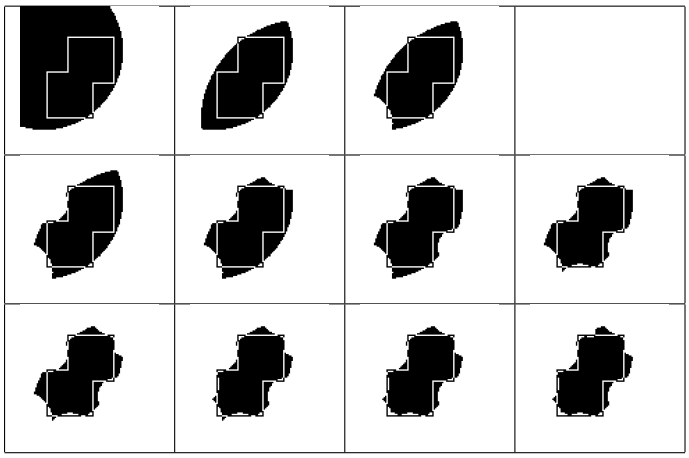

Figure 6. Sequential refinement of the classifier through stages and sub-stages. Each row corresponds to a stage and each column to a sub-stage within it. The region inside the lines is the true support of the target class. The three stages have one, three and six target-class prototypes, respectively. The first stage has three sub-stages, while the later ones have four each.

boosted CCCD classifier, and each column represents a sub-stage within each stage, so the figures should be read from left to right and top to bottom. Each figure shows the estimate of the effective support for the target class corresponding to the respective stage and substage. The three stages have one, three and six target-class prototypes, respectively, and the allowed error for each substage is 0.5% (thus the total error on the target class is bounded by 5.5%). Note that the first stage has only three sub-stages, while stages two and three have four sub-stages each. We see how easy non-target observations can be correctly classified using only a small portion of the classifier, and those rare ones near the discriminant boundary must be processed by later stages or sub-stages.

Figure 7 shows classifier performance on each class as a function of the total sub-stage count, for a 42 sub-stage sequentially boosted classifier. This was evaluated using a set of several thousand faces and non-faces, disjoint from the training set. Note how the performance on the face class decreases as the number of sub-stages increases,[1] while the error rate on the background class decreases. A straight sum of the error rates is not a good criterion of performance, however, as the priors are severely skewed toward the background class. Hence, even though it would appear that the optimal classifier in this case has around 10 sub-stages, the full cascade indeed has a better detection to false alarm ratio.

1. The error rate on the face class is bounded above by the number of sub-stages times the error bound on each individual sub-stage.

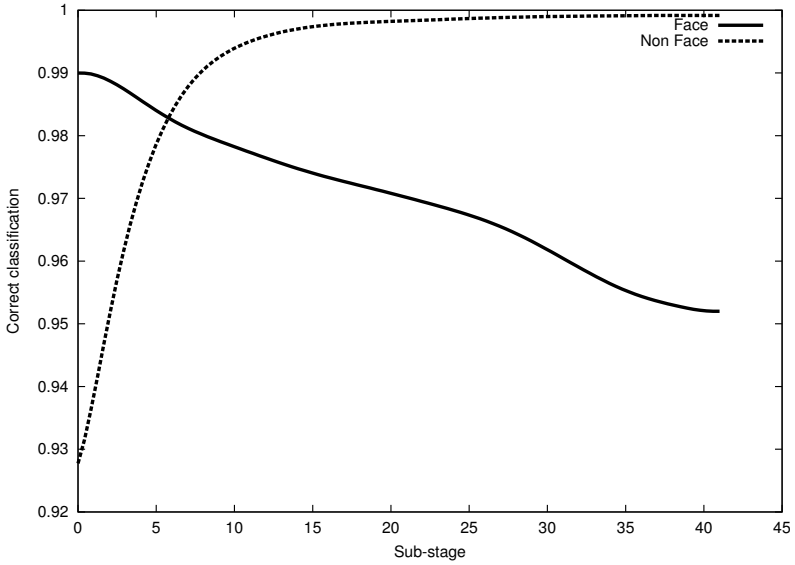

Figure 7. Sequentially boosted performance as a function of total sub-stage count, for a 42 sub-stage sequentially boosted classifier.

3.5 Fast Training of a Sequential Classifier

The main emphasis of our work lies in lowering the classifier error rate and speeding up its application at time of detection. Training time is not a major factor, since training occurs off-line and once completed does not need to occur again; that is, once a classifier has been trained, it can be used indefinitely without modification. However, if training times are prohibitively long, then it is not possible to obtain the desired classifier. In our case, the large number of training observations used, especially for the non-target class, would make standard CCCD training as in Priebe et al. (2003) and DeVinney et al. (2002) all but impossible, since the algorithm there requires the computation of all distances between training observations. Even though it may be clear to the reader, we should remark that a very large number of non-target class observations are normally required to train a sequentially boosted CCCD classifier because of the filtering inherent to the training process. In order to train a given sub-stage, we must collect a sufficiently large number of non-target observations that have been incorrectly classified by all previous stages and sub-stages. This becomes increasingly hard for later stages, as the partial classifier is normally fairly good early in the training process. As a concrete example, when training a face detector (Socolinsky et al. 2003) we have used tens of millions of non-face examples. Note that this does not mean collecting tens of millions of images, but rather using that many subwindows of a few hundred images manually determined to

contain no faces.

In order to avoid the quadratic growth problem, we apply the following stochastic search strategy. At each sub-stage of training, random subsamples from the class-0 and class-1 training data are drawn, consisting of a few hundred observations each. These subsets are used to train the sub-stage as above, and the empirical performance of the resulting stage on class-1 data is evaluated on the class-1 data not used for training (the complement of the random sample). Recall that the performance on class-0 data is enforced explicitly, so it is not necessary to evaluate it. This process is repeated multiple times for new random samples of the training data, and the sub-stage yielding highest empirical performance is used in the final classifier. We can think of this as a degenerate form of bagging (Breiman 1998).

Experimentally, we have observed that if the random training subsets are not too small, this procedure yields a classifier whose performance is indistinguishable from that of one trained on the full set of training data, but requiring only a small fraction of the computation time. We normally use on the order of 50 random iterations of the above procedure for each sub-stage of the classifier tree.

4. Biasing Performance for Optimal Sequential Testing

Given a training dataset, the CCCD training process in Section 2 seeks to find the classifier minimizing the empirical error rate on that set. This is not the desired outcome if the relative priors in the training set do not reflect those in the real world. For a detection problem, where the real-world priors differ by many orders of magnitude, it is not feasible to work with a representative training set, so we resort to biasing the classifier performance, as in Section 3.2. Likewise, when our classifier is to be used within a sequential testing process, it becomes necessary to bias its performance. In this case, it is important that the error rate on the target class for each test in the sequence be much lower than that on the non-target class. It is easy to see that if for a true target observation a sub-stage of our classifier makes a mistake, the final classification result will be incorrect. On the other hand, if a mistake is made for a true non-target observation, we simply incur the cost of (at least) another test. Hence, errors on the target class are unrecoverable, whereas those on the non-target class are potentially recoverable at the price of additional computation.

It follows from the previous remarks that it is a valid strategy to seek, for any given sub-stage, the highest performance on the non-target class for a fixed maximum error rate on the target class. Note that given the structure of our sequential testing procedure, the overall error (of all tests in sequence) on the target class is bounded above by the sum of the target-class errors of the individual tests. We do not have a comparable bound on the non-target class

error, and in any case such a bound would not be of much use, since we have no control over that error during training. It follows that if we seek to train a sequentially boosted CCCD classifier with a given maximum allowable error on the target class, we must simply control the individual sub-stage empirical errors so that their sum does not exceed the allowed error. This recipe leaves us with a great deal of freedom as to how to distribute the per-sub-stage errors, or equivalently as to the choice of the corresponding biasing factors. When training the boosted classifier, the choice of biasing factor is of critical importance. Not only will it affect the detection rate of the classifier, but it will affect the computational cost as well, especially in the earlier stages. We show below how different choices of per-substage-error with equal overall error result in classifiers with substantially different run-time speeds.

4.1 Varying the Biasing Factor on a Per-Sub-Stage Basis

For this discussion, let $\varepsilon_i(k)$ be the class-i error for sub-stage k only, where the error rate is only evaluated for class-i samples for which all sub-stages from 1 to $k-1$ assign the class-0 label.[2] Let $\varepsilon_i^*(k)$ be the cumulative class-i error for sub-stages 1 through k. Furthermore, let $\bar{X}$ denote the set complement. Based on the algorithm for boosted classification 2, the cumulative error rates are as follows:

$$\varepsilon_0^*(k) = \begin{cases} 0 & \text{when } k = 0, \\ \varepsilon_0(k)\bar{\varepsilon}_0^*(k-1) + \varepsilon_0^*(k-1) & \text{otherwise.} \end{cases} \tag{8}$$

$$\varepsilon_1^*(k) = \begin{cases} 1 & \text{when } k = 0, \\ \varepsilon_1(k)\varepsilon_1^*(k-1) & \text{otherwise.} \end{cases} \tag{9}$$

To analyze the effects of varying the biasing factor, let us first consider the simple case when $\varepsilon_0(k)$ is constant, so $\varepsilon_0(k) = \varepsilon_0(l)$, for all $k, l \geq 1$. This can be achieved by adjusting the biasing factor at each sub-stage, and allowing the class-1 error to vary from sub-stage to sub-stage. In this case it is simple to compute ε_0^* as follows:

$$\varepsilon_0^*(k) = \varepsilon_0(k) \sum_{i=0}^{k-1} \bar{\varepsilon}_0(k)^i \leq k\, \varepsilon_0(k) \tag{10}$$

On the other hand, $\varepsilon_1^*(k)$ will in general depend on the data, according to the receiver operating characteristic (ROC) curves for the individual sub-stages. See Figures 8 and 9.

---

2. Recall that if any sub-stage prior to k had assigned a class-1 label to the observation, then it would never have been evaluated by sub-stage k.

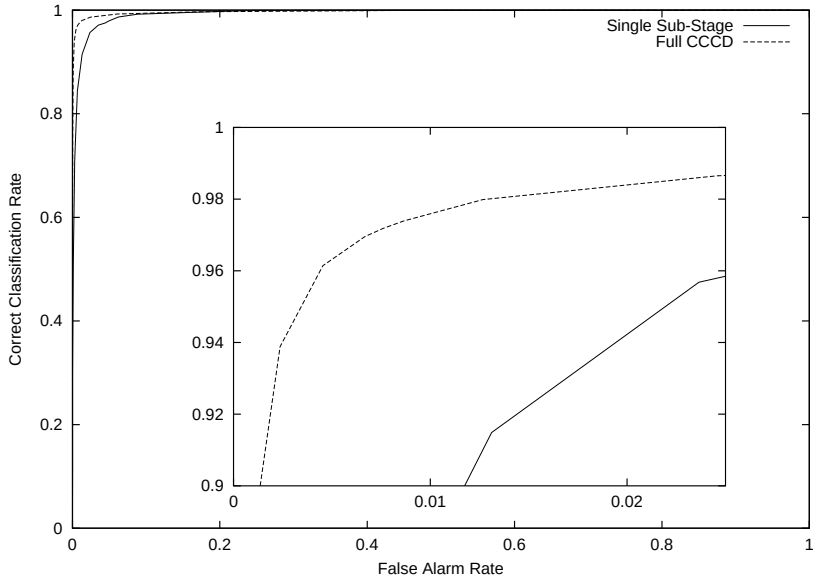

Figure 8. An ROC curve for a typical sub-stage as compared to a standard full CCCD classifier. While it is a weaker classifier, it is faster to evaluate, and can be used with sequential testing to achieve high performance. Inset shows a magnification.

Now assume that since the prior on the non-target class is much larger than that on the target class, there is a fixed acceptable target error rate, and that the goal of training is to get the best possible non-target rejection rate for that amount of target-class error. The simplest way to achieve this is to take the total allowed error, divide that by a constant number, and limit each sub-stage error to that. If the total error is small, then each step up in error will be nearly the same size according to Equation (10). When we compute empirical ROC curves for a sequentially boosted CCCD classifier trained in this fashion, we see that each sub-stage corresponds to a roughly constant step down (increasing error on the target class), and an incrementally decreasing leftward motion (improving error on the non-target class), as is shown as the solid line in Figure 8.

The expected computational cost of evaluating a non-target sample up to no more than k sub-stages is proportional to the average number sub-stages that need to be evaluated,

$$U_1(k) \propto \sum_{i=1}^{k} \varepsilon_k^*(k). \tag{11}$$

Since U_1 depends on the training and testing data, it is hard to estimate ahead of time, but there are several strategies for reducing it.

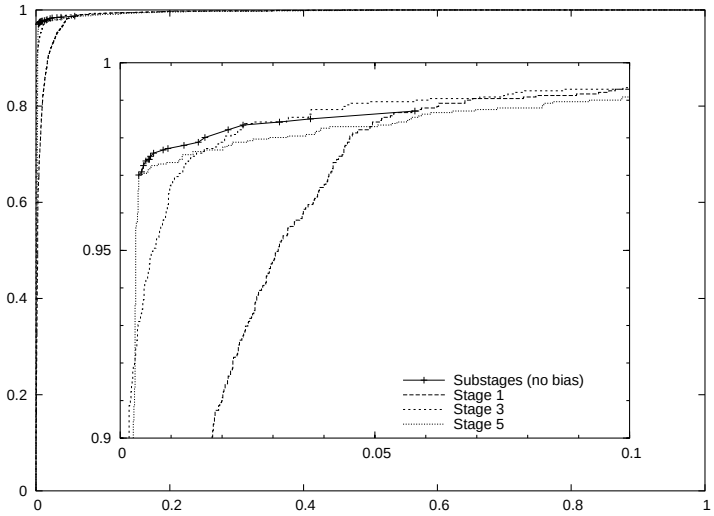

Figure 9. ROC curves for the first, third, and fifth stages in a boosted CCCD classifier. The ROC curves are generated by adjusting the bias between the nearest face and non-face prototypes explored. Also shown is an ROC curve generated by varying the number of sub-stages used with no bias. Inset shows a magnification.

Recall that after a sample has been classified as non-target, computation stops, thus reaching a high rejection rate early in the sequential process will result in a lower average number of sub-stages requiring evaluation. Since a higher rejection rate on the non-target class will necessitate a higher error rate on the target class, it is not possible to have many large reductions in false alarms before exceeding the limit on correct classification. It is the early stages that have the greatest effect on classification speed, so after initial stages of high non-target rejection rates and high target error rates, we can switch to biasing factors that yield lower target-class error for the remaining stages. See Figure 9.

Figure 10 shows three strategies for varying biasing factors that yield the same final correct classification rate of 98% on the target class, but rapidly achieve most of the non-target rejection in the first few sub-stages. The first method is to allow a constant maximum error of 0.2%. The second scheme is to allow a large error in the first step, and then a constant smaller error thereafter. This scheme is labeled “Stepped” in Figure 10, and uses a 1% initial step, followed by a 0.1% limit, as compared to the 0.2% limit in the constant case. Both the “constant” and “step” strategies take 10 steps to reach 98% correct target classification. The third scheme is to take an exponential decay rate of $1/2$ the target error each step to converge on a final rate of 98%.

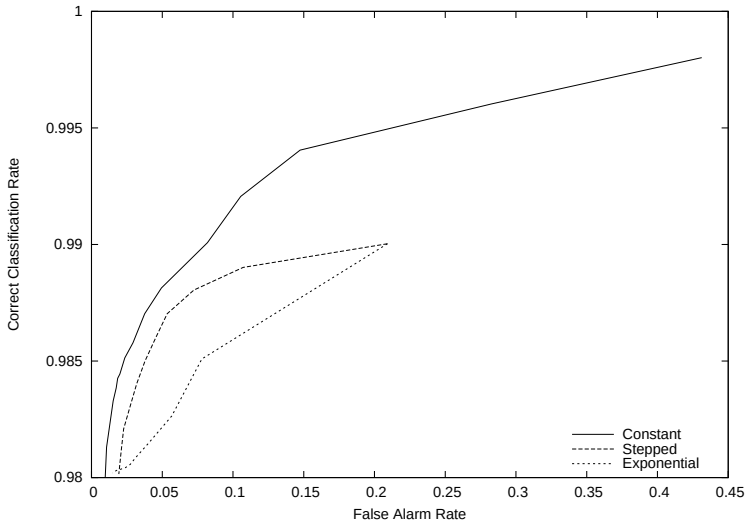

Figure 10. ROC curves for three different biasing strategies. The operational point of the final classifier is at 98% correct classification. While the linear strategy has a better rejection rate, the computational expense of the other two strategies are significantly lower.

Table 1. Average number of sub-stages evaluated for non-face samples under three different biasing schemes with equal final correct classification rate of 98% on the face class.

Linear	Step	Exponential
2.15	1.40	1.35

Performance results for these three different biasing strategies are shown in Table 1. The performance is measured by the average number of sub-stages that must be evaluated for a non-target observation. This is estimated using a large set of non-target observations not used for training. It should be clear that since the relative frequency of non-targets is overwhelmingly higher than that of targets, the computational cost of classifying a non-target observation determines the run-time speed of the classifier when used in the real world. Results show that both of the dynamic strategies for adjusting the biasing factor markedly improve performance. Between the two strategies, the exponential decay of target error has slightly superior performance. Since the exponential decay strategy is also able to achieve a higher non-target rejection rate for the same correct classification, it should be preferred over the step method. The constant bias factor will be able to achieve the best non-target rejection rates of all however, so should be considered when speed is not as important.

5. Adaptive CCCD Trees

In Section 3.4, we discussed the fact that as we train the second and subsequent stages of a sequentially boosted CCCD classifier, we allow multiple prototypes to be used for the target class. This is so that we can more accurately model the support of the target class. In the classification stage, however, we are only concerned with whether the unlabeled sample was closer to any target or non-target prototype. In particular, if a sample is found to be closer to some target prototype than to all non-target prototypes within a stage, the specific closest target prototype is irrelevant. This is a waste of valuable information, since the set of target-class protoypes induce a partition space. More explicitly, for any given CCCD stage, we can view the set of target-class prototypes as a set of cluster representatives. Now, almost every sample (target or non-target) can be assigned to one target-class cluster using the nearest neighbor rule with respect to (1). Figure 11 shows the cluster means obtained by training a CCCD stage on a set of zero mean unit norm face images with fixed eye locations. Even though the choice of prototypes is a greedy optimization of the graph dominating set, we see that structure within the data arises naturally. In this case, the three prototypes encode the type of illumination under which the face image was acquired. In this section we introduce a variant of the sequentially boosted CCCD classifier that exploits the partition induced by the target representatives to yield better performance and faster run times. The inevitable trade-off is more complex and lengthier training.

In order to train a boosted tree, we proceed as in Section 3.4 to obtain the first two stages of a sequentially boosted CCCD classifier. As we pointed out before, for reasons of run-time speed, one normally uses a single target-class prototype for the first stage, and multiple ones for subsequent stages. In order to train the third level stages of the tree, we run the training data through the partial classifier and discard, as before, any training observations that are labeled as non-target. We separate the remaining training observations into groups according to the nearest target-class prototype in the second stage. Now, for each one of these groups (one per target-class prototype) we train a sequential CCCD stage using only the data in the group. As in the sequentially boosted CCCD training, we need only specify the maximum allowable number of target prototypes, and the algorithm will select at most such a number. For example, when training the stage whose target clusters are shown in Figure 11, the training algorithm was told to select no more than four prototypes, yet it chose to stop at three, since the data would only support three natural clusters. Figure 12 shows a boosted CCCD tree. Each stage is shown as a box containing the target-class prototypes along the bottom and the non-target prototypes along the right side. A rightward-pointing arrow indicates classification as non-target. A downward-

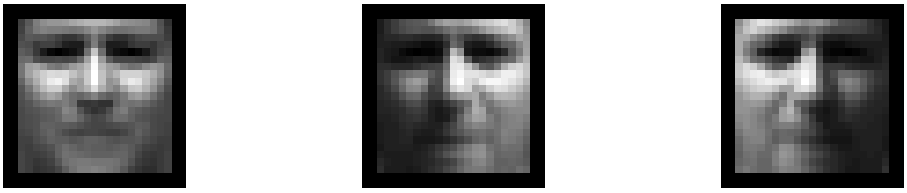

Figure 11. Clusters found by the CCCD classifier while training a boosted tree classifier. Each image represents the mean face within that cluster. While the clusters are ultimately chosen to maximize classification performance, they do correspond to the major image classes that are intuitively expected. Frontal, left and right illumination are all the dominant clusters.

pointing arrow leads to either further testing or classification as target.

Once a boosted CCCD tree is trained, classification is a straightforward extension of the sequentially boosted case. Algorithm 3 shows pseudo-code for the boosted CCCD tree classifier. We simply compare the unlabeled sample sequentially against the target and non-target prototypes in a stage, as before, and if the sample is closer to the target class, we choose the next set of target/non-target prototypes based on the closest target prototype, and continue. If at any point the sample is closer to some non-target prototype than to all target-class prototypes, then it is labeled as non-target and the process is halted. Alternatively, if the sample is never found to be closer to a non-target prototype, it is declared target class. By partitioning the data at training time, we tune the later-stage CCCD classifiers to local regions of the discriminant boundary. As a result, each later stage is required to represent only a region of the boundary in the vicinity of its target-class prototypes, thus allowing finer modeling of the class support with the same number of prototypes. Note that even though a boosted CCCD tree may have a large number of target class prototypes, only those along the path explored by a given unlabeled sample are used. Therefore the increase in representational capacity does not result in longer run times. On the other hand, training is a lengthier process, and more data is needed. In some cases, the need for extremely large amounts of training data may render the tree training infeasible after a few stages, especially for the branches exploring rare cases. In order to obtain sufficient training data to extend these branches it is often necessary to filter millions of non-target training data through the previous tree stages.

Performance when using a CCCD tree is improved on two fronts. Firstly, overall accuracy is better, with results shown in Figure 13. The reason for this is that the number of effective target prototypes is increased. In the sequentially boosted CCCD classifier, only a few target prototypes have to represent the entire target-class distribution, which does not substantially change from

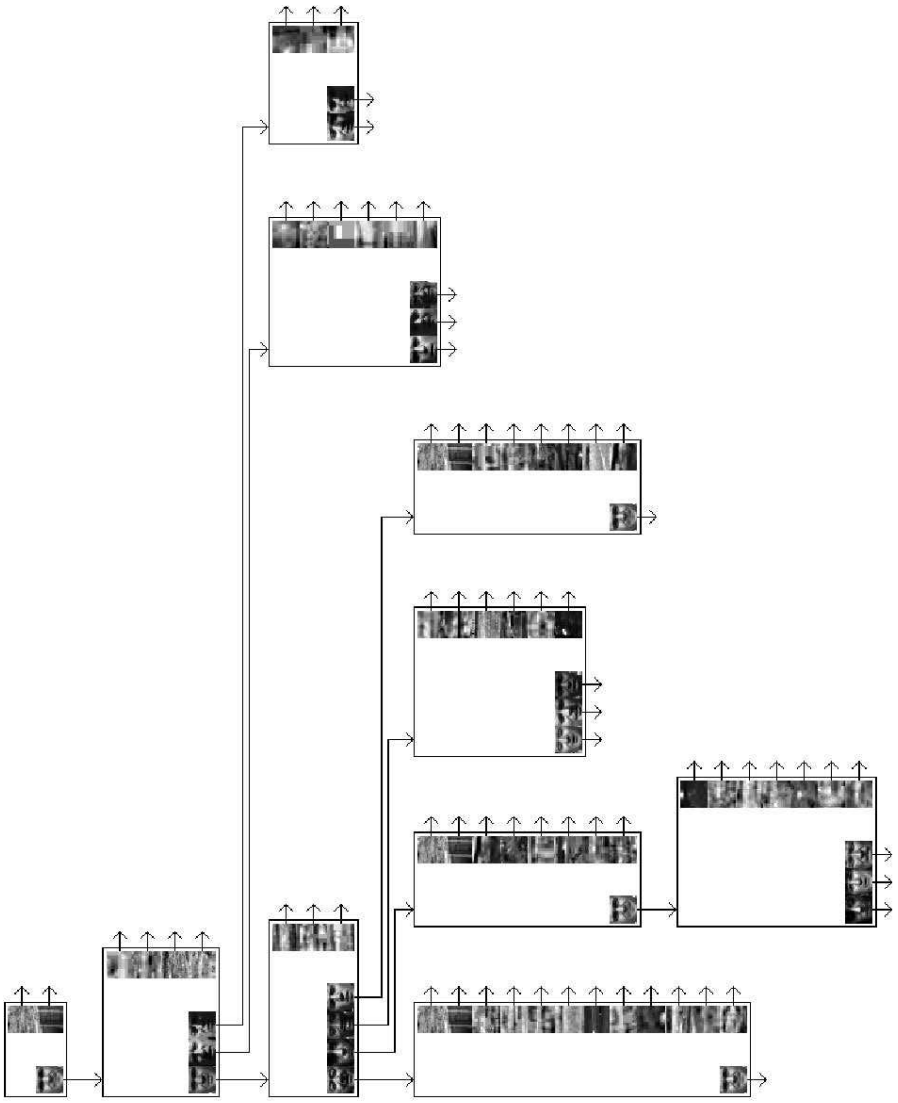

Figure 12. A boosted CCCD tree classifier. Terminal arrows facing right indicate a target-class decision, while arrows facing up indicate a non-target decision. Non-target prototypes (sub-stages) are evaluated from left to right.

Algorithm 3: Boosted tree CCCD classifier

Let $C_k(S)$ be the set of class-k prototypes for the stage S.
Let $c_k(S, j)$ be the class-k prototype for the j-th sub-stage of the stage S.
Let $r_k(S, j)$ be the class-k radius for the j-th sub-stage of the stage S.
Let $T(S, j)$ be the biasing factor for the j-th sub-stage of the stage S.
Let Child(S, j) be the j-th child stage of the stage S.

$S \leftarrow$ The root stage.
while $S \neq \emptyset$ **do**
  for $j \leftarrow 1 \ldots |C_0(S)|$ **do**
    $d_j \leftarrow d(x, c_0(S, j))$
  end for
  for $j \leftarrow 1 \ldots |C_1(S)|$ **do**
    $b \leftarrow \arg\min_{k \in [1, |C_0(S)|]} \frac{d_k}{T(S,j) r_0(S,k)}$
    $r'_0 \leftarrow \frac{d_b}{T(S,j) r_0(S,b)}$
    $r'_1 \leftarrow \frac{d(x, c_1(S,j)) T(S,j)}{r_0(S,j)}$
    if $r'_0 \geq r'_1$ **then**
      return 1
    end if
  end for
  $S \leftarrow$ Child(S, b)
end while
return 0

stage to stage. With the boosted CCCD tree classifier, the same number of face prototypes can be used to model only the cluster within the face distribution matching the previous decisions in the tree. As the target class distribution will decrease in complexity as we narrow our scope, a small number of prototypes normally suffices for good performance.

A secondary performance improvement comes from the fact that the average cost of evaluating a target sample is lower than in the sequential case. This is clear from the fact that for a face sample to be correctly classified by the sequential process, it must be compared to all target prototypes. The average number of comparisons for the tree case is lower, thus yielding faster run-times. Of course, it is the expected cost of evaluating a non-target sample that normally drives the run-time speed of a detector. However, there are cases where the cost of evaluating a target sample is a significant run-time factor.[3] Table 2 shows the average number of sub-stages evaluated when processing a target-class sam-

3. For example, if one were to use the detector within a target tracking framework, the likelihood of observing a target would be relatively high as long as the tracker maintains a good estimate. In this case the relative class priors are no longer so severely skewed, and the cost of classifying a true target observation is relevant to overall speed.

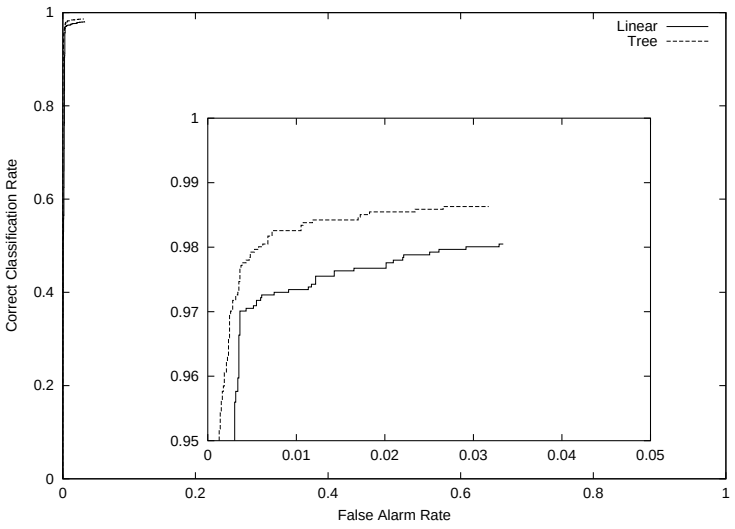

Figure 13. ROC curves for sequentially boosted and boosted tree CCCD classifiers. Insert shows a magnification.

ple using sequentially boosted and boosted tree CCCD classifiers. We see that the number of sub-stage evaluations required to process a target sample is substantially lower when using a tree-based classifier. The number of sub-stages required for evaluating a non-target sample is not very different. This should be no surprise, as the majority of non-target samples are correctly classified using the first stage only, which occurs before the tree begins to branch.

6. Performance and Speed Analysis

This section provides a number of performance comparisons between the proposed CCCD approaches and support vector machines (Vapnik 1998). The comparison to SVMs is particularly relevant since CCCDs have many conceptual similarities with them, especially SVMs based on radial basis functions. While SVMs model the discriminant boundary by choosing class representatives close to that boundary, CCCDs do so by using representatives in regions of high statistical depth.[4] Both methods restrict their choice of representatives to elements of the training set, unlike other methods for prototype selection, such as k-means. In both cases, this restriction is imposed in order to make

4. The statistical depth of a point can be thought of as its closeness to the center of a dataset. For a more detailed discussion, see Zuo and Serfling (2000).

Table 2. Average number of sub-stages that must be evaluated for each two classifiers of roughly equal classification performance on both targets and non-targets.

	Targets	Non-Targets
Sequential CCCD:	20.58	1.44
Tree CCCD:	12.31	1.36

training computationally tractable. Unlike SVMs, however, CCCDs use a sub-optimal procedure for picking the prototype set, since the optimal method is NP-complete.

In some cases CCCD classifiers have been shown to have similar or superior performance to SVMs (DeVinney et al. 2002). For many of the experiments presented in this paper, SVMs have shown better classification performance than CCCD classifiers, although at considerably higher computational cost. Figure 14 shows ROC curves for SVMs with linear, polynomial (degree two), and Gaussian kernels as compared to a CCCD classifier. Multiple CCCD classifiers were trained on random subsets of the data, and 95% confidence intervals on their performance are shown in the graphs. Note that the comparisons are between standard CCCD classifiers with no bound on the number of prototypes (not boosted ones) and SVMs. We have observed that a boosted CCCD classifier will have roughly similar classification performance as a standard one with no bound on the number of prototypes per class, although at a much lower computational cost. Therefore, these comparisons should reflect the expected performance of the boosted classifiers. Unfortunately, training the boosted classifiers is a lengthy process, and it would be infeasible to train a large enough sample in order to have meaningful confidence intervals. As we can see in Figure 14, SVMs with quadratic and Gaussian kernels outperform a CCCD, while the CCCD outperforms a linear SVM. All of them were trained with the same data set.

Figures 15, 16 and 17 explore the generalization capability of CCCD classifiers versus linear, quadratic and Gaussian SVMs, trained and tested on the same data. The training set consisted of face images with frontal or near frontal illumination, and the test set consisted of face images with severly lateral (left or right) illumination. As we can see, CCCDs generalize statistically significantly better than linear SVMs. For Gaussian SVMs, CCCDs appear to generalize slightly worse, but it is unclear whether this difference is significant. The difference appears to be more significant with quadratic SVMs, although once again the confidence intervals are wide enough as to make this unclear.

The clear advantage of CCCDs over SVMs comes when we consider the computational cost of applying the classifier. Table 3 shows the number of support vectors for each of the three tested support vector machines. In each case,

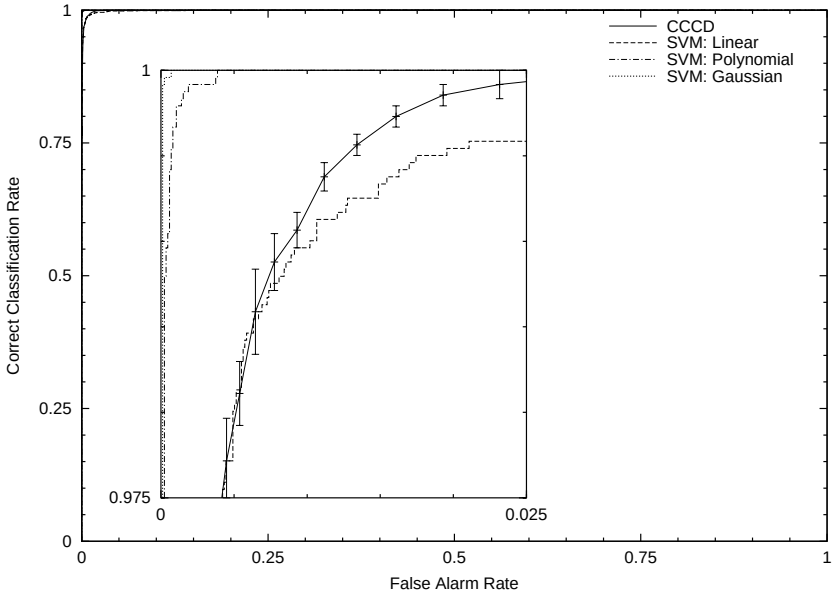

Figure 14. Performance of CCCD *versus* several SVMs. The same training and testing data are used for both. Since the CCCD training algorithm is non-deterministic, 95% confidence intervals are shown. Insert shows a magnification. SVM performance is best with Gaussian kernels, followed by polynomial and linear kernels.

all support vectors must be evaluated in order to obtain a classification. For the boosted CCCD classifiers, we noted that the average number of sub-stages evaluated when processing a non-target (the driving component of the run-time cost) is about 1.4, which would result in an average of 2.4 prototype comparisons per non-target sample. (One for the face class, and an average of 1.4 non faces.) Since the computational cost of processing a support vector is the same as that of processing a CCCD prototype, this is about 450 times fewer comparisons than the various SVMs, and hence about 450 times faster. A number of authors have proposed methods for reducing the complexity of SVMs (Burgess 1996; Schölkopf, Burgess, Knirsch, Müller, Rätsch, and Smola 1999). Using the algorithm in Schölkopf et al. (1999), and the Gaussian SVM whose performance is displayed in Figure 14, we created a reduced SVM with computational complexity comparable to a boosted CCCD tree. Figure 18 shows ROC curves for the full Gaussian SVM, the reduced one, and a boosted CCCD tree. We can now see that for the same computational cost, the boosted CCCD tree is superior to the Gaussian SVM.

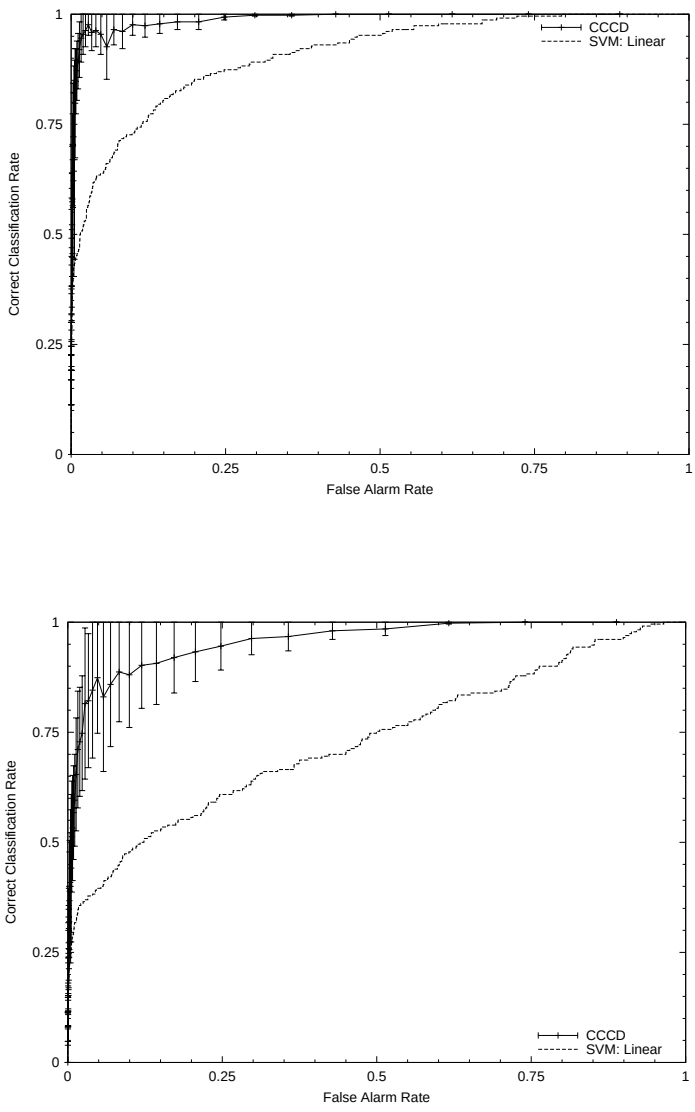

Figure 15. The generalization capability of an SVM with a linear kernel *versus* a CCCD classifier. Testing is on lateral illumination (top) and severe lateral illumination (bottom). In both cases, the training data was frontal or near frontal illumination. Error bars show the 95% confidence intervals after 25 random restarts for the CCCD classifier. Insert shows a magnification.

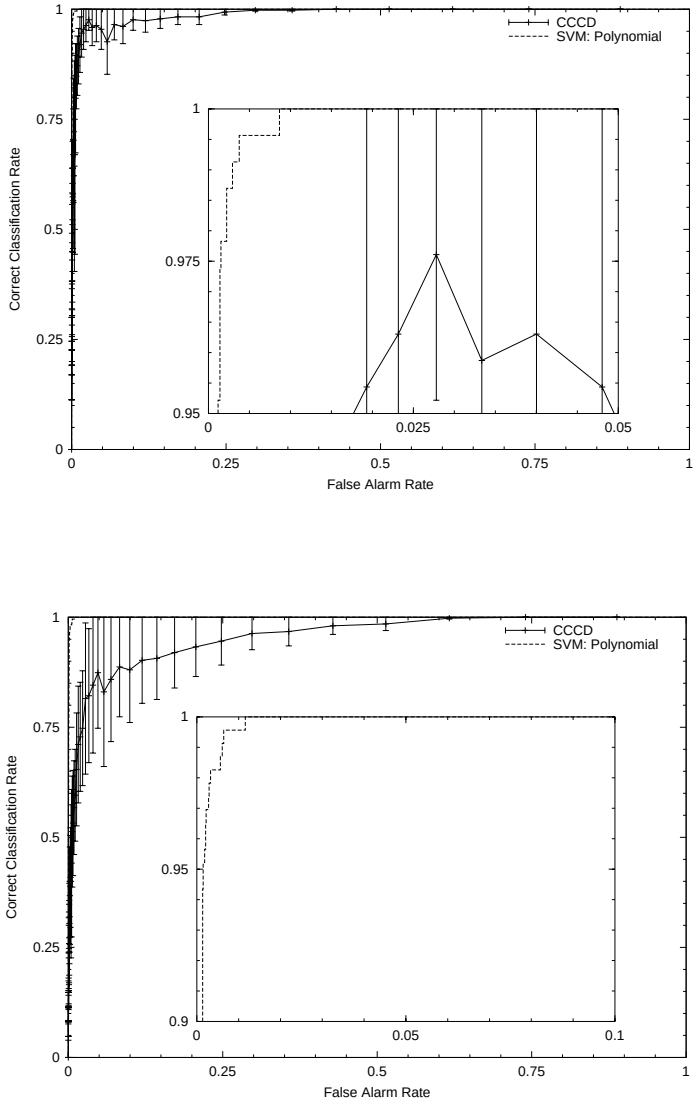

Figure 16. The generalization capability of a SVM with a degree-2 polynomial *versus* a CCCD classifier. Testing is on lateral illumination (top) and severe lateral illumination (bottom). In both cases, the training data was frontal or near frontal illumination. Error bars show the 95% confidence intervals after 25 random restarts for the CCCD classifier. Insert shows a magnification.

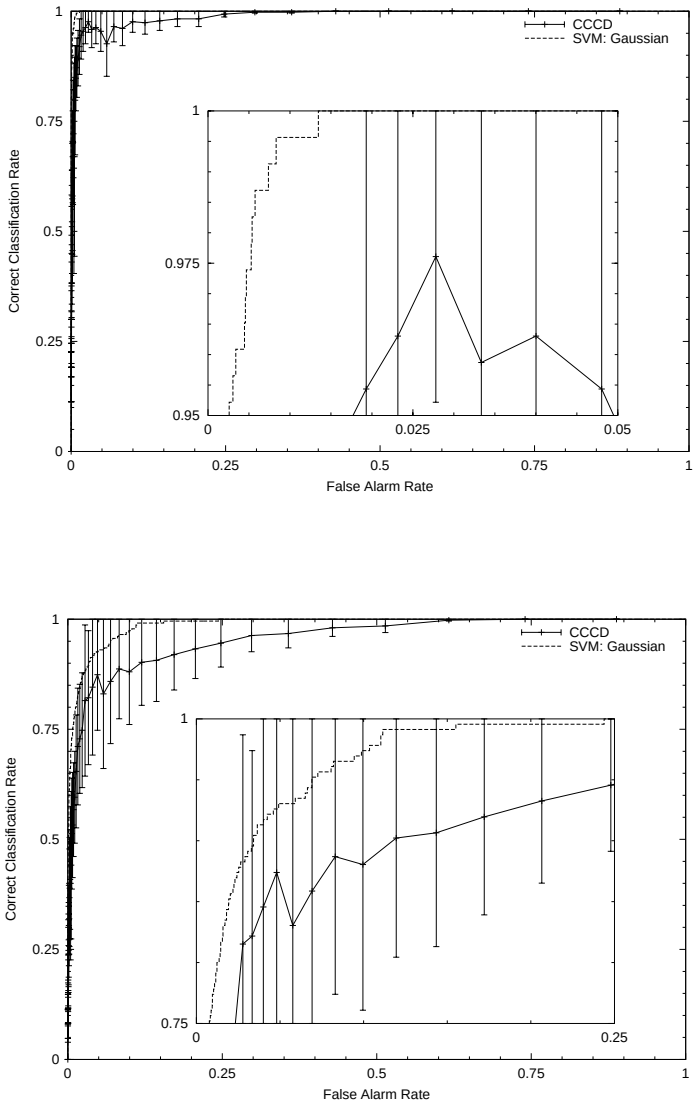

Figure 17. The generalization capability of a SVM with a Gaussian kernel *versus* a CCCD classifier. Testing is on lateral illumination (top) and severe lateral illumination (bottom). In both cases, the training data was frontal or near frontal illumination. Error bars show the 95% confidence intervals after 25 random restarts for the CCCD classifier. Insert shows a magnification.

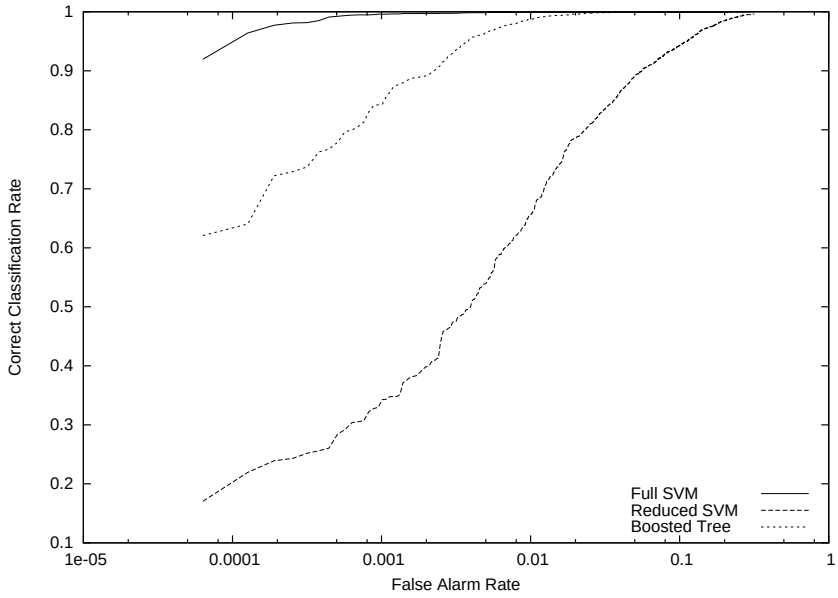

Figure 18. ROC curves for a full Gaussian SVM, reduced Gaussian SVM and boosted CCCD tree, all trained on the same data. The CCCD tree's accuracy lies between that of the full SVM and the reduced set SVM. The reduced set SVM and CCCD tree have the same computational cost, while the full SVM is considerably more expensive.

7. Conclusion

We introduced a novel extension to the CCCD family of classifiers. The proposed methods are specifically designed for the one-class, or detection, problem where the natural abundance of one class is overwhelmingly larger than that of the other. In this context, processing speed for the classifier is primarily driven by the cost of evaluating a sample from the non-target class. As a result, we structured the new classifiers in such a way as to minimize the cost of evaluating such samples. This allowed us to reduce the average cost of evaluating a sample more than tenfold with no degradation in classification performance. The key insight was to structure the classifiers as either cascades or trees with a maximal rejection bias.

Due to the large amount of data needed to train the proposed classifiers, it was necessary to introduce a fast training method based on a bagging technique. Fast training makes it possible to use millions of training observations (for the non-target class) in a manageable period of time. Previous CCCD training implementations were limited to a few thousand training observations.

Comparisons with support vector machines were provided both for accuracy and speed. While the classification performance of certain SVMs appears

Table 3. Number of support vectors used in various experiments.

Name	Kernel	Support Vectors
Linear	$K(\mathbf{x},\mathbf{y}) \propto \mathbf{x}\cdot\mathbf{y}$	887
Polynomial	$K(\mathbf{x},\mathbf{y}) \propto (\mathbf{x}\cdot\mathbf{y})^2$	1306
Gaussian	$K(\mathbf{x},\mathbf{y}) \propto \exp\ -\frac{(\mathbf{x}-\mathbf{y})^2}{2}$	922

to be higher on the sample problem, their computational cost is several hundred times higher, thus making them unsuitable for applications requiring very high processing throughput. As the proposed CCCD classifiers were designed for use in real-time applications, we find this comparison to be a positive one.

The most valuable contribution of this paper, in the authors' view, is the adaptive CCCD tree. The combination of an early rejection option with what amounts to adaptive classifier selection is a powerful tool for fast and accurate classification. An effort is currently underway to construct a similar classifier based on SVMs instead of CCCDs, the results of which will be reported elsewhere.

References

BREIMAN, L. (1998), "Arcing Classifiers (With Discussion)," *Annals of Statistics, 26*, 801–849.

BURGESS, C.J.C. (1996), "Simplified Support Vector Decision Rules," in M. Kaufman (Ed.), *Proceeding of the 13th International Conference on Machine Learning*, pp. 71–77.

DEVINNEY, J., PRIEBE, C.E., MARCHETTE, D.J., and SOCOLINSKY, D. (2002), "Random Walks and Catch Digraphs in Classification," *Computing Science and Statistics*, 34.

DUDA, R.O., and HART, P.E. (1973), *Pattern Classification and Scene Analysis*, New York: John Wiley & Sons.

DUDA, R.O., HART, P.E., and STORK, D.G. (2000), *Pattern Classification*, New York: John Wiley & Sons.

FLEURET, F., and GEMAN, D. (2001), "Coarse-to-Fine Face Detection," *International Journal of Computer Vision, 41*, 85–107.

FUKUNAGA, K. (1990), *Introduction to Statistical Pattern Recognition* (2nd ed.), San Diego: Academic Press.

HASTIE, T., TIBSHIRANI, R., and FRIEDMAN, J. (2001), *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, New York: Springer.

JAIN, A.K., DUIN, R.P.W., and MAO, J. (2000), "Statistical Pattern Recognition: A Review," *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*, 4–37.

KULKARNI, S., LUGOSI, G., and VENKATESH, S. (1998), "Learning Pattern Classification - A Survey," *IEEE Transactions on Information Theory, 44*, 2178–2206.

MCLACHLAN, G.J., and PEEL, D. (2000), *Finite Mixture Models*, New York: John Wiley & Sons.

PRIEBE, C., MARCHETTE, D., DEVINNEY, J., and SOCOLINSKY, D. (2003), "Classification Using Class Cover Catch Digraphs," *Journal of Classification, 20*, 2–23.

RIPLEY, B. (1996), *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

SCHÖLKOPF, B., BURGESS, C.J.C., KNIRSCH, P., MÜLLER, K.-R., RÄTSCH, G., and SMOLA, A. (1999), "Input Space vs. Feature Space in Kernel-Based Methods - Support Vector Learning", *IEEE Transactions on Neural Networks*, *10*, 1000–1017.

SCOTT, D.W. (1992), *Multivariate Density Estimation: Theory, Practice, and Visualization*, New York: John Wiley & Sons.

SILVERMAN, B.W. (1986), *Density Estimation for Statistics and Data Analysis*, New York: Chapman and Hall.

SOCOLINSKY, D.A., NEUHEISEL, J.D., PRIEBE, C.E., MARCHETTE, D., and DEVINNEY, J.G. (2003), "A Boosted CCCD Classifier for Fast Face Detection", *Computing Science and Statistics*, 35.

VAPNIK, V. (1998), *Statistical Learning Theory*, New York: Wiley Interscience.

VIOLA, P., and JONES, M. (2001a), "Rapid Object Detection using a Boosted Cascade of Simple Features", in *Computer Vision and Pattern Recognition*, IEEE Computer Society, pp.I:511–518.

VIOLA, P., and JONES, M. (2001b), "Robust Real-Time Face Detection", in *International Conference on Computer Vision*, IEEE Computer Society, p.II:747.

ZUO, Y., and SERFLING, R. (2000), "General Notions of Statistical Depth Function", *Annals of Statistics*, *28*, 461–482.