DynamicBoost: Boosting Time Series Generated by Dynamical Systems

René Vidal
Center for Imaging Science, Dept. of BME,
Johns Hopkins University, Baltimore MD, USA
rvidal@cis.jhu.edu http://www.vision.jhu.edu

Paolo Favaro
Department of Electrical Engineering and
Physics, Heriot-Watt University, Edinburgh, UK
p.favaro@hw.ac.uk http://www.eps.hw.ac.uk/~pf21

Abstract

Boosting is a remarkably simple and flexible classification algorithm with widespread applications in computer vision. However, the application of boosting to non-Euclidean, infinite length, and time-varying data, such as videos, is not straightforward. In dynamic textures, for example, the temporal evolution of image intensities is captured by a linear dynamical system, whose parameters live in a Stiefel manifold: clearly non-Euclidean.

In this paper, we present a novel boosting method for the recognition of visual dynamical processes. Our key contribution is the design of weak classifiers (features) that are formulated as linear dynamical systems. The main advantage of such features is that they can be applied to infinitely long sequences and that they can be efficiently computed by solving a set of Sylvester equations. We also present an application of our method to dynamic texture classification.

1. Introduction

Classification and recognition problems have been the mainstream areas of research in machine learning for the past decades. Such problems have been motivated and constantly driven by several applications in bioinformatics, speech processing, medical imaging and computer vision, such as DNA sequence classification, speech recognition, handwritten digit recognition, object recognition, etc.

Given a set of points $x_i \in \mathcal{X}$ with corresponding labels $z_i \in \mathcal{Z} = \{-1, +1\}$, the goal of classification is to find a mapping (classifier) $h : \mathcal{X} \to \mathcal{Z}$ that maps each new data x into its corresponding label $z = +1$ or $z = -1$ by minimizing a loss function $L : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}^+$. Among the several existing methods for classification *e.g*. k-nearest neighbors, linear classifiers, decision trees, neural networks, Bayesian networks, support vector machines, hidden Markov models, *etc*. [3], boosting [9, 5] is a remarkably simple approach that has become particularly attractive over the past few years. The key idea behind boosting is to first use *weak classifiers* to deal with *easy* examples, to then learn *strong classifiers* that deal with *hard* examples. More specifically, boosting learns the classification function h incrementally from a given set of *weak classifiers*, *i.e*. classifiers that perform slightly better than chance. At each stage, boosting chooses the weak classifier that minimizes the current loss. The output of this weak classifier is then used to modify the loss function so that data points that are currently misclassified get *boosted* in importance. In this way, other weak classifiers will be chosen so as to correct for these mistakes.

Boosting has found numerous applications in computer vision, most notably in face detection [4, 13]. The most typical setup consists of extracting Haar-like features in the image, such as the difference between the sum of the pixel intensities within two adjacent rectangular regions. The values of such features $f(x)$ are then used to construct the weak classifiers as $h(x) = \text{sign}(f(x) - \theta)$, where x is *e.g*. a 24×24 pixel sub-window of the image and θ is a threshold. As boosting proceeds, good weak classifiers are selected, which in turns results in the automatic selection of *face-like* features. As these features can be computed extremely fast using the integral image representation, face detection can be performed in real-time.

However, the detection process is performed frame-by-frame, without considering the dynamics of the video. While this may be fine for problems such as face detection, there are other problems in computer vision where exploiting dynamical properties of the video is fundamental. Consider for example video sequences of dynamic textures, such as water, foliage, steam, etc. As shown in [2], such videos can be accurately modeled as the output of linear dynamical systems (LDS). The parameters of such LDS can be learned from data using techniques from system identification [8]. Once the parameters have been learned, one may use them not only for synthesizing novel sequences, but also for classification and recognition. For instance, if one has a distance between two LDS, one can perform classification using k-nearest neighbors. Of course the results are not ideal, not only because k-nearest neighbors is too simple of a classification method, but also because defining a distance between dynamical models is not straightforward, as

reported by recent works in this area [6, 1, 14].

The main contribution of this paper is to propose a generalization of boosting to the temporal domain for the purpose of discriminating between time series data generated by different dynamical systems. The development of such a generalization faces significant technical challenges.

1. First of all, one could argue that the problem could be easily solved by building an integral volume, computing scale-time Haar-like features, and then using standard AdaBoost. However, as capturing dynamical phenomena may require several frames, computing such features is no longer computationally straightforward. Furthermore, such a construction fails to incorporate the dynamical properties of the data to begin with.

2. Second, the discrimination of two different dynamical systems must be performed from a finite set of samples. Therefore, one cannot hope to be able to discriminate between any two stochastic processes. Often assumptions of stationarity and ergodicity are required.

3. Even for simple dynamical systems, such as LDS, the model parameters do not live in a Euclidean space. The fundamental questions are then 1) how to define *weak classifiers* for non Euclidean data, and 2) how to adapt AdaBoost to operate on non Euclidean data.

In this paper, we address some of these issues by proposing *DynamicBoost*, a generalization of AdaBoost to time series data generated by linear dynamical systems. The key to our approach is to use *dynamical systems* to classify *dynamical systems*. More specifically, we choose linear classifiers in the Hilbert space of outputs of a LDS as weak classifiers. Such classifiers can be computed efficiently, not in terms of the infinite time series data, but rather in terms of the parameters of the LDS identified from finite samples. The exact formula is in fact a dot product between a LDS associated with the weak classifier and the LDS identified from time series data. Thus, the weak classifiers are in fact dynamical systems. Since these classifiers are also linear classifiers in the space of outputs, the incremental selection of weak classifiers can be performed using standard AdaBoost. Therefore, DynamicBoost requires only a particular definition of weak classifiers, without any modification to the boosting part. We also present experiments testing the performance of DynamicBoost on the classification of dynamic textures.

To the best of our knowledge, our work is the first one to extend AdaBoost to the temporal domain [15]. Existing works have simply combined boosting with HMMs [7], or run AdaBoost at each frame in a way that uses previous history in evaluating the current frame [11]. However, none of these works formally exploits dynamical systems theory. For recent work extending boosting to the classification of data on Riemannian manifolds the reader is referred to [12].

2. Preliminaries

2.1. AdaBoost

We begin with a description of (discrete) Adaboost in the case of two classes. Our description is taken from [5], where the reader is referred to for further details and extensions.

Given a training set $\{(x_i, z_i)\}_{i=1}^N$, where $x_i \in \mathcal{X}$ is a vector and $z_i \in \mathcal{Z} = \{-1, +1\}$ is a label, the goal of classification is to find a mapping $h : \mathcal{X} \to \mathcal{Z}$ that maps each new data $x \in \mathcal{X}$ to its corresponding label $z \in \mathcal{Z}$. Boosting is a method for combining many *weak classifiers* $\{h_m(x)\}_{m=1}^M$, *i.e*. classifiers that perform slightly above chance, to produce a *strong classifier* $h(x) = \text{sign}\big(\sum_{m=1}^M c_m h_m(x)\big)$, where $\{c_m\}$ are constants to be determined. AdaBoost proceeds by sequentially applying a classification algorithm to re-weighted versions of the training data and then taking a weighted majority vote of the sequence of classifiers thus produced. More specifically, AdaBoost operates as follows:

1. Start with weights $w_i = 1/N$, $i = 1, \ldots, N$.

2. Repeat for $m = 1, 2, \ldots, M$

   (a) Fit the classifier $h_m(x) \in \{-1, 1\}$ using weights w_i on the training data.

   (b) Compute $\epsilon_m = E_w[\mathbf{1}_{z \neq h_m(x)}]$, $c_m = \log\big(\frac{1-\epsilon_m}{\epsilon_m}\big)$.

   (c) Set $w_i \leftarrow \frac{w_i \exp[c_m \mathbf{1}_{z_i \neq h_m(x_i)}]}{\sum_j w_j \exp[c_m \mathbf{1}_{z_j \neq h_m(x_j)}]}$.

3. Output the classifier $\text{sign}\big(\sum_{m=1}^M c_m h_m(x)\big)$.

The function $\mathbf{1}_A$ is the indicator function, *i.e*. $\mathbf{1}_A = 1$ if A is true, and 0 otherwise.

2.2. Dynamic Textures

A dynamic texture [2] is a generative model of videos defined by a random process (y_t, x_t). The observed variable $y_t \in \mathbb{R}^p$ encodes the video frame at time t and the hidden state variable $x_t \in \mathbb{R}^n$ (typically $n \ll p$) encodes the evolution of the video over time. The state and observed variables are related through the following linear dynamical system (LDS) equations

$$\begin{aligned} x_{t+1} &= Ax_t + v_t \\ y_t &= Cx_t + \mu + w_t, \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{n \times n}$ is the state transition matrix, $C \in \mathbb{R}^{m \times n}$ is a matrix containing the principal components of the video sequence, and $\mu \in \mathbb{R}^p$ is the temporal mean of the video sequence. The driving noise process is $v_t \overset{i.i.d.}{\sim} \mathcal{N}(0, Q)$, with $Q \in \mathbb{R}^{n \times n}$, and the observed noise is $w_t \overset{i.i.d.}{\sim} \mathcal{N}(0, R)$, with $R \in \mathbb{R}^{m \times m}$, where $\mathcal{N}(0, \Phi)$ is a zero-mean Gaussian distribution with covariance Φ. We will also assume that the initial state x_0 is distributed as $x_0 \overset{i.i.d.}{\sim} \mathcal{N}(0, \Sigma)$.

The dynamic texture model is completely specified by the parameters $\theta \doteq (A, Q, C, R, \mu, \Sigma) \in \Theta$. However, notice that the space of LDS Θ is not a Euclidean space. This is because for any invertible linear transformation $T \in GL(n)$, the models $\theta_1 = (A, Q, C, R, \mu, \Sigma)$ and $\theta_2 = (TAT^{-1}, TQT^\top, CT^{-1}, R, \mu, T\Sigma T^\top)$ generate the same random process y_t.

A number of methods are available to learn the parameters of the dynamic texture from a training video sequence, including asymptotically efficient methods such as N4SID [8], maximum likelihood methods such as expectation-maximization [10], and a suboptimal (but computationally efficient) solution based on the singular value decomposition [2]. In order for these methods to operate correctly, usually a number of assumptions on the dynamical systems are made. In particular, it is usually assumed that the process x_t is stationary and ergodic. Under these assumptions, the matrix Σ must satisfy the Lyapunov equation[1]

$$\Sigma = A\Sigma A^\top + Q. \tag{2}$$

Another typical assumption is that v_t and $w_{t'}$ are independent for all t and t', and that x_t is independent of $v_{t'}$ and $w_{t'}$ for all $t' \geq t$. Loosely speaking, these assumptions guarantee that one can identify the parameters θ from a single sample path of the random process y_t.

3. DynamicBoost

In this section, we present the proposed boosting algorithm for classifying time series data generated by LDS. As we will see shortly, the main difference between DynamicBoost and AdaBoost is in the selection of the weak classifiers, not on the boosting of the weak classifiers. Therefore, we will restrict our attention to the two-class problem, as the multi-class case can be dealt with in the same way as with normal AdaBoost.

Let $\{\theta_i, z_i\}_{i=1}^N$ be a given collection of dynamical systems $\theta_i = (A_i, Q_i, C_i, R_i, \mu_i, \Sigma_i) \in \Theta$ with corresponding labels $z_i \in \mathcal{Z}$. The goal is to learn a classifier $h : \Theta \to \mathcal{Z}$ that maps each new dynamical system $\theta \in \Theta$ to its corresponding label $z \in \mathcal{Z}$. As discussed at the end of the previous section, a time series data generated by a LDS is simply a sample path from a random process y_t which can be characterized by a set of parameters $\theta \in \Theta$. Thus, from now on we will speak indistinctively of classifying time series generated by a LDS and classifying LDS.

For the purpose of defining a set of weak classifiers, imagine we are given a time series $y_t \in \mathcal{Y}$, where $\mathcal{Y}$ is the Hilbert space of all possible outputs of a dynamical system. For any two time series y_t and y'_t, we will use the standard dot product on $\mathcal{Y}$, which is given by:

$$< y_t, y'_t >= \sum_{t=0}^{\infty} \lambda^t y_t^\top y'_t, \tag{3}$$

where $\lambda \in (0, 1)$ is an exponential discount factor that allows one to control the convergence of the series.

A hyperplane on $\mathcal{Y}$ with normal vector $h_t \in \mathcal{Y}^*$ (the dual of $\mathcal{Y}$ as a vector space) is simply given by

$$\mathcal{H} = \{y_t \in \mathcal{Y} :< h_t, y_t >= \phi\} \tag{4}$$

for some $\phi \in \mathbb{R}$. Therefore, we can define a weak classifier simply as a linear classifier on $\mathcal{Y}$, *i.e.* a classifier of the form

$$h(y) = \text{sign}\big(\sum_{t=0}^{\infty} \lambda^t h_t^\top y_t - \phi\big), \tag{5}$$

where $h_t \in \mathbb{R}^p$ and $\phi \in \mathbb{R}$ are the parameters of the linear classifier, and need to be chosen.

At a first sight, one may think that such a choice of weak classifiers does not make much sense, because in order to classify a time series $y_t \in \mathcal{Y}$, we need to generate an infinite dimensional object $h_t \in \mathcal{Y}^*$. Similarly to the *kernel trick*, our idea is to exploit the fact that y_t is the output of a LDS so as to show that, under certain conditions, the linear classifier h yields a finite dimensional classifier on Θ. The details of the derivation are left to the following two subsections.

3.1. Static weak classifiers

For the sake of simplicity, let us first assume that h_t is constant, *i.e.* $h_t = h_0$ for all $t \geq 0$. Also, for the sake of simplicity, assume that the time series is generated without noise, *i.e.* $v_t = 0$ and $w_t = 0$. From equation (1) we have

$$y_t = CA^t x_0 + \mu. \tag{6}$$

Therefore, we have that

$$\begin{aligned} < h_t, y_t > &= \sum_{t=0}^{\infty} \lambda^t h_0^\top (CA^t x_0 + \mu) \\ &= h_0^\top C(I - \lambda A)^{-1} x_0 + (1 - \lambda)^{-1} h_0^\top \mu. \end{aligned} \tag{7}$$

The last step requires that λA be stable, *i.e.* all the eigenvalues μ of λA must be such that $|\mu| < 1$. As a consequence, we obtain a family of classifiers on $\theta = (A, C, x_0)$

$$h(\theta) = \text{sign}\big(h_0^\top C(I - \lambda A)^{-1} x_0 + (1 - \lambda)^{-1} h_0^\top \mu - \phi\big) \tag{8}$$

parameterized by the choice of h_0, ϕ and λ.

An important property of the proposed classifier is that it can be computed in closed form, as we have shown. However, when y_t is an image sequence, $h_0 \in \mathbb{R}^p$ is an image. Therefore, even after the huge simplification of choosing h_t to be constant, we still have a problem of high dimensionality. To deal with this issue, similarly to the case of face recognition, we can choose h_0 to be a Haar-like feature. In fact, as $C \in \mathbb{R}^{p \times n}$, where the order of the system n is much smaller than the number of pixels p, each column of C can

[1] Well know results from linear systems theory ensure that for all $Q \succ 0$, the Lyapunov equation (2) has a unique solution $\Sigma \succ 0$ when A is stable, *i.e.* all eigenvalues μ of A are such that $|\mu| < 1$.

also be seen as an image. Therefore, from a computational perspective, the proposed weak classifiers can be thought of as applying Haar-like features to n images. As the number of frames in a video is $F \gg n$, we see already that by exploiting the fact that the video is the output of a dynamical model, we obtain significant gains in complexity.

Another property of the classifier is that it is invariant with respect to a change of basis for x_0. Recall from the previous section that if we apply a transformation $x_0 \rightarrow Tx_0$, then $A \rightarrow TAT^{-1}$ and $C \rightarrow CT^{-1}$. Therefore,

$$\begin{aligned} h_0^\top C(I-\lambda A)^{-1}x_0 &\rightarrow h_0^\top CT^{-1}(I-\lambda TAT^{-1})^{-1}Tx_0 \\ &= h_0^\top (T^{-1}(I-\lambda TAT^{-1})T)^{-1}x_0 \\ &= h_0^\top C(I-\lambda A)^{-1}x_0 \end{aligned} \quad (9)$$

and so changes of basis for the representation of the dynamical system do not affect the classifier.

3.2. Dynamic weak classifiers

In the previous subsection, we assumed that the classifier h_t is constant, *i.e.* $h_t = h_0$ for all $t \geq 0$. Another choice for h_t is to assume that it is the output of a dynamical system $\tilde{\theta} = (\tilde{A}, \tilde{C}, \tilde{x}_0, \tilde{\mu})$ of the same form as in equation (1), but without noise. More specifically, we define h_t as

$$h_t = \tilde{C}\tilde{A}^t\tilde{x}_0 + \tilde{\mu}, \quad (10)$$

where $\tilde{A} \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$, $\tilde{C} \in \mathbb{R}^{p\times\tilde{n}}$ and $\tilde{\mu} \in \mathbb{R}^p$. Notice that the order of the classifier $\tilde{n}$, need not coincide with the order of the LDS n, but the dimension of the output p must be the same.

As in the previous subsection, the question is whether we can evaluate this weak classifier efficiently on a dynamical system θ. Again, under the assumption of no noise, *i.e.* $v_t = 0$ and $w_t = 0$, we have that

$$\begin{aligned} < h_t, y_t > &= \sum_{t=0}^{\infty} \lambda^t (\tilde{x}_0^\top (\tilde{A}^t)^\top \tilde{C}^\top + \tilde{\mu}^\top)(CA^t x_0 + \mu) \\ &= \tilde{x}_0^\top P x_0 + \tilde{x}_0^\top (I - \lambda\tilde{A}^\top)^{-1}\tilde{C}^\top\mu + \\ &\quad \tilde{\mu}^\top C(I-\lambda A)^{-1}x_0 + (1-\lambda)^{-1}\tilde{\mu}^\top\mu, \end{aligned} \quad (11)$$

where $P = \sum_{t=0}^{\infty} \lambda^t (\tilde{A}^t)^\top \tilde{C}^\top C A^t \in \mathbb{R}^{\tilde{n}\times n}$. In order for the sum to converge, and so for P to be well defined, it is necessary that $\lambda A \otimes \tilde{A}$ be stable. Then, after separating the infinite sum into the first term and the remaining ones, it can be seen that P must satisfy the Sylvester's equation[2]

$$P = \tilde{C}^T C + \lambda \tilde{A}^T P A. \quad (12)$$

We thus obtain a family of classifiers on $\theta = (A, C, x_0, \mu)$, parameterized by the choice of $\tilde{A}$, $\tilde{C}$, $\tilde{x}_0$, $\tilde{\mu}$ and λ

$$h(\theta) = \text{sign}(\tilde{x}_0^\top P x_0 + \cdots + (1-\lambda)^{-1}\tilde{\mu}^\top\mu - \phi). \quad (13)$$

[2]Notice that the Lyapunov equation in (2) is a special case of the Sylvester's equation with $\tilde{A} = A$, $\tilde{C} = C$ and $\lambda = 1$.

As in the case of static classifiers, notice that the dynamic classifiers can also be computed in closed form by solving a Sylvester's equation, which is linear in P.

Remark 1 (Static versus dynamic classifiers) *Notice that when $\tilde{n} = 1$ and $\tilde{A} = 1$, then $\tilde{C} \in \mathbb{R}^{p\times 1}$, $P \in \mathbb{R}^{1\times n}$, and $P(I - \lambda A) = \tilde{C}^\top C$. Hence $P = \tilde{C}^T C(I - \lambda A)^{-1}$. Therefore, $\tilde{x}_0^T P x_0 = \tilde{x}_0^T \tilde{C}^T C(I - \lambda A)^{-1}x_0$, and so by letting $h_0 = \tilde{C}\tilde{x}_0 + \tilde{\mu}$ we obtain the static classifier $h_t = h_0$ described in the previous subsection.*

The calculation of the dynamic classifiers involves the computation of $\tilde{C}^T C$, which has linear complexity in p, plus the computation of P from the Sylvester's equation, which has cubic complexity in $n\tilde{n}$, because to compute P one needs to invert $(I_{n\tilde{n}} - \lambda\tilde{A}\otimes A) \in \mathbb{R}^{n\tilde{n}\times n\tilde{n}}$. As typically $p \gg \max(n, \tilde{n})$, the bottleneck is on the computation of $\tilde{C}^\top C$. As before, in order to reduce the computational complexity, we can choose $\tilde{C}$ in a particular way. Recall that each column of C, C_i $i = 1, \ldots n$, is an image. Therefore, $\tilde{C}C_i$ can be thought of as applying $\tilde{n}$ filters to an image. By choosing such filters to have a small support, *e.g.* Haar-like features, the computational complexity can be significantly reduced. In face classification, for example, one typically chooses image features that are supported on a few pixels in the image (say a 20×20 window). In what regards the choice of the dimension $\tilde{n}$, it is unclear how one may go about choosing it. Intuitively, $\tilde{n}$ should be of the same order or smaller than the dimension of the individual LDS n_i. For instance, one may choose $\tilde{n} = \min_{i=1,\ldots,N}\{n_i\}$.

Remark 2 (Invariance with respect to initial conditions) *In some applications, one may be interested in comparing dynamical systems in a way that does not depend on the initial conditions. However, the weak classifiers we have proposed have been defined in such a way that they depend explicitly on the initial conditions, as they involve expressions of the form $\tilde{x}_0^\top P x_0$. As proposed in [14], one may achieve invariance with respect to the initial conditions by taking the expectation over $\tilde{x}_0, x_0$. This yields a classifier of the form $h(\theta) = sign(trace(PS) + (1-\lambda)^{-1}\tilde{\mu}^\top\mu - \phi)$, where $S = E(x_0\tilde{x}_0^\top)$. For simplicity, one may set $S = I$.*

3.3. Boosting weak classifiers

In the previous two sections, we proposed a family of weak classifiers for dynamical systems. Such classifiers can be applied either to a time series data y_t or to a LDS identified from it. Notice that the Adaboost algorithm described in Section 2.1 depends solely on the values of the weak classifiers h_m at the data points. Therefore, in order to boost the weak classifiers for dynamical systems, we simply apply AdaBoost or any of its extensions to the chosen weak classifiers. In other words, the boosting part is not modified, in spite of the fact that the data lives on a non-Euclidean space.

4. Experiments

Experiments on synthetic data. We first compare the performance of the two proposed classifiers (*static* and *dynamic*) against a simple *texture classifier* $\text{sign}(h_0^\top y_0 - \phi)$ applied to the first measurement y_0 of the time series data.

The comparison is done on the synthetic data set shown in Figure 1. On the left we show the first frame of several 8×8 pixel and 20 frame long subsequences extracted from a simulated snow fall sequence. Synthetic snowflakes move from top to bottom while oscillating sideways. On the right we show the first frame of several subsequences extracted from another sequence where the snowflakes undergo a circular motion. The main motivation for choosing these two sequences is that it is very difficult to distinguish the two classes from a single snapshot; instead, discrimination is easier when dynamics are taken into account.

In our comparison, we use $N = 1000$ sample subsequences from both sequences. For each sample, we identify a LDS (A, C, x_0, μ) of order $n = 3$. We use $M = 400$ weak classifiers, with randomly chosen parameters h_0, (h_0, λ), and $(\tilde{A}, \tilde{C}, \tilde{x}_0, \lambda)$ for the texture, static, and dynamic classifiers, respectively. For the static and dynamic classifiers, we do not use the temporal mean, *i.e.* we set $\mu = 0$ and $\tilde{\mu} = 0$ in the corresponding formulae for the weak classifiers. For the dynamic classifier, the state dimension is chosen as $\tilde{n} = 4$.

Figure 2 shows the performance (test error) of AdaBoost for the three weak classifiers. As one can notice, the proposed dynamic classifiers achieve the best performance.

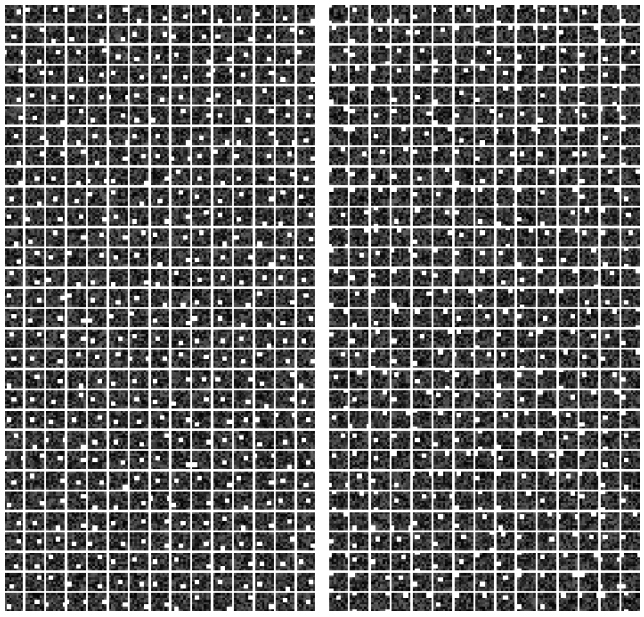

Figure 1. Synthetic data set. The left image shows the first frame of several patches from an artificial snow-fall sequence. The right image shows the first frame of several patches from a sequence with snowflakes that undergo a circular motion. Notice that it is very hard to distinguish the two sequences from a single snapshot.

Experiments on real data. The real experiments are performed on the dynamic texture data sets shown in Figure 3, which consist of several sequences from 6 different classes: bushes, flags, fountains, stairs, trees, and waves. We extract subsequences of 35×35 pixels by 20 frames from each one of the original sequences. We then discard any subsequence that does not show any motion. This gives us 3289 valid subsequences for the bush sequences; 2214 for the flag sequences; 214 for the fountain sequences; 1911 for the stair sequences; 581 for the tree sequences; and 5785 for the wave sequences. For each subsequence, we identify a LDS (A, C, x_0, μ) of order $n = 7$. We use $M = 100$ dynamic classifiers $\text{sign}(\text{trace}(P) + (1 - \lambda)^{-1}\tilde{\mu}^\top \mu)$ with randomly chosen parameters $(\tilde{A}, \tilde{C}, \tilde{\mu}, \lambda)$ and with a state dimension of $\tilde{n} = 7$. For each weak classifier, the threshold ϕ is found automatically. Table 1 shows the percentage of correct classification for each pair of classes and Figure 4 shows examples of the evolution of the test error. Notice that for most class pairs the error is within 22%, although in some cases the algorithm gives large errors of up to 39.5%.

Sequence	bushes	flags	fountains	stairs	trees	waves
bushes	–	14.5	11.5	4.0	20.0	22.0
flags	14.5	–	5.0	6.0	17.5	11.5
fountains	11.5	5.0	–	0.5	3.5	8.0
stairs	4.0	6.0	0.5	–	31.5	20.5
trees	20.0	17.5	3.5	31.5	–	39.5
waves	22.0	11.5	8.0	20.5	39.5	–

Table 1. Results on the real data set. Percentage of correct classification given by DynamicBoost with dynamic features for each pair of classes.

5. Conclusions

We have presented a novel extension of boosting for the recognition of visual dynamical processes. Our key contribution is the design of weak classifiers (features) that are formulated as linear dynamical systems. The main advantages of such features are that they can be applied to infinitely long sequences and that they can be efficiently computed by solving a set of Sylvester equations. We demonstrated our approach on classification of dynamic textures.

Acknowledgements

This work has been funded by Johns Hopkins startup funds, a VIBOT fellowship, and grants NSF EHS-0509101, ONR N00014-05-10836, and NSF CAREER IIS-0447739.

References

[1] K. D. Cock and B. D. Moor. Subspace angles and distances between ARMA models. *System and Control Letters*, 46(4):265–270, 2002.

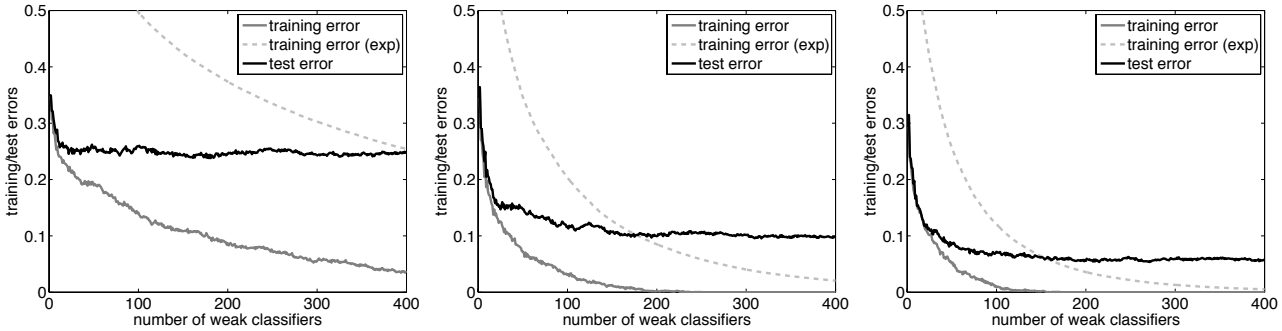

Figure 2. Performance results on the synthetic data set. Left image: test error when texture features are used. Middle image: test error when static features are used. Right image: test error when dynamic features are employed. Every plot shows the empirical loss training error (dark gray solid curve), the exponential loss training error (light gray and dashed curve), and the empirical loss test error (black solid curve). Notice how the dynamic features achieve the lowest test error.

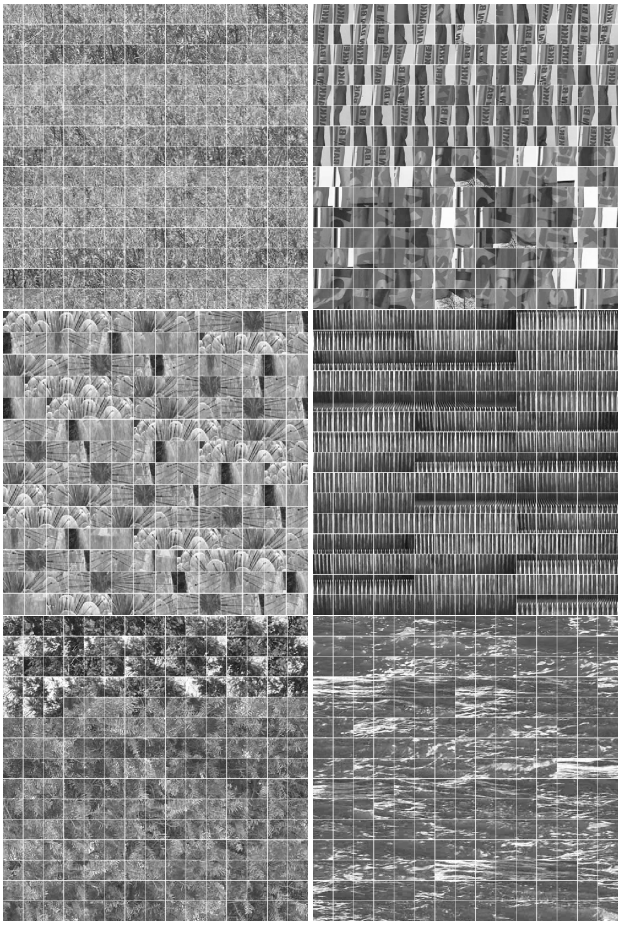

Figure 3. Real data set. From left to right and top to bottom we show a snapshot of patches extracted from: bushes, flags, fountains, stairs, trees, and waves. Each subsequence is 35×35 pixels and 20 frames long.

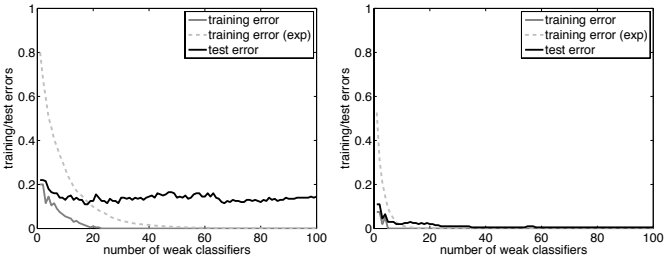

Figure 4. Performance results on the real data set. Left image: errors for bushes vs flags. Right image: errors for flags vs fountains.

[2] G. Doretto, A. Chiuso, Y. Wu, and S. Soatto. Dynamic textures. *International Journal of Computer Vision*, 51(2):91–109, 2003.

[3] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, New York, 2nd edition, 2000.

[4] F. Fleuret and D. Geman. Coarse-to-fine face detection. *International Journal of Computer Vision*, 41(1-2):85–107, 2001.

[5] J. H. Friedman, T. Hastie, and R. Tibshirani. Special invited paper. additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–374, 2000.

[6] R. Martin. A metric for ARMA processes. *IEEE Transactions on Signal Processing*, 48(4):1164–1170, 2000.

[7] K. Okuma, A.Taleghani, N. Freitas, J. Little, and D. Lowe. Boosted particle filter: multitarget detection and tracking. In *European Conference on Computer Vision*, 2004.

[8] P. V. Overschee and B. D. Moor. Subspace algorithms for the stochastic identification problem. *Automatica*, 29(3):649–660, 1993.

[9] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Computational learning theory*, pages 80–91, 1998.

[10] R. Shumway and D. Stoffer. An approach to time series smoothing and forecasting using the EM algorithm. *Journal of Time Series Analysis*, 3(4):253–264, 1982.

[11] P. Smith, N. da Vitoria, and M. Shah. TemporalBoost for event recognition. In *IEEE International Conference on Computer Vision*, pages 733–740, 2005.

[12] O. Tuzel, F. Porikli, and P. Meer. Human detection via classification on riemannian manifolds. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

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

[14] S. Vishwanathan, A. Smola, and R. Vidal. Binet-Cauchy kernels on dynamical systems and its application to the analysis of dynamic scenes. *Int. Journal of Computer Vision*, 73(1):95–119, 2007.

[15] P. Yin, I. Essa, and J. Rehg. The segmental boosting algorithm for time-series feature selection. In *Learning Workshop*, 2007.