Multi-camera Spatio-temporal Fusion and Biased Sequence-data Learning for Security Surveillance

Gang Wu, Yi Wu, Long Jiao, Yuan-Fang Wang, Edward Y. Chang
Electrical & Computer Engineering and Computer Science
University of California, Santa Barbara, CA 93106

gwu@engineering.ucsb.edu,wuyi@ece.ucsb.edu,longjiao@cs.ucsb.edu
yfwang@cs.ucsb.edu,echang@ece.ucsb.edu

ABSTRACT

We present a framework for multi-camera video surveillance. The framework consists of three phases: *detection*, *representation*, and *recognition*. The *detection* phase fuses video streams from multiple cameras for efficiently and reliably extracting motion trajectories from video. The *representation* phase summarizes raw trajectory data to construct hierarchical, invariant, and content-rich descriptions of the motion events. Finally, the *recognition* phase deals with event classification and identification on the data descriptors. Because of space limits, we describe only briefly how we detect and represent events, but we provide in-depth treatment on the third phase: event recognition. For effective recognition, we devise a sequence-alignment kernel function to perform sequence data learning for identifying suspicious events. We show that when the positive training instances (i.e., suspicious events) are significantly outnumbered by the negative training instances (benign events), then SVMs (or any other learning methods) can suffer a high incidence of errors. To remedy this problem, we propose the kernel boundary alignment (KBA) algorithm to work with the sequence-alignment kernel. Through empirical study in a parking-lot surveillance setting, we show that our spatio-temporal fusion scheme and biased sequence-data learning method are highly effective in identifying suspicious events.

Categories and Subject Descriptors: I.2.6 [Artificial Intelligence]: Learning—concept learning; I.4.7 [Image Processing and Computer Vision]: Feature Measurement—feature representation

General Terms: algorithms, performance, theory

1. INTRODUCTION

United States policymakers, especially in security and intelligence services, are increasingly turning toward video surveillance as a means to combat terrorist threats and increase public security. With the proliferation of inexpensive cameras and the availability of high-speed, broad-band wired/wireless networks, it has become economically and technically feasible to deploy a large number of cameras for security surveillance [33, 34]. However, several important research questions must be addressed before we can rely upon video surveillance as an effective tool for crime prevention.

A surveillance task can be divided into three phases: *event detection*, *event representation*, and *event recognition* [11]. The *detection* phase handles multi-source spatio-temporal data fusion for efficiently and reliably extracting motion trajectories from video. The *representation* phase summarizes raw trajectory data to construct hierarchical, invariant, and content-rich representations of the motion events. Finally, the *recognition* phase deals with event recognition and classification. The research challenges of the three phases are summarized as follows:

- *Event detection from multiple cameras.* Objects observed from multiple cameras should be integrated to build spatio-temporal patterns that correspond to 3-dimensional viewing. Such integration must handle spatial occlusion and temporal shift (e.g., camera recording without timing synchronization and videotaping with differing frame rates). In addition, a motion pattern should not be affected by varying camera positions/poses and incidental environmental factors that can alter object appearance.

- *Hierarchical and invariant event description.* Invariant descriptions are those that are not affected by incidental change of environment factors (e.g., lighting) and sensing configuration (e.g., camera placement). The concept of invariancy is applicable at multiple levels of event description. We distinguish two types of invariancy: *fine-grain* and *coarse-grain*. *Fine-grain invariancy* captures the characteristics of an event at a detailed, numeric level. Fine-grain invariant descriptors are therefore suitable for "intra-class" discrimination of similar event patterns (e.g., locating a particular event among multiple events depicting the same circling behavior of vehicles in a parking lot). *Coarse-grain invariancy* captures the motion traits at a concise, semantic level. Coarse-grain invariant descriptions are thus suitable for "inter-class" discrimination, e.g., discriminating a vehicle's circling behavior from, say, its parking behavior. These two types of descriptors can be used synergistically to accomplish a recognition task.

- *Event recognition.* Event characterization deals with mapping motion patterns to semantics (e.g., benign and suspicious events). Traditional machine learning algorithms such as SVMs and decision trees cannot be directly applied

MM'03, November 2–8, 2003, Berkeley, California, USA.

to such infinite-dimensional data, which may also exhibit temporal ordering. Furthermore, positive events (i.e., the sought-for hazardous events) are always significantly outnumbered by negative events in the training data. In such an imbalanced set of training data, the class boundary tends to skew towards the minority class and becomes very sensitive to noise. (An example is presented in Section 4.2 to illustrate this problem.)

Due to space limitations, we provide in-depth treatment in this paper only on event recognition, briefly describing how we perform event detection and representation to prepare for event recognition. For event detection and representation, we configure two-level Kalman filters to fuse multi-source video data; and we employ a variant of dynamic programming to construct event descriptors. For effective event recognition, we first discretize a continuous spatio-temporal sequence. We propose a sequence-alignment kernel, which we show to be a legitimate *kernel function*, to discriminate events. For tackling the imbalanced training-data problem, we propose the kernel boundary alignment (KBA) algorithm, which adaptively modifies the kernel matrix according to the training-data distribution. One particular application scenario we utilize to evaluate our algorithms is detecting suspicious activities in a parking lot. Our empirical study shows that our spatio-temporal fusion scheme can efficiently and reliably reconstruct scene activities even when individual cameras may have spatial or temporal lapses, and that our sequence-alignment kernel and KBA algorithm are highly effective in identifying suspicious events.

The rest of the paper is organized as follows. In Section 2 we discuss related work. Section 3 briefly discusses our sensor data fusion scheme and sequence-data representation. Section 4 presents event characterization and recognition methods. Section 5 presents empirical results. Finally, in Section 6 we offer concluding remarks.

2. RELATED WORK

We divide our discussion of related work into two parts. The first part surveys related work in fusing data from multi-cameras. The second part discusses related research in sequence-data learning.

2.1 Sensor Data Analysis and Fusion

Sensor-data fusion from multiple cameras is an important problem with many potential applications. The work in multi-source fusion can be divided into two categories based on camera configurations: spatially non-overlapping and spatially overlapping. The study of [25] attempts to fuse data from several non-overlapping cameras using a Bayesian reasoning approach. Since there might be significant gaps between the fields of view of the cameras, the precision in prediction may suffer. Most fusion algorithms assume an overlapping camera configuration, and they concentrate on fusing local coordinate frames of multiple cameras into one global coordinate system. For example, [26] assumes that only the intrinsic camera parameters are known, and the cameras are registered into a common frame of reference by aligning moving objects observed in multiple cameras spatially and temporally [24]. The DETER project [32] also uses an overlapping camera configuration. The homography between images from two cameras is computed, the images are mosaiced together to perform seamless tracking, and all data processing is then performed in the synthesized image plane. In this paper, we use a two-level hierarchy of Kalman filters for trajectory tracking and data fusion from multiple cameras. The advantage of our formulation is that it enables both bottom-up fusion and top-down guidance, and hence is robust even with partial occlusion.

The Kalman filter is an important theoretical development for data smoothing and prediction. The traditional Kalman filter is a linear algorithm that operates under a prediction-correction paradigm. The quantities of a system to be estimated are summarized in an internal state vector, which is constrained by the observation of the system's external behavior. The prediction mechanism is used for propagating the system's internal state over time and the correction mechanism is for fine tuning the state propagation with external observations [39].

The Kalman filter and its variants are powerful tools for tracking inertial systems. Generally speaking, the standard Kalman filter performs satisfactorily for tracking the position of a moving object. However, for tracking the orientation of an object, where the governing equations in state propagation and observation may be nonlinear, the extended Kalman filter (e.g., non-linear variants of the Kalman filter) must be used [29, 30]. For example, [2] uses a standard Kalman filter to predict the head position, and an extension of it to estimate the head orientation of a user in a virtual reality system. In this paper, we are interested in summarizing the trajectory of a vehicle, and hence, only the position, not the orientation, of a vehicle is needed. We have thus employed the traditional Kalman filter for the efficient tracking purpose.

In addition to the Kalman filter, the Hidden Markov Model (HMM) has been used for object tracking [4]. Both the Kalman filter and HMM can be used to estimate the internal states of a system. However, HMM is not an attractive online tracking method due to its high computational intensive with respect to the number of states. For tracking objects(s), where the number of possible locations (number of states) of the tracked object(s) is theoretically infinite, the Kalman filter is the popular choice [4]. In general, HMM is more suitable for recognition tasks [5, 36], where the number of states is relatively small.

2.2 Sequence-data Modeling and Learning

Generative and discriminative models have been proposed to classify sequence data. The Hidden Markov Model (HMM) is the most widely used generative model [3] for describing sequence data. The HMM requires building a model for each pattern family. When an unknown pattern is to be classified, the Bayes rule returns the most likely model to depict it. For surveillance, however, building an HMM for each motion pattern, benign or suspicious, may not be realistic because of the scarcity of positive training data. (Learning an HMM, even with a moderate number of states, requires a large number of training instances [8].)

SVMs [37] are the most popular discriminative models, which directly estimate a discriminant function for each class. SVMs have been proven superior to generative models in classification problems for both effectiveness and efficiency. SVMs are applied to training data that reside in a vector space. The basic form of an SVM kernel function which classifies an input vector $\mathbf{x}$ is expressed as

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + b = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \quad (1)$$

where ϕ is a non-linear mapping function which maps input vectors into the feature space, operator '$\cdot$' denotes the inner product operator, and $\mathbf{x}_i$ is the i^{th} training sample. Parameters y_i and α_i are class label and Lagrange multiplier of $\mathbf{x}_i$, respectively. K is a kernel function, and b is the bias.

For sequence data, in particular variable-length sequences, the basis function ϕ does not exist, nor does the feature space. Several attempts (e.g., [27, 28]) have been made to convert sequence data to fixed-length segments and represent them in vector spaces, but these forced mappings of variable-length to fixed-length segments often result in loss of information. Furthermore, these models cannot capture the temporal relationship between spatial instances, nor do they consider secondary variables (e.g., velocity can be a secondary variable when traveling direction is the one).

Another vector-space approach is the SVM-Fisher kernel. The SVM-Fisher kernel [20, 21] computes features from probabilistic models $p(x|\theta)$, in which θ is learned from the HMM training. It then uses the tangent vector of the log marginal likelihood $\log p(x|\theta)$ as the feature vectors. The SVM-Fisher kernel considers only positive training instances, and it does not take advantage of negative training instances in learning.

Fortunately, kernel methods (SVMs are a member of the kernel method family) require only a positive definite kernel matrix $K(\mathbf{x}_i, \mathbf{x}_j)$, which consists of a similarity measure between all pairs of training instances, without explicitly representing individual instances in a vector space [22]. More specifically, we can treat each sequence instance $\mathbf{x}_i$ as a random variable z_i. All we need is a kernel function that generates pair-wise similarity between all pairs of z_i and z_j, and that can ensure that the generated similarity matrix is positive definite. To put this another way, as long as we have a similarity function that can produce a positive definite kernel matrix, we can use the kernel method to conduct sequence-data learning. Hence, our design task is reduced to formulating a function that can characterize the similarity between sequences, provided that the pair-wise similarity matrix is positive definite. In this paper, we borrow an idea from sequence-data (DNA, RNA, and protein) analysis in biological science to measure sequence similarity based on alignment. We show in Section 4.1 the flexibility and effectiveness of this similarity function. Furthermore, we propose the *sequence-alignment* kernel, which fuses symbolic summarizations (in a non-vector space) with numeric descriptions (in a vector space). Because its ability to fuse primary structures with secondary structures that do or do not have a vector-space representation, our proposed kernel features great modeling flexibility.

Finally, to tackle the imbalanced-data learning problem, we propose the kernel boundary alignment (KBA) scheme. For related work of imbalanced-data learning, please refer to [40].

3. EVENT DETECTION AND REPRESENTATION

This section presents the methods we employ for event detection (Section 3.1) and event representation (Section 3.2).

3.1 Event Detection

We use the Kalman filter [6, 31] as the tool for fusing information spatially and temporally from multiple cameras for motion event detection. The Kalman filter is an optimal linear data-smoothing and prediction algorithm. It has been applied extensively in control, signal processing, and navigation applications since its introduction in 1960. Our contribution is in using two-level Kalman filters to fuse data from multiple sources.

The Kalman filter has been widely used to estimate the internal state of a system based on the observation of the system's external behavior [6, 31]. Furthermore, a system's state estimate can be computed and then updated by incorporating external measurements iteratively—*without* recomputing the estimate from scratch each time a new measurement becomes available. Such an iterative process is optimal in the sense that the Kalman filter incorporates all available information from past measurements, weighted by their precision. Optimal information fusion is achieved by combining three factors: (1) knowledge of the system and measurement device dynamics, (2) the statistical description of the system noises, measurement errors, and uncertainty in the system model, and (3) relevant initial state description [31]. While the Kalman filter is optimal only among *linear* estimators, and when certain assumptions about the noise processes are valid, it is easy to implement and is efficient at run time. Work has also been done on relaxing some of the assumptions, such as the Gaussian noise assumption and the linearity assumption [23].

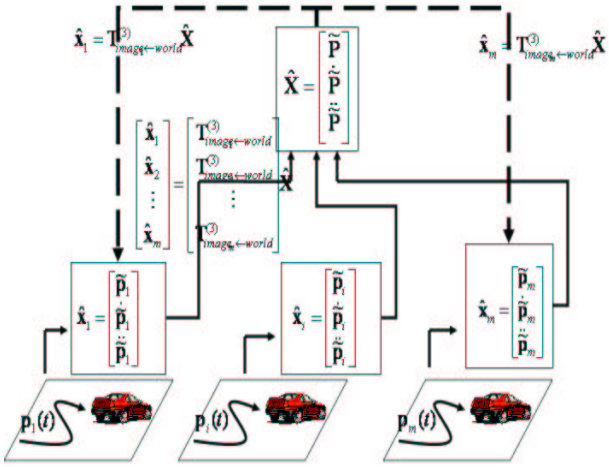


Figure 1: Two-level hierarchical Kalman filter configuration

Suppose that a vehicle is moving in the parking lot. Its trajectory is described in the global reference system by

$$\mathbf{P}(t) = [X(t), Y(t), Z(t)]^T.$$

The trajectory may be observed in camera i, as

$$\mathbf{p}_i(t) = [x_i(t), y_i(t)]^T,$$

where $i = 1, \cdots, m$ (m is the number of cameras used).[1] The question is then how to best estimate $\mathbf{P}(t)$ given $\mathbf{p}_i(t), i = 1, \cdots, m$.

We formulate the solution as a two-level hierarchy of the Kalman filters. Referring to Fig. 1, at the bottom level of the hierarchy, we employ for each camera a Kalman filter to estimate, independently, the position $\mathbf{p}_i(t)$, velocity $\dot{\mathbf{p}}_i(t)$, and acceleration $\ddot{\mathbf{p}}_i(t)$ of the vehicle, based on the tracked image trajectory of the vehicle in *the local camera reference frame*. Or in the Kalman filter jargon, the position, velocity, and acceleration vectors establish the "state" of the system while the image trajectory serves as the "observation" of the system state. At the top level of the hierarchy, we use a sin-

[1]There might be multiple moving vehicles in a busy parking lot, and it may be difficult to synchronize the activities observed with multiple cameras. The question is then how we disambiguate the correspondence of multiple trajectories both spatially and temporally. Spatial and temporal trajectory correspondence can be established through the camera registration and stereopsis correspondence processes [9, 12, 15, 18, 19, 35, 41, 42], which are well established techniques in photogrammetry and computer vision. For our discussion, we will assume that these problems can be solved and we can achieve spatial and temporal registration of vehicle trajectories.

gle Kalman filter to estimate the vehicle's position $\mathbf{P}(t)$, velocity $\dot{\mathbf{P}}(t)$, and acceleration $\ddot{\mathbf{P}}(t)$ in *the global world reference frame*—this time, using the estimated positions, velocities, and accelerations from multiple cameras ($\mathbf{p}_i(t)$, $\dot{\mathbf{p}}_i(t)$, $\ddot{\mathbf{p}}_i(t)$) as observations (the solid feed-upward lines in Fig. 1). This is possible because camera calibration and registration [9, 15, 19, 41, 42] are used for deriving the transform matrix $\mathbf{T}_{image_i \leftarrow world}$. This matrix allows $\mathbf{p}_i$, measured in the reference frame of an individual camera, to be related to $\mathbf{P}$ in the global world system. We also allow dissemination of fused information to individual cameras (the dashed feed-downward lines in Fig. 1) to help to guide image processing.

3.2 Hierarchical, Invariant Representation

Raw trajectory data derived above are in terms of either local or global Cartesian coordinates. Such a representation suffers from at least two problems: (1) the same motion trajectory observed by different cameras will have different representations, and (2) the representation is difficult for a human operator to understand. Our solution is to summarize such raw trajectory data using syntactic and semantic descriptors that are not affected by incidental changes in environmental factors and camera poses. We briefly describe our semantic descriptors here.

We first segment a raw trajectory fused from multiple cameras into fragments. Using a constrained optimization approach under the EM (expectation-maximization) framework, we then label these fragments semantically (e.g., a fragment representing a left turn action). We approximate the acceleration trajectory of a vehicle as a piecewise constant (zeroth-order) or linear (first-order) function in terms of its direction and its magnitude. When the magnitude of acceleration is first order ($\mathbf{r}(t) = \mathbf{r}_o + t\mathbf{r}_1$ in Eq. 5), it gives rise to a motion trajectory that is a concatenation of piecewise polynomials that can be as high as third order (cubic). This is often considered sufficient to describe a multitude of motion curves in the real world (e.g., in computer-aided design [14] and computer graphics [16], piecewise third-order Hermite, Beźier, and B-spline curves are universally used for design and manufacturing). We chop the whole acceleration trajectory $\ddot{\mathbf{P}}(t)$, from $t = t_{min}$ to t_{max} (where $[t_{min}, t_{max}]$ is the time interval that a vehicle is observed by one or more of the surveillance cameras) into, say, k pieces such that $t_o \leq t_1 \leq \cdots \leq t_k$ and $t_o = t_{min}, t_k = t_{max}$

$$\ddot{\mathbf{P}}(t) = \mathbf{r}(t)e^{i\theta(t)}, where$$

$$\begin{cases} \mathbf{r}(t) = \mathbf{r}_o^{(i)} \ \ or \ \ \mathbf{r}_o^{(i)} + t\mathbf{r}_1^{(i)} \\ \theta(t) = \theta_o^{(i)} \ \ or \ \ \theta_o^{(i)} + t\theta_1^{(i)} \end{cases} \quad t_i \leq t < t_{i+1}, i = 0, \cdots, k-1. \tag{5}$$

We employ an iterative expectation and maximization (EM) algorithm [13] to segment trajectories. The EM algorithm consists of two stages: (1) The E-stage hypothesizes the number of segments with their start and stop locations, and (2) the M-stage optimizes the fitting parameters based on the start and stop locations and the number of segments derived from the E-stage. These two steps iterate until the solution converges. Table 1 sketches the pseudo-code of the algorithm (using fitting $\theta(t)$ as an illustration).

We label each segmented fragment based on its acceleration and velocity statistics. More specifically, we denote the initial vehicle velocity when each segment starts as $\mathbf{V}_o$, which can be either zero or nonzero. The acceleration (Eq. 5) can be either of a constant or linearly-varying magnitude and/or of a constant or a linearly-varying direction. For example, if $|\mathbf{r}| \approx 0$, the motion pattern is either "constant speed" or "stop." Segmentation based on θ is meaningful and necessary only when $|\mathbf{r}| > 0$. If $|\mathbf{r}| > 0$, possible motion patterns include "speed up," "slow down," "left turn," and "right turn." "Speed up" and "slow down" can be determined by the sign of $\ddot{\mathbf{P}} \cdot \dot{\mathbf{P}}$. "Left turn" and "right turn" are determined by the sign of $(\ddot{\mathbf{P}} \times \dot{\mathbf{P}})_z$. If $(\ddot{\mathbf{P}} \times \dot{\mathbf{P}})_z > 0$ it is a right turn; otherwise, it is a left turn.

4. EVENT RECOGNITION

Recap our discussion in Section 3. Our descriptors summarize a motion trajectory as an ordered sequence of labels, and each label corresponds to a motion segment with a humanly understandable action. Recognizing and identifying events on such descriptors must handle the ordered nature of the descriptors. Furthermore, in a surveillance setting, positive (suspicious) events are always significantly outnumbered by negative events. As we will explain shortly through an example, this imbalanced training-data situation can skew the decision boundary toward the minority class, and hence cause high rates of false negatives (i.e., failure to identify suspicious events). In this section, we first design a sequence-alignment kernel function to work with SVMs for correlating events. We then propose using kernel boundary alignment (KBA) to deal with the imbalanced training-data problem.

4.1 Sequence Alignment Learning

In the previous section, we have labeled each segmented fragment of a trajectory with a semantic label and its detailed attributes including velocity and acceleration statistics. For convenience, we use a symbol to denote the semantic label. We use 'C' for "**C**onstant speed," 'D' for "slow **D**own," 'L' for "**L**eft turn," 'R' for "**R**ight turn," and 'U' for "speed **U**p." We label each segment with a two-level descriptor: a primary segment symbol and a set of secondary variables (e.g., velocity and acceleration). We use $\mathbf{s}$ to denote a sequence, which comprises the concatenation of segment symbols $s_i \in \mathcal{A}$, where $\mathcal{A}$ is the legal symbol set. We use $\mathbf{v}_i$ to denote the vector of the i^{th} secondary variable.

The following example depicts a sequence with this two-level descriptor. Sequence $\mathbf{s}$ denotes the segmented trajectory with $\mathbf{v}_1$ representing the velocity and $\mathbf{v}_2$ the acceleration. For velocity and acceleration, we use their average values taken place in a segment.

$\mathbf{s}:$	C	D	U	C	L	R	R	L
$\mathbf{v}_1:$	0.7	0.5	0.8	0.8	0.7	0.8	0.6	0.5
$\mathbf{v}_2:$	0.0	-0.2	0.3	0.0	-0.1	0.1	-0.2	-0.1

Now, the trajectory learning problem is converted to the problem of sequence-data learning with secondary variables. For this purpose, we construct a new *sequence-alignment kernel* that can be applied to measure pair-wise similarity between sequences with secondary variables.

4.1.1 Tensor Product Kernel

The sequence-alignment kernel will take into consideration both the degree of conformity of the symbolic summarizations and the similarity between the secondary numerical descriptions (i.e., velocity and acceleration) of the two sequences. Two separate kernels are used for these two criteria and are then combined into a single sequence-alignment kernel through *tensor product*. These are explained below.

Let $\mathbf{x} \in \mathcal{X}$ be a composite structure and $x_1, ..., x_N$ be its "parts," where $x_n \in \mathcal{X}_n$, $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_N$, and N is a positive integer. For our sequence data, $\mathbf{x}$ is a sequence with both primary segment

Table 1: The motion event segmentation process

1) Initialization. Compute a linear fit to the $\theta(t)$ curve between the specified end points, denoted as $[t_{min}, t_{max})$. Using the notation $\theta_{max} = \theta(t_{max})$ and $\theta_{min} = \theta(t_{min})$, we have

$$(\theta_{max} - \theta_{min})t + (t_{min} - t_{max})\theta + (\theta_{min} - \theta_{max})t_{min} + (t_{max} - t_{min})\theta_{min} = 0 \quad (2)$$

2) Refinement. Compute location t_{maxdev} in between t_{min} and t_{max} as the largest deviation of the true acceleration curve from the fitting,

$$t_{maxdev} = argmax_t|(\theta_{max} - \theta_{min})t + (t_{min} - t_{max})\theta(t) + (\theta_{min} - \theta_{max})t_{min} + (t_{max} - t_{min})\theta_{min}|/\Delta \quad (3)$$

$$maxdev = |(\theta_{max} - \theta_{min})t_{maxdev} + (t_{min} - t_{max})\theta(t_{maxdev}) + (\theta_{min} - \theta_{max})t_{min} + (t_{max} - t_{min})\theta_{min}|/\Delta \quad (4)$$

where $\Delta = \sqrt{(\theta_{max} - \theta_{min})^2 + (t_{max} - t_{min})^2}$

3) Iteration. If $maxdev$ is above a preset threshold, break the curve into two sections $[t_{min}, t_{maxdev})$ and $[t_{maxdev}, t_{max})$ and repeat the first two steps using these two new intervals.

symbols and secondary variables. Let x_1 denote its primary symbol sequence, and each other x_i be its $(i-1)^{th}$ secondary vector. Assume that $\mathcal{X}$, $\mathcal{X}_1$,...,$\mathcal{X}_N$ are nonempty sets. We define the *tensor product kernel* as follows:

Definition 1. Tensor Product Kernel. *Given* $\mathbf{x} = (x_1, ..., x_N) \in \mathcal{X}$ *and* $\mathbf{x}' = (x'_1, ..., x'_N) \in \mathcal{X}$. *If* K_1,...,K_N *are (positive definite) kernels defined on* $\mathcal{X}_1 \times \mathcal{X}_1$,..., $\mathcal{X}_N \times \mathcal{X}_N$ *respectively, then their tensor product,* $K_1 \otimes \cdots \otimes K_N$, *defined on* $\mathcal{X} \times \mathcal{X}$ *is*

$$K_1 \otimes \cdots \otimes K_N(\mathbf{x}, \mathbf{x}') = K_1(x_1, x'_1) \cdots K_N(x_N, x'_N). \square$$

Since kernels are closed under product [38], it is easy to see that the tensor product kernel is positive definite if each individual kernel is positive definite.

4.1.2 Sequence-alignment Kernel

To measure the similarity between two sequences, our idea is to first compare their similarity at the symbol level. After the similarity is computed at the primary level, we consider the similarity at the secondary variable level. We then use the tensor product kernel to combine the similarity at the primary and secondary level.

At the primary (segment-symbol) level, we use kernel $K_s(\mathbf{s}, \mathbf{s}')$ to measure symbol-sequence similarity. We define $K_s(\mathbf{s}, \mathbf{s}')$ as a joint probability distribution (p.d.) that assigns a higher probability to more similar sequence pairs. We employ pairs-HMM (PHMM) [38], a generative probability model, to model the joint p.d. of two symbol sequences. (Notice that PHMM is different from HMM, which aims to model the *evolution* of individual sequence data. PHMM is one of many *dynamic-programming* based methods that one can employ to perform string alignment.)

A realization of PHMM is a sequence of states, starting with **START** and finishing with **END**; and in between there are three possible states: states **AB**, **A**, and **B**. State **AB** emits two symbols, state **A** emits one symbol for sequence $\mathbf{a}$ only, and state **B** emits one symbol for sequence $\mathbf{b}$ only. State **AB** has an emission probability distribution $p_{\mathbf{a}_i\mathbf{b}_j}$ for emitting an aligned $\mathbf{a}_i : \mathbf{b}_j$, and states **A** and **B** have distributions $q_{\mathbf{a}_i}$ and $q_{\mathbf{b}_j}$, respectively, for emitting a symbol against a gap, such as $\mathbf{a}_i$: '$-$' and '$-$' : $\mathbf{b}_j$. Parameter δ denotes the transition probability from **AB** to an insert gap state **A** or **B**, ε the probability of staying in an insert state, and τ the probability of a transition into the **END** state. Any particular pair of sequences $\mathbf{a}$ and $\mathbf{b}$ may be generated by exponentially many different realizations. The dynamic programming algorithms can sum over all possible realizations to calculate the joint probability of any two sequences. The overall computational complexity is $O(mn)$, in which m and n are the lengths of the two sequences respectively.

To compute the similarity at the secondary level, we can concatenate all variables into one vector, and employ a traditional vector-space kernel such as an RBF function. Let $K_v(\mathbf{v}, \mathbf{v}')$ denote such a kernel measuring the distance between $\mathbf{v}$ and $\mathbf{v}'$. (Notice that vectors $\mathbf{v}$ and $\mathbf{v}'$ may differ in length since $\mathbf{s}$ and $\mathbf{s}'$ may have different length. We will discuss shortly how we align two vectors into the same length via an example.) Finally, we define the tensor product on $(\mathcal{S} \times \mathcal{S}) \times (\mathcal{V} \times \mathcal{V})$ as

$$(K_s \otimes K_v)((\mathbf{s}, \mathbf{v}), (\mathbf{s}', \mathbf{v}')) = K_s(\mathbf{s}, \mathbf{s}')K_v(\mathbf{v}, \mathbf{v}'). \quad (6)$$

In the following we present an example to show the steps of computing similarity between two sequences using our sequence-alignment kernel.

Example 1. Suppose we have two sequences $(\mathbf{s}, \mathbf{v})$ and $(\mathbf{s}', \mathbf{v}')$ as depicted next. The similarity between the sequences is computed in the following three steps:

$\mathbf{s}$:	C	D	U	C	L	R	R	L
$\mathbf{v}$:	0.7	0.5	0.8	0.8	0.7	0.8	0.6	0.5
$\mathbf{s}'$:	C	U	C	L	R	L	C	
$\mathbf{v}'$:	0.5	0.4	0.4	0.5	0.6	0.6	0.6	

Step 1. Primary symbol-level similarity computation: $K_s(\mathbf{s}, \mathbf{s}')$. By using PHMM, we can obtain the joint p.d. $K_s(\mathbf{s}, \mathbf{s}')$ between symbol sequences $\mathbf{s}$ and $\mathbf{s}'$. As a part of the PHMM computation, two sequences are aligned as follows:

C	D	U	C	L	R	R	L	$-$
C	$-$	U	C	L	R	$-$	L	C

Step 2. Secondary variable-level similarity computation: $K_v(\mathbf{v}, \mathbf{v}')$. The unaligned positions in $\mathbf{v}$ and $\mathbf{v}'$ are padded by zero. We obtain two equal-length vectors, and can compute their similarity by using a traditional SVM kernel, e.g., an RBF function.

0.7	0.5	0.8	0.8	0.7	0.8	0.6	0.5	**0.0**
0.5	**0.0**	0.4	0.4	0.5	0.6	**0.0**	0.6	0.6

Step 3. Tensor fusion: $(K_s \otimes K_v)((\mathbf{s}, \mathbf{v}), (\mathbf{s}', \mathbf{v}'))$. $\square$

There are three advantages of the above *sequence-alignment kernel*. First, it can use any sequence-alignment algorithms to obtain a pair-wise probability distribution for measuring variable-length sequence similarity. (Again, we employ PHMM to perform the measurement.) Second, the kernel considers not only the alignment of symbol strings but also secondary variables, making the similarity measurement between two sequences more informative.

Third, compared with the SVM-Fisher kernel (discussed in Section 2), our sequence-alignment kernel adds the ability to learn from negative training instances, as well from positive training instances.

4.2 Imbalanced Learning via Kernel Boundary Alignment

Skewed class boundary is a subtle but severe problem that arises in using an SVM classifier—in fact in using *any* classifier—for real world problems with imbalanced training data. To understand the nature of the problem, let us consider it in a binary (positive vs. negative) classification setting. Recall that the Bayesian framework estimates the posterior probability using the class conditional and the prior [17]. When the training data are highly imbalanced, it can be inferred that the state of the nature favors the majority class much more than the other. Hence, when ambiguity arises in classifying a particular sample because of similar class conditional densities for the two classes, the Bayesian framework will rely on the large prior in favor of the majority class to break the tie. Consequently, the decision boundary will skew toward the minority class.

To illustrate this skew problem graphically, we use a 2D checkerboard example. The checkerboard divides a 200×200 square into four quadrants. The top-left and bottom-right quadrants contain negative (majority) instances while the top-right and bottom-left quadrants are occupied by positive (minority) instances. The lines between the classes are the "ideal" boundary that separates the two classes. In the rest of the paper, we will use *positive* when referring to minority instances, and *negative* when referring to majority instances.

Figure 2 exhibits the boundary distortion between the two left quadrants in the checkerboard under two different negative/positive training-data ratios, where a black dot with a circle represents a support vector, and its radius represents the weight value α_i of the support vector. The bigger the circle, the larger the α_i. Figure 2(a) shows the SVM class boundary when the ratio of the number of negative instances (in the quadrant above) to the number of positive instances (in the quadrant below) is $10:1$. Figure 2(b) shows the boundary when the ratio increases to $10,000:1$. The boundary in Figure 2(b) is much more skewed towards the positive quadrant than the boundary in Figure 2(a), and hence causes a higher incidence of false negatives.

While the Bayesian framework gives the optimal results (in terms of the smallest average error rate) in a theoretical sense, one has to be careful in applying it to real-world applications. In a real-world application such as security surveillance, the risk (or consequence) of mispredicting a positive event (a false negative) far outweighs that of mispredicting a negative event (a false positive). It is well known that in a binary classification problem, Bayesian risks are defined as:

$$\begin{aligned} R(\alpha_p|\mathbf{x}) &= \lambda_{pp}P(\omega_p|\mathbf{x}) + \lambda_{pn}P(\omega_n|\mathbf{x}) \\ R(\alpha_n|\mathbf{x}) &= \lambda_{np}P(\omega_p|\mathbf{x}) + \lambda_{nn}P(\omega_n|\mathbf{x}) \end{aligned} \tag{7}$$

where p and n refer to the positive and negative events, respectively, λ_{np} refers to the risk of a false negative, and λ_{pn} the risk of a false positive. Which action (α_p or α_n) to take—or which action has a smaller risk—is affected not just by the event likelihood (which directly influences the misclassification error), but also by the risk of mispredictions (λ_{np} and λ_{pn}).

For security surveillance, positive (suspicious) events often occur much less frequently than negative (benign) events. This fact causes imbalanced training data, and thereby results in higher incidence of false negatives. To remedy this boundary-skew problem, we propose an adaptive conformal transformation algorithm. In the remainder of this section, we first outline how our prior work [40] deals with the problem in a vector space (Section *4.2.1*). We then present our solution to sequence-data learning where a discretized variable-length sequence may not have a vector-space representation (Section *4.2.2*).

4.2.1 Conformally Transforming K

In [40], we proposed feature-space adaptive conformal transformation (ACT) for imbalanced-data learning. We showed that conducting conformal transformation adaptively to data distribution, and adjusting the degree of magnification based on feature-space distance (rather than on input-space distance as proposed by [1]) can remedy the imbalanced-data learning problem.

A conformal transformation, also called a conformal mapping, is a transformation T which takes the elements $X \in D$ to elements $Y \in T(D)$ while preserving the local angles between the elements after mapping, where D is a domain in which the elements X reside [10].

Kernel-based methods, such as SVMs, introduce a mapping function Φ which embeds the input space I into a high-dimensional feature space F as a curved Riemannian manifold S where the mapped data reside [1, 7]. A Riemannian metric $g_{ij}(\mathbf{x})$ is then defined for S, which is associated with the kernel function $K(\mathbf{x}, \mathbf{x}')$ by

$$g_{ij}(\mathbf{x}) = \left(\frac{\partial^2 K(\mathbf{x}, \mathbf{x}')}{\partial x_i \partial x'_j} \right)_{\mathbf{x}'=\mathbf{x}}. \tag{8}$$

The metric g_{ij} shows how a local area around $\mathbf{x}$ in I is magnified in F under the mapping of Φ. The idea of conformal transformation in SVMs is to enlarge the margin by increasing the magnification factor $g_{ij}(\mathbf{x})$ around the boundary (represented by support vectors) and to decrease it around the other points. This could be implemented by a conformal transformation of the related kernel $K(\mathbf{x}, \mathbf{x}')$ according to Eq. 8, so that the spatial relationship between the data would not be affected too much [1]. Such a conformal transformation can be depicted as

$$\tilde{K}(\mathbf{x}, \mathbf{x}') = D(\mathbf{x})D(\mathbf{x}')K(\mathbf{x}, \mathbf{x}'). \tag{9}$$

In the above equation, $D(\mathbf{x})$ is a properly defined positive conformal function. $D(\mathbf{x})$ should be chosen in a way such that the new Riemannian metric $\tilde{g}_{ij}(\mathbf{x})$, associated with the new kernel function $\tilde{K}(\mathbf{x}, \mathbf{x}')$, has larger values near the decision boundary. Furthermore, to deal with the skew of the class boundary caused by imbalanced classes, we magnify $\tilde{g}_{ij}(\mathbf{x})$ more in the boundary area close to the minority class. In [40], we demonstrate that an RBF distance function such as

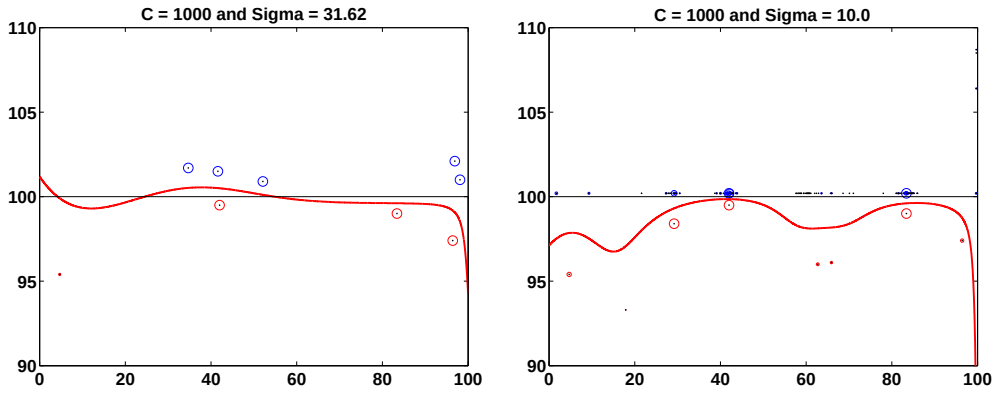
$$D(\mathbf{x}) = \sum_{k \in \mathbf{SV}} \exp(-\frac{|\mathbf{x} - \mathbf{x}_k|}{\tau_k^2}) \tag{10}$$

is a good choice for $D(\mathbf{x})$.

In Eq. 10, we can see that if τ_k^2's are fixed for all support vectors $\mathbf{x}_k$'s, $D(\mathbf{x})$ would be very dependent on the density of support vectors in the neighborhood of $\Phi(\mathbf{x})$. To alleviate this problem, we adaptively tune τ_k^2 according to the spatial distribution of support vectors in F [40]. This goal can be achieved by the following equations:

$$\tau_k^2 = \mathrm{AVG}_{i \in \{\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2 < M,\ y_i \neq y_k\}} \left(\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2 \right). \tag{11}$$

In this equation, the average on the right-hand side comprises all support vectors in $\Phi(\mathbf{x}_k)$'s neighborhood within the radius of M but having a different class label. Here, M is the average distance of the nearest and the farthest support vectors from $\Phi(\mathbf{x}_k)$. Setting



(a) 10:1 (b) 10,000:1

Figure 2: Boundaries of Different Ratios.

τ_k^2 in this way takes into consideration the spatial distribution of the support vectors in F. Although the mapping Φ is unknown, we can play the kernel trick to calculate the distance in F:

$$\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2 = K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_k, \mathbf{x}_k) - 2 * K(\mathbf{x}_i, \mathbf{x}_k). \quad (12)$$

Substituting Eq. 12 into Eq. 11, we can then calculate the τ_k^2 for each support vector, which can adaptively reflect the spatial distribution of the support vector in F, not in I.

When the training dataset is very imbalanced, the class boundary would be skewed towards the minority class in the input space I. We hope that the new metric $\tilde{g}_{ij}(\mathbf{x})$ would further magnify the area far away from a minority support vector $\mathbf{x}_i$ so that the boundary imbalance could be alleviated. Our algorithm thus assigns a multiplier for the τ_k^2 in Eq. 11 to reflect the boundary skew in $D(\mathbf{x})$. We tune $\tilde{\tau}_k^2$ as $\eta_p \tau_k^2$ if $\mathbf{x}_k$ is a minority support vector; otherwise, we tune it as $\eta_n \tau_k^2$. Examining Eq. 10, we can see that $D(\mathbf{x})$ is a monotonously increasing function of τ_k^2. To increase the metric $\tilde{g}_{ij}(\mathbf{x})$ in an area which is not very close to the support vector $\mathbf{x}_k$, it would be better to choose a larger η_p for the τ_k^2 of a minority support vector. For a majority support vector, we can choose a smaller η_n, so as to minimize influence on the class-boundary. We empirically demonstrate that η_p and η_n are proportional to the skew of support vectors, or η_p as $O(\frac{|\mathbf{SV}^-|}{|\mathbf{SV}^+|})$, and η_n as $O(\frac{|\mathbf{SV}^+|}{|\mathbf{SV}^-|})$, where $|\mathbf{SV}^+|$ and $|\mathbf{SV}^-|$ denote the number of minority and majority support vectors, respectively. (Please see [40] for the details of ACT.)

4.2.2 *Modifying* $\mathbf{K}$

For data that do not have a vector-space representation (e.g., sequence data), ACT may not be applicable. In this work we thus propose KBA, which modifies kernel matrix $\mathbf{K}$ based on training-data distribution. Kernel matrix $\mathbf{K}$ contains the pairwise similarity information between all pairs of instances in a training dataset. Hence, in kernel-based methods, all we need is a kernel matrix to learn the classifier, even the data do not reside in a vector space. Notice that KBA is certainly applicable to data that *do* have a vector-space representation since $\mathbf{K} = (k_{xx'}) = K(\mathbf{x}, \mathbf{x}')$.

Now, since a training instance $\mathbf{x}$ might not be a vector, in this paper we introduce a term, *support instance*, to denote $\mathbf{x}$ if its embedded point via $\mathbf{K}$ is a support vector[2]. In this situation, we cannot choose $D(\mathbf{x})$ as in Eq. 10. (It is impossible to calculate the Euclidean distance $|\mathbf{x} - \mathbf{x}_i|$ for non-vector data.) In Section *4.2.1*, we show that $D(\mathbf{x})$ should be chosen in such a way that the spatial resolution of the manifold S would be magnified around the support instances. In other words, if $\mathbf{x}$ is close to a support instance $\mathbf{x}_k$ in F (or in its neighborhood), we hope that $D(\mathbf{x})$ would be larger so as to achieve a greater magnification. In KBA, we use the pairwise-similarity $k_{\mathbf{x}\mathbf{x}_k}$ to measure the distance of $\mathbf{x}$ from $\mathbf{x}_k$ in F. Therefore, we choose $D(\mathbf{x})$ as

$$D(\mathbf{x}) = \sum_{k \in \mathbf{SI}} \exp\left(-\frac{\frac{1}{k_{\mathbf{x}\mathbf{x}_k}} - 1}{\tau_k^2} \right), \quad (13)$$

where $\mathbf{SI}$ denotes the support-instance set, and τ_k^2 controls the magnitude of $D(\mathbf{x})$.

Figure 3 illustrates a $D(\mathbf{x})$ for a given support instance $\mathbf{x}_k$, where we can see that $D(\mathbf{x})$ (y-axis) becomes larger when an instance $\mathbf{x}$ is more similar to $\mathbf{x}_k$ (a larger $k_{\mathbf{x}\mathbf{x}_k}$ in the x-axis), so that there would be more magnification on the spatial resolution around the support vector embedded by $\mathbf{x}_k$ in F. Notice in the figure that $D(\mathbf{x})$ can be shaped very differently with different τ_k^2. We thus need to adaptively choose τ_k^2 as

$$\tau_k^2 = \mathrm{AVG}_{i \in \{Dist^2(\mathbf{x}_i, \mathbf{x}_k) < M,\ y_i \neq y_k\}} \left(Dist^2(\mathbf{x}_i, \mathbf{x}_k) \right), \quad (14)$$

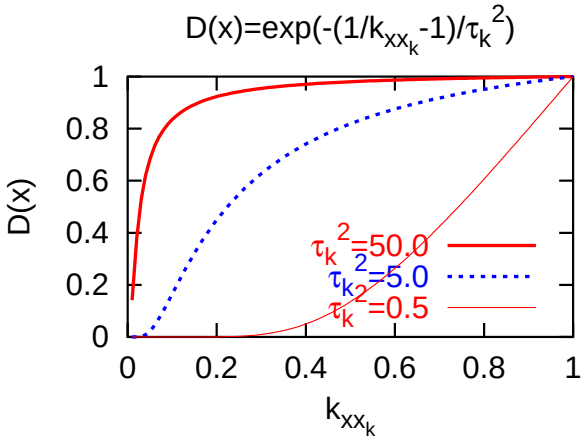


Figure 3: $D(\mathbf{x})$ with Different τ_k^2.

[2]In KBA algorithm, if $\mathbf{x}$ is a support instance, we call both $\mathbf{x}$ and its embedded support vector via $\mathbf{K}$ in F *support instance*.

Input:
X_{train}, X_{test}, $\mathbf{K}$;
θ; /* *stopping threshold* */
T; /* *maximum running iterations* */
Output:
$\mathcal{C}$; /* *output classifier* */
Variables:
$\mathbf{SI}$; /* *support-instance set* */
M; /* *neighborhood range* */
$\mathbf{s}$; /* *a support instance* */
$\mathbf{s}.\tau$; /* *parameter of* $\mathbf{s}$ */
$\mathbf{s}.y$; /* *class label of* $\mathbf{s}$ */
Function Calls:
SVMTrain(X_{train}, $\mathbf{K}$); /* *train classifier* $\mathcal{C}$ */
SVMClassify($X_{test}, \mathcal{C}$); /* *classify* X_{test} *by* $\mathcal{C}$ */
ExtractSI($\mathcal{C}$); /* *obtain* $\mathbf{SI}$ *from* $\mathcal{C}$ */
ComputeM($\mathbf{s}, \mathbf{SI}$); /* *compute* M */
Begin
1) $\mathcal{C} \leftarrow$ SVMTrain($X_{train}, \mathbf{K}$);
2) $\varepsilon_{old} \leftarrow \infty$;
3) $\varepsilon_{new} \leftarrow$ SVMClassify($X_{test}, \mathcal{C}$);
4) $t \leftarrow 0$;
5) **while** $((\varepsilon_{old} - \varepsilon_{new} > \theta) \&\& (t < T))$ {
6) $\mathbf{SI} \leftarrow$ExtractSI($\mathcal{C}$);
7) $\eta_p \leftarrow O(\frac{|\mathbf{SI}^-|}{|\mathbf{SI}^+|})$, $\eta_n \leftarrow O(\frac{|\mathbf{SI}^+|}{|\mathbf{SI}^-|})$;
8) **for** each $\mathbf{s} \in \mathbf{SI}${
9) $M \leftarrow$ComputeM($\mathbf{s}, \mathbf{SI}$);
10) $\mathbf{s}.\tau \leftarrow \sqrt{\text{AVG}_{i \in \{Dist^2(\mathbf{s}_i, \mathbf{s}) < M,\ \mathbf{s}_i.y \neq \mathbf{s}.y\}} (Dist^2(\mathbf{s}_i, \mathbf{s}))}$;
11) **if** $\mathbf{s} \in \mathbf{SI}^+$ **then** /* *a minority* */
12) $\mathbf{s}.\tau \leftarrow \sqrt{\eta_p} \times \mathbf{s}.\tau$;
13) **else** /* *a majority* */
14) $\mathbf{s}.\tau \leftarrow \sqrt{\eta_n} \times \mathbf{s}.\tau$;}
15) $D(\mathbf{x}) = \sum_{\mathbf{s} \in SI} \exp\left(-\frac{\frac{1}{k_{\mathbf{xs}}} - 1}{\mathbf{s}.\tau^2}\right)$
16) **for** each k_{ij} in $\mathbf{K}${
17) $k_{ij} \leftarrow D(\mathbf{x}_i) \times D(\mathbf{x}_j) \times k_{ij}$;}
18) $\mathcal{C} \leftarrow$ SVMTrain($X_{train}, \mathbf{K}$);
19) $\varepsilon_{old} \leftarrow \varepsilon_{new}$;
20) $\varepsilon_{new} \leftarrow$ SVMClassify($X_{test}, \mathcal{C}$);
21) $t \leftarrow t + 1$;}
22) **return** $\mathcal{C}$;
End

Figure 4: The KBA Algorithm.

where the distance $Dist^2(\mathbf{x}_i, \mathbf{x}_k)$ between two support instances $\mathbf{x}_i$ and $\mathbf{x}_k$ is calculated via the kernel trick as

$$Dist^2(\mathbf{x}_i, \mathbf{x}_k) = k_{\mathbf{x}_i\mathbf{x}_i} + k_{\mathbf{x}_k\mathbf{x}_k} - 2 * k_{\mathbf{x}_i\mathbf{x}_k}. \quad (15)$$

The neighborhood range M in Eq. 14 is chosen as the average of the minimal distance $Dist^2_{min}$ and the maximal distance $Dist^2_{max}$ from $\mathbf{x}_k$. In addition, τ_k^2 is scaled in the same way as we did in Section *4.2.1* for dealing with the imbalanced training-data problem.

Figure 4 summarizes the KBA algorithm. We apply KBA on the training dataset X_{train} until the testing accuracy on X_{test} cannot be further improved. In each iteration, KBA adaptively calculates τ_k^2 for each support instance (step 10), based on the distribution of support instances in feature space F. KBA scales the τ_k^2 according to the negative-to-positive support-instance ratio (steps 11 to 14). Finally, KBA updates the kernel matrix and performs retraining on X_{train} (steps 15 to 18).

5. EXPERIMENTAL RESULTS

We have conducted experiments on detecting suspicious events in a parking-lot setting to validate the effectiveness of our proposed methods. We recorded one hour and a half's video at parking lot-20 on UCSB campus using two cameras. We collected trajectories depicting five motion patterns: *circling*, *zigzag-pattern* or *M-pattern*, *go-straight*, *back-and-forth* and *parking*. We classified these events into benign and suspicious categories. The benign-event category consists of patterns *go-straight* and *parking*, and the suspicious-event category consists of the other three patterns. We are most interested in detecting suspicious events accurately. Specifically, we would like to answer the following three questions:

1. Can the use of the two-level Kalman filter successfully reconstruct motion patterns?
2. Can our sequence-data characterization and learning methods (in particular, the tensor product kernel) work effectively to fuse the degree of conformity of the symbolic summarizations and the similarity between the secondary descriptions?
3. Can KBA reduce the incidence of false negatives while maintaining low incidence of false positives?

We use specificity and sensitivity as the evaluation criteria. We define the *sensitivity* of a learning algorithm as the ratio of the number of true positive (TP) predictions over the number of positive instances (TP+FN) in the test set, or Sensitivity = TP/(TP+FN). The *specificity* is defined as the ratio of the number of true negative (TN) predictions over the number of negative instances (TN + FP) in the test set. For surveillance applications, we care more about the sensitivity and at the same time, hopefully the specificity will not suffer too much from the other side.

Table 2 depicts the two datasets, a balanced and a skewed dataset, which we used to conduct the experiments. The balanced dataset was produced from the recorded video. We then added synthetic trajectories to produce the skewed dataset. For each experiment, we chose 60% of the data as the training set, and the remaining 40% as our testing data. We used PHMM for sequence alignment and selected an RBF function for $K_v(\mathbf{v}, \mathbf{v}')$ that works the best on the dataset. (The kernel and the parameter selection processes are rather routine, so we do not report them here.) We employed the best parameter settings obtained through running a five-fold cross validation, and report average class-prediction accuracy.

Motion Pattern	*Balanced Data Set* # *of Instances*	*Skewed Data Set* # *of Instances*
Circling	22	30
M − pattern	19	22
Back − and − forth	38	40
Benign event	41	3,361

Table 2: Datasets.

Experiment #1: *Kalman filter evaluation.*
For this experiment, two cameras were used to record the activities in the parking lot. Sample images for a circling pattern are shown in Fig. 5(a) and (b).[3] We employed a simple mechanism

[3]To conserve space and to better illustrate the motion trajectories, we superimposed multiple video frames into a single picture for display.

for figure-background separation. Since in our current experiment the camera positions were fixed, we detected the presence of moving objects by performing a simple difference operation between adjacent video frames. We then extracted the moving objects by another difference operation with an adjacent video frame that showed no motion. The Kalman filter was used to track the moving vehicles. It helped in smoothing the trajectories, fusing the trajectories from the two cameras, and providing velocity and acceleration estimates from the raw trajectories. Fig. 5(c) shows the fused raw vehicle trajectories from the two cameras. Sample raw and filtered vehicle trajectories are shown in Fig. 5(d) where the black (dark) curve is the raw vehicle trajectory and the red (light) curve is the Kalman filtered and fused trajectory. The agreement of the two curves demonstrates the effectiveness of our fusion and trajectory reconstruction method.

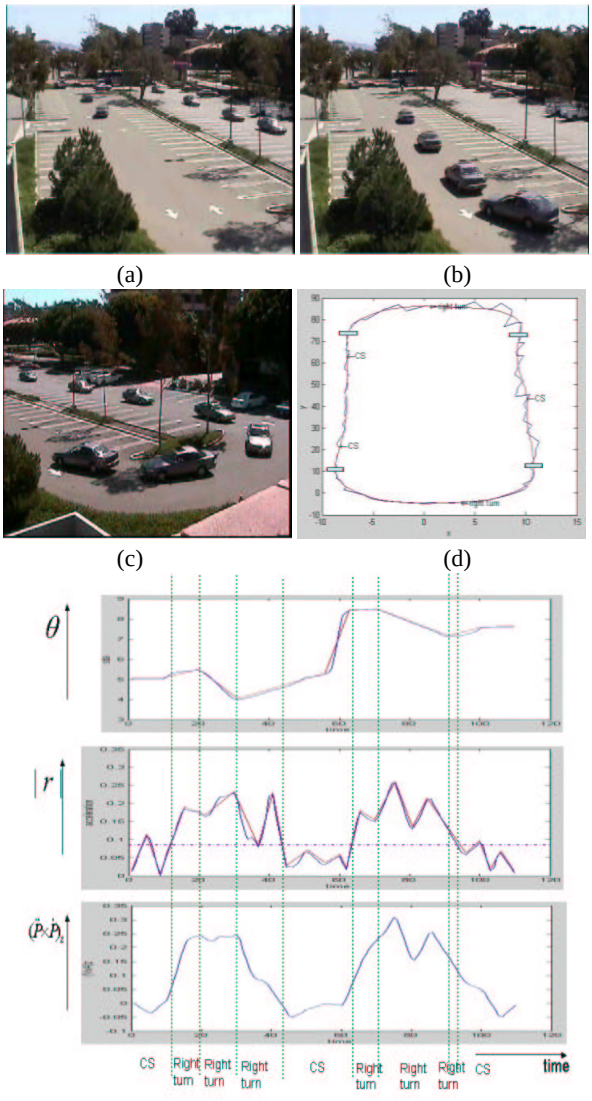


Figure 5: A Circling Pattern. (a) and (b) condensed video footages from the left camera, (c) condensed video footage from the right camera, (d) raw (black or dark) and the Kalman filtered (red or light) trajectories with segment boundaries and labels, and (e) acceleration curves used in segmentation.

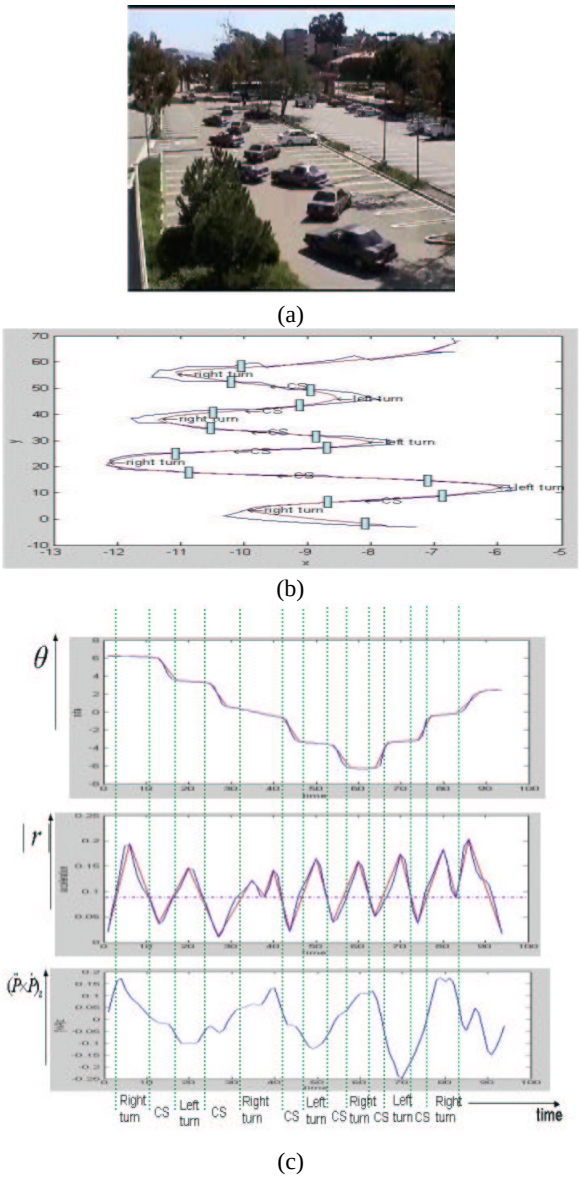


Figure 6: An M-pattern. (a) condensed video footages, (b) raw (black or dark) and the Kalman filtered (red or light) trajectories with segment boundaries and labels, and (c) acceleration curves used in segmentation.

For trajectory segmentation, we imposed the piecewise linearity constraint on both $|\mathbf{r}|$ and θ of the acceleration curve after the trajectory was segmented into two types: where $|\mathbf{r}| > 0$, and where $|\mathbf{r}| \approx 0$. In our experiment, the threshold for $|\mathbf{r}|$ to be considered roughly zero was 0.9. (This level is indicated as the horizontal dashed line in Figs. 5(e) and 6(c).)

In Fig. 5 (d) and (e), we show the sample results of segmenting a circling pattern. Fig. 5(e) depicts the $|\mathbf{r}|$, θ, and $(\ddot{\mathbf{P}} \times \dot{\mathbf{P}})_z$ curves used in segmentation. The θ and $|\mathbf{r}|$ trajectories estimated from the Kalman filter are shown in black while the piecewise linear approximations of these curves using the EM algorithm described before are shown in red. Vertical lines show the beginning and end of each segment. For illustration, the boundaries between adjacent segments and the segment labels are shown in Fig. 5(d) as well.

Fig. 6 shows another result of segmenting an M-pattern. Fig. 6(a) depicts the raw, condensed footage from the left camera only. Fig. 6(b) depicts the raw (in black) and the Kalman filtered (in red) trajectories, and (c) the $|\mathbf{r}|$, θ, and $(\ddot{\mathbf{P}} \times \dot{\mathbf{P}})_z$ curves used in segmentation. The segment boundaries and labels are superimposed on Fig. 6(b). As can be seen from Figs. 5 and 6, the Kalman filter was able to smooth the noisy raw trajectories and arrived at reasonable velocity and acceleration estimates. And our EM segmentation algorithm was able to segment the trajectories into pieces that conformed to the intuitive notion of a human observer. These results demonstrate that our tracking and segmentation algorithms work correctly.

Experiment $\#2$: *Sequence-alignment kernel evaluation*.
We used the balanced dataset to conduct this experiment. We compared the classification accuracy between when we use the primary segment symbols and when we also consider secondary description *velocity*. Figures 7(a) and 7(b) show that when the secondary structure was considered, both sensitivity and specificity were improved. The improvement is marked (about 6%) in sensitivity. In the rest of the experiments, we thus considered both the primary and secondary information.

Experiment $\#3$: KBA *evaluation*.
In this experiment, we examined the effectiveness of KBA on two datasets of different benign/suspicious ratios. The balanced dataset (the second column in Table 2) has a benign/suspicious ratio of about 50%. Figures 7(c) and 7(d) show that the employment of KBA improves sensitivity significantly by 39%, whereas it degrades specificity by just 4%. Next, we repeated the KBA test on the skewed dataset (the third column in Table 2), where the benign/suspicious ratio is less than 3%. Figures 7(e) and 7(f) show that the average sensitivity suffers from a drop from 68% to 35%. After applying KBA, the average sensitivity improved to 70% by giving away just 3% in specificity.

6. CONCLUSIONS

In this paper, we have presented methods for 1) fusing multi-camera surveillance data, 2) characterizing motion patterns and their secondary structure, 3) and conducting statistical learning in an imbalanced training-data setting for detecting rare events. For fusing multi-source data from cameras with overlapping spatial and temporal coverage, we proposed using a two-level hierarchy of Kalman filters. For efficiently summarizing motion events, we studied hierarchical and invariant descriptors. For characterizing motion patterns, we proposed our sequence-alignment kernel, which uses tensor product to fuse a motion sequence's symbolic summarizations (e.g., left-turn and right-turn, which cannot be represented in a vector space) and its secondary numeric characteristics (e.g., velocity, which can be represented in a vector space). When the positive training instances (i.e., suspicious events) are significantly outnumbered by the negative training instances, we showed that kernel methods can suffer from high event-detection errors. To remedy this problem, we proposed an adaptive conformal transformation algorithm to work with our sequence-alignment kernel. Through extensive empirical study in a parking-lot surveillance setting, we showed that our system is highly effective in identifying suspicious events.

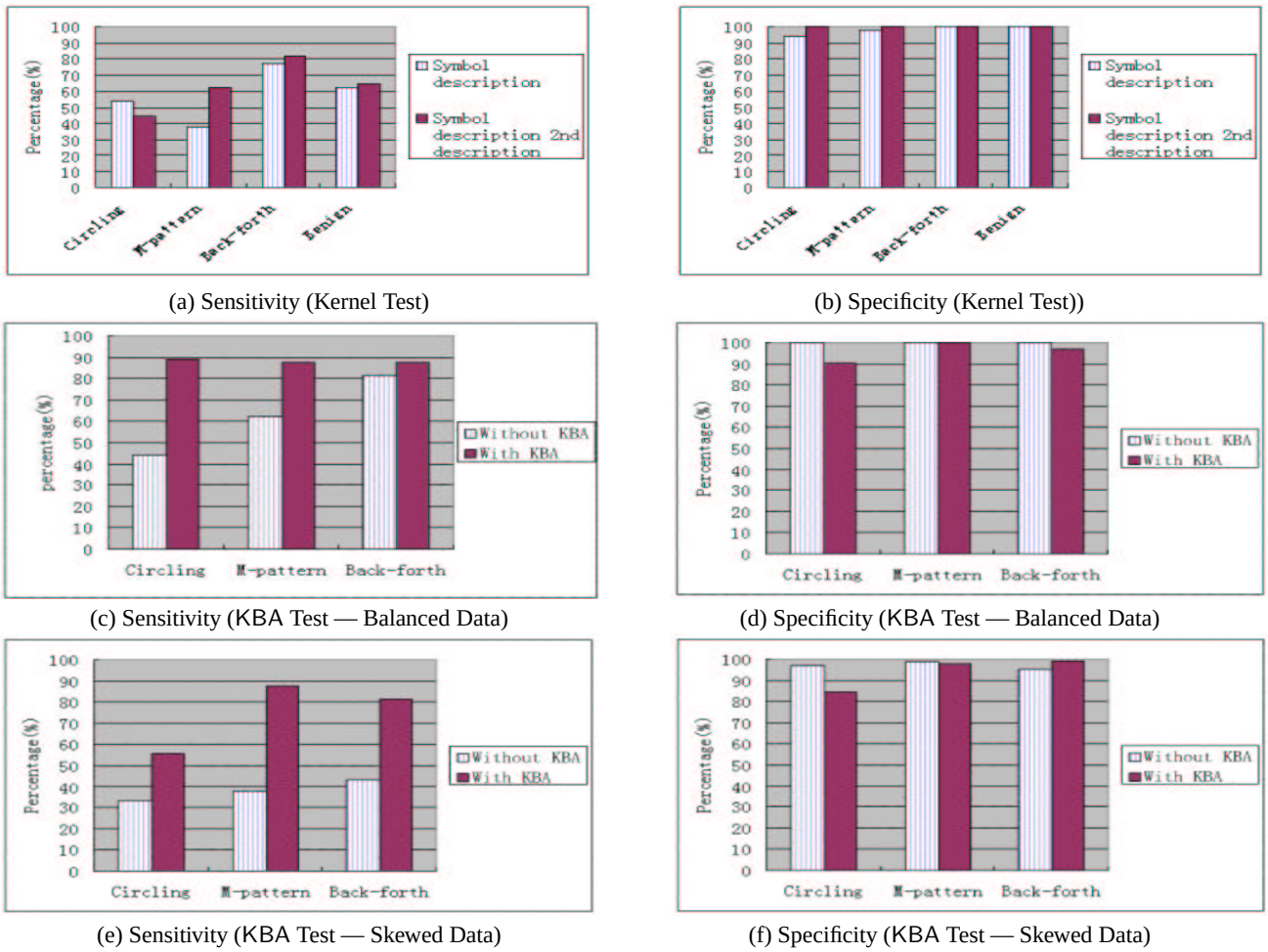
We are currently building a surveillance system with low-resolution Web cams and high-resolution zoom/tilt/pan cameras. We are particularly interested in testing the scalability of our multi-camera fusion scheme (the hierarchical Kalman-filter scheme) with respect to both the number of cameras and the number of objects that are simultaneously tracked. We also plan to investigate more robust parameter-tuning methods for enhancing our kernel-boundary alignment (KBA) algorithm.

Acknowledgment

We would like to acknowledge the support of NSF grants IIS-0133802 (CAREER), IIS-0219885, IIS-9908441, and EIA-0080134.

7. REFERENCES

[1] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel fun ctions. *Neural Networks*, 12(6):783–789, 1999.
[2] R. Azuma. Predicative tracking for augmented reality. *UNC-CS at Chapel Hill, TR-95-007*, 1995.
[3] Y. Bengio. Markovian models for sequential data. *Neural computing serveys*, 1998.
[4] J. Boyd, E. Hunter, P. Kelly, L. Tai, C. Phillips, and R. Jain. Mpi-video infrastructure for dynamic environments. *IEEE International Conference on Multimedia Systems 98*, June 1998.
[5] T. Bozkaya and M. Ozsoyoglu. Distanced-based indexing for high-dimensional metric spaces. *Proc. of ACM SIGMOD*, pages 357–368, 1997.
[6] R. G. Brown. *Introduction to Random Signal Analysis and Kalman filtering*. Wiley, New York, NY, 1983.
[7] C. Burges. *Geometry and Invariance in Kernel Based Methods. In Adv. in Kernel Methods: Support Vector Learning*. MIT Press, 1999.
[8] D. Chudova and P. Smyth. Pattern discovery in sequences under a markov assumption. *ACM SIGKDD*, 2002.
[9] E. Church. Revised Geometry of the Aerial Photograph. *Bulletin of Aerial Photogrammetry*, 15, 1945.
[10] H. Cohn. *Conformal Mapping on Riemann Surfaces*. Dover Pubns, 1980.
[11] R. T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, and L. Wixson. A system for video surveillance and monitoring (VSAM project final report). *CMU Technical Report CMU-RI-TR-00-12*, 2000.
[12] D. F. DeMenthon and L. S. Davis. Model-based object pose in 25 lines of code. *Int. J. Comput. Vision*, 15:123–141, 1995.
[13] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification, 2nd Ed.* Wiley, New York, 2001.
[14] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, San Diego, CA, 4 edition, 1997.
[15] O. Faugeras. *Three-Dimensional Computer Vision*. MIT Press, Cambridge, MA, 1993.
[16] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice, 2nd ed.* Addison-Wesley, Reading, MA, 1990.
[17] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, Boston, MA, 2 edition, 1990.
[18] R. Haralick, H. Joo, C. Lee, X. Zhuang, V. Vaidya, and M. Kim. Pose estimation from corresponding point data. *IEEE Trans. Syst., Man, Cybern.*, 19:1426–46, 1989.
[19] R. Horaud, F. Dornaika, B. Lamiroy, and S. Christy. Object pose: The link between weak perspective, paraperspective and full perspective. *Int. J. Comput. Vision*, 22:173–189, 1997.
[20] T. S. Jaakkola, M. Diekhans, and D. Haussler. Using the fisher kernel method to detect remote protein homologies. *Proceedings of the seventh international conference on intellige nt systems for molecular biology*, 1999.
[21] T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. *In Advances in Neural Information Processing Systems*, 1998.
[22] T. S. Jaakola and D. Haussler. Probabilistic kernel regression models. In *Conference of AI and Statistics*, 1999.
[23] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte. A New Approach for Filtering Nonlinear Systems. In *Proc. Amiercan Control Conference*, Seattle, WA, 1995.
[24] K. Kanatani. Optimal homography computation with a reliability measure. *Proc. IAPR Workshop Machine Vision Applications*, Nov. 1998.
[25] V. Kettnaker and R. Zabih. Bayesian multi-camera surveillance. *CVPR*, 1999.



(a) Sensitivity (Kernel Test)

(b) Specificity (Kernel Test))

(c) Sensitivity (KBA Test — Balanced Data)

(d) Specificity (KBA Test — Balanced Data)

(e) Sensitivity (KBA Test — Skewed Data)

(f) Specificity (KBA Test — Skewed Data)

Figure 7: Sensitivity and Specificity of Three Test Cases.

[26] L. Lee, R. Romano, and G. Stein. Monitoring activities from multiple video streams: Establishing a common coordinate system. *IEEE Trans. on PAMI*, 22(8), Aug. 2000.

[27] Leslie, Christin, E. Eskin, A. Cohen, J. Weston, and W. S. Noble. Mismatch string kernels for svm protein classification. *Neural Information Processing Systems*, 2002.

[28] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: a string kernel for svm protein classification. *Proceedings of the Pacific Symposium on Biocomputing. World Scient ific*, 2002.

[29] J. Lou, H. Yang, W. Hu, and T. Tan. Visual vehicle tracking using an improved ekf. *ACCV*, 2002.

[30] S. Maybank, A. Worrall, and G. Sullivan. Filter for car tracking based on acceleration and steering angle. *Proc. of British Machine Vision Conference*, 1996.

[31] P. S. Maybeck. *Stochastic Models, Estimation, and Control, vol. 1*. Academic Press, New York, NY, 1979.

[32] I. Pavlidis and V. Morellas. Two examples of indoor and outdoor surveillance systems: Motivation, design, and testing. *Proceedings 2nd European Workshop on Advanced Video-Based Surveillance*, 2001.

[33] I. Pavlidis, V. Morellas, P. Tsiamyrtzis, and S. Harp. Urban surveillance systems: From the laboratory to the commercial world. *Proc. of the IEEE*, 89(10), 2001.

[34] C. Regazzoni and P. K. Varshney. Multisensor surveillance systems based on image and video data. *Proc. of the IEEE Conf. on Image Proc.*, 2002.

[35] F. W. Sears. *Optics, 3rd Ed.* Addison-Wesley, Reading, MA, 1958.

[36] T. Starner and A. Pentland. Visual recognition of american sign language using hidden markov models. Technical Report Master's Thesis, MIT, Program in Media Arts & Sciences, MIT Media Laboratory, 1994.

[37] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[38] C. Watkins. Dynamic alignment kernels. *Technical Report, Department of Computer Science, University of London*, 1999.

[39] G. Welch and G. Bishop. An introduction to the kalman filter. *UNC-Chapel Hill, TR 95-041*, 2002.

[40] G. Wu and E. Chang. Adaptive feature-space conformal transformation for learning imbalanced data. *International Conference of Machine Learning (ICML)*, August 2003.

[41] G. Xu and Z. Zhang. *Epipolar Geometry in Stereo, Motion and Object Recognition*. Kluwer Academic Publishers, The Netherlands, 1996.

[42] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Analy. Machine Intell.*, 22:1330–4, 2000.