

Boosting Adaptive Linear Weak Classifiers for Online learning and Tracking

Toufiq Parag
Dept of Computer Science
Rutgers University
Piscataway, NJ 08854

Fatih Porikli
Mitsubishi Electric Research Labs
Cambridge, MA 02139

Ahmed Elgammal
Dept of Computer Science
Rutgers University
Piscataway, NJ 08854

Abstract

Online boosting methods have recently been used successfully for tracking, background subtraction etc. Conventional online boosting algorithms emphasize on interchanging new weak classifiers/features to adapt with the change over time. We are proposing a new online boosting algorithm where the form of the weak classifiers themselves are modified to cope with scene changes. Instead of replacement, the parameters of the weak classifiers are altered in accordance with the new data subset presented to the online boosting process at each time step. Thus we may avoid altogether the issue of how many weak classifiers to be replaced to capture the change in the data or which efficient search algorithm to use for a fast retrieval of weak classifiers. A computationally efficient method has been used in this paper for the adaptation of linear weak classifiers. The proposed algorithm has been implemented to be used both as an online learning and a tracking method. We show quantitative and qualitative results on both UCI datasets and several video sequences to demonstrate improved performance of our algorithm.

1. Introduction

Studies on online learning algorithms originated in computational learning community. The initial algorithms train several experts based on the labeled samples arriving sequentially and later combine the predictions of these experts to categorize any new example. The algorithms popular to the machine learning researchers like the weighted majority algorithm or winnow algorithm, as discussed by Littlestone and Warmuth in [11] and by Littlestone in [10] respectively, belong to this family¹. Both the weighted majority and winnow algorithms works as a committee of hypotheses to classify target samples. The popular (offline) Adaboost classifier [7] resembles these classifiers in the sense that it also

combines several ‘weak’hypotheses in classifying new observations. An online version of the boosting classifiers in has been recently proposed in a relatively recent study by Oza et.al [12].

Online learning algorithms can be immediately applied to the object tracking scenario. The tracker employs an online learning method to learn the object from frames arriving sequentially and then apply the classifier to detect the target in the next frame. Recently, the work of Avidan [2] and Grabner et.al. [8] has shown impressive results of using classifier based tracking methods. The classifier based methods initially learn a binary classifier to distinguish the object of interest from the (neighboring) background and then apply it at each new frame to locate the position of the object. Both the works of [2] and [8] uses AdaBoost [7] or an online variant of AdaBoost classifier.

Several previous works have addressed object tracking problem by approximating the distribution of feature responses representing an object using Kalman Filter [6], Particle Filter [9], Mean-shift method [5] etc. Density approximation tracking algorithms result in inferior performances to classifier based tracking algorithms in the cases when the target appearance undergoes substantial change over time or when there are similar objects nearby [2]. Results of our experiments will show how the proposed method is able to track objects in such scenarios where meanshift tracker fails.

One important issue to resolve in classifier based trackers is how to adapt with scene change and remain capable of identifying the object correctly. We should keep in mind that both the background and the object of interest may change with time. The previous studies [2, 8] replace a subset of current weak classifiers with a new one to cope with scene changes. Ensemble tracking [2] does not update the weak classifiers themselves. Instead at every frame, it replaces some of the older weak classifiers with several new weak classifiers. Grabner’s online boosting method [8] models the feature densities by simple Gaussians and update their parameters at each frame using Kalman filtering method. Even if we assume that a Gaussian distribution is

¹A more detailed discussion on online learning algorithms is provided by Blum in [3]

sufficient to model feature densities (which is usually not the case), the updating mechanism soon becomes complicated and slow if we wish to use higher dimensional weak classifiers. Furthermore, the method of [8] immediately throws away any weak classifier generating an error greater than or equal to 50%. To fill up the space evacuated by the leaving weak classifier, the methods in both [2] and [8] have to search a large subset of weak classifiers for replacement.

The *primary contribution* of this paper is an adaptation scheme for the weak classifiers themselves to conform with the changes over time. Similar to the method of ensemble tracking [2], we combine linear weak classifiers, learned in a least square fashion, and learn incrementally (online boosting). However, we neither replace weak classifiers for each data set nor do we throw out any weak classifier during training phase. Instead, the proposed method modifies the internal parameters of the base learners for the final classifier to blend with the change as long as the base learner has an error rate below 50%. The online boosting algorithm described in this paper uses linear regressors as the base learners. The adaptation scheme does not require us to keep the previous examples in the memory and does not need complex filtering techniques. This paper also demonstrates how the adaptation process can also ‘forget’ previous observations.

In the previous studies of online boosting that [2, 8], it has not been guaranteed that, interchanging a fixed number of weak classifiers will be able to identify and capture the change in pattern induced by new samples. Therefore, the number of weak classifiers to be replaced is an external parameter to these methods and the choice of such parameter is still unresolved. Furthermore, the time complexity of these methods will also increase with the increase in the number of base learners to be replaced. The proposed method, instead of replacement, keeps modifying the form of as many hypotheses as necessary to adapt to any new trend in the dataset. Therefore, the proposed method is capable of identifying the object with substantial change in the appearance during tracking. We show results where our algorithm supersedes previous method of tracking for tracking such objects in the results section. Our method also produces comparable performance w.r.t. its offline counterpart in classifying the UCI data.

The paper is organized as follows. We start with describing offline and online boosting algorithms in sections 2.1 and 2.2 respectively. sections 3 and 4 illustrates the least square fitting of linear regressors, how to modify them to cope with new data and how to incorporate these linear functions into online boosting. With a brief description of how to apply online boosting for object tracking in section 5, section 6 analyzes the performance of the proposed algorithm. Finally, we summarize our findings in Section 7.

2. Background and Related Work

2.1. Offline Boosting

Boosting was proposed as a classification algorithm in [7]. Any input $\mathbf{x} \in \mathbb{R}^L$ is categorized as one of the classes 1 or -1 using the sign of the function $H(\mathbf{x})$. The function $H : \mathbb{R}^L \rightarrow \{-1, 1\}$, also known as the strong classifier, is a linear combination of several other functions $f^k(\mathbf{x})$, $k = 1, 2, \dots, K$.

$$H(\mathbf{x}) = \text{sign} \left[\sum_{k=1}^K c^k f^k(\mathbf{x}) \right]. \quad (1)$$

The functions $f(\mathbf{x}) : \mathbb{R}^L \rightarrow \{-1, 1\}$, known as the base learners or weak classifiers in the boosting literature, are also classifier functions except (as their name implies) they do not possess a high rate of accuracy. It has been proved in [7] that, even if the individual performances of the weak learners are barely satisfactory (error rate $> 50\%$), their combination could be highly effective in terms of error.

The AdaBoost classifier is trained in an iterative fashion on the whole dataset $\tilde{X} \in \mathbb{R}^{N \times (L)}$ and their corresponding labels $\tilde{y} \in \{-1, 1\}^N$. First, each of the samples are imposed a uniform boosting weight $w_i^0 = \frac{1}{N}$ and $\mathbf{w}^0 = [w_i^0]_{i=1}^N$ such that $\sum_i w_i = 1$. Then, at k -th step the algorithm searches the $f(\mathbf{x})$ producing the lowest expected error w.r.t. the boosting weights \mathbf{w} . The mixing weight for the linear combination given in Eqn 1 is calculated by $c^k = \frac{1}{2} \log \frac{1-\epsilon^k}{\epsilon^k}$. In the next iteration of boosting, the weights are modified by $w_i^{k+1} = \frac{w_i^k \exp(c^k y_i f^k(\mathbf{x}_i))}{Z^k}$ so that the examples missed by $f^k(\mathbf{x})$ receives a higher weight. The weights are then normalized by Z^k to maintain \mathbf{w} as a probability vector. In the following description, we will omit the the superscripts of \mathbf{w}^k and $f(\mathbf{x})$ unless when they seem essential for precise description.

2.2. Online boosting

The underlying idea for development of online boosting classifier is to learn incrementally. We wish to build the classifier in an environment where the samples arrive one after another as opposed to batch learning, where the whole dataset is available to learn from. The work by Oza et. al. [12] models the sequential arrival of samples by a Poisson distribution. Each of the weak classifiers of the pool is learned and updated on each sample k times in a row where k is a random number generated by Poisson(λ). If any example is misclassified by a base learner, the Poisson parameter (λ) increases. Therefore, the next base learner will concentrate more on learning the misclassified sample due to a large value of k . The value of Poisson parameter λ is also accumulated to calculate the mixing weights of the hypotheses (e.g. c^k in Eqn 1).

However, Oza et.al. did not discuss much about how to update these weak hypotheses. Grabner et. al. [8] introduced online boosting algorithm to the vision community and showed results on different problems. To update the weak hypotheses, the authors of [8] proposes to incrementally model the sample distribution with the help of a Kalman filter [13]. Their implementation also replaces a set of weak classifiers with a new one. Replacing a set of base learners with a new set has also been proposed by Avidan [2]. Avidan’s work was primarily concentrated on tracking using a boosting method that adapts to the change in scenes by adding new members and removing the old ones from the weak classifier pool. In this work, we only update the linear weak classifiers (learned in a Least Square method) to cope with the new data samples. The following subsections describe the update procedure of weak classifiers and how they are incorporated in boosting framework.

3. Adaptive Linear Weak Classifier

3.1. Weighted Linear Regressor

Let us suppose $X \in \mathbb{R}^{N \times (L+1)}$ is the data matrix where there are N observations $\mathbf{x} \in \mathbb{R}^L$ (the last column of X is a vector of all ones that is used for calculating the intercept). The corresponding labels for these examples are stored in $\mathbf{y} \in \{-1, 1\}^N$. To solve a linear relation $X\hat{\beta} = \mathbf{y}$ by least squares method, we have to minimize the error function $(\mathbf{y} - X\hat{\beta})^T(\mathbf{y} - X\hat{\beta})$. If the different samples have different importance weights quantified by the vector \mathbf{w} , as they do in AdaBoost, we have to minimize the error function $(\mathbf{y} - X\hat{\beta})^T W(\mathbf{y} - X\hat{\beta})$ where W is a diagonal matrix with \mathbf{w} on its diagonal. The linear coefficients β that minimizes the error is given by the following expression.

$$\beta = (X^T W X)^{-1} X^T W \mathbf{y}. \quad (2)$$

Denoting the quantities $X^T W X$, $X^T W \mathbf{y}$ as $P \in \mathbb{R}^{(L+1) \times (L+1)}$ and $\mathbf{s} \in \mathbb{R}^{L+1}$ respectively, the expression can be abbreviated as $\beta = P^{-1} \mathbf{s}$. The base learners we used in this study are linear classifiers $f(\mathbf{x}) : \mathbb{R}^L \rightarrow \{-1, 1\}$. The response of $f(\mathbf{x})$ is calculated as follows:

$$f(\mathbf{x}) = \begin{cases} 1, & \text{if } [\mathbf{x}^T \ 1] \beta > 0; \\ -1, & \text{otherwise.} \end{cases} \quad (3)$$

3.2. Adaptive Linear Regressor

As we mentioned earlier, our weak learners recalculate their parameter values for each new data subset. Let X_τ denote the examples seen so far up to frame t and β_τ denote the parameters (linear coefficients) of the regressor we learned from X_τ . Then, for a new subset X_ν , the new value of the linear coefficients $\beta_{\tau+1}$ should be learned on $X_{\tau+1} = [X_\tau^T \ X_\nu^T]^T$ and $\mathbf{y}_{\tau+1} = [\mathbf{y}_\tau^T \ \mathbf{y}_\nu^T]^T$ by the linear regression .

$$\beta_{\tau+1} = P_{\tau+1}^{-1} \mathbf{s}_{\tau+1}. \quad (4)$$

Here, $P_{\tau+1} = X_{\tau+1}^T W_{\tau+1} X_{\tau+1}$, $\mathbf{s}_{\tau+1} = X_{\tau+1}^T W_{\tau+1} \mathbf{y}_{\tau+1}$, $W_{\tau+1} = \begin{pmatrix} W_\tau & 0 \\ 0 & W_\nu \end{pmatrix}$ where W_ν is a diagonal matrix having the boosting weights on the new samples \mathbf{w}_ν on its diagonal. It can be easily verified that, since $X_{\tau+1} = [X_\tau^T \ X_\nu^T]^T$ and $\mathbf{y}_{\tau+1} = [\mathbf{y}_\tau^T \ \mathbf{y}_\nu^T]^T$, the two quantities required for computing $\beta_{\tau+1}$ can be decomposed and expressed as a recursive summation of previous and new samples:

$$\begin{aligned} P_{\tau+1} &= X_\tau^T W_\tau X_\tau + X_\nu^T W_\nu X_\nu = P_\tau + P_\nu \\ \mathbf{s}_{\tau+1} &= X_\tau^T W_\tau \mathbf{y}_\tau + X_\nu^T W_\nu \mathbf{y}_\nu = \mathbf{s}_\tau + \mathbf{s}_\nu. \end{aligned} \quad (5)$$

We are describing in the next section how to calculate $P_{\tau+1}$ and $\mathbf{s}_{\tau+1}$ without storing all the previous examples seen so far.

3.3. Weak Classifier Memory

Based on the decomposition provided in the previous section, to update the parameter $\beta_{\tau+1}$, we only need to store a matrix and a vector of sizes $(L+1) \times (L+1)$ and $L+1$ respectively and compute the two quantities P_ν , \mathbf{s}_ν only for the new samples. The simplistic form of Eqn 5 suggests that we can also discard the quantities (i.e. ‘forget’ them) relating to very old examples if we wish to restrict ourselves to only the recent changes. So after the algorithm received a specific number of datapoints, the update equation changes to the following:

$$\begin{aligned} P_{\tau+1} &= P_\tau + P_\nu - P_{\tau-\omega} \\ \mathbf{s}_{\tau+1} &= \mathbf{s}_\tau + \mathbf{s}_\nu - \mathbf{s}_{\tau-\omega}. \end{aligned} \quad (6)$$

The value of ω decides for how long do we wish to ‘remember’ the contribution of any datapoint to our method. Obviously, we need to store all the quantities $P_\tau, P_{\tau-1}, \dots, P_{\tau-\omega}$ and $\mathbf{s}_\tau, \mathbf{s}_{\tau-1}, \dots, \mathbf{s}_{\tau-\omega}$ in a queue. Since the dimensionality L of the data is usually much smaller than the number of samples N_ν in the new set, the space complexity for this queue is not high.

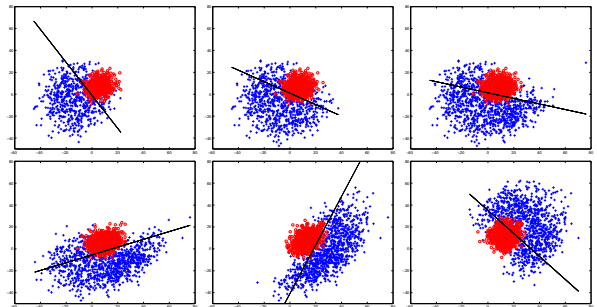


Figure 1. Adaptation of linear classifier. Top row: without forgetting, bottom row: with forgetting.

To visualize how this adaptation scheme works on any linear classifier, we generated synthetic two-class dataset. As Figure 1 shows, the blue dots and red circles represent

positive and negative examples respectively. The black line is the classifier learned by the proposed method with uniform weights ($W = I$). In the first three image of top row, we are incrementally adding new samples to the dataset and learning the weak classifier according to the update equation 5 without forgetting. In three images of the bottom row, we are also removing the oldest subset and training the linear classifier by the update equation 6 with forget mechanism. It can be easily observed how the linear classifier is correctly following the changing pattern of the dataset.

3.4. Temporal weighting

According to the update equation 5, all the new samples have equal importance and hence have the same contribution towards the modification of β . Therefore, after several time steps, the weak learner will become biased to the recent samples and contributions of the first set of examples will be gradually lost. As a result, the resulting classifier will be unable to classify observations similar to the first ones. Therefore, in this paper, we scaled the quantities used in equation 5 with respect to time.

This study uses a temporal weight ρ_τ (decreasing with time) on the quantities P_τ and s_τ so that the updated values of them becomes a weighted cumulative sum.

$$\begin{aligned} P_{\tau+1} &= P_\tau + \rho_\tau P_\nu \\ s_{\tau+1} &= s_\tau + \rho_\tau s_\nu \end{aligned} \quad (7)$$

The values of ρ_τ are predefined and decreases with τ . In our implementation we used $\rho_\tau = \exp(-\tau/\sigma)$ where σ is a predefined constant. In case of the ‘forgetting’, $P_{\tau-\omega}$ and $s_{\tau-\omega}$ have to be multiplied by their respective temporal weights before being subtracted in Eqn 6. But then, all the P and s in the queue need to be re-weighted so that $P_{\tau-\omega+1}$ receives the largest weight followed by that of $P_{\tau-\omega+2}$ and so on. Therefore, the new values of P and s becomes

$$\begin{aligned} P_{\tau+1} &= P_\tau - \rho_1 P_{\tau-\omega} + \sum_{l=1}^{\omega-1} \rho_l P_{\tau-\omega+l} + \rho_\omega P_\nu \\ s_{\tau+1} &= s_\tau - \rho_1 s_{\tau-\omega} + \sum_{l=1}^{\omega-1} \rho_l s_{\tau-\omega+l} + \rho_\omega s_\nu \end{aligned} \quad (8)$$

The effect of temporal weights on online boosting will be revisited in section 4.

4. Combining adaptive linear regressors for online boosting

Similar to the previous studies [12, 8], we apply the Adaboost training algorithm (modified) on every new subset of data X_ν . All samples in X_ν are assigned uniform weights. For each new subset X_ν of data and their labels y_ν , we calculate the updated value of $\beta_{\tau+1}^j$ (using Eqn 7 or Eqn 8 depending on whether or not we wish to forget) of j -th weak

hypothesis $f^j(\mathbf{x})$ to determine which $f^j(\mathbf{x})$ can most suitably conform itself with the changes. This base learner is immediately included in the strong classifier and the boosting weights w are updated accordingly. For rest of the hypothesis, β_τ is not updated to $\beta_{\tau+1}$ until they are found to be the one generating minimum error w.r.t w_ν and included in the strong classifier.

We also added some modifications to the original boosting method to incorporate the adaptive linear regressors as the weak learners for online boosting. The following paragraphs will describe these changes to Adaboost.

Need-based inclusion of base learners: We start the learning process with K base learners. The initial data subset is used to train few of the K available base classifiers until all samples are adequately learned. We claim that, when the sum of boosting weights w_ν of the new samples decreases below a threshold, the examples need not be learned by any more hypothesis. So, we stop training new weak learners when total weight is below a specific value and the remaining of the base learners remain dormant in the pool of weak classifiers. The same strategy has been followed for consequent datasets X_ν and their labels y_ν . At $k = 1$, instead of updating the linear base classifiers, first we apply them on the samples in X_ν to determine if any of the present learners can already classify them accurately or not. The update equation is enforced only when the minimum classification error with the current set of learners is not zero and therefore, the importance weight is significantly large.

Updating Boosting weights w : Every example receives the weight $w_i^0 = 1, i = 1, 2, \dots, N$ initially in our algorithm. We update the weights of the weak classifiers based on the performance of the weak classifier chosen in the latest iteration. But, if we normalize the importance weights after a perfectly correct classification (all samples were classified accurately), $w_i, i = 1, 2, \dots, N_\nu$ will retain their previous values and their sum will not fall below the specified threshold. Therefore, the proposed online boosting method does not normalize the importance weight after updating.

Effect of temporal weighting: It may appear to the reader that, since we are using a temporal weight decreasing with time, after several time step the new datapoints would not have any influence on the boosting algorithm. To comprehend why that does not happen, it is important to understand an important fact that not every classifier has a chance to observe each of the samples. This is due to the fact that, we stop training weak learners whenever $w_\nu^T \mathbf{1}$ decreases below a specific threshold. Therefore, the new subset arriving at time $t = 10$ may not be the $\tau = 10$ th subset that $f^j(\mathbf{x})$ (where $1 \leq j \leq K$) experienced. Recall that, we are decreasing the value of ρ according to the value τ which denotes the number of subset the corresponding weak classifier has actually learned on.

OnlineBoostFor each subsetset of new data X_ν and their labels \mathbf{y}_ν

1. Start with uniform distribution \mathbf{w}_ν and $dontLearn = 1$.
2. for $k = 1, 2, \dots, K$
 - (a) if $dontLearn = 0$ then $\text{WeakLearn}(\mathbf{w}_\nu)$.
 - (b) $err^j = \text{CalcResp}(f^j, X_\nu)$.
 - (c) $f^k = f^{j^*}$ where $j^* = \arg \min_j err^j$.
 - (d) $c^k = \frac{1}{2} \log \frac{1-err^{j^*}}{err^{j^*}}$
 - (e) $\forall w_i \in \mathbf{w}_\nu, w_i = w_i * \exp(-c^k y_i f^k(\mathbf{x}_i))$.
 - (f) if $\mathbf{w}_\nu^T \mathbf{1} < w_{th}$ then $dontLearn = 1$.
 - (g) if $err^k > 0.5$ ignore X_ν, \mathbf{y}_ν and return.

WeakLearn(\mathbf{w}_ν)

1. Compute new quantities $P_{\tau+1}$ and $\mathbf{s}_{\tau+1}$.
2. Learn $\beta_{\tau+1}$ according to Equation 7 or Equation 8.

CalcResp($f^j, X_\nu, \mathbf{w}_\nu$) returns err^j

1. $\lambda_{corr}^j = \lambda_{corr}^j + \sum_i w_i \delta(f^j(\mathbf{x}_i), y_i)$ and
 $\lambda_{miss}^j = \lambda_{miss}^j + \sum_i w_i (1 - \delta(f^j(\mathbf{x}_i), y_i))$
 where $\mathbf{x}_i \in X_\nu, y_i \in \mathbf{y}_\nu$ and $w_i \in \mathbf{w}_\nu$.
2. $err^j = \frac{\lambda_{miss}^j}{\lambda_{corr}^j + \lambda_{miss}^j}$.
3. $c^j = \frac{1}{2} \log \frac{1-err^j}{err^j}$.

Note: Here $\mathbf{1}$ is a vector of all ones and $\delta(a, b) = 1$ when $a = b$ and $\delta(a, b) = 0$ otherwise.

Table 1. Algorithm: Boosting with Linear Adaptive Classifier.

So, for the subset that was fed to the online boosting algorithm at time t , it will be the τ_1 -th and τ_2 -th new subset for two hypothesis $f^{j_1}(\mathbf{x})$ and $f^{j_2}(\mathbf{x})$ respectively. Therefore, even if for some weak learner the temporal weight ρ_{τ_1} is very small, for other classifier $f^{j_2}(\mathbf{x})$, the weight ρ_{τ_2} will be substantially large. Nonetheless, at some point of time, when the value of τ is large for all the base learners, any new subset will not receive the necessary attention. This is exactly the time when we should start ‘forgetting’ to make room for new observations.

Calculating c^k : The linear coefficients c^j combining the base learners are determined using the overall performance, that is, the overall classification accuracy (or error) on the whole set of training examples. These quantities are cumulatively stored into λ_{corr}^j (λ_{miss}^j) where λ_{corr}^j accumulates the summation of boosting weights of samples correctly (incorrectly) classified. For details on these quantities, please refer to [12]. We need to keep in mind that, when we are ‘forgetting’ a subset, their corresponding λ_{corr}^j and λ_{miss}^j also need be discarded. The complete algorithm for proposed method of boosting adaptive linear weak hypotheses is stated in Table 1.

5. Application to tracking

An online learning method can be readily applied for tracking objects in a video. Avidan used a modified version of AdaBoost[2] for tracking objects in consecutive images. We have closely followed the implementation of [2] to apply our online boosting algorithm for tracking. The target is identified in the first frame by the smallest rectangle \mathcal{R}_{inner} containing only the object itself. Then a larger rectangle \mathcal{R}_{outer} is selected around the inner rectangle \mathcal{R}_{inner} to mark the background pixels. All the pixels in \mathcal{R}_{inner} are considered as positive examples (i.e. $y_i = 1$) and all the pixels in rectangle \mathcal{R}_{outer} are considered as negative examples (i.e. $y_i = -1$) for learning. In the next frame, the boosting classifier is applied on all the pixels in \mathcal{R}_{outer} to generate the responses the strong classifier. A meanshift algorithm [4] is applied to determine the new location of our target on this response image (also called the confidence map). Once the meanshift algorithm converges, two new rectangles \mathcal{R}_{inner} and \mathcal{R}_{outer} are redrawn around the new location to label the pixels and the strong classifier is retrained using the new data subset and their labels.

6. Experiments and Results

6.1. Synthetic data

The 2D synthetic data of $N = 440$ samples was generated from a mixture of Gaussians as shown in the top-left image of Figure 2. The blue ‘*’ and red ‘+’ denote the positive and negative examples respectively. We passed a pair of positive and negative examples to the proposed online boosting algorithm at a time. Since we do not wish to forget any observations, weak learner memory is not used. The learned classifier was tested on another test dataset generated from the same mixture of Gaussians.

Fig 2 visualizes how the weak classifiers are being generated and modified according to the changes on these synthetic dataset. We are using only $K = 10$ base learners for this illustration. Each hypothesis correspond to a line in Fig 2. Initially, with $t < 20$, the algorithm tends to generate new weak classifiers and add them to strong classifier to learn the new examples (top row of Fig 2). Then, when there are no more hypothesis left unused, the base learners start adapting to the changes (as can be seen in 2nd row of Fig 2). Once the algorithm received sufficient samples, the set of weak classifiers are stabilized and remains almost unchanged till we finish learning (last row). One can easily notice how the the converged forms of the classifier are separating the boundaries of two classes.

To compare the performance of the online boosting method, we generated the classification error on the same dataset produced by an off-line Adaboost algorithm with logistic function as the base learner. We used the AdaBoost

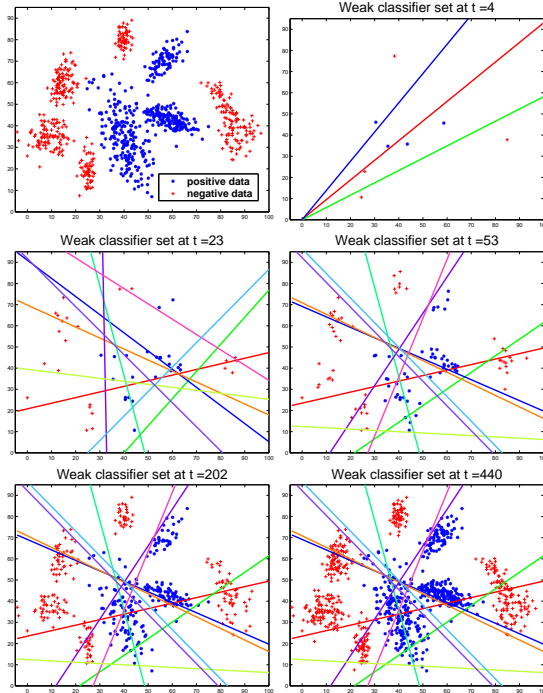


Figure 2. Synthetic 2D data (top-left image) and gradual adaptation of weak classifier at times $t = 4, 23, 53, 202, 440$.

implementation of machine learning software Weka [14]. The total number of weak classifier for online and offline boosting was the same. Although we expect any online learning algorithm to exhibit inferior performance to that of its offline counterpart, the proposed algorithm works much better than the offline boosting for this synthetic dataset (as shown in the first row of Table 2).

| Dataset | Training set | Size of test set | Proposed % correct | Offline % correct |
|---------------|--------------|------------------|--------------------|-------------------|
| synthetic | 440 + 440 | 440 + 440 | 94.43 | 72.61 |
| ionosphere | 75 + 75 | 150 + 51 | 84.07 | 83.08 |
| breast cancer | 100 + 100 | 112+257 | 90.24 | 91.32 |
| diabetes | 130 + 130 | 138 + 370 | 63.18 | 72.63 |
| spam | 500 + 500 | 1313 + 2288 | 87.67 | 89.58 |

Table 2. Classification rates on synthetic and UCI data

6.2. UCI datasets

We followed the same strategy we used for synthetic data on UCI [1] datasets ‘ionosphere’, ‘breast cancer’, ‘diabetes’ and ‘spam’. First, each of these datasets were separated into training and testing subsets. Then, training samples were supplied to the learning algorithm in pairs of a positive and a negative example and none of the observations were ‘forgotten’. The results, using both the proposed online boosting method and the offline AdaBoost implementation of Weka on several UCI datasets are given in Table 2. Both online and offline boosting classifiers com-

prise $K = 30$ bse learners for all dataset but ‘spam’. Since ‘spam’ dataset is considerably larger than others, the total number of base learners used to learn it was $K = 50$. The results suggests that we can have competitive performance by online boosting with adaptive linear weak classifiers for real datasets too.

6.3. Tracking examples

For tracking, we processed the images of video following the procedures used in Avidan’s work [2]. As described in section 5, the samples within the inner rectangle \mathcal{R}_{inner} were regarded as positive examples and that within \mathcal{R}_{outer} were considered as negative examples. In all the following experiments, the features used to represent the pixels were only R, G, B values.

The total number of weak learners used in all the tracking examples is 8. We fixed a the number of previous frames that we wish to ‘remember’ and discarded earlier observations from the data. The queue length for weak learner memory was $\omega = 15$, i.e., the learners ‘forgets’ everything happened before 15 frames. The temporal weights are calculated with $\sigma = 3$. The procedure for outlier detection of [2] was also followed in our implementation. In what follows in this section, we will show some image sequences where the proposed method outperforms the meanshift [4] and ensemble trackers [2] respectively.

Comparison with meanshift: Figures 3 (a), (b) and (c) compare the performance of the proposed online boosting algorithm to a meanshift tracker. Our first dataset (as shown in Figure 3) (a) contains images of a police car chase. At a certain stage of chasing, the car being chased (also the object which we are tracking) completely turns around and then tries to flee again in another direction. We show in Figure 3(a) that even though meanshift tracker was able to track the car after the collision, it fails to follow the object due to an occlusion by roadside pole and trees. But, our online learning adapts itself to the changes very rapidly and tracks it correctly. Since the current implementation of our algorithm can not modify the target window according to rotation or scale changes in the object, the target window was not redrawn according to the new appearance of the object.

The second dataset were recorded by a moving camera. In Figure 3(b), three vehicles with similar appearances cross each other in the opposite direction. While the meanshift tracker confuses the target truck with the other one, the proposed learning algorithm remains capable of distinguishing the target. The last comparison (Figure 3(c)) manifests how our algorithm adapts with illumination change (notice Frames 38 and 199) whereas meanshift tracker cannot.

Comparison with ensemble tracker: The ensemble tracking method [2] cope with the change in scene by replacing a set of base learners with new ones. The number of

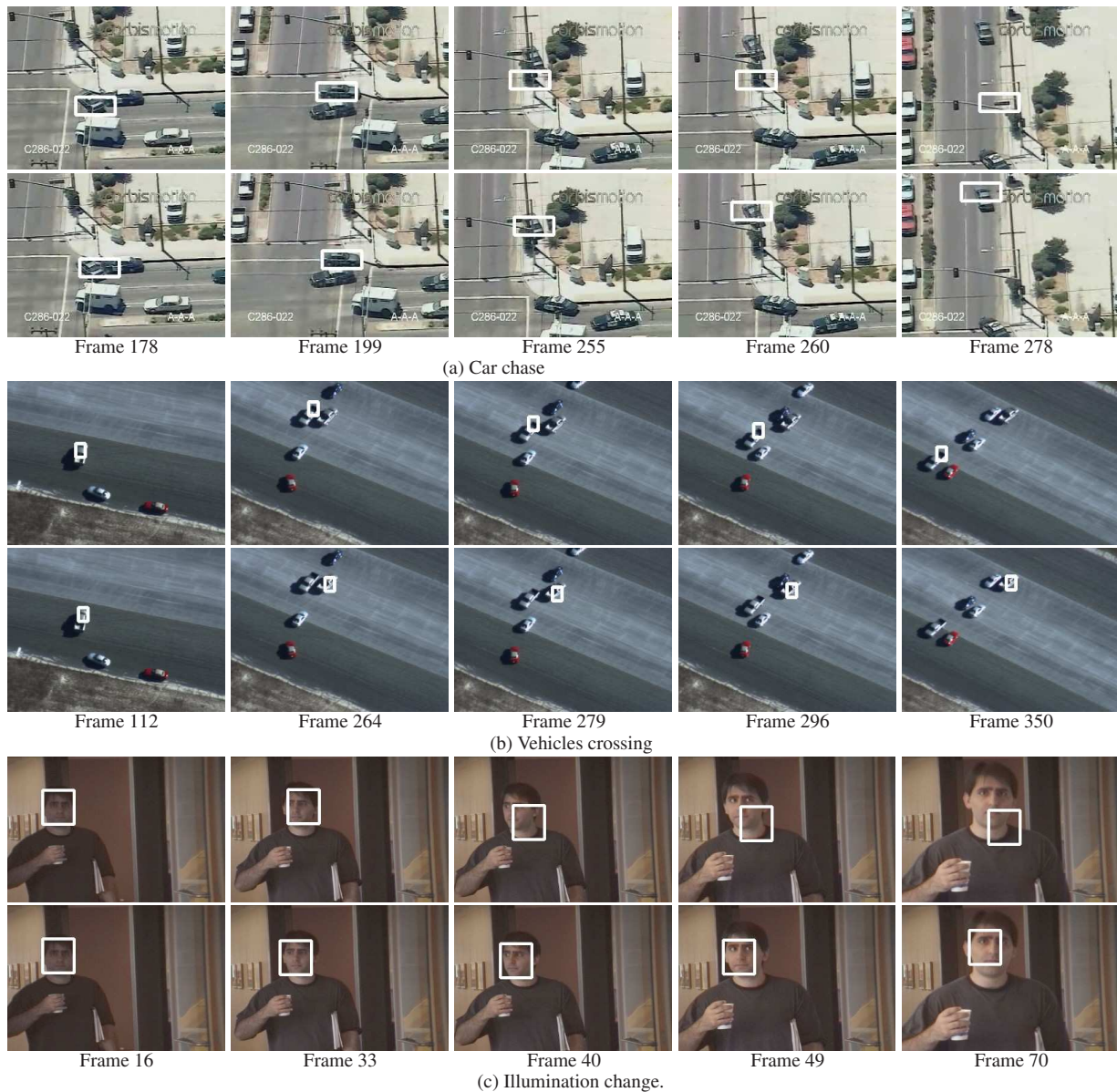


Figure 3. Performance comparison (a), (b) and (c) with meanshift tracker. For each output sequence, top row: meanshift tracker, bottom row: proposed method

weak learners to exchange is an external parameter to the algorithm. We can not expect that replacing any fixed number of weak classifiers can always capture the change with time. This is exactly what happens when ensemble tracker loses the target object in Figures 4 (a) and (b). In the video of cars on a city street at night (Figure 4 (a)), the ensemble tracker gets distracted by the rotating red light of the police car and eventually ends up on another car facing in the reverse direction. Since we are modifying all the base learners (if necessary), the online boosting method tracks the object accurately. In Figure 4(b), the tracker confuses the tracked person wearing a red jacket with another person wearing similar attire. The output of the proposed learning

method, as shown in the bottom row(s), clearly exhibits the robustness of method for both identification and capturing the changes.

Table 3 displays number of frames of the aforementioned datasets correctly tracked by the proposed method and other methods. The number of frames were calculated manually from the output images. If in any frame, 25% of the target window does not contain the object (approximately, except CarChase sequence), we classify the frame as being incorrectly tracked. As we can see, in all the videos where the traditional methods fail, our method can track the target for almost the full length of the sequence.

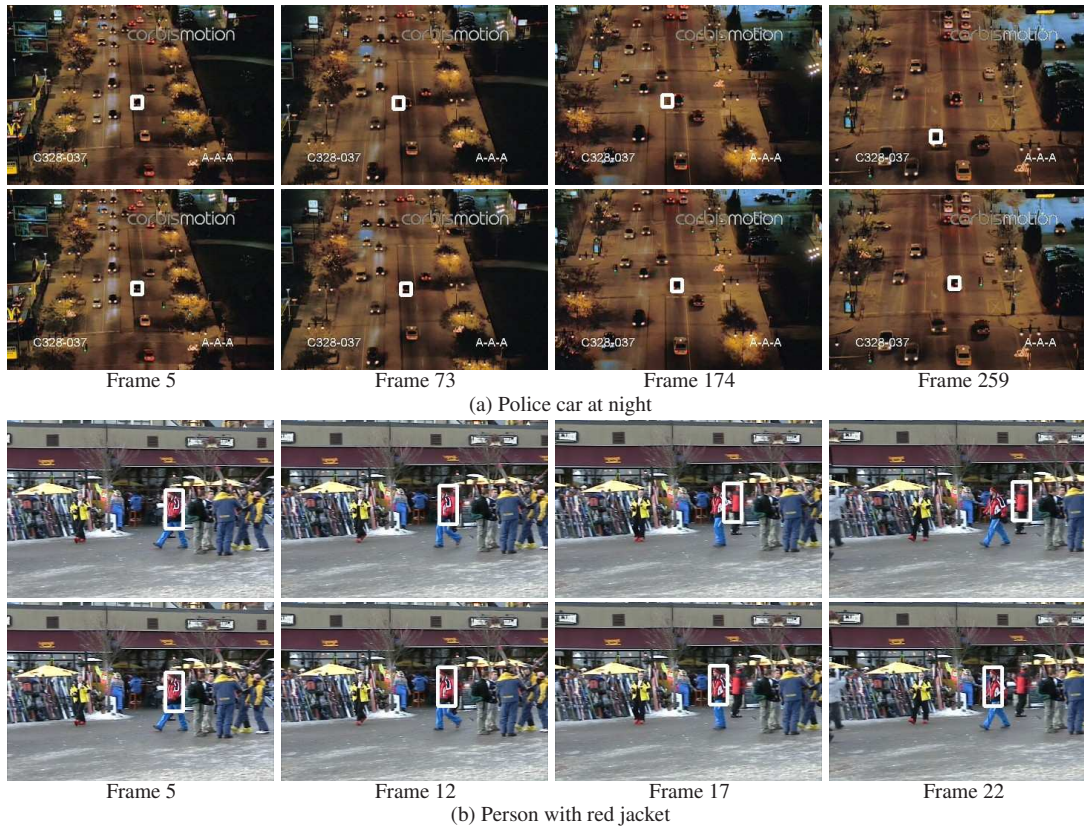


Figure 4. Performance comparison (a) and (b) with ensemble tracker. For each output sequence, top row: ensemble tracker, bottom row: proposed method.

| Dataset | Object | meanshift | ensemble | Proposed |
|--------------------|--------|-----------|----------|----------|
| CarChase | car | 250/285 | - | 285/285 |
| VehiclesCrossing | car | 260/395 | - | 390/395 |
| IlluminationChange | person | 38/76 | - | 92/92 |
| PoliceCarNight | car | - | 40/290 | 280/290 |
| PersonRedJacket | person | - | 10/45 | 45/45 |

Table 3. Frames exactly tracked by the proposed & other methods

7. Conclusion

This study proposes a new online boosting by continuous updating of weak classifiers. Results on artificial and real datasets shows the better performances achieved for both online learning and object tracking purposes by the proposed method than that of previous methods.

Acknowledgement: This research was partially funded by NSF CAREER award IIS-0546372 and Mitsubishi Electric Research Labs.

References

- [1] A. Asuncion and D. Newman. Uci machine learning repository, 2007. UC Irvine, School of ICS. **6**
- [2] S. Avidan. Ensemble tracking. In *CVPR*, pages 494–501, 2005. **1, 2, 3, 5, 6**
- [3] A. Blum. On-line algorithms in machine learning. *Online Algorithms: The State of the Art*, LNCS 1442, 1998. **1**
- [4] D. Comanciu, R. Visvanathan, and P. Meer. Kernel-based object tracking. *TPAMI*, 25(5):564–575, 2003. **5, 6**
- [5] D. Comanciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *CVPR*, 2000. **1**
- [6] F. Dellaert and C. Thorpe. Robust car tracking using kalman filtering and bayesian templates. In *Conference on Intelligent Transportation Systems*, 1997. **1**
- [7] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. **1, 2**
- [8] H. Grabner and H. Bischof. On-line boosting and vision. In *CVPR*, pages 260–267, 2006. **1, 2, 3, 4**
- [9] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *IJCV*, 29(1):5–28, 1998. **1**
- [10] N. Littlestone. Redundant noisy attributes, attribute errors and linear threshold learning using winnow. In *Fourth Annual Workshop on COLT*, pages 147–156. Morgan Kaufmann, 1991. **1**
- [11] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994. **1**
- [12] N. Oza and S. Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics*, pages 105–112, 2001. **1, 2, 4, 5**
- [13] G. Welch and G. Bishop. An introduction to kalman filter, 1995. Tech Report., Univ of NC-CH, Dept of Computer Science. **3**
- [14] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, San Francisco, 2005. **6**