Fast Asymmetric Learning for Cascade Face Detection

Jianxin Wu, S. Charles Brubaker, Matthew D. Mullin, and James M. Rehg, *Member, IEEE,*

***Abstract*— A cascade face detector uses a sequence of node classifiers to distinguish faces from non-faces. This paper presents a new approach to design node classifiers in the cascade detector. Previous methods used machine learning algorithms that simultaneously select features and form ensemble classifiers. We argue that if these two parts are decoupled, we have the freedom to design a classifier that explicitly addresses the difficulties caused by the asymmetric learning goal. There are three contributions in this paper. The first is a categorization of asymmetries in the learning goal, and why they make face detection hard. The second is the Forward Feature Selection (FFS) algorithm and a fast precomputing strategy for AdaBoost. FFS and the fast AdaBoost can reduce the training time by approximately 100 and 50 times, in comparison to a naive implementation of the AdaBoost feature selection method. The last contribution is Linear Asymmetric Classifier (LAC), a classifier that explicitly handles the asymmetric learning goal as a well-defined constrained optimization problem. We demonstrated experimentally that LAC results in improved ensemble classifier performance.**

***Index Terms*— Face detection, cascade classifier, asymmetry, feature selection.**

I. Introduction

THERE has been much progress in frontal face detection in recent years. State of the art face detection systems can reliably detect frontal faces at video rate. Various face detection methods have been proposed [1]–[7].

Most face detectors use a pattern classification approach. A classifier that can discriminate face patches from background non-face patches is trained from a set of training examples. When a new test image is presented, patches of all possible sizes and positions are extracted and scaled to the same size as the training samples. The trained classifier then decides whether a patch is a face or not. This brute-force search strategy is used in most of the face detection methods.

The classifiers used in early work on face detection, e.g. neural networks [2] and SVM [5], were complex and computationally expensive. Instead of designing a single complex classifier and applying it to every possible patch, recently coarse-to-fine search has been used to achieve computational efficiency. In early work, Amit and Geman [8] designed a *visual selection* strategy which was in effect a coarse-to-fine search process. The coarse-to-fine search idea was popularized by Viola and Jones [7] using explicitly a *cascade* of classifiers with increasing complexity, illustrated in Fig. 1. An input patch was classified as a face only if it passed tests in all the nodes. Most non-face patches were quickly rejected by the early nodes. Cascade detectors have demonstrated impressive detection speed and high detection rates.

Manuscript received ; revised .

The authors are with the School of Interactive Computing, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332.

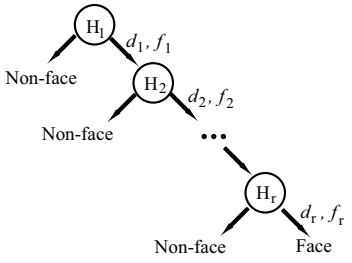


Fig. 1. Illustration of the cascade structure with r nodes, where H_i is the ith node classifier, and d_i and f_i are the detection rate and false positive rate of the ith node, respectively.

In this paper we use the cascade structure, in order to ensure high testing speed.[1]

There were three contributions in the Viola-Jones face detection system: the integral image representation, the cascade framework, and the use of AdaBoost to train cascade nodes. The cascade framework allows background patches to be filtered away quickly. The integral image representation can calculate the image features extremely fast, which are called 'rectangle features' and are then used in the node classifiers. The AdaBoost algorithm [10] is used to select rectangle features and combine them into an ensemble classifier in a cascade node. The integral image and the cascade framework make the detector run fast, and AdaBoost is the key to a cascade's high detection rate.

AdaBoost performs two tasks simultaneously when it trains a node classifier: selecting several rectangle features, and forming an ensemble classifier using combination of these features. However, these two processes are not necessarily tied together. In this paper, we show that by decoupling the problems of feature selection and ensemble classifier design, we can address the fundamental difficulties in the learning problem explicitly. The result is improved classification performance and faster training speed. The contributions of this paper are summarized as three points.

First, we categorize different forms of asymmetries in the face detection problem and explain how they make face detection hard. For example, while the positive class contains only faces and requires only thousands of training images, the negative class contains image patches from all over the world and requires billions of training samples.

Second, we propose the Forward Feature Selection (FFS) algorithm as an alternative way to select features in a cascade node. FFS has similar detection accuracy as AdaBoost. It also reduces the cascade's training time by two orders of magnitude compared to a naive implementation of AdaBoost. We also present a faster implementation for the AdaBoost algorithm. FFS

[1]There exist other fast detection methods, e.g. the synergistic face detection method [9].

is approximately 2 to 3 times faster than this new implementation of AdaBoost. In addition, FFS only requires about 3% memory usage as that of the faster AdaBoost implementation, which makes it a natural choice in applications with a huge training set and a big pool of features.

Third, we then propose the Linear Asymmetric Classifier (LAC) to form ensemble classifiers. Decoupled from the feature selection process, LAC has the freedom to explicitly address the asymmetries in the face detection problem. Thus, LAC can improve the node classifiers' performances. For example, applying LAC to features selected by AdaBoost, we obtain a better classifier than that provided by AdaBoost itself. LAC is efficient and is easy to implement.

The rest of this paper is organized as follows. Section II explains how the cascade framework helps solving difficulties in the face detection problem and section III provides a survey of related methods. FFS and a faster implementation for AdaBoost are introduced in section IV, and LAC is described in section V. Experimental results comparing FFS/LAC to other methods are also presented. Section VI concludes this paper with discussions of future work.

Preliminary versions of portions of this work have been published in [11] and [12]. The FFS algorithm presented in this paper is an improved version of the FFS algorithm in [11]. Some new results are also presented in this paper, including analysis of asymmetries in the face detection problem (section II and III), a fast implementation of the AdaBoost method (section IV-C), validity of LAC's assumptions (section V-B), and additional experimental results (Fig. 6, 9(c), and 10).

II. Analysis of the cascade face detector

A. Asymmetries in face detection

We observe three asymmetries in the face detection problem: uneven class priors, goal asymmetry, and unequal complexity within the positive and negative classes. In this section, we will discuss why these asymmetries make the classifier design problem difficult. We want to point out that these asymmetries also apply to the detection of all other objects (e.g. car detection).

The first asymmetry comes from the uneven class priors. Among the millions of image patches generated from an input image, only very few contain faces. The occurrence of a face in an image is a rare event. In this sense, face detection (and all other detection problems in vision) are rare event detection problems. Methods designed to minimize error rate will classify all instances as negative on such extremely uneven data sets. Thus all faces will be missed. Another difficulty associated with this unevenness is that the negative class usually has a huge amount of data. Approximately 350 million non-face patches are used in [7] to train a cascade. The flood of non-face patches makes the learning algorithm train very slowly.

The second asymmetry comes from the difference in positive and negative class learning goals. A high (e.g. 95%) detection rate is required, because we do not want to lose a single face. However, because of the huge amount of non-face data, an extremely low false positive rate (e.g. 10^{-7}) is necessary for reliable detection. It is difficult for a single classifier to achieve such a learning goal.

The last asymmetry comes from the different composition of the two classes. The positive class consists of only faces. However, the negative class consists of image patches from a wide range of scenes, which include a very large number of different object categories: animals, trees, man-made objects, buildings, and more. It is not hard to distinguish faces from cars. However, it is much harder to distinguish faces from all other objects.

B. Cascade approach to asymmetric problems

The cascade structure alleviates the difficulties associated with the three asymmetries described above.

Cascade classifier deals with the uneven data set via sampling. The training set for each node classifier is balanced by sampling roughly the same amount of non-face patches as the number of faces. After a new node is trained, all non-face patches which are correctly classified by this node are removed from the pool of non-faces. So the number of non-face patches in the pool decreases at an exponential speed. This data bootstrapping strategy deals effectively with the huge amount of non-face data. It was introduced in [1] as a way to find non-face samples that are difficult to separate from faces. It is worth noting that sampling is a widely used strategy to deal with uneven data sets in the machine learning and data mining domains [13].

Goal asymmetry is also addressed by the cascade classifier. Consider a cascade which consists of a set of nodes $H_1, H_2, \ldots, H_r$. Let O be the event that the testing instance is a true face, and A_i be the event that H_i classifies it as a face. Then, the detection rate D and false positive rate F of the cascade are computed as

$$D = \Pr\left[A_r, \ldots, A_1 | O\right] = \prod_{i=1}^{r} d_i \tag{1}$$

$$F = \Pr\left[A_r, \ldots, A_1 | \bar{O}\right] = \prod_{i=1}^{r} f_i \tag{2}$$

by the chain rule, where $d_i = \Pr\left[A_i | A_{i-1}, \ldots, A_1, O\right]$ and $f_i = \Pr\left[A_i | A_{i-1}, \ldots, A_1, \bar{O}\right]$ are the detection rate and false positive rate of the ith node. The above equations do not assume that the nodes make independent errors. The false positive rate F drops to 0 exponentially with the depth of the cascade.

As a consequence of (1) and (2), it is natural to define the learning goal of a cascade as: for every node, design a classifier with very high (e.g. 99.9%) detection rate and only moderate (e.g. 50%) false positive rate.

This node learning goal is in principle easier to achieve in comparison to the daunting 10^{-7} false positive rate goal for a single classifier. However, the result is a cost-sensitive learning problem. A false negative clearly costs more than a false positive since we allow about 50% errors in the negative class, but nearly no error can be allowed in the face class. Many machine learning algorithms (including AdaBoost) are designed to minimize error rates and usually do not work well on cost-sensitive problems [13]. Viola and Jones proposed the AsymBoost method [14] to handle this asymmetry. We will discuss the drawbacks of AsymBoost and other asymmetric learning methods in the related work (section III). We then propose the Linear Asymmetric Classifier to deal with this cost asymmetry in section V.

The asymmetry in class composition is taken care of by increasing the complexity of the node classifiers. When more nodes are used, the data bootstrapping process will make the node negative training set contain more "face-like" patches, which are hard to separate from faces. The node classifiers become more complex, consequently. In [7], only 2 features were used in the

first node, while 200 features were used for the last node. Since most non-face patches are rejected by early nodes, a small number of features are evaluated for these patches. This fact enables the cascade to run at video rate.

III. Related work

We now review the literature on cascade-structured detectors for faces and other objects. A comprehensive review of other face detection approaches and techniques can be found in [15].

A. 'Project and Reject' detectors

One key aspect of the cascade detector is the ability to quickly reject some candidate image patches. This intuition is utilized in many face detection systems implicitly or explicitly. For example, the neural network-based detector of Rowley *et al*. [2] incorporated a manually-designed two node cascade structure for improving the detection speed.

There are some other detectors which explicitly exploited the idea of rejecting non-face patches in a cascade-like structure. Most of these methods follow a 'Project and Reject' approach. In the maximal rejection classifier approach [16] and the Antiface approach [17], in each stage the input patch was projected in a given direction and was rejected if the projected value was beyond certain thresholds. In [18], a set of reduced set vectors was calculated from a support vector machine. These vectors were applied as projection directions sequentially. An alternative cascade framework for SVM classifiers was proposed by Heisele *et al.* [19]. Baker and Nayar proposed a theory of pattern rejection for object recognition [20] using the project and reject procedure.

There are two major differences between these methods and the Viola-Jones detector. First, cascade detectors used Haar-like rectangle features [21]. Rectangle features can be extracted more quickly, in comparison to the projection operation. Second, earlier nodes in a cascade have smaller complexity than deeper nodes. Since most non-face patches are rejected by these early nodes, the cascade is able to run faster than those detectors whose nodes all have the same complexity. Recently, Sahbi and Geman also used a hierarchy of SVMs (organized as a tree) to detect faces [22].

B. Node training in a cascade framework

The remaining works that we will discuss are focused on the Viola-Jones cascade framework. The central learning problem is to construct a single node which satisfies the node learning goal. Successive iterations of this procedure will result in a cascade.

As discussed above, it is the cost asymmetry (or cost-sensitive) nature of this learning goal that makes it difficult. Many cost-sensitive learning methods have been proposed. We are specifically interested in those variants of AdaBoost since the Viola-Jones work has demonstrated that AdaBoost is an effective method to learn a node. A naive approach based on modifying the initial weight distribution was used by Schapire *et al.* in text filtering [23]. However, it is pointed out by Viola and Jones in [14] that, even if positive examples are assigned much higher initial weights than negative examples, the difference is absorbed quickly by AdaBoost. They proposed AsymBoost [14] as a remedy, which continuously gave positive examples higher weights at every training round. They applied AsymBoost to face detection and showed that it had fewer false positives than standard AdaBoost. The key idea is to put more weights on positive examples than negative ones. Many other strategies to apply this idea have been proposed in the machine learning and data mining literature, e.g. AdaUBoost [24], AdaCost [25], and the CSB family [26]: CSB0, CSB1 and CSB2. One characteristic of these methods is that they all conflate the problem of selecting features with the problem of designing an ensemble classifier.

There are other methods that are related to the node learning goal, e.g. BMPM [27] and MRC [16]. BMPM is an asymmetric learning algorithm, which maximizes an lower bound of the detection rate while keeping the false positive rate smaller than a constant. Additional discussion on these methods will be presented in section V-D.

Several related works provide theoretical underpinnings for cascade classifiers. Blanchard and Geman [28] analyzed the designs of hierarchical testings from the statistical point of view, which includes the cascade classifier as a special case. There are also tools to analyze the generalization ability of a cascade. The cascade structure is a special case of decision list, a data structure introduced by Rivest [29]. Anthony gave generalization bounds for threshold decision lists [30], which can be applied to cascade detectors.

C. Other methods related to the cascade detector

We also briefly mention some other improvements to the cascade detector, including new features, node and cascade classifier training methods, and applications.

New features have been proposed, e.g. the rotated Haar-like features [31]. Levi and Weiss studied various features to reduce the number of training images [32]. Torralba *et al.* proposed a way to efficiently share features for multi-class object detection, although they did not use a cascade [33].

The learning algorithms used to train node classifiers are another topic of interest. Lienhart *et al.* [31] experimentally evaluated different boosting algorithms and different weak classifiers. They argued that Gentle AdaBoost and CART decision trees had the best performance. Xiao *et al.* proposed the Boosting Chain algorithm [34] to integrate "historical knowledge" into the ensemble classifier. Li *et al.* incorporated floating search into the AdaBoost algorithm (FloatBoost) for detecting multi-view faces [35]. Liu and Shum proposed KLBoost to train a node classifier, in which the weak classifiers were based on histogram divergence of linear features [36]. It is worth noting that in KLBoost the classifier design was also decoupled from feature selection. In KLBoost, a linear classifier was learned using gradient descent after features were selected.

The aforementioned research focused on improving a single node classifier. In [37], Sun *et al.* considered the problem of connecting the learning objectives of a node to the overall cascade performance. They proposed a cascade indifference curve framework to select the point of operation in a node classifier's ROC curve.

Besides improving the cascade face detector, researchers have used the framework in other fields. A cascade approach has been used (in some cases with substantial modifications) to detect multi-view faces [35], [38], text [39], wiry objects [40], and pedestrians [41].

IV. Fast Feature Selection

Algorithm 1 The cascade framework

1: {Given a set of positive examples $\mathcal{P}$, a set of initial negative examples $\mathcal{N}$, and a database of bootstrapping negative examples $\mathcal{D}$. }
2: {Given a learning goal $\mathcal{G}$.}
3: {The output is a cascade $H = (H_1, H_2, \ldots, H_r)$.}
4: $i \Leftarrow 0$, $H \Leftarrow \emptyset$
5: **repeat**
6: $\quad i \Leftarrow i + 1$
7: $\quad$ *NodeLearning* { Learn H_i using $\mathcal{P}$ and $\mathcal{N}$, add H_i to H }
8: $\quad$ Remove correctly classified non-face patches from $\mathcal{N}$
9: $\quad$ Run the current cascade H on $\mathcal{D}$, add any false detection to $\mathcal{N}$ until $\mathcal{N}$ reaches the same size as the initial set.
10: **until** The learning goal $\mathcal{G}$ is satisfied

The cascade learning algorithm is shown in Algorithm 1. The two major blocks in training a cascade are node learning (line 7) and data bootstrapping (line 9).

In the Viola-Jones detector, the node learning algorithm is AdaBoost, which does feature selection and classifier design simultaneously. We decouple '*NodeLearning*' into two separate parts and propose Forward Feature Selection to perform the feature selection task. We also describe a fast implementation of AdaBoost to select features.

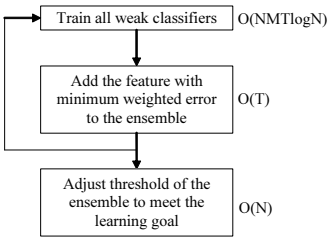
A. The Forward Feature Selection algorithm

In [7], AdaBoost was used to train the node classifier H_i in line 7 of Algorithm 1. AdaBoost is an iterative method for obtaining an ensemble of weak classifiers by evolving a distribution of weights, D_t, over the training set. In the Viola-Jones approach, each iteration t of the boosting process adds the rectangle feature h_t with the lowest weighted error to the ensemble classifier. After T rounds of boosting, the decision of the AdaBoost ensemble classifier is defined as $H(\mathbf{x}) = \text{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) - \theta\right)$, where the α_t's are the ensemble weights obtained by the AdaBoost algorithm and θ is threshold of the ensemble. The flowchart of a straightforward implementation of AdaBoost is shown in Fig. 2(a), in which N and M are the numbers of training examples and features, respectively.

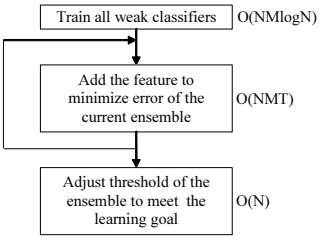
AdaBoost picks the rectangle feature with the smallest weighted error with respect to the weight distribution D_t. D_t is updated every round, which in turn requires that the weak classifiers were re-trained at every round, as indicated in Fig. 2(a). In the face detection application, the number of training examples and rectangle features are both in the order of thousands. Thus, the re-training of rectangle features is the most time consuming component in the algorithm. In [7], it took a few weeks to train a complete cascade face detector.

We propose a new feature selection method based on Forward Feature Selection (FFS), a greedy feature selection method [42]. After the features are selected, an ensemble classifier can be formed by simple voting of the selected features. Pseudo-code for the FFS algorithm for selecting features and building an ensemble classifier for a single cascade node is given in Algorithm 2. The corresponding flowchart is illustrated in Fig. 2(b).

Before getting into details of FFS, the flowcharts reveal the intuition behind it. In Fig. 2(b), 'Train all weak classifiers', the most time-consuming component, is moved out of the loop. That



(a) Naive AdaBoost implementation



(b) Forward Feature Selection

Fig. 2. Diagram comparing the naive AdaBoost implementation and the FFS algorithm for selecting features and forming a single node classifier.

is, the weak classifiers are trained for only once. In AdaBoost, they need to be trained T times, where T is the number of features in the AdaBoost ensemble. Since most of the training time is used to train weak classifiers, FFS requires only $1/T$ training time as that of AdaBoost. In summary, the key intuition in FFS is *pre-computing*: the results of trained weak classifiers are stored and re-used.

Both AdaBoost and FFS are greedy feature selection methods: they pick up the 'best' feature they find in every round, based on different criteria. There are three major differences between FFS and AdaBoost.

First, there is no distribution maintained over the training set in FFS. Each training example is treated equally. Thus, the rectangle features are trained only once. Their corresponding weak classifiers' classification results are stored into a table V. In the following feature selection part, table lookup operations provide all the information needed about the rectangle features. Thus, FFS greatly expedites the training process, with the cost of the storage of a table V.

Second, after the features in an ensemble have been selected, the ensemble classifier $H(\mathbf{x}) = \text{sgn}\left(\sum_{h \in S} h\left(\mathbf{x}\right) - \theta\right)$ is a majority vote. The total vote $\sum_{h \in S} h\left(\mathbf{x}\right)$ is an integer between 0 and T, the number of selected features. In contrast, the total vote in AdaBoost is a real number.

Third, the criterion used in FFS to select features is that the selected feature should make the ensemble classifier has smallest error on the training set. In AdaBoost, the criterion is to choose a feature with the smallest weighted error on the training set.

The FFS algorithm operates as follows. The first step is to train all weak classifiers and store their classification results into a table V. The set of selected features, S, is initialized to an empty set. At every round, we examine every possible feature and select the feature that most reduces the ensemble classifier's error.

Given the candidate feature h_i and S, it is easy to compute the error rates of $S' = S \cup \{h_i\}$. The sum of votes v' for S' is obtained by a vector addition (line 11 in Algorithm 2). As

Algorithm 2 The Forward Feature Selection algorithm

1: {Given a set of examples $\{\mathbf{x}_i, y_i\}_{i=1}^N$, where N is the size of the training set.}
2: {Given a set of rectangle features $\{h_i\}_{i=1}^M$, where M is the number of rectangle features.}
3: {The output is an ensemble classifier, whose false positive rate is 0.5. S is the set of selected features.}
4: **for** $i = 1$ to M **do**
5: Pick appropriate threshold for rectangle feature h_i, such that h_i has smallest error on the training set
6: **end for**
7: Make a table V_{ij} such that $V_{ij} = h_i(\mathbf{x}_j)$, $1 \le i \le M, 1 \le j \le N$
8: $S \Leftarrow \emptyset, v \Leftarrow \mathbf{0}_{1\times N}$, where $\mathbf{0}_{1\times N}$ is a row vector of zeros.
9: **for** $t = 1$ to T **do**
10: **for** $i = 1$ to M **do**
11: $S' \Leftarrow S \cup h_i, v' \Leftarrow v + V_{i:}$, where $V_{i:}$ is the ith row of V.
12: $\{H'(\mathbf{x}) = \text{sgn}\left(\sum_{h\in S'} h\left(\mathbf{x}\right) - \theta\right)$ is the classifier associated with S', and we can compute its value using $H'(\mathbf{x}_i) = \text{sgn}(v'_i - \theta).\}$
13: Find the θ that makes H' has the smallest error rate
14: $\epsilon_i \Leftarrow$ the error rate of H' with the chosen θ value
15: **end for**
16: $k \Leftarrow \arg\min_{1\le i\le M} \epsilon_i$
17: $S \Leftarrow S \cup h_k$, $v \Leftarrow v + V_{k:}$
18: **end for**
19: {The output is $H(\mathbf{x}) = \text{sgn}\left(\sum_{h\in S} h\left(\mathbf{x}\right) - \theta\right)\}$
20: Adjust the value of θ such that H has a 50% false positive rate on the training set.

discussed above, the components of v' are integers between 0 and t, the round index number. Thus, the threshold θ can only take values in the set $\{0, 1, \ldots, t\}$. A histogram for v' can be built (how many positive/negative examples have 0 votes, 1 votes, etc.) With this histogram, it is trivial to find the optimal value for θ, which allows the ensemble classifier associated with S' to have the smallest error rate (line 13). We will later show that to train a single feature requires $O(N \log N)$ time steps (refer to section IV-B). It is easy to see that the overall time complexity of the FFS algorithm is $O(NMT + NM \log N)$. These computational complexity results are also shown in Fig. 2(a) and 2(b).

B. Complexity of the AdaBoost algorithm

Similarly, we can analyze the complexity of the AdaBoost algorithm. The first step is to analyze the complexity of the weak classifier training algorithm. When a rectangle feature is given, the optimal threshold τ can only take value from a finite set: the feature values at the training samples. After the feature values are sorted, the error rate of a weak classifier with different thresholds can be updated sequentially at all possible threshold values. This algorithm is described in Algorithm 3, which is $O(N \log N)$. The AdaBoost algorithm is thus $O(NMT \log N)$. As discussed in section IV-A, FFS has complexity $O(NMT + NM \log N)$. In face detection we have $N \gg T \gg \log N$, which means that FFS requires approximately $\frac{1}{T} + \frac{1}{\log N}$ of the training time of AdaBoost.

Algorithm 3 Training a weak classifier

1: {Given a training set $\{\mathbf{x}_i, y_i\}_{i=1}^N$ with weights $\{w_i\}_{i=1}^N$, a rectangle feature h, and its corresponding mask $\mathbf{m}$}
2: Compute the feature values, $v_1, \ldots, v_N$, where $v_i = \mathbf{x}_i^T \mathbf{m}$.
3: Sort the feature values as $v_{i_1}, \ldots, v_{i_N}$ such that $(i_1, \ldots, i_N)$ is a permutation of $(1, \ldots, N)$, and $v_{i_1} \le \cdots \le v_{i_N}$
4: $\epsilon \Leftarrow \sum_{y_i=-1} w_i$
5: **for** $k = 1$ to N **do**
6: **if** $y_{i_k} = -1$ **then**
7: $\epsilon \Leftarrow \epsilon - w_{i_k}, \epsilon_i \Leftarrow \epsilon$
8: **else**
9: $\epsilon \Leftarrow \epsilon + w_{i_k}, \epsilon_i \Leftarrow \epsilon$
10: **end if**
11: **end for**
12: $k = \arg\min_{1\le i\le N} \epsilon_i$, $\tau = \mathbf{x}_{i_k}^T \mathbf{m}$
13: The output is a weak classifier $h(\mathbf{x}) = \text{sgn}(\mathbf{x}^T \mathbf{m} - \tau)$

C. Faster AdaBoost implementation

Careful examination of Algorithm 3 reveals that AdaBoost can also be expedited by a pre-computing strategy. The feature values v_i and the permutations $i_1, \ldots, i_N$ do not change in different rounds of AdaBoost. This part (line 2 and line 3 in Algorithm 3) only needs to be done once and the permutation vectors can be stored in a table V for future use. Using such a permutation table V, the weak classifier training is now $O(N)$, and the AdaBoost algorithm is $O(NMT + NM \log N)$. Other researchers also independently suggested using a similar pre-computing idea, e.g. in [43], [44].

We want to point out that FFS has lower memory requirement than the faster AdaBoost implementation. In FFS, every entry in the table V is a binary value and requires only 1 bit. However, every entry in the table V in AdaBoost is an integer which uses 32 bits (in a 32-bit CPU). In applications with large number of training examples and a large feature set (and consequently a large table V), FFS may be a preferred feature selection method.

D. Experiments comparing FFS and AdaBoost

Although the computational complexity of the FFS and AdaBoost algorithms can be analyzed, their detection performances need to be compared experimentally. Both algorithms are used to build cascade face detectors, and their performances are compared on the MIT+CMU test set [2]. Two cascade face detectors were trained using the FFS and AdaBoost algorithm, respectively. All other aspects of the experimental setup were the same: the two cascades were trained with the same training set, validation set, abstract cascade algorithm, and learning goal. In all cascades we trained, cascade nodes at the same depth all have the same number of features. All cascades were evaluated using the same test set and post-processing step.

Our training set contained 5000 example face images and 5000 initial non-face examples, all of size 24x24. We had a set of 4832 face images for validation purposes. We used approximately 2284 million non-face patches to bootstrap the non-face examples between nodes (line 9 of Algorithm 1). We used 16233 features sampled uniformly from the entire set of rectangle features. For testing purposes we used the MIT+CMU frontal face test set in all experiments. Although many researchers use automatic procedures to evaluate their algorithm, we decided to manually

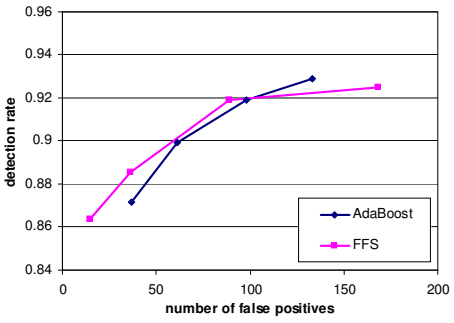


Fig. 3. ROC curves comparing cascade detectors using the AdaBoost and FFS algorithm on the MIT+CMU test set.

count the missed faces and false positives.[2] When scanning a test image at different scales, the image is re-scaled repeatedly by a factor of 1.25. Post-processing is similar to [7].

In the AdaBoost cascade, every node classifier is an AdaBoost ensemble. In each node classifier, the weights of selected features are set by the AdaBoost training algorithm. In the FFS cascade, every node is an ensemble classifier too. In each node classifier, we use FFS to select features, and the weights for all of the selected features are set to 1. In both cascades, we adjust thresholds of the node classifiers such that they have 50% false positive rates.

Our cascade training algorithm terminates when the bootstrapping non-face image database is depleted. Since our learning goal requires that every cascade node has a false positive rate of 50%, all cascades should have approximately the same number of nodes. However, we find that the AdaBoost cascade has 21 nodes, but the FFS cascade has only 17 nodes. This discrepancy comes from the fact that in FFS, every weak classifier has integer votes (1 or -1). With only integer votes, FFS can not achieve an exact 50% false positive rate and we choose lower false positive rates for each node in our experiments. The average false positive rate of all nodes in the FFS cascade is 43.48%. So the FFS cascade has fewer nodes[3].

ROC curves of the AdaBoost cascade and the FFS cascade are shown in Fig. 3. We construct the ROC curves by repeatedly removing nodes from the cascades to generate points with increasing detection and false positive rates. The curve between two consecutive points is approximated by a line segment. The ROC curves show that the FFS cascade has very close performance to the AdaBoost cascade. In regions with more false positives (>100), the AdaBoost classifier's performance is slight better than that of the FFS classifier. In regions with less false positives, the FFS cascade has slightly better performance.

Experiments also showed other properties of the FFS algorithm. Since the same cascade framework was used in the FFS cascade and each node had the same number of features as the AdaBoost cascade, the detection speed (test speed) should be about the same as the AdaBoost cascade. The experiments showed that both cascade detectors did have very close testing speed.

Furthermore, although both FFS and faster implementation of the AdaBoost algorithm has the same complexity $O(NMT + NM \log N)$, the constant factor in FFS is smaller. To train a node having the same number of features, experiments showed that the faster implementation of AdaBoost usually took 2.5–3.5 times of the training time of FFS, and the original AdaBoost implementation needed 50–150 times of the training time of FFS.

The fact that FFS has similar performance as AdaBoost suggests that AdaBoost is not the only choice for the feature selection method in training a cascade node. Other feature selection methods, such as FFS, can be used in place of AdaBoost and still get good or even better performance. For a complete comparison of feature selection methods in the cascade framework (including FFS, CMIM [45] and various boosting methods), we refer the reader to [46].

V. Linear Asymmetric Classifier

We have presented FFS as an alternative to AdaBoost for feature selection in cascade classifiers. This raises the question of whether alternative methods for forming an ensemble classifier from selected features could lead to performance improvements. In particular, neither FFS or AdaBoost explicitly addresses the difficulties caused by the asymmetries discussed in section II-A. In this section, we propose the Linear Asymmetric Classifier (LAC) which is designed to handle the asymmetric node learning goal in the cascade framework: for every node, design a classifier with very high (e.g. 99.9%) detection rate and only moderate (e.g. 50%) false positive rate.

A. The Linear Asymmetric Classifier

For the purpose of the LAC, we assume that appropriate features have already been selected by other algorithms. For example, AdaBoost, FFS, or information theory-based method from [45] can be used to select features. We study the problem of how to find an optimal linear classifier for the node learning goal given these features. A detailed comparison to related work on similar problems will be presented in section V-D.

The problem can be formalized as follows. Let $\mathbf{x} \sim (\bar{\mathbf{x}}, \Sigma_{\mathbf{x}})$ denote that $\mathbf{x}$ is drawn from a distribution with mean $\bar{\mathbf{x}}$ and covariance matrix $\Sigma_{\mathbf{x}}$. Note that we do not assume any specific form of the distribution, except that its mean and covariance can be estimated from samples. We are dealing with binary classification problems with two classes $\mathbf{x} \sim (\bar{\mathbf{x}}, \Sigma_{\mathbf{x}}), \mathbf{y} \sim (\bar{\mathbf{y}}, \Sigma_{\mathbf{y}})$, which are fixed but unknown. Here $\mathbf{x}$ denotes a vector of feature values of a positive example and $\mathbf{y}$ denotes that of a negative example. Note that the notations in this section are different from previous sections. Both $\mathbf{x}$ and $\mathbf{y}$ are used to denote feature vectors, in order to emphasize the fact that the learning goals are asymmetric and to make presentations more clear. Class labels of training examples are obvious from the notation ($\mathbf{x}$ for positive and $\mathbf{y}$ for negative). We use $\mathbf{z}$ to denote an example with unknown class label. The linear classifier to be learned can be written as $H = (\mathbf{a}, b)$:

$$H(\mathbf{z}) = \begin{cases} +1 & \text{if } \mathbf{a}^T\mathbf{z} \geq b \\ -1 & \text{if } \mathbf{a}^T\mathbf{z} < b. \end{cases}$$

The asymmetric node learning goal is expressed as:

$$\begin{aligned} \max_{\mathbf{a}\neq 0, b} \quad & \Pr_{\mathbf{x}\sim(\bar{\mathbf{x}},\Sigma_{\mathbf{x}})}\{\mathbf{a}^T\mathbf{x} \geq b\} \\ \text{s.t.} \quad & \Pr_{\mathbf{y}\sim(\bar{\mathbf{y}},\Sigma_{\mathbf{y}})}\{\mathbf{a}^T\mathbf{y} \leq b\} = \beta. \end{aligned} \tag{3}$$

[2]We found that the criterion for automatically finding detection errors in [31] was too loose. This criterion yielded higher detection rates and lower false positive rates than manual counting.

[3]Since every node in the FFS cascade has a false positive rate lower than 50%, it will consume more bootstrapping non-face data than a cascade node trained by AdaBoost (with 50% false positive rate). Thus the FFS cascade consumes the bootstrapping non-face data more quickly and consequently has fewer number of nodes.

In general this problem has no closed-form solution. In this section, we will develop an approximate solution for it. Empirical results show that it is effective to set $\beta = 0.5$ for all cascade nodes. Thus, we will give a closed-form approximate solution when $\beta = 0.5$.

Note that an AdaBoost classifier is a linear combination of weak classifiers:

$$H(\mathbf{x}) = \text{sgn}(\Sigma_{t=1}^{T} a_t h_t(\mathbf{x}) - b) = \text{sgn}(\mathbf{a}^T \mathbf{h}(\mathbf{x}) - b) \quad (4)$$

in which $\mathbf{h}(\mathbf{x})$ is the vector of weak classifiers' outputs. Thus $H(\mathbf{x})$ is a linear classifier in the feature space defined by $\mathbf{h}(\mathbf{x})$. However, there is no guarantee that the $(\mathbf{a}, b)$ selected by AdaBoost will satisfy (3) for a given choice of β. The same argument applies to FFS. We seek a linear discriminant $(\mathbf{a}, b)$ which maximizes the node learning goal in (3).

The key idea to solve this learning problem is to use the cumulative distribution functions of $\mathbf{a}^T\mathbf{x}$ and $\mathbf{a}^T\mathbf{y}$ to replace the $\Pr\{\}$ function.

Let $\mathbf{x_a}$ denote the standardized version of $\mathbf{a}^T\mathbf{x}$ ($\mathbf{x}$ projected onto the direction of $\mathbf{a}$), i.e.

$$\mathbf{x_a} = \frac{\mathbf{a}^T(\mathbf{x} - \bar{\mathbf{x}})}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{x} \mathbf{a}}}, \quad (5)$$

obviously we have $\mathbf{x_a} \sim (0, 1)$. Let $\Psi_{\mathbf{x},\mathbf{a}}$ denotes the cumulative distribution function (c.d.f.) of $\mathbf{x_a}$, i.e.

$$\Psi_{\mathbf{x},\mathbf{a}}(b) = \Pr\{\mathbf{x_a} \le b\}. \quad (6)$$

$\mathbf{y_a}$ and $\Psi_{\mathbf{y},\mathbf{a}}$ are defined similarly as

$$\mathbf{y_a} = \frac{\mathbf{a}^T(\mathbf{y} - \bar{\mathbf{y}})}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{y} \mathbf{a}}}, \quad (7)$$

$$\Psi_{\mathbf{y},\mathbf{a}}(b) = \Pr\{\mathbf{y_a} \le b\}. \quad (8)$$

Thus, the constraint in (3) can be re-written as

$$\beta = \Pr\left\{\mathbf{a}^T\mathbf{y} \le b\right\} = \Pr\left\{\frac{\mathbf{a}^T(\mathbf{y} - \bar{\mathbf{y}})}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{y} \mathbf{a}}} \le \frac{b - \mathbf{a}^T\bar{\mathbf{y}}}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{y} \mathbf{a}}}\right\}$$
$$= \Psi_{\mathbf{y},\mathbf{a}}\left(\frac{b - \mathbf{a}^T\bar{\mathbf{y}}}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{y} \mathbf{a}}}\right),$$

which in turn gives an expression for the optimal value of b:

$$b = \mathbf{a}^T\bar{\mathbf{y}} + \Psi_{\mathbf{y},\mathbf{a}}^{-1}(\beta)\sqrt{\mathbf{a}^T \Sigma_\mathbf{y} \mathbf{a}} \quad (9)$$

where $\Psi_{\mathbf{y},\mathbf{a}}^{-1}$ is the inverse function of $\Psi_{\mathbf{y},\mathbf{a}}$. Note that $\Psi_{\mathbf{y},\mathbf{a}}^{-1}$ depends on both $\mathbf{y}$ and $\mathbf{a}$. Similarly, the objective function in (3) can be re-written as

$$1 - \Psi_{\mathbf{x},\mathbf{a}}\left(\frac{b - \mathbf{a}^T\bar{\mathbf{x}}}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{x} \mathbf{a}}}\right)$$

Using (9) to eliminate b and we obtain

$$1 - \Psi_{\mathbf{x},\mathbf{a}}\left(\frac{\mathbf{a}^T(\bar{\mathbf{y}} - \bar{\mathbf{x}}) + \Psi_{\mathbf{y},\mathbf{a}}^{-1}(\beta)\sqrt{\mathbf{a}^T \Sigma_\mathbf{y} \mathbf{a}}}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{x} \mathbf{a}}}\right).$$

Thus the constrained optimization problem (3) is equivalent to

$$\min_{\mathbf{a} \ne 0} \Psi_{\mathbf{x},\mathbf{a}}\left(\frac{\mathbf{a}^T(\bar{\mathbf{y}} - \bar{\mathbf{x}}) + \Psi_{\mathbf{y},\mathbf{a}}^{-1}(\beta)\sqrt{\mathbf{a}^T \Sigma_\mathbf{y} \mathbf{a}}}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{x} \mathbf{a}}}\right). \quad (10)$$

In (10), $\Psi_{\mathbf{x},\mathbf{a}}$ and $\Psi_{\mathbf{y},\mathbf{a}}^{-1}$ depend on the distributions of $\mathbf{x}$ and $\mathbf{y}$, in addition to the projection direction $\mathbf{a}$. Because we have no knowledge of these distributions, we cannot solve (10) analytically. We need to make some approximations to simplify it.

First, because the c.d.f. is monotonic, instead of minimizing the complex function $\Psi_{\mathbf{x},\mathbf{a}}(\cdot)$, we simply minimize its argument. In other words, we approximately solve (10) by solving

$$\min_{\mathbf{a} \ne 0} \frac{\mathbf{a}^T(\bar{\mathbf{y}} - \bar{\mathbf{x}}) + \Psi_{\mathbf{y},\mathbf{a}}^{-1}(\beta)\sqrt{\mathbf{a}^T \Sigma_\mathbf{y} \mathbf{a}}}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{x} \mathbf{a}}}, \quad (11)$$

or, equivalently,

$$\max_{\mathbf{a} \ne 0} \frac{\mathbf{a}^T(\bar{\mathbf{x}} - \bar{\mathbf{y}}) - \Psi_{\mathbf{y},\mathbf{a}}^{-1}(\beta)\sqrt{\mathbf{a}^T \Sigma_\mathbf{y} \mathbf{a}}}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{x} \mathbf{a}}}. \quad (12)$$

There are several motivations that inspire the transformation from (10) to (11).

- This transformation is *approximate* because in (10), the function $\Psi_{\mathbf{x},\mathbf{a}}$ depends on $\mathbf{a}$, while $\mathbf{a}$ also appears in the argument of $\Psi_{\mathbf{x},\mathbf{a}}$.
- If we assume that $\mathbf{a}^T\mathbf{x}$ is Gaussian for any $\mathbf{a}$, then $\mathbf{x_a}$ is the standard normal distribution. Under this assumption, $\Psi_{\mathbf{x},\mathbf{a}}$ does not depend on $\mathbf{a}$ any more, and (11) is exactly equivalent to (10).
- Empirical evidence in section V-B suggests that $\mathbf{a}^T\mathbf{x}$ is similar to a Gaussian. However, the tails of $\mathbf{a}^T\mathbf{x}$ gradually move away from an ideal normal distribution.
- At the same time, the fact that $\mathbf{a}^T\mathbf{x}$ is similar to a Gaussian in the face detection problem suggests that our approximation will work well in practice.

Second, we assume that the median value of the distribution $\mathbf{y_a}$ is close to its mean. This assumption is true for all symmetric distributions and is reasonable for many others. Under this assumption, we have $\Psi_{\mathbf{y},\mathbf{a}}^{-1}(0.5) \approx 0$. Thus for $\beta = 0.5$, (12) can be further approximated by

$$\max_{\mathbf{a} \ne 0} \frac{\mathbf{a}^T(\bar{\mathbf{x}} - \bar{\mathbf{y}})}{\sqrt{\mathbf{a}^T \Sigma_\mathbf{x} \mathbf{a}}}. \quad (13)$$

Note that if we can make the stronger assumption that $\mathbf{y_a}$ is a symmetric distribution and $\beta = 0.5$, then we have $\Psi_{\mathbf{y},\mathbf{a}}^{-1}(0.5) = 0$. The implication is that under these assumptions, (13) is exactly equivalent to the node learning goal in (3). We call the linear discriminant function determined by (13) the Linear Asymmetric Classifier (LAC) and use it in the cascade learning framework.

The form of (13) is similar to the Fisher Discriminant Analysis (FDA), which can be written as:

$$\max_{\mathbf{a} \ne 0} \frac{\mathbf{a}^T(\bar{\mathbf{x}} - \bar{\mathbf{y}})}{\sqrt{\mathbf{a}^T (\Sigma_\mathbf{x} + \Sigma_\mathbf{y}) \mathbf{a}}}. \quad (14)$$

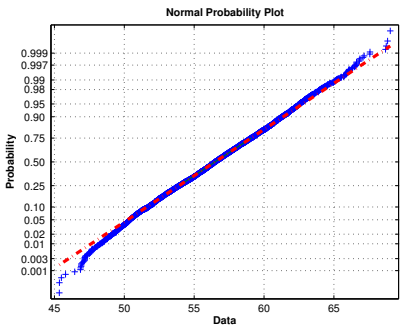
The only difference between FDA and LAC is that the pooled covariance matrix $\Sigma_\mathbf{x} + \Sigma_\mathbf{y}$ is replaced by $\Sigma_\mathbf{x}$. This analogy immediately gives us the solution to (13) as:

$$\mathbf{a}^* = \Sigma_\mathbf{x}^{-1}(\bar{\mathbf{x}} - \bar{\mathbf{y}}), b^* = \mathbf{a}^{*T}\bar{\mathbf{y}}, \quad (15)$$

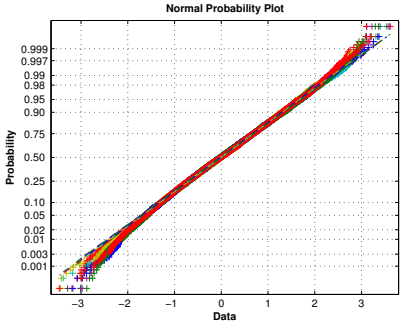
under the assumption that $\Sigma_\mathbf{x}$ is positive definite. In applications where $\Sigma_\mathbf{x}$ happens to be positive semi-definite, $\Sigma_\mathbf{x} + \lambda I$ can be used to replace $\Sigma_\mathbf{x}$, where λ is a small positive number.

B. Empirical support for approximate Gaussianity

We have shown that if for any $\mathbf{a}$, $\mathbf{a}^T\mathbf{x}$ is Gaussian and $\mathbf{a}^T\mathbf{y}$ is symmetric, then LAC is guaranteed to be the optimal linear classifier for the node learning goal. In this section we verify that these assumptions are valid in the cascade face detector.



(a) Single



(b) Overlapped

Fig. 4. Normality test for $\mathbf{a}^T\mathbf{x}$, in which $\mathbf{x}$ is a feature vector extracted from face data, and $\mathbf{a}$ is drawn from the uniform distribution $[0\ 1]^T$. Fig. 4(a) shows result for a single $\mathbf{a}$. Fig. 4(b) shows overlapped results for 10 different $\mathbf{a}$'s. Data in Fig. 4(b) are centered (i.e. means are subtracted).

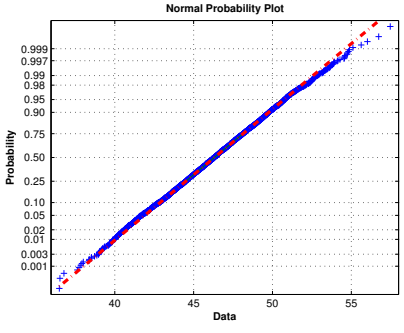


Fig. 5. Normality test for $\mathbf{a}^T\mathbf{y}$, in which $\mathbf{y}$ is a feature vector extracted from non-face data, and $\mathbf{a}$ is drawn from the uniform distribution $[0\ 1]^T$.

Probability theory shows that $\mathbf{x}$ is Gaussian if and only if $\mathbf{a}^T\mathbf{x}$ is Gaussian for all $\mathbf{a}$'s. Obviously $\mathbf{x}$ is not Gaussian since all of its components are binary random variables. However, experiments show that $\mathbf{a}^T\mathbf{x}$ is approximately Gaussian for most reasonable instantiations of $\mathbf{a}$. Fig. 4(a) shows the normal probability plot of $\mathbf{a}^T\mathbf{x}$ for $\mathbf{a}$ randomly drawn from the uniform distribution. $\mathbf{a}^T\mathbf{x}$ fits closely to a normal distribution, only with small deviations at the tails. Fig. 4(b) shows that $\mathbf{a}^T\mathbf{x}$ is approximately Gaussian for all different $\mathbf{a}$'s in our experiments.

Experiments also show that $\mathbf{a}^T\mathbf{y}$ fits nearly exactly to a Gaussian (refer to Fig. 5). We tested the normality of $\mathbf{a}^T\mathbf{y}$ for many non-face training data sets and different instantiations of $\mathbf{a}$, and $\mathbf{a}^T\mathbf{y}$ always fits a Gaussian distribution. Since centered Gaussian distributions are symmetric, we can safely assume that $\mathbf{a}^T\mathbf{y}$ is symmetric for all $\mathbf{a}$'s.

TABLE I
SUMMARY OF THE KURTOSIS OF $\mathbf{a}^T\mathbf{x}$ AND $\mathbf{a}^T\mathbf{y}$ FOR 1000 DIFFERENT $\mathbf{a}$'S RANDOMLY DRAWN FROM THE UNIFORM DISTRIBUTION $[0\ 1]^T$.

	Kurt($\mathbf{a}^T\mathbf{x}$)	Kurt($\mathbf{a}^T\mathbf{y}$)
mean	-0.23	-0.02
standard deviation	0.05	0.07
min	-0.38	-0.06
max	-0.22	0.23

The normal probability plot is a way to visually examine whether a distribution is normal or not. The kurtosis of a one dimensional distribution provides a numerical evaluation of the normality, since the kurtosis of a normal distribution is 0 and nearly all non-Gaussian distributions have non-zero kurtosis. Table I summarizes the kurtosis value of $\mathbf{a}^T\mathbf{x}$ and $\mathbf{a}^T\mathbf{y}$ for 1000 different $\mathbf{a}$'s, all of which drawn randomly from the uniform distribution $[0\ 1]^T$. The result in table I confirms that for most reasonable $\mathbf{a}$, $\mathbf{a}^T\mathbf{x}$ is approximately Gaussian and $\mathbf{a}^T\mathbf{y}$ fits very close to a normal distribution.

C. LAC in the cascade framework

When the rectangle features are used, the vector of their corresponding weak classifiers' output $\mathbf{h}(\mathbf{x})$ is often used as features (refer to (4)). The rectangle features can be selected by any feature selection method, e.g. AdaBoost, AsymBoost, or FFS. The abstract cascade learning algorithm remains unchanged (Algorithm 1), while the node learning algorithm is replaced by a feature selector plus LAC. The new node learning algorithm is shown in Algorithm 4.

Algorithm 4 LAC as a new node learning algorithm

1: {Given a training set composed of positive examples $\{\mathbf{x}_i\}_{i=1}^{n_\mathbf{x}}$ and negative examples $\{\mathbf{y}_i\}_{i=1}^{n_\mathbf{y}}$, and a set of rectangle features.}
2: {Given a feature selection method $\mathcal{F}$}
3: {The output is a classifier with false positive rate 0.5}
4: Select T weak classifiers $\mathbf{h} = (h_1, h_2, \ldots, h_T)$ using $\mathcal{F}$, where $h_i(\mathbf{z}) = \mathrm{sgn}(\mathbf{z}^T\mathbf{m}_i - \tau_i)$
5: For each training example, build a feature vector $\mathbf{h}(\mathbf{z}) = (h_1(\mathbf{z}), h_2(\mathbf{z}), \ldots, h_T(\mathbf{z}))$.
6: Estimate the mean and covariance:

$$\bar{\mathbf{x}} = \frac{\sum_{i=1}^{n_\mathbf{x}} \mathbf{h}(\mathbf{x}_i)}{n_\mathbf{x}}, \bar{\mathbf{y}} = \frac{\sum_{i=1}^{n_\mathbf{y}} \mathbf{h}(\mathbf{y}_i)}{n_\mathbf{y}},$$

$$\Sigma_\mathbf{x} = \frac{\sum_{i=1}^{n_\mathbf{x}} (\mathbf{h}(\mathbf{x}_i) - \bar{\mathbf{x}})(\mathbf{h}(\mathbf{x}_i) - \bar{\mathbf{x}})^T}{n_\mathbf{x}},$$

$$\Sigma_\mathbf{y} = \frac{\sum_{i=1}^{n_\mathbf{y}} (\mathbf{h}(\mathbf{y}_i) - \bar{\mathbf{y}})(\mathbf{h}(\mathbf{y}_i) - \bar{\mathbf{y}})^T}{n_\mathbf{y}}.$$

7: Applying (15) to get

$$\mathbf{a} = \Sigma_\mathbf{x}^{-1}(\bar{\mathbf{x}} - \bar{\mathbf{y}}), b = \mathbf{a}^T\bar{\mathbf{y}}$$

8: The output is a classifier

$$H(\mathbf{z}) = \mathrm{sgn}\left(\sum_{t=1}^{T} \mathbf{a}_t\mathbf{h}_t(\mathbf{z}) - b\right) = \mathrm{sgn}\left(\mathbf{a}^T\mathbf{h}(\mathbf{z}) - b\right) \quad (16)$$

When we want to apply FDA instead of LAC, we replace (15)

with the following FDA solution:

$$\mathbf{a} = \left(\Sigma_{\mathbf{x}} + \Sigma_{\mathbf{y}}\right)^{-1} \left(\bar{\mathbf{x}} - \bar{\mathbf{y}}\right). \tag{17}$$

It is tempting to use the integral feature values $\mathbf{z}^T\mathbf{m}_k - \tau_k$ directly as features in the LAC or FDA algorithm. However, since $\mathbf{z}^T\mathbf{m}_t - \tau_t$ is a linear function of the input $\mathbf{z}$, $H(\mathbf{z}) = \operatorname{sgn}\left(\sum_{t=1}^{T} \mathbf{a}_t(\mathbf{z}^T\mathbf{m}_t - \tau_t) - b\right)$ is still a linear discriminant function and will not work for problems that are not linearly separable. Since most visual classification tasks are not linearly separable, we need to inject some non-linearity into the linear asymmetric classifier. The sgn function in $\operatorname{sgn}(\mathbf{z}^T\mathbf{m}_t - \tau_t)$ introduces the necessary non-linearity into the features.

Both AdaBoost and LAC have the same form $H(\mathbf{z}) = \operatorname{sgn}\left(\mathbf{a}^T\mathbf{h}(\mathbf{z}) - b\right)$ and they share the same feature vector $\mathbf{h}(\mathbf{z})$. The only difference between these two classifiers are the parameters of the linear discriminant $(\mathbf{a}, b)$. In AdaBoost, a_i is chosen in step i of the AdaBoost procedure to minimize a margin-based cost function [47]. This is a greedy procedure and a_i is never changed after its value is determined. Furthermore, AdaBoost does not take into account the fact that the two classes are asymmetric. The linear asymmetric classifier, on the contrary, is a global procedure to seek the optimal vector $\mathbf{a}$ which optimizes the asymmetric loss in (3).

Viola and Jones proposed AsymBoost [14] to accommodate the asymmetry. In AsymBoost, sample weights were updated using

$$D_{t+1}(i) = \frac{D_t(i)\exp\left(-y_i h_t(x_i)\right)\exp\left(y_i \log\sqrt{k}\right)}{Z_t}$$

instead of the standard AdaBoost updating rule

$$D_{t+1}(i) = \frac{D_t(i)\exp\left(-y_i h_t(x_i)\right)}{Z_t}.$$

The extra term $\exp\left(y_i \log\sqrt{k}\right)$ causes the algorithm to gradually pay more attention to positive samples in each round of boosting, in which k is a parameter representing the level of asymmetry. However, the resulting linear discriminant $(\mathbf{a}, b)$ is determined in the same way as ordinary AdaBoost.

D. Comparison to previous work

There are other methods that are similar to the form of (13), e.g. [27], [48], [49]. Researchers have also presented methods that are related to the node learning goal. However, the node learning goal was not explicitly defined and solved in these methods. In this section we will examine the relationship between the proposed LAC and other related classifiers.

Applying the Chebyshev inequality to (10), we can obtain a related approximation to (10):

$$\max_{\mathbf{a}\neq 0} \frac{\mathbf{a}^T\left(\bar{\mathbf{x}} - \bar{\mathbf{y}}\right) - \kappa(\beta)\sqrt{\mathbf{a}^T\Sigma_{\mathbf{y}}\mathbf{a}}}{\sqrt{\mathbf{a}^T\Sigma_{\mathbf{x}}\mathbf{a}}}, \tag{18}$$

where $\kappa(\beta) = \sqrt{\frac{\beta}{1-\beta}}$. This approximation is used in the Biased Minimax Probability Machine (BMPM) [27]. (18) is similar to our objective (12). However, BMPM is derived under the minimax principle [48], aiming at minimizing a worst case lower bound of the objective function (10). LAC, on the other hand, uses domain knowledge to eliminate the dependency on conditional distributions, and provides an approximate solution to (10). In the special case when $\mathbf{y}$ is Gaussian, BMPM gives the same solution as LAC. [27, Theorem 5] also gives an interesting hint of why $\mathbf{a}^T\mathbf{x}$ converges to Gaussian in high dimensional spaces. In short, BMPM solves a more general class of problems than LAC (β is not confined to 0.5), while LAC incorporates more domain knowledge and gives better solutions for a more general set of distributions of $\mathbf{y}$ (with equal mean and median for $\mathbf{a}^T\mathbf{y}$).

Another related objective function comes from the Maximum Rejection Classifier (MRC) [16], which can be written:

$$\max_{\mathbf{a}\neq 0} \frac{\left(\mathbf{a}^T\bar{\mathbf{y}} - \mathbf{a}^T\bar{\mathbf{x}}\right)^2 + \mathbf{a}^T\Sigma_{\mathbf{y}}\mathbf{a}}{\mathbf{a}^T\Sigma_{\mathbf{x}}\mathbf{a}}. \tag{19}$$

The solution of (19) requires solving a generalized eigenvalue problem. The intuition behind (19) is to make the overlap between the projections $\mathbf{x_a}$ and $\mathbf{y_a}$ small. The derivation of (19) in [16] treats the two classes equally. Asymmetry in the MRC framework results from the fact that the two classes have different prior probabilities with $P(\mathbf{x}) \ll P(\mathbf{y})$. However, the effect of the prior on $\mathbf{y}$ is reduced quickly as the stage-wise rejection process continues. After a few rejections, $P(\mathbf{x})$ is no longer negligible in comparison to $P(\mathbf{y})$. Under such conditions, (19) is not an appropriate objective function.

A final comparison can be made between LAC and FDA. FDA and LAC both have their own merits and drawbacks. We have shown that when $\mathbf{x_a}$ is normal, $\mathbf{y_a}$ is symmetric, and $\beta = 0.5$, LAC is indeed the optimal solution to the node learning goal. However, when these assumptions are broken, LAC may be suboptimal. The intuition in FDA is to maximize the (normalized) separation between the two class means. It does not minimize the error rate or the node learning goal. The advantage of FDA is that it does not have constraints – performance will be reasonably good if the class means are far apart. If we assume that $\mathbf{x}$ and $\mathbf{y}$ have equal covariance matrices, then LAC is equivalent to FDA.

E. Experiments on LAC

We tested the performance of the linear asymmetric classifier on both a synthetic data set and the face detection task. In the synthetic data set, LAC is compared against BMPM, MRC and FDA. For detection of faces, the cascade framework is used. Three feature selectors are used: AdaBoost, AsymBoost, and FFS. We compare three different ways to determine the linear discriminant $(\mathbf{a}, b)$ after the features are selected. The first method is to use the weights $\mathbf{a}$ and threshold b found by AdaBoost or AsymBoost; the second method uses the proposed linear asymmetric classifier; the third method uses Fisher Discriminant Analysis. We use "X+Y" to denote the methods used in experiment, e.g. AdaBoost+LAC means that the features are selected by AdaBoost and the linear discriminant function is trained by LAC.

1) Results on Synthetic Data: Fig. 6 gives some intuition of the difference between LAC and FDA. The positive data is drawn from a normal distribution with mean $(1.23, 1.23)$ and covariance $[10\ 0; 0\ 1]$. The negative data is drawn from a normal distribution with mean $(0, 0)$ and covariance $[1\ 0; 0\ 10]$. LAC is guaranteed to be optimal in this data set. It rejects 50% of the negative data, while keeping almost all positive data. On the contrary, FDA returns a boundary that also rejects 50% of the negative data, but more than 13% of the positive data are rejected.

We also tested LAC on data sets where its assumptions were broken. A synthetic data set was generated using the following steps. Three distributions were created as: $\mathbf{d}_i = A_i\mathbf{c}_i + \mathbf{m}_i, i = 1, 2, 3$, where $\mathbf{c}_i \sim N(0, I), \mathbf{m}_i \sim N(0, 0.1I)$, and the elements

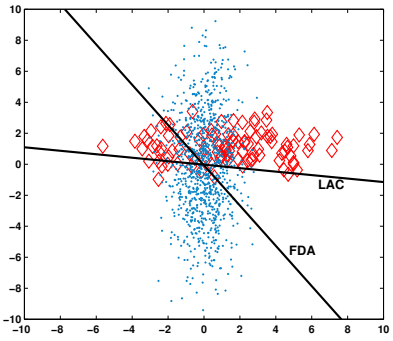


Fig. 6. Comparing LAC and FDA on synthetic data set when both $\mathbf{x}$ and $\mathbf{y}$ are Gaussians.

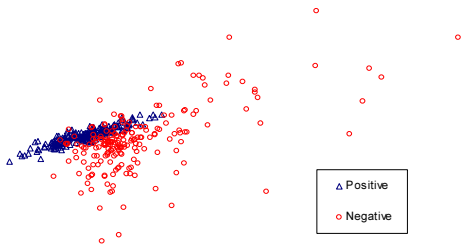


Fig. 7. Example of a synthetic data set where $\mathbf{y}$ is not symmetric.

of A were drawn randomly from a uniform distribution in $[0\ 1]$. The positive examples $\mathbf{x}$ were drawn from $\mathbf{X} = \mathbf{d}_1$, and negative examples $\mathbf{y}$ were drawn from $\mathbf{Y} = \mathbf{d}_2^2 - \mathbf{d}_3^2$. This choice produced a $\mathbf{Y}$ which is not symmetric, and has a reasonable overlap with $\mathbf{X}$. One example of such a data set is shown in Fig. 7. The training and test sets both contain 1000 samples, including 500 positive and 500 negative samples. Four linear discriminant methods (LAC, FDA, MRC, and BMPM) were compared.[4] In each method, we determined the projection direction $\mathbf{a}$ using the corresponding method. The threshold b was determined such that on the training set the false positive rate was 50%. For every method, the experiments were repeated 100 times. The averaged test set accuracy on both classes are reported in Table II.

Although the negative class is not symmetric, LAC gives the best performance, followed closely by FDA. Two-tailed paired t-test shows that there is no significant difference between LAC and FDA. Both the difference between LAC and MRC, and the difference between LAC and BMPM are significant, at the 0.01 level.

In cascaded classifiers, the imbalance between the positive class and the negative class is absorbed by the cascade structure. In each node of a cascade, balanced training sets are usually used. This is why we used a balanced training set in the above synthetic data set. We also tested the performance of these classifiers on imbalanced training set. Two extra sets of experiments were performed. The training sets still had 500 positive examples, but the negative class had 1000 and 1500 examples, respectively. The examples were drawn from the same distributions as described above. All four classifiers' performances remained approximately the same, despite the increase in the number of negative training examples. Thus detailed error rates are not presented. Under both imbalance levels, LAC performed about the same as FDA, and both LAC and FDA were better than MRC and BMPM.

[4]We used the Matlab toolbox for BMPM from `http://www.cse.cuhk.edu.hk/~miplab/mempm_toolbox/`.

TABLE II
RESULTS ON SYNTHETIC DATA SET.

Classifier	Positive Accuracy	Negative Accuracy
LAC	96.11	50.04
FDA	95.12	49.96
MRC	90.19	49.96
BMPM	87.99	50.26

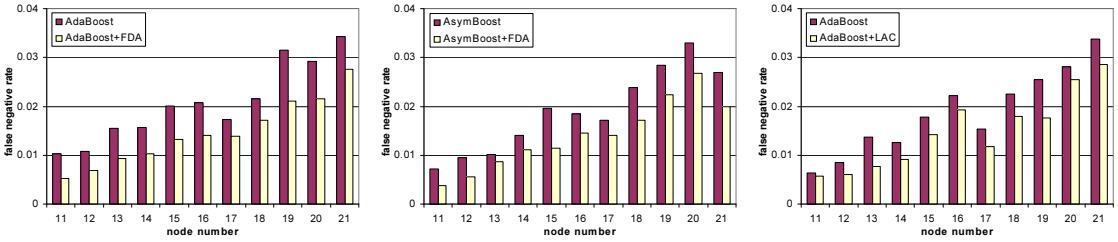
2) Results on Face Detection: For face detection, we trained 9 different cascades, using the three feature selectors (AdaBoost, AsymBoost, and FFS) and three linear discriminant functions (using weights provided by the feature selector, LAC, or FDA). Each cascade has 21 nodes, except that the AsymBoost+LAC cascade has 22 nodes and the FFS cascade has 17 nodes. We require that every node have 50% false positive rates and the cascade training process is terminated when there are not enough non-face patches to bootstrap. In order to make the face detector run at video speed, the first node uses only 7 features. We use more features as the node index increases (the last node used 200 features).

We consider two types of performance measures: node and cascade. The node performance measure is the classifiers' ability to achieve the node learning goal. Given a trained cascade, each node has an associated training set, which is generated by the bootstrapping process. We collected all such training sets from the 9 trained cascades. Given one such training set, different algorithms are required to achieve the criteria in (3). Their performance is evaluated using the validation set. The node performance measure is useful because it directly compares the ability of each method to achieve the node learning goal. The cascade performance measure compares the performance of the entire cascade. The performance of a cascade depends upon more than just the classifier that is used to train the nodes. The background data bootstrapping step and post processing step in face detection also have significant effect on each cascade's performance. The cascade performance measure is evaluated using the MIT+CMU benchmark test set.

The node comparison results are shown in Fig. 8. We did not perform the node comparison for the FFS algorithm, since FFS enforced all weights to be 1 and was not in the same hypothesis space as AdaBoost, FDA, or LAC. We collected the training set from the remaining 6 cascades, using the other two feature selectors (AdaBoost and AsymBoost) and three linear discriminant functions. The AdaBoost cascade is the same one as used in section IV-D.[5]

We are able to observe the effects of using FDA or LAC to train a linear discriminant function instead of using the values provided by the AdaBoost (or AsymBoost) algorithm. From the results in Fig. 8, it is obvious that both FDA and LAC can greatly reduce the false negative rates (i.e. increase the detection rates). In Fig. 8(a), averaged over the 11 nodes shown, AdaBoost+FDA reduces the false negative rates by 31.5% compared to AdaBoost, while in Fig. 8(c) AdaBoost+LAC reduces it by 22.5%. When AsymBoost

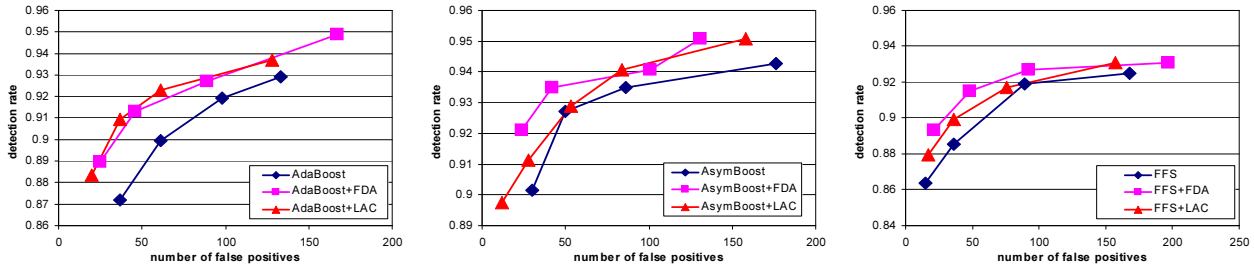
[5]The source code for training a cascade using methods described in this paper is available online at `http://www.cc.gatech.edu/~wujx`. We also provide trained cascades, demo executables, and a video showing testing results.



(a) AdaBoost vs FDA (b) AsymBoost vs FDA (c) AdaBoost vs LAC

(d) AsymBoost vs LAC (e) AdaBoost vs FDA & LAC (f) AsymBoost vs FDA & LAC

Fig. 8. Experiments comparing different linear discriminant functions. The y axis shows the false negative rate when $\beta = 0.5$. In 8(a), training sets are collected from the AdaBoost+FDA cascades' node 11 to 21 (x axis shows the node number). AdaBoost and AdaBoost+FDA are compared using these training sets. Similarly, 8(b)-8(f) used training sets from the AsymBoost+FDA, AdaBoost+LAC, AsymBoost+LAC, AdaBoost, and AsymBoost cascades respectively. We do not show results when the data set index is less than 11, for space constraint.



(a) AdaBoost vs FDA & LAC (b) AsymBoost vs FDA & LAC (c) FFS vs FDA & LAC

Fig. 9. Experiments comparing cascade performances on the MIT+CMU test set. The x axis is the number of false positives. The y axis is the detection rate.

is used as the feature selector, the reductions are 27.3% and 17.3%, respectively. In Fig. 8(a) to 8(d), training sets came from FDA or LAC cascades. We also compare node performance when the training sets came from the AdaBoost or AsymBoost cascade. Results are shown in Fig. 8(e) and 8(f). Both FDA and LAC work better than the original AdaBoost and AsymBoost.

Cascade comparison results are shown in Fig. 9. The x axis is the total number of false positives in the MIT+CMU test set. The y axis is the detection rate. Fig. 9(a)-9(c) show the results when AdaBoost, AsymBoost, or FFS is used as the feature selector, respectively. These ROC curves show that both FDA and LAC have significant advantages over the linear discriminant provided by AdaBoost, AsymBoost, and FFS. It coincides well with the node performances in Fig. 8.

Another way to interpret Fig. 9 is to compare the number of false positives at a same detection rate. LAC can greatly reduce false positives. For example, in Fig. 9(a), averaged over the range of possible detection rates, AdaBoost+LAC reduces the number of false positives by 36.1% compared with AdaBoost.

3) Discussions: Three observations from the experimental results are worthy of further discussions.

First, the improvement of LAC over AsymBoost is smaller

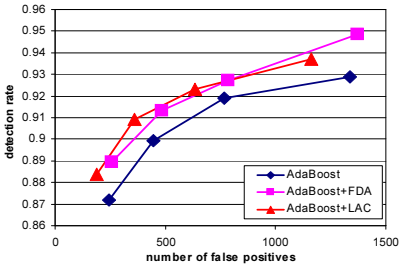


Fig. 10. Experiments comparing cascade performances on the MIT+CMU test set. The x axis is the number of false positives without post-processing.

than that of LAC over AdaBoost. Our conjecture is that since AsymBoost already takes into account the asymmetry when it selects features, LAC has smaller space to improve.

Second, the error reduction effects of FDA or LAC in Fig. 8 are more significant than those in Fig. 9. We conjecture that the background data bootstrapping and post processing remove part of the error reducing effects.

The effect of post-processing can be studied by evaluating the cascade detector without performing the post-processing step.

The results without post-processing when AdaBoost was used are shown in Fig. 10. Comparing Fig. 9(a) and 10, we find that the post-processing step does not change the relative performances of different algorithms, since these two figures are very similar to each other. However, post-processing can greatly reduce the absolute number of false positives.

Third, we find that FDA works better than LAC in a few cases. LAC is derived under the assumptions that $\mathbf{x_a}$ is Gaussian and $\mathbf{y_a}$ is symmetric. However, from Fig. 4(a) and Fig. 5 we find that although $\mathbf{y_a}$ is always symmetric, $\mathbf{x_a}$ slightly deviates from the Gaussian distribution at the tails. This might be the reason why LAC does not perform the best in some cases.

VI. Conclusions

We have presented a new approach to designing a node classifier in a cascade detector. Previous methods used machine learning algorithms that simultaneously select features and form ensemble classifiers. We analyzed the asymmetries inherent in the face detection problem and why these asymmetries make the problem difficult. We then argued that if we decouple the feature selection step from the classifier design step, we have the freedom to use different feature selection methods. More importantly, we have the freedom to design an ensemble classifier that explicitly addresses the asymmetries in its learning goal. We proposed FFS as the new feature selection method, and LAC as the new classifier.

The contributions of this paper can be summarized into three points. The first contribution is an analysis of the cascade detector. Three types of asymmetries are categorized: uneven class priors, goal asymmetry, and the unequal complexity of the positive and negative class. We argued that these asymmetries are the characteristics of the face detection problem that make it hard to solve. A literature survey of computer vision and machine learning researches to deal with these asymmetries are also provided.

The second contribution is Forward Feature Selection, the feature selection part of our new decoupled node learning algorithm. We also propose a faster implementation method for the AdaBoost algorithm. On one hand, FFS provides an alternative feature selection method. The classifier formed by voting the FFS features has similar accuracy as the AdaBoost method. On the other hand, FFS is computationally attractive. FFS is two orders of magnitude faster than the naive implementation of AdaBoost. FFS is also 2.5 to 3.5 times faster than the faster implementation of AdaBoost, but only requires about 3% memory usage as that of AdaBoost.

The third contribution is Linear Asymmetric Classifier, the classifier design part of the decoupled node learning algorithm. The asymmetries are taken care of by LAC as a well-defined constrained optimization problem. By incorporating domain knowledge (or assumptions about the data), LAC solves this complex optimization problem approximately in closed form and a computationally efficient manner. The derivation of LAC also gives some hints to interesting characteristics of the face detection data sets. Experiments on both synthetic and MIT+CMU benchmark show that LAC can greatly reduce the errors. In addition, we also applied Fisher's Discriminant Analysis to features extracted by AdaBoost and got improved results.

Despite its effectiveness, there are limitations in our node learning algorithm. We describe these limitations and propose some future work that are possible ways to address these limitations.

- One of the conditions for LAC to be optimal is $\beta = 0.5$. However, in applications other than face detection, $\beta \neq 0.5$ may be required. Other asymmetric learning methods, such as BMPM [27], might be used in these cases.
- LAC is a linear classifier. It is possible to extend LAC to a non-linear classifier (for example, use the kernel method). Will this non-linear extension improve the classifier's performance?
- We observed that FDA outperformed LAC in some cases and we conjecture that this is because $\mathbf{a}^T\mathbf{x}$ is only approximately Gaussian. Is it possible to design a new feature selection method which guarantees $\mathbf{a}^T\mathbf{x}$ to be normally distributed?
- It is also desirable to have a feature selection method that take into account the asymmetric learning goal in a principled way.

Acknowledgment

This work was supported by US National Science Foundation (NSF) Grants ITR-0205507.

References

[1] K. Sung and T. Poggio, "Example-based learning for view-based human face detection," *IEEE Trans. PAMI*, vol. 20, no. 1, pp. 39–51, 1998.
[2] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Trans. PAMI*, vol. 20, no. 1, pp. 23–38, 1998.
[3] H. Schneiderman and T. Kanade, "A statistical model for 3D object detection applied to faces and cars," in *Proc. CVPR*, 2000, pp. 746–751.
[4] M.-H. Yang, D. Roth, and N. Ahuja, "A SNoW-based face detector," in *NIPS 12*, 2000, pp. 862–868.
[5] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," in *Proc. CVPR*, 1997, pp. 130–136.
[6] F. Fleuret and D. Geman, "Coarse-to-fine face detection," *IJCV*, vol. 41, no. 1-2, pp. 85–107, 2001.
[7] P. Viola and M. Jones, "Robust real-time face detection," *IJCV*, vol. 57, no. 2, pp. 137–154, 2004.
[8] Y. Amit and D. Geman, "A computational model for visual selection," *Neural Computation*, vol. 11, no. 7, pp. 1691–1715, 1999.
[9] R. Osadchy, M. Miller, and Y. LeCun, "Synergistic face detection and pose estimation with energy-based model," in *NIPS 17*, 2005, pp. 1017–1024.
[10] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods," *The Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686, 1998.
[11] J. Wu, J. Rehg, and M. Mullin, "Learning a rare event detection cascade by direct feature selection," in *NIPS 16*, 2004, pp. 1523–1530.
[12] J. Wu, M. Mullin, and J. Rehg, "Linear asymmetric classifier for cascade detectors," in *Proc. 22nd International Conference on Machine Learning*, 2005, pp. 993–1000.
[13] G. M. Weiss, "Mining with rarity: a unifying framework," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 7–19, 2004.
[14] P. Viola and M. Jones, "Fast and robust classification using asymmetric AdaBoost and a detector cascade," in *NIPS 14*, 2002, pp. 1311–1318.
[15] M.-H. Yang, D. J. Kriegman, and N. Ahujua, "Detecting faces in images: a survey," *IEEE Trans. PAMI*, vol. 24, no. 1, pp. 34–58, 2002.
[16] M. Elad, Y. Hel-Or, and R. Keshet, "Pattern detection using a maximal rejection classifier," *Pattern Recognition Letters*, vol. 23, no. 12, pp. 1459–1471, 2002.
[17] D. Keren, M. Osadchy, and C. Gotsman, "Antifaces: A novel, fast method for image detection," *IEEE Trans. PAMI*, vol. 23, no. 7, pp. 747–761, 2001.
[18] S. Romdhani, P. Torr, B. Schoelkopf, and A. Blake, "Computationally efficient face detection," in *Proc. ICCV*, 2001, pp. 695–700.
[19] B. Heisele, T. Serre, S. Mukherjee, and T. Poggio, "Feature reduction and hierarchy of classifiers for fast object detection in video images," in *Proc. CVPR*, 2001, pp. II:18–24.
[20] S. Baker and S. Nayar, "Pattern rejection," in *Proc. CVPR*, 1996, pp. 544–549.
[21] C. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Proc. ICCV*, 1998, pp. 555–562.

[22] H. Sahbi and D. Geman, "A hierarchy of support vector machines for pattern detection," *Journal of Artificial Intelligence Research*, vol. 7, pp. 2087–2123, 2006.
[23] R. E. Schapire, Y. Singer, and A. Singhal, "Boosting and rocchio applied to text filtering," in *SIGIR*, 1998, pp. 215–223.
[24] G. J. Karakoulas and J. Shawe-Taylor, "Optimizing classifiers for imbalanced training sets," in *NIPS 11*, 1999, pp. 253–259.
[25] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan, "Adacost: Misclassification cost-sensitive boosting," in *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999, pp. 97–105.
[26] K. M. Ting, "A comparative study of cost-sensitive boosting algorithms," in *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000, pp. 983–990.
[27] K. Huang, H. Yang, I. King, M. Lyu, and L. Chan, "The minimum error minimax probability machine," *Journal of Machine Learning Research*, vol. 5, pp. 1253–1286, 2004.
[28] G. Blanchard and D. Geman, "Sequential testing designs for pattern recognition," *Annals of Statistics*, vol. 33, no. 3, pp. 1155–1202, 2005.
[29] R. L. Rivest, "Learning decision lists," *Machine Learning*, vol. 2, no. 3, pp. 229–246, 1987.
[30] M. Anthony, "Generalization error bounds for threshold decision lists," *Journal of Machine Learning Research*, vol. 5, pp. 189–217, 2004.
[31] R. Lienhart, A. Kuranov, and V. Pisarevsky, *Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection*, ser. Lecture Notes in Computer Science, 2003, vol. 2781, pp. 297–304.
[32] K. Levi and Y. Weiss, "Learning object detection from a small number of examples: the importance of good features," in *Proc. CVPR*, 2004, pp. II:53–60.
[33] A. Torralba, K. Murphy, and W. Freeman, "Sharing features: Efficient boosting procedures for multiclass object detection," in *Proc. CVPR*, 2004, pp. II:762–769.
[34] R. Xiao, L. Zhu, and H.-J. Zhang, "Boosting chain learning for object detection," in *Proc. ICCV*, 2003, pp. 709–715.
[35] S. Li, Z. Zhang, H.-Y. Shum, and H. Zhang, "FloatBoost learning for classification," in *NIPS 15*, 2003, pp. 993–1000.
[36] C. Liu and H.-Y. Shum, "Kullback-leibler boosting," in *Proc. CVPR*, 2003, pp. I:587–594.
[37] J. Sun, J. M. Rehg, and A. F. Bobick, "Automatic cascade training with perturbation bias," in *Proc. CVPR*, 2004, pp. II:276–283.
[38] M. J. Jones and P. Viola, "Fast multi-view face detection," TR2003-96, MERL, Tech. Rep., 2003.
[39] X. Chen and A. Yuille, "Detecting and reading text in natural scenes," in *Proc. CVPR*, 2004, pp. II:366–373.
[40] O. Carmichael and M. Hebert, "Shape-based recognition of wiry objects," in *Proc. CVPR*, 2003, pp. II:401–408.
[41] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," in *Proc. ICCV*, 2003, pp. 734–741.
[42] A. R. Webb, *Statistical Pattern Recognition*. New York: Oxford University Press, 1999.
[43] S. Avidan and M. Butman, "The power of feature clustering: An application to object detection," in *NIPS 17*, 2005, pp. 57–64.
[44] L. Ren, G. Shakhnarovich, J. K. Hodgins, H. Pfister, and P. A. Viola, "Learning silhouette features for control of human motion," *ACM Trans. Graph.*, vol. 24, no. 4, pp. 1303–1331, 2005.
[45] F. Fleuret, "Fast binary feature selection with conditional mutual information," *Journal of Machine Learning Research*, vol. 5, pp. 1531–1555, 2004.
[46] S. Brubaker, J. Wu, J. Sun, M. Mullin, and J. Rehg, "On the design of cascades of boosted ensembles for face detection," GVU center, Georgia Institute of Technology, Tech. Rep. GIT-GVU-05-28, 2005.
[47] L. Mason, J. Baxter, P. L. Bartlett, and M. Frean, "Functional gradient techniques for combining hypotheses," in *Advances in Large Margin Classifiers*. MIT Press, 2000, pp. 221–246.
[48] G. Lanckriet, L. E. Ghaoui, C. Bhattacharyya, and M. Jordan, "A robust minimax approach to classification," *Journal of Machine Learning Research*, vol. 3, pp. 555–582, 2002.
[49] S.-J. Kim, A. Magnani, S. Samar, S. Boyd, and J. Lim, "Pareto optimal linear classification." in *Proc. ICML*, 2006, pp. 473–480.

Jianxin Wu received the BS degree and ME degree in computer science, both from the Nanjing University, China. He is currently working toward the PhD degree in Georgia Institute of Technology under the supervision of Dr. James M. Rehg. His research interests are computer vision and machine learning.

S. Charles Brubaker received the BS degree in engineering from Swarthmore College with high honors in 2002 and the MS degree from the Georgia Institute of Technology in 2006. His research interests include computer vision, computational learning theory, and algorithms.

Matthew D. Mullin received a BA degree in Mathematics from Princeton University in 1990. He is currently a research scientist at the Georgia Institute of Technology in the College of Computing. His research interests include computer vision and machine learning.

James M. Rehg received his PhD degree in Electrical and Computer Engineering from the Carnegie Mellon University. He is an Associate Professor in the College of Computing at the Georgia Institute of Technology. He is a member of the Graphics, Visualization, and Usability Center and co-directs the Computational Perception Lab. His research interests are computer vision, robotics, machine learning, and computer graphics. He is a member of the IEEE.