Time Series Analysis Using Geometric Template Matching

Jordan Frank, *Student Member*, *IEEE*, Shie Mannor, *Senior Member*, *IEEE*, Joelle Pineau, and Doina Precup

Abstract—We present a novel framework for analyzing univariate time series data. At the heart of the approach is a versatile algorithm for measuring the similarity of two segments of time series called geometric template matching (GeTeM). First, we use GeTeM to compute a similarity measure for clustering and nearest-neighbor classification. Next, we present a semi-supervised learning algorithm that uses the similarity measure with hierarchical clustering in order to improve classification performance when unlabeled training data are available. Finally, we present a boosting framework called TDEBOOST, which uses an ensemble of GeTeM classifiers. TDEBOOST augments the traditional boosting approach with an additional step in which the features used as inputs to the classifier are adapted at each step to improve the training error. We empirically evaluate the proposed approaches on several datasets, such as accelerometer data collected from wearable sensors and ECG data.

Index Terms—Activity recognition, gait recognition, supervised learning, unsupervised learning, wearable computing, time series classification

1 Introduction

Time series are challenging for machine learning practitioners. Traditional assumptions about data on which classification algorithms operate, namely that the data are independent, do not hold when elements of a time series are considered as individual objects. Moreover, notions of similarity and dissimilarity for time series can be difficult to define and are often problem specific. For instance, one may be interested in comparing trends that occur in time series (e.g., a stock price rising, then falling sharply), without regard for the magnitude of the values (amplitude and offset invariance), the temporal location at which the trend starts (phase invariance), the temporal length of the trend (temporal scaling invariance), or some combination of these [1].

A typical approach for the treatment of time series is to consider windows (i.e., subsequences) of the data and treat the elements in the window as coordinates of a point in a high-dimensional space, where traditional notions of similarity such as euclidean distance can be used. To combat the curse of dimensionality, transformations such as the discrete Fourier transform (DFT), the discrete wavelet transform (DWT), or principal components analysis (PCA) are often applied, and uninformative components or coefficients are discarded.

An alternative approach is to treat the data as a sequence of observations of a dynamical system. Methods in this category usually consider a particular class of dynamical system models and compare two time series by fitting models to the data, then comparing properties of the models. For example, Kalpakis et al. [2] fit autoregressive integrated moving average (ARIMA) models to segments of data and compare the parameters. The intuition is that two time series should be considered similar when the underlying physical models that generated them are the same or close. The approach that we propose in this paper follows this line of reasoning. However, we replace the parameterized linear dynamical system models that are typically used [2], [3], [4] with nonparametric nonlinear dynamical system models called time-delay embeddings.

Time-delay embeddings are an effective technique for creating models of periodic time series [5]. They have been used extensively in the analysis of chaotic systems [6], [7], but their use in other fields is limited. A time-delay embedding of a time series is a reconstruction of a nonlinear dynamical system that could have generated the observed data. In the idealized case in which the observations are noise free and capture the degrees of freedom inherent in the dynamical system, time-delay embeddings can be shown to perfectly reconstruct both the state and the dynamics of arbitrary finite-dimensional deterministic dynamical systems [8]. This is of great interest as it allows assessing properties such as dimensionality or predictability (by estimating Lyapunov exponents, for example) of a latent dynamical system of arbitrarily high complexity through a sequence of univariate measurements.

Since the models are nonparametric, the process of comparing two models is more involved. The main contribution of this work is an algorithm called Geometric Template Matching (GeTeM) which uses time-delay embeddings for building models from segments of time series and then computes a score that represents how similar the reconstructed dynamical systems are in terms of their state space as well as their dynamics. GeTeM is computationally efficient and is designed to be used with resource-constrained

- *J. Frank. J. Pineau, and D. Precup are with the School of Computer Science, McGill University, McConnell Engineering Bldg, 3480 University Street, Montreal, QC H3A 0E9, Canada. E-mail: {jordan.frank, jpineau, dprecup}@cs.mcgill.ca.*
- *S. Mannor is with the Department of Electrical Engineering, Faculty of Electrical Engineering, Technion, Fishbach Bldg, Rm. 456, Haifa 32000, Israel. E-mail: shie.mannor@ee.technion.ac.il*

Manuscript received 9 Dec. 2011; revised 30 Mar. 2012; accepted 12 May 2012; published online 22 May 2012.
Recommended for acceptance by A.J. Storkey.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-2011-12-0880.
Digital Object Identifier no. 10.1109/TPAMI.2012.121.

embedded devices. To assess the quality of the GeTeM similarity measure, we consider its use in a number of different frameworks. First, we treat the GeTeM scores as a measure of similarity between segments for clustering and nearest neighbor classification. Second, we present a semi-supervised learning algorithm that uses GeTeM to find structure in unlabeled training data in order to augment small labeled datasets. Finally, we treat time-delay embedding models as pseudo-radial-basis functions, using the GeTeM score as the distance function. This provides a means for extracting features from time series, which can be used to train classifiers. We propose a boosting framework based on these features, which constructs both an ensemble of classifiers as well as an ensemble of feature extractors, in order to minimize the training error. For all of the proposed algorithms, we evaluate their performance on real data, comparing against existing techniques where appropriate.

This paper is organized as follows: In Section 2, we provide an overview of existing approaches for treating time series in a machine learning context and a brief introduction to time-delay embeddings. In Section 3, we present the main contribution of this work, the GeTeM algorithm. Section 4 describes the use of GeTeM for clustering, classification, and semi-supervised learning, and includes an empirical evaluation with real data for each of these three settings. We present a more powerful classification algorithm for long time series, called TDEBOOST, which uses GeTeM for feature extraction, in Section 5. TDEBOOST is evaluated on a large, challenging dataset that we collected and are making available for benchmarking time series classification algorithms, particularly those suited for gait recognition. Finally, we conclude with a discussion and ideas for future work in Section 6.

2 Background

In this section, we review a number of existing approaches for machine learning in time series and give a brief overview of time-delay embeddings, which form the basis of our technique.

2.1 Related Work

We begin by giving an overview of some existing approaches for treating time series in a machine learning context. We acknowledge that this is a large field, and our selection of topics in this section is guided by the specific problem settings considered in this paper.

2.1.1 Machine Learning with Time Series

In this paper, we consider supervised, unsupervised, and semi-supervised learning with time series data. For supervised learning, we consider time series which are associated with some finite (typically small) set of output (class) labels. Each time series can have a single class label, or each value in the time series can have a class label. The former is typically used with short time series and the latter with long time series. The goal is to produce a classifier that can label unseen data, based on a labeled training set. Performance is typically measured as the accuracy of the classifier on data on which it was not trained. When each individual time series is associated with a single class label, standard metrics for supervised learning algorithms, such as precision and recall, are used. If labels are associated with the values in the time series, other metrics can provide additional insight. Ward et al. [9] present a thorough taxonomy of performance metrics for these types of problems. For example, in physical activity detection, a single time series may contain segments of walking, running, etc., which are referred to as *events*. One may be interested only in detecting whether or not an event occurred, and may be less interested in detecting at exactly what time the event began or finished. Therefore, classifying any individual value or window (*frame*) within an event could be considered a correct labeling for the entire event, even if some of the frames are mislabeled. For our purposes, we ignore these event-based metrics, and only consider standard frame metrics (in the language of [9]) in which scores are computed based on the number of labels correctly predicted.

Unsupervised learning is concerned with finding structure in unlabeled data. For example, in clustering one tries to group the data into clusters. A good clustering is one in which the data within a cluster are more similar to each other than to data in other clusters. This definition relies on having a notion of similarity. Unlike in supervised learning, where performance is easy to measure, the performance of an unsupervised learning algorithm is more difficult to assess. One approach is to compute a score based on the distance between points within a single cluster and the distance between points in different clusters. Two examples of such scores are the Dunn index [10] and the Davies-Bouldin index [11]. Alternatively, if labeled data are available, one can ignore the labels when clustering and assess the quality of a clustering based on whether data within each cluster have consistent class labels.

Semi-supervised learning considers the case in which training data contain both labeled and unlabeled instances. In many real-world problems, unlabeled data are plentiful, but labeling data are costly. For instance, in medical applications, one could obtain large quantities of x-ray images, but labeling the images would require medical doctors or experts in the field. Semi-supervised algorithms exploit the structure of the unlabeled training data to obtain classifiers that perform better than ones trained only on the labeled portion of the data [12].

2.1.2 Types of Time Series Data

In this work, we consider univariate, real-valued time series of the type generated by measurements of some latent dynamical system. Before we discuss methods for processing such data, it is important to note the distinction between *time series* and *sequential data*. In both cases, the data consist of a sequence, or list of values, in which the order is important. Time series is a subclass of sequential data where the longitudinal component corresponds to time. Not all datasets that are considered "time series" benchmarks in the literature have this property. For instance, a number of the datasets in the UCR Time Series Classification Database [13] consist of sequential data that are not time series. An example is the Swedish Leaf dataset [14], where each object is generated by taking a static image of a leaf, choosing a starting point, tracing the outline, and measuring the distance from the centroid of the leaf. This gives a sequence of values which the authors call a "pseudo time series"; however, the temporal aspect in this case is artificial.

We emphasize this distinction because, for many instances of sequential data, the assumptions that underlie

dynamical systems-based approaches for the classification of time series are not satisfied. In these situations, pattern-matching approaches are more appropriate as the data are typically well segmented and well aligned. Surprisingly, though, as we will see later in this paper, approaches based on fitting dynamical systems often outperform pattern-matching approaches even on these types of data.

Time series also come in different flavors. Most important for our work is the distinction between segmented (short) time series and long time series; techniques developed for the former often require special care when applied to the latter, as we will see in the following sections. There are a large number of techniques for analyzing time series, and the list of techniques we describe in the following sections is not meant to be exhaustive. We have selected techniques that are commonly used and for which published results exist on the datasets that we use to evaluate our proposed approach.

2.1.3 Segmented Time Series

In the time series classification, clustering, and indexing literature, most work focuses on applications in which there is some set of patterns of interest, or motifs, in the data. The data are segmented such that some fixed number (typically 1) of repetitions of a motif occur in each segment. For example, an ECG dataset might contain 100 segments, where each segment represents one period of the heart's electrical cycle. In classification, each class is associated with a particular motif, and the goal is to determine which motif occurs in each segment. For clustering, the goal is typically to partition the data such that segments within a partition have motifs that are similar. In the indexing problem, the goal is to compile a set of data into a data structure such that, given a new segment, segments that most closely resemble it can quickly be retrieved. In all these applications, the segments can be treated as individual objects.

The simplest approach for comparing segments of equal length is to treat the n elements of the segment as coordinates of points in $\mathbb{R}^n$, then compute some traditional measure of distance, typically an L_p norm such as the euclidean distance or auto-correlation. Normalizing the magnitude of the values in the data typically improves the performance of this approach.

When data are not well aligned or the segments are of unequal length, dynamic time warping (DTW) [15] can be used to align the segments before computing the distance between them. DTW stretches sequences in the temporal dimension in order to minimize some measure of distance (typically euclidean) between them. It is common to impose a locality constraint which limits the amount of stretching between subsequent elements in a sequence. Constrained DTW (cDTW) is computationally more efficient than DTW, and has been shown to outperform DTW when used as a distance measure for certain classification problems [16]. Computing the DTW distance can be done using dynamic programming at a computational cost of $O(nm)$ (where n and m are the lengths of the two sequences) or $O(rn)$ if the locality constraint with threshold r is used.

Time series segments often contain hundreds or thousands of measurements. The approaches reviewed so far involve distances in high-dimensional spaces, and therefore often suffer from the "curse of dimensionality." It is well known that nearest neighbor classification in high dimensions is wrought with pitfalls, from both a computational perspective (data structures that work well in low dimensions do not offer the same computational advantages in high dimensions) and in terms of the quality of their results (the difference in distance between one's nearest and farthest neighbors in high dimensions is often small) [17]. To address these problems, the dimensionality of the sequences is often reduced. Two approaches that are well suited for time series data are the discrete Fourier transform[1] [18], [19] and the discrete Wavelet transform [20]; traditional dimensionality reduction techniques such as PCA [21] can also be used. All three of these approaches operate by projecting the data into an alternative basis, then discarding dimensions that are deemed less informative. The DFT decomposes a signal into its constituent frequencies and represents it as coefficients of a linear combination of sinusoids. The DWT is similar to the DFT, but augments the frequency information with temporal locality information by using a basis generated by a wavelet function. Commonly used wavelet functions include the Haar wavelet and the Daubechies wavelet [22]. Unlike the DFT and DWT, for which the bases are predetermined, PCA finds the orthonormal basis such that the greatest variance by any projection of the data lies on the first axis, the second greatest variance on the second axis, and so on. Dimensionality reduction involving the DFT and the DWT typically assume that high-frequency components are more likely to account for noise, and can be discarded without losing information about the signal. In the case of PCA, the assumption is that directions on which the data have low variance can be discarded.

The approaches mentioned so far all attempt to match shapes or motifs by looking at the raw data or some projection thereof. We will refer to these types of techniques as *pattern matching*. An alternative viewpoint is to consider the raw data to be observations from some underlying dynamical system, and to compare properties of the systems that generated the different sequences of data. These approaches operate by choosing some class of dynamical systems, typically parameterized by some finite set of parameters, fitting a model to each sequence, and then computing a measure of distance based on the parameters of the fitted models.

One popular family of dynamical system models are autoregressive models ($\mathrm{AR}(p)$) [23], where each value in the sequence is assumed to be a linear function of some number p of previous elements. Dynamical system models are often used for data generated by periodic dynamical systems, such as measurements of cardiac or pulmonary systems. For example, Ge et al. [3] modeled ECG data with an $\mathrm{AR}(p)$ model, then used the parameters in a linear classifier and achieved over 92 percent accuracy in identifying cardiac disorders. Kalpakis et al. [2] used the euclidean distance between the LPC cepstrum, which is a function of the $\mathrm{AR}(p)$ parameters. This distance measure was used to classify a number of datasets, including ECG data, per capita

1. The discrete Fourier transform is often called the fast Fourier transform (FFT), which simply refers to the algorithm commonly used to calculate the DFT.

personal income, temperature readings, and population levels. The performance of the LPC cepstrum was competitive with the DFT, DWT, and PCA-based approaches.

Another popular dynamical system model is the hidden Markov model (HMM) [24], which assumes an underlying Markov process over a set of hidden states. To perform time series classification, one can learn an HMM for each class from training data. For new data, each HMM can be used to estimate the likelihood of the observations; the class associated with the HMM giving the highest likelihood is then picked. Similarly, for clustering, each cluster has an associated HMM and the likelihood is used as a similarity measure to associate sequences with the clusters [25]. Alternatively, one can learn an HMM for each sequence and compute a measure of similarity between the HMM parameters, such as the Kullback-Leibler divergence between the distributions [26].

2.1.4 Long Time Series

In the real world, time series are more likely to arise as the result of some measurement apparatus, such as a wearable sensor, an ECG or EEG recording device, or a stock price index. For these types of data, there is often no natural notion of a segment. Even if segments exist, the data may be too noisy for automatic segmentation; manual segmentation on long sequences is also very tedious. To avoid segmentation, a common approach is to use a sliding window of data and extract overlapping segments. By increasing the amount of overlap, one increases the chances that a segment will contain a pattern of interest, which might otherwise be split among neighboring segments. However, increasing the size of the overlap increases the number of segments considered. The size of the window greatly affects the performance of this approach. If the window is too small, longer patterns of interest will be missed. If the window is too large, dimensionality reduction techniques are required, which may discard properties of interest (such as high-frequency components). A standard approach is to take multiple passes over the data with different window sizes. Segments produced by a sliding window can then be compared using the approaches for segmented data described in the previous section.

While the goal in the supervised learning setting with segmented data is to assign a label to each segment, with long time series, the goal is typically to assign a label to each element in the time series. With a sliding window, each data element will be present in multiple segments; typically, each segment is labeled using one of the methods described in the previous section, and then each element of the time series is assigned the majority label over the segments that contain it.

One example application of long time series is classifying primitive physical activities from data provided by wearable sensors, such as accelerometers. For the purpose of detecting physical activities, nearly all existing techniques identify periodic motifs in the data, such as FFTs and cepstral coefficients. For example, the activity recognition system of Bao and Intille [27] uses mean, energy, spectral entropy, and correlation features, while the experiments of Heinze et al. [28] considered mean, integration, and variance features. Ravi et al. [29] extracted mean, standard deviation, energy, and correlation features, and Huynh and Schiele [30] analyzed mean, variance, energy, spectral entropy, correlation coefficients, and log FFT frequency bands. Lester et al. [31] used linear and log-scale FFT frequency coefficients, cepstral coefficients, spectral entropy, band-pass filter coefficients, correlations, integrals, means, and variances over windows of various sizes.

2.2 Technical Background

We now present a brief overview of time-delay embedding, the main approach on which we build; we refer the reader to the standard text by Kantz and Schreiber [6] for further details.

The purpose of time-delay embeddings is to reconstruct the state and dynamics of an unknown stationary, deterministic dynamical system from measurements or observations of that system taken over time. The state of the dynamical system at time t, $x_t \in \mathbb{R}^k$, contains all the information necessary to compute the future of the system at all times following t (determinism), and the dynamics are assumed to not change over time (stationarity). The set of all states is called *phase space*. The state cannot be measured directly, and even the true dimension of the phase space k may not be known. However, a time series $o = \langle o_t = \omega(x_t) \rangle$ can be obtained by an unknown measurement function $\omega : \mathbb{R}^k \to \mathbb{R}$, which is a smooth map from the phase space to a scalar value. Throughout this paper, we use "smooth" to mean continuously differentiable. While individual measurements do not provide a complete description of the system (with the exception of 1D systems), sequences of successive measurements do. Intuitively, the sequence $\langle o_t, o_{t-\delta 1}, \ldots, o_{t-\delta_{m-1}} \rangle$, referred to as *delay coordinates*, for sufficiently large m and sufficiently small δ_i, contains information about the derivatives up to order $m-1$. Therefore, if the dimension of the system is less than m, one should have enough information to describe the evolution as a system of $m-1$ differential equations.

The key problem is to determine how many measurements m have to be considered, and at what times they should be taken (i.e., what are the values of δ_i), in order to capture the system dynamics well. In particular, the map from $\mathbb{R}^k$ to $\mathbb{R}^m$, encapsulating the measurement and subsequent projection to delay coordinates, should be one-to-one and preserve local structure. If this were the case, by looking at the time series in $\mathbb{R}^m$ (which is also called the *reconstruction space*), one would essentially have all the information from the true system state and dynamics.

Let F be a function of a sequence o of T measurements, a time index t, and time-delay parameters m and τ, defined as:

$$F(o, m, \tau, t) = \langle o_t, o_{t+\tau}, o_{t+2\tau}, \ldots, o_{t+(m-1)\tau} \rangle. \quad (1)$$

The function F represents the state of the system at time t by an m-length vector of τ-lagged measurements. The true state of the system x_t is assumed to lie on an attractor $A \subset \mathbb{R}^k$ in an unknown phase space of dimension k. We use the term "attractor," as in dynamical system theory, to mean a closed subset of states toward which the system will evolve from many different initial conditions; once the state reaches the attractor, it will typically remain inside it indefinitely (in the absence of external perturbations). An

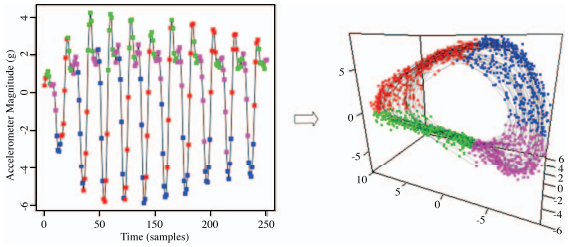


Fig. 1. Example of time-delay embedding for accelerometer data from biking activity. Left: Raw accelerometer data. Right: Time-delay embedding with $m = 3, \tau = 11$. The colored points in the left diagram are mapped to points of the same color in the embedding.

embedding is a map from A into the reconstruction space $\mathbb{R}^m$ that is one-to-one and preserves local differential structure. In differential topology, such a map is called a local diffeomorphism on A.

Takens showed that if A is a d-dimensional smooth compact manifold, then provided $m > 2d$, $\tau > 0$, and the attractor contains no periodic orbits of length τ or 2τ and only finitely many periodic orbits of length $3\tau, 4\tau, \ldots, m\tau$, then for almost every smooth function ω, the map from $\mathbb{R}^k$ to the time-delay reconstruction (i.e., the measurement function ω composed with F) is an embedding [8]. In other words, any time-delay reconstruction will accurately preserve the dynamics of the underlying dynamical system, provided m is large enough and τ does not conflict with any periodic orbits in the system. However, from a practical standpoint, m should be as small as possible as it controls the dimensionality of the reconstruction as well as the size of the time window considered for each point in the reconstruction. If the data are sampled at a rate of r measurements per second, then the time window captured by each point is $((m-1)\tau + 1)/r$ seconds.

Takens' theorem only holds when the measurement function ω is deterministic, but in practice measurements are corrupted by noise from a variety of sources, ranging from the measurement apparatus to rounding errors due to the finite precision of a computer. In the presence of noise, the choice of m and τ becomes very important. There are a number of techniques for choosing good values for m and τ based on geometrical and spectral properties of the data [32], [33]. In practice, since computing the reconstruction for a given parameter setting requires little computation, it is feasible to perform a grid search over a reasonable parameter space and cross validation using a problem-specific performance criterion. Empirically, we have found that for a choice of τ that does not conflict with periodic events in the time series, the quality of the embedding is robust to the choice of m. Choosing a small value for τ will make it less likely that there is a conflict, but for smaller values of τ, the effects of noise substantially increase with the value of m. Therefore, we prefer larger values of τ, which make the quality of the embedding more robust to the choice of m.

Fig. 1 illustrates the concept of time-delay embedding using a segment of data captured from an accelerometer placed on the hip of a subject while riding a stationary bicycle. Clearly, the system evolves along a set of tightly clustered trajectories, in a periodic fashion. This is quite intuitive, considering that cycling involves a periodic, roughly 2D leg movement. However, it is very nice that this structure can be reconstructed automatically from the time series on the left.

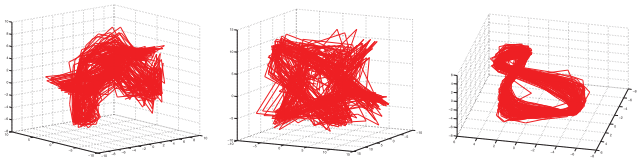


Fig. 2. Time-delay embedding of three example activities for walking (left), running (middle), and biking (right), $m = 3, \tau = 4$.

Methods developed for nonlinear time series analysis have not been widely used in machine learning, but are quite popular in other areas of research, especially in the study of electrophysiological time series data [34 and references therein]. By estimating the dimensionality of the underlying attractors captured in electroencephalographic (EEG) and intracranial electroencephalogram (IEEG) data, researchers have been able to characterize various changes in the dynamics of the brain, such as different stages of sleep [35], and epileptic seizures [36]. More recently, time-delay embeddings have been used to model epileptic seizures in order to learn effective treatment regimes [37].

3 Geometric Template Matching

Given a time series o consisting of T univariate measurements, the *time-delay reconstruction* of o is given by the following sequence of points in $\mathbb{R}^m$:

$$u = \langle \mathbf{u}_t = F(o, m, \tau, t) \rangle_{t=1}^{M},$$

where $M = T - (m-1)\tau$. We refer to u as the *model* for the sequence o. The points u_t represent the reconstructed states, while vectors connecting successive pairs of points u_i and u_{i+1} represent the dynamics. The Geometric Template Matching algorithm is an approach for comparing two models in terms of both their states and dynamics.[2] To gain an intuition as to how well such a method may work, consider three examples of embedding results from noisy accelerometer data for three different activities (walking, running, and biking), presented in Fig. 2. All are embedded in a 3D space. By visual inspection, the three manifolds look rather different; intuitively, a distance measure on the state and dynamics of these models should be able to distinguish between them easily. Note that walking and running appear more similar than biking, as one would expect.

To compare $o = \langle o_i \rangle_{i=1}^{N}$ with another time series $o' = \langle o'_i \rangle_{i=1}^{N'}$, GeTeM starts by building models $u = \langle \mathbf{u}_i \rangle_{i=1}^{M}$ and $u' = \langle \mathbf{u}'_i \rangle_{i=1}^{M'}$ for o and o', respectively, using the same time-delay parameters m and τ. Let $n_{i,1}, \ldots, n_{i,k}$ be the indices in u of the k nearest neighbors of $\mathbf{u}'_i$ in terms of euclidean distance. That is,

$$n_{i,1} = \operatorname*{argmin}_j \|\mathbf{u}'_i - \mathbf{u}_j\|,$$

$$n_{i,2} = \operatorname*{argmin}_{j \neq n_{i,1}} \|\mathbf{u}'_i - \mathbf{u}_j\|, \text{ etc.}$$

2. Reference implementation available at https://github.com/jwf/tdetools.

Let $\bar{\mathbf{u}}_i = k^{-1}\sum_{j=1}^{k}\mathbf{u}_{n_{i,j}}$ and $\bar{\mathbf{u}}_{i+1} = k^{-1}\sum_{j=1}^{k}\mathbf{u}_{1+n_{i,j}}$ be the mean of the k nearest neighbors of $\mathbf{u}'_i$ in u and the mean of the subsequent points[3] in u, respectively.

The similarity between models u and u' is defined as

$$S(u,u') = \frac{1}{M'-1}\sum_{i=1}^{M'-1}\frac{(\bar{\mathbf{u}}_{i+1}-\bar{\mathbf{u}}_i)\cdot(\mathbf{u}'_{i+1}-\mathbf{u}'_i)}{\max(|\bar{\mathbf{u}}_{i+1}-\bar{\mathbf{u}}_i|,|\mathbf{u}'_{i+1}-\mathbf{u}'_i|)^2},$$

where $|\mathbf{x}|$ denotes the vector norm and $\cdot$ the vector dot-product. Each term in the sum computes the cosine between a vector formed by subsequent points in u' and the "average" of nearby vectors formed by subsequent points in u, multiplied by the ratio of the length of the smaller of the two vectors to the length of the larger of the two vectors. Therefore, $-1 \le S(u,u') \le 1$. Note that the similarity scores are not symmetric and unless $k = 1$ in general $S(u,u) < 1$ due to the effect of averaging over nearest neighbors. To convert this measure of similarity to a measure of distance, we define

$$d(u,u') = \exp(-S(u,u')). \tag{2}$$

We could also make this measure symmetric by letting

$$d(u,u') = \exp(-(S(u,u') + S(u',u))/2),$$

but we have found that this makes little difference in practice and is not worth the extra computational cost.

GeTeM has a number of attractive properties. Most importantly, by matching nearest neighbors in the reconstruction space, GeTeM allows for the comparison of unaligned segments. It does this in a manner that is quite different from DTW, which stretches the segments; GeTeM finds the closest match between the state and dynamics of an underlying model and compares those. Unless the data are simply out of phase, DTW will perform a nonlinear warping, which may perturb useful characteristics of the data. As evidence that this warping can distort important properties of the data, it has been shown that on many real datasets, constrained DTW, which limits the amount of distortion, outperforms standard DTW [16] (i.e., the optimal DTW alignment often distorts the data, so pairs of segments appear more similar than they really are). GeTeM, on the other hand, does not distort the data. Two segments are similar, according to GeTeM, if the trajectories they follow through a nonlinear dynamical system have a similar shape, regardless of the starting and ending points of the trajectories.

For similar reasons, GeTeM is well suited for comparing segments of different lengths. If two segments contain different numbers of repetitions of the pattern of interest, then DTW will perform poorly as it tries to globally align the segments. Since GeTeM only matches short subsegments, it is suitable for this type of situation. Techniques based on DFT and DWT also allow for comparison of unaligned segments of differing lengths, but both are parametric, making strong assumptions on the structure of the data.

Additionally, GeTeM is computationally efficient. For segments of lengths m and n, GeTeM requires $O(n\log m + m\log m)$ time. The $m\log m$ term is the time required to build a structure suitable for nearest neighbor queries (e.g., a kd-tree), and $n\log m$ is required for the nearest neighbor queries. If the lengths of the two sequences differ, we let m be the length of the shorter sequence (i.e., let the shorter sequence be o, and the longer o'). Many practical applications involve finding occurrences of a short segment within a long time series, in which case the time to build the nearest neighbor data structure on the short segment is negligible and the running time is dominated by the $n\log m$ term. Furthermore, if we wish to perform repeated comparisons, for instance when computing pairwise distances between points for clustering, we only need to build the nearest neighbor structures for each segment once, and can reuse them for future comparisons. Constrained DTW with a locality constraint of length r requires $O(rn)$ time, and, in practice, r depends linearly on m, making it effectively $O(nm)$.

In practice, approaches such as approximate nearest neighbor [38] can be used to speed up GeTeM. For repeated comparisons, the nearest neighbor data structures can be cached. It has also been shown (empirically) that for large datasets, the cost of DTW amortized over the dataset is closer to $O(n)$ [14]. For DTW, lower bounding techniques [39] can greatly improve performance. Hence, in practice, both techniques can be quite efficient, and, for large datasets, both algorithms are easily parallelizable.

4 Clustering, Classification, and Semi-Supervised Learning

In order to analyze the utility of the GeTeM-based distance measure for time series analysis (both segmented and long time series), we consider three different applications. First, we use GeTeM to cluster an ECG dataset from the PhysioNet ECG database [40], which has been analyzed previously by Kalpakis et al. [2]. Second, we use the distance measure with a 1-nearest-neighbor classifier (1-NN) to classify all of the available datasets from the UCR segmented time series database [13].[4] Finally, we introduce a dataset collected from wearable sensors while subjects performed different exercise routines, and demonstrate that given only sparsely labeled training data, clusters found using GeTeM perform well in a semi-supervised classification algorithm. We conclude this section with a discussion of the results and the efficiency of the algorithms that were considered.

4.1 Clustering ECG Data

To demonstrate the use of GeTeM for clustering periodic time series, we used a dataset composed of 40 2-minute ECG time series from the PhysioNet ECG database, which has previously been used as a clustering benchmark by Kalpakis et al. [2]. The authors used several distance measures and performed hierarchical clustering with the goal of separating the 18 traces that correspond to normal sinus rhythm from the 22 traces taken from patients with malignant ventricular arrhythmia. They considered distance measures based on differences in coefficients from DFT, DWT, PCA, and their own approach based on Linear Predictive Coding (LPC) cepstrum. Like GeTeM, the LPC cepstral distance involves fitting a dynamical model to the data, but unlike GeTeM, it uses linear modeling. The best

3. It is important to note that these points are not necessarily the nearest neighbors of $\mathbf{u}'_{i+1}$.

4. Data were retrieved on 30 September 2011.

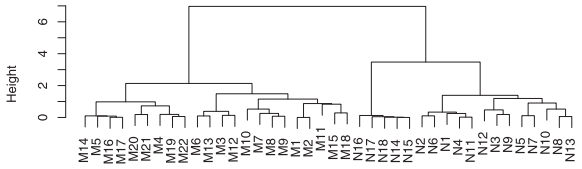


Fig. 3. Clustering dendrogram for ECG data using GeTeM. We clearly see two distinct clusters, one containing only malignant traces and one containing only normal traces.

reported performance is achieved using the DFT and autocorrelation coefficients, which grouped three malignant traces with the normal traces, and one normal trace with the malignant traces. The authors justified the mislabeling by examining the raw data and concluding that the malignant traces qualitatively "look more similar to the normal time series than to the malignant arrhythmia time series."

We constructed one time-delay model for each of the 40 traces using parameters $m = 7$ and $\tau = 11$. These parameters were chosen using the false nearest neighbors technique [6]. We computed the similarity matrix S using GeTeM and converted it to a distance matrix according to (2). In the reported results, we only consider the lower triangular portion of D, that is, the distance between points with indices i and j was $D_{i,j}$ if $i > j$, or $D_{j,i}$ otherwise. Symmetrization, as described in Section 3, of D leads to almost identical results, and therefore did not justify the extra computational cost. Finally, we performed hierarchical clustering [41] with Ward's minimum-variance criteria for cluster merging [42]. The resulting cluster dendrogram, shown in Fig. 3, contains two clearly separated clusters, one containing only malignant samples (M) and the other containing only normal samples (N). The time-delay embedding is able to tease out the underlying dynamical systems, and the distances produced by the GeTeM algorithm enable a standard hierarchical clustering algorithm to clearly separate the data into normal and malignant traces, despite their visual similarity. We also performed the same hierarchical clustering using DTW with euclidean distance, as described in Section 2.1.3. This approach found a cluster with nine of the 22 malignant traces, and placed all of the remaining traces in the other cluster.

Note that the approaches considered by Kalpakis et al. [2] required a nontrivial amount of preprocessing: removing trends, normalization, etc. Our approach works with the raw data and the clustering takes roughly 1 minute of computation on a desktop computer. To demonstrate the robustness of GeTeM to the time delay embedding parameters m and τ, we performed the clustering for a number of different settings. For all of the 20 possible combinations of $m \in \{4, \ldots, 8\}$ and $\tau \in \{3, \ldots, 6\}$, a perfect clustering was found. Expanding the range of parameters to the 88 possible combinations of $m \in \{3, \ldots, 10\}$ and $\tau \in \{3, \ldots, 13\}$, 39 of the parameter settings resulted in a perfect clustering (with the poorest clusterings occurring toward the extreme ranges of parameter settings, namely for small τ and large m), and 75 of the combinations found clusterings that were at least as good (in terms of accuracy) as the best clustering found by any of the methods in Kalpakis et al. [2].

4.2 Nearest-Neighbor Classification

Next, we assess the performance of the GeTeM-based distance in a nearest-neighbor (1-NN) classifier for segmented data. We use the UCR time series database [13], which contains 43 sequential datasets ranging in length from 24 to 1,882 samples, with most falling in the 60 to 600 range. The UCR database contains classification accuracies for each dataset using 1-NN with euclidean distance, 1-NN with DTW, and 1-NN with cDTW with the locality constraint tuned by cross validation [13]. All of these results are computed on a single training and testing split provided in the database. To facilitate comparison with the published results, we also present our results on these same splits.

We perform 1-NN classification using the GeTeM distance measure. We use leave-one-out cross validation on the training set to find optimal time-delay parameters m and τ, and the number of neighbors k for computing the similarity measure. Fig. 4 depicts the performance of GeTeM compared to euclidean distance, DTW distance, and constrained DTW with the optimal locality constraint tuned on the training set, as described in Section 2.1.3. The full results are in Table 1. On 19 of the 43 datasets, GeTeM has higher accuracy than any of the other approaches. The 1-NN with euclidean distance approach performs best on three of the datasets, the tuned constrained DTW approach performs best on 15 of the datasets, and the standard DTW approach performs best on 12 of the datasets. Since results are computed on a single split of the data and in many cases the error rates are close, these values should not be

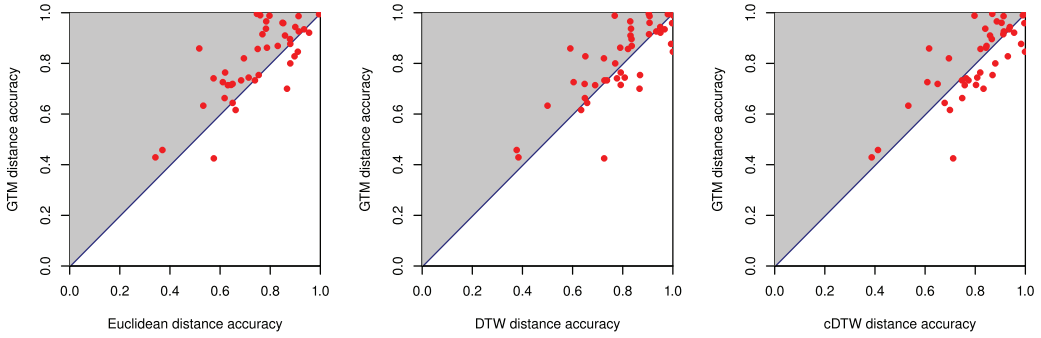


Fig. 4. Performance of GeTeM distance for 1-NN, compared to the euclidean distance, DTW distance, and constrained DTW with tuned locality constraint, from left to right, respectively, on the 43 datasets in the UCR time series database [13]. The shaded area represents the region in which GeTeM is outperforming the other algorithm.

TABLE 1
Results on the UCR Database [13]

Name	Number of Classes	Size of Training Set	Size of Testing Set	Time Series Length	1-NN Euclidean Dist.	1-NN Best Warping Window DTW (r)	1-NN DTW, No Warping Window	1-NN GeTeM Dist. (m,τ,k)
50Words	50	450	455	270	0.369	**0.242 (6)**	0.310	0.286 (16,7,3)
Adiac	37	390	391	176	0.389	0.391 (3)	0.396	**0.274 (6,1,2)**
Beef	5	30	30	470	0.467	0.467 (0)	0.5	**0.367 (10,16,3)**
CBF	3	30	900	128	0.148	0.004 (11)	**0.003**	0.0411 (3,13,1)
Chlorine Conc.	3	467	3840	166	0.35	0.35 (0)	0.352	**0.281 (12,13,1)**
CinC ECG torso	4	40	1380	1639	0.103	**0.07 (1)**	0.349	0.172 (16,7,3)
Coffee	2	28	28	286	0.25	0.179 (3)	0.179	**0.143 (12,16,1)**
Cricket X	12	390	390	300	0.426	0.236 (7)	**0.223**	0.259 (15,1,1)
Cricket Y	12	390	390	300	0.356	**0.197 (17)**	0.208	0.285 (15,1,1)
Cricket Z	12	390	390	300	0.38	**0.18 (7)**	0.208	0.236 (15,1,1)
Diatom Sz. Red.	4	16	306	345	0.065	0.065 (0)	**0.033**	0.0654 (4,13,3)
ECG 200	2	100	100	96	**0.12**	**0.12 (0)**	0.23	0.2 (4,16,3)
ECG Five Days	2	23	861	136	0.203	0.203 (0)	0.232	**0.0116 (7,13,3)**
Face (all)	14	560	1690	131	0.286	**0.192 (3)**	**0.192**	0.256 (9,4,1)
Face (four)	4	24	88	350	0.216	0.114 (2)	0.170	**0.0341 (7,10,3)**
FacesUCR	14	200	2050	131	0.231	0.088 (12)	0.0951	**0.0849 (7,7,2)**
Fish	7	175	175	463	0.217	0.16 (4)	0.167	**0.063 (7,4,3)**
Gun-Point	2	50	150	150	0.087	0.087 (0)	0.093	**0.0133 (4,10,3)**
Haptics	5	155	308	1092	0.63	0.588 (2)	0.623	**0.542 (16,16,5)**
Inline Skate	7	100	550	1882	0.658	0.613 (14)	0.616	**0.571 (15,1,1)**
Italy Pwr. Dem.	2	67	1029	24	**0.045**	**0.045 (0)**	0.0496	0.0787 (6,1,1)
Lightning-2	2	60	61	637	0.246	**0.131 (6)**	**0.131**	0.246 (3,1,1)
Lightning-7	7	70	73	319	0.425	0.288 (5)	**0.274**	0.575 (15,1,1)
MALLAT	8	55	2345	1024	0.086	0.086 (0)	**0.066**	0.0738 (15,13,1)
Medical Images	10	381	760	99	0.316	**0.253 (20)**	0.263	0.267 (15,1,1)
MoteStrain	2	20	1252	84	0.121	0.134 (1)	0.165	**0.104 (3,1,1)**
OliveOil	4	30	30	570	0.133	0.167 (1)	**0.133**	0.3 (7,13,3)
OSU Leaf	6	200	242	427	0.483	0.384 (7)	0.409	**0.141 (10,7,4)**
Sony AIBO Surf	2	20	601	70	0.305	0.305 (0)	0.275	**0.18 (6,1,1)**
Sony AIBO Surf2	2	27	953	65	0.141	0.141 (0)	0.169	**0.09 (6,4,1)**
Star Light Crvs.	3	1000	8236	1024	0.151	0.095 (16)	0.093	**0.0397 (12,10,1)**
Swedish Leaf	15	500	625	128	0.213	0.157 (2)	0.210	**0.138 (9,1,1)**
Symbols	6	25	995	398	0.1	0.062 (8)	**0.0503**	0.0563 (4,10,3)
Synthetic Control	6	300	300	60	0.12	0.017 (6)	**0.007**	0.123 (3,13,1)
Trace	4	100	100	275	0.24	0.01 (3)	**0**	0.01 (7,13,1)
Two Patterns	4	1000	4000	128	0.09	0.0015 (4)	**0**	0.154 (6,13,1)
TwoLeadECG	2	23	1139	82	0.253	0.132 (5)	0.096	**0.00439 (3,13,1)**
uWave X	8	896	3582	315	0.261	**0.227 (4)**	0.273	0.267 (15,16,4)
uWave Y	8	896	3582	315	0.338	**0.301 (4)**	0.366	0.384 (15,16,3)
uWave Z	8	896	3582	315	0.35	**0.322 (6)**	0.342	0.356 (12,10,2)
Wafer	2	1000	6174	152	**0.005**	**0.005 (1)**	0.020	0.00584 (9,7,1)
Word Synonyms	25	267	638	270	0.382	**0.252 (8)**	0.351	0.337 (9,13,3)
Yoga	2	300	3000	426	0.170	0.155 (2)	0.164	**0.131 (16,4,5)**

Shaded cells indicate lowest error rate.

interpreted as saying that any one algorithm is significantly better than the others. Comparisons of this nature require that the results be reliable estimates of the algorithms' performance on each dataset, and values computed on a single training and testing split, as given in the UCR database, do not meet this criteria.

To address this shortcoming, we also evaluated 1-NN using GeTeM, euclidean, and DTW distances on each dataset using 10-fold cross validation. By performing a more thorough evaluation of the performance of each of these algorithms, we are able to make a stronger statement about the performance of the algorithms under the reasonable assumption that the datasets are independent. For all results, we use significance level $\alpha = 0.05$. We use Friedman's method [43], which found a statistically significant difference between the performance of the three algorithms, with GeTeM being ranked the best performing, DTW second best, and euclidean the worst (p-value computed by Iman and Daveport test was 0.0028). With Bergmann-Hommel's posthoc procedure [44] we found that the differences in accuracies for GeTeM versus DTW and GeTeM versus euclidean were both statistically significant (adjusted p-values were 0.029 and 0.0022, respectively). The difference in accuracies for DTW versus euclidean was not, however, found to be statistically significant (adjusted p-value was 0.23).

Note that DTW is more suited (by design) for these datasets, given that the data are segmented, the segments are short, and the problem is to find a single matching pattern or motif in pairs of segments. The data are not periodic, as GeTeM (in principle) requires, but GeTeM still outperforms DTW.

4.3 Semi-Supervised Learning

Semi-supervised learning considers the case where the training set contains both labeled and unlabeled examples. The algorithm we present is intentionally simple so that the performance explicitly depends on the quality of the distances used for clustering. We note that this does not reflect the state of the art in semi-supervised algorithms (see [12] for a thorough survey), and it is likely that a more powerful algorithm would achieve better results.

First, we perform hierarchical clustering on the entire training set, as in Section 4.1, to produce a dendrogram, then

we traverse it in a depth-first manner. Each node represents a set of points, some of which may be labeled. If all of the labels on points in the node agree, we assign that label to all of the points in the node and cut the tree at that point. If there are no labels, then we also cut the tree at that node, but leave the points unlabeled. If there are multiple unique labels in the set of nodes, we move on and consider the children of that node. Once finished, we will have a larger set of labeled nodes, but some unlabeled nodes may remain.

In the second pass, we assign labels to the remaining unlabeled nodes. To do this, we compute the distance from each unlabeled node to its nearest labeled neighbor. The unlabeled node that is closest to a labeled node is assigned the label of its nearest labeled neighbor. If there are unlabeled nodes remaining, we repeat the process until all nodes have been assigned a labeled neighbor. In the experiments, we found that often there are no unlabeled nodes, so this second pass can be skipped. In some cases, however, as many as 30 percent of the nodes remained unlabeled after the first pass.

We collected data from four individuals, three males and a female, who were asked to perform a sequence of four activities at the gym for approximately 2 minutes each. The activities were: using an elliptical machine, riding a stationary bike, using a stationary rowing machine, and using a stair-climbing device. The three male subjects also ran on a treadmill, and one of the male subjects only completed the treadmill and biking activities. Data were collected by custom software running on an HTC G1© mobile phone clipped to the waistband of the subject's clothing. We collected data from the triaxial acceleration sensor in the G1 at the maximum frequency allowed by the phone's operating system, 35 Hz. From the three input channels, x, y, and z, we formed a univariate time series by taking the euclidean vector norm.

The data were manually segmented into the various activities, then split into 5-second windows in which the first and last second of each segment overlapped with the previous or next segment, respectively. The first and last 5 seconds of each segment were discarded, and a time-delay model was built for each window. This resulted in 27-46 models per activity, per subject. There were 689 models in total, corresponding to 689 5-second segments. Each segment was labeled according to the subject and activity that took place during the majority of the time period covered by the segment.

We removed all but n labels from each of the 15 subject-activity pairs, with n from 1 to 25, and used the semi-supervised technique with the GeTeM-based and DTW-based distances. Since we had the correct labels, we were able to assess the accuracy of the labeling, for both the activity and the subject. For each value of n, we ran the experiment 100 times, picking random examples to be unlabeled each time. The results are shown in Fig. 5, with error bars representing 95 percent confidence intervals around the mean. GeTeM consistently outperforms DTW for all values of n, and is able to correctly classify the subject and activity with over 90 percent accuracy using only two labels per subject-activity pair. The small size of the confidence intervals is not surprising given the repetitive nature of these activities.

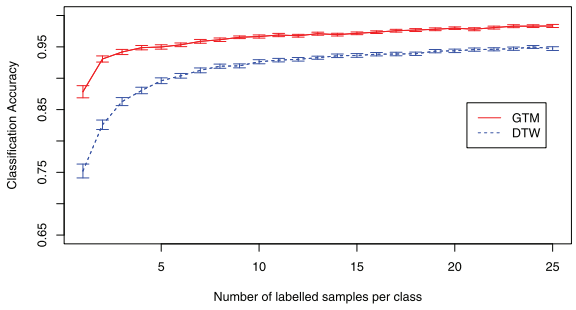


Fig. 5. Result of applying the semi-supervised technique to the activity dataset. The x-axis represents the number of labeled examples for each class and the accuracy is computed over data with missing labels. Error bars represent 95 percent confidence intervals.

4.4 Discussion

The results in this section demonstrate the versatility of the GeTeM distance measure. While originally intended for extracting features from long, unsegmented sequences of time series, GeTeM also provides good distance measures for clustering and nearest-neighbor classification of short segmented time series. When data can be carefully segmented so that the patterns sought are temporally aligned between pairs of segments, then treating n-length sequences as points in $\mathbb{R}^n$ and using the euclidean distance between segments to train a 1-NN classifier is hard to beat [45]. When the patterns in the sequences are out of phase, running an alignment technique such as DTW [46] can compensate and 1-NN with euclidean distance between aligned elements also performs well.

As described in Section 3, GeTeM allows for comparisons between unaligned segments without distorting their temporal properties, while DTW performs a warping in the temporal domain. This is likely the cause of the poor performance of DTW on the ECG data in Section 4.1. In the semi-supervised experiment in Section 4.3, the data are segmented into windows of fixed time lengths. Therefore, segments contain unaligned data and may also contain different numbers of repetitions of the patterns of interest. It is most likely due to these properties that GeTeM greatly outperforms DTW on this data. Based on these results, GeTeM appears preferable for real data, which is usually not well segmented or aligned.

In terms of computational cost, GeTeM required less time than DTW. For the ECG clustering example in Section 4.1, the 40×40 GeTeM distance matrix took approximately 190 seconds of CPU time to compute, while the DTW distance matrix took approximately 426 seconds. For the semi-supervised experiment in Section 4.3, the 689×689 GeTeM distance matrix took approximately 36 minutes to compute, while DTW took approximately 70 minutes. We note that neither algorithm was optimized and that there are techniques (see Section 3) for speeding up both GeTeM and DTW.

5 Boosted Time-Delay Classifiers

In this section, we focus on analyzing long, unsegmented time series data, which arise from many real-world problems, such as activity recognition from wearable sensor

data [31], [47], heart rate and breathing monitoring, seizure detection and financial prediction. As described in Section 2.1.4, the goal in this case is to identify the segments that correspond to a particular class. We use GeTeM in this context in order to extract useful features of the data.

More precisely, we work in the framework of radial-basis functions, in which one picks a set of points in the space of the input values and then represents new input data using the set of distances to each of these points, which become input features. We propose to compute features for new time series by computing the GeTeM distances between the data and a set of selected segments from a training set. The challenge is how to pick the segments from the training data that will provide the most informative features. Our algorithm selects segments using a boosting-based approach. We evaluate performance on a new, challenging benchmark problem, comparing against two strong baseline approaches.

5.1 TDEBOOST

We propose an algorithm called TDEBOOST for classifying time series. TDEBOOST is based on the SAMME.R algorithm [48], a boosting algorithm for multiclass classification. Boosting is a powerful ensemble method for classification, exemplified by the AdaBoost algorithm [49] for binary classification problems. AdaBoost associates a weight with each example in a training set of labeled data $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$. The weights are initially all equal. At iteration l, a base classifier h_l is trained on the weighted dataset and then used to classify the entire training set. The weights of misclassified examples are increased (relative to the other examples). The weights are then renormalized to sum to one, and the next iteration begins. Once a sufficient number of iterations M has been performed, a classifier is formed as a weighted majority vote of the outputs of the base classifiers trained on each iteration, $h(x) = \sum_{l=1}^M \alpha_l h_l(x)$. The weight α_l for a base classifier h_l is computed based on its accuracy on the weighted dataset on which it was trained.

For multiclass problems (i.e., $y_i \in \{1, \ldots, K\}$), a common approach is to break the problem into multiple two-class problems, training multiple binary classifiers and aggregating their results. AdaBoost requires that the base classifiers have an error rate at most $1/2$ at every iteration, in order to work correctly. For the K-class problem, this requirement is often hard to satisfy. SAMME uses a correction term in the weight calculation such that the accuracy of the base classifier at each round need only be at least $1/K$. A recent theoretical result [50] shows that this approach is actually too permissive, and proposes a better theoretical error bound. However, this new method has not yet been thoroughly tested in practice (though initial results are promising). We base the experiments in this paper on a variant of SAMME called SAMME.R that builds an ensemble of base classifiers, each of which output a probability distribution (or set of confidence scores) over the class labels, rather than selecting a single class label.

The most straightforward approach to boosting in our setting would be to build classifiers directly out of the time-delay embedding models by taking a collection of models (one for each class) and defining the classifier for a new snippet of data by picking the class with the most similar model. However, this approach can be problematic for time series (especially for the practical applications we envision). If the models picked initially are noisy, the base classifier is bound to have very high error, and even if we continue picking new data segments and reweighting them, the classifiers would continue to be poor. Thus, there is no way to recover from a bad initial model.

Input: A training set $\langle(o_i, y_i)\rangle_{i=1}^N$, a specified number of rounds L, model and window-length parameters W and s, time-delay embedding parameters m and τ, and a hypothesis class $\mathcal{H}$.
Let $N' = N - s + 1$, set weights, $w_i = 1/N', i = 1, \ldots, N'$.
for $l = 1, \ldots, L$ **do**
  If $l = 1$, build a set of M time-delay models $\mathcal{M}_1 = \{u_1, \ldots, u_M\}$, otherwise update the set of models $\mathcal{M}_{l-1}$ to produce $\mathcal{M}_l$.
  for $t = 1, \ldots, N'$ **do**
    Build model v_t for subsequence $\langle o_t, \ldots, o_{t+s-1}\rangle$.
    Compute features $\mathbf{x}_t = (S(u_1, v_t), \ldots, S(u_M, v_t))$.
  end for
  Fit a classifier $h_l \in \mathcal{H}$ to $\langle(\mathbf{x}_t, y_t)\rangle_{t=1}^{N'}$ using weights w_i.
  For each class label $k = 1, \ldots, K$, obtain the weighted hypothesis

$$g_l^k(\mathbf{x}) \leftarrow (K-1)\left(\log h_l^k(\mathbf{x}) - \frac{1}{K}\sum_{k'=1}^{K} \log h_l^{k'}(\mathbf{x})\right).$$

  For $i = 1, \ldots, N'$, update the weights

$$w_i \leftarrow w_i \exp\left(-\frac{K-1}{K}\sum_{k=1}^{K}\left(-\frac{1}{K-1}\right)^{\mathbb{I}(y_i \neq k)} h_l^k(\mathbf{x_i})\right).$$

  Renormalize the weights w_i.
end for
Return: Models $\mathcal{M}_1, \ldots, \mathcal{M}_L$ and classifier

$$h(\mathbf{x}) = \operatorname*{argmax}_k \sum_{m=1}^{M} g_m^k(\mathbf{x}).$$

Fig. 6. TDEBOOST algorithm pseudocode.

To prevent the problem of bad initial models, we modify the boosting framework to allow the set of features to change over time. More precisely, at each boosting round we use GeTeM with an ensemble of time-delay models to extract from the time series the features that are used by the base classifiers. We allow the set of models to change between boosting rounds, which means that the set of features used by the base learners also changes. The algorithm is presented in Fig. 6.

The algorithm begins by generating a set of M models, $\mathcal{M} = \{u_1, \ldots, u_M\}$. Each model u_i is generated by first selecting an index j at random or according to some application-specific criteria and constructing a time-delay model for the subsequence $\langle o_j, \ldots, o_{j+W-1}\rangle$, where W is a model-size parameter. The set of models $\mathcal{M}$ is used to extract an M-dimensional feature vector $\mathbf{x}_t$: A time-delay model v is constructed from $\langle o_t, \ldots, o_{t+s-1}\rangle$, where s is the segment-length parameter, and for $i = 1, \ldots, M$, the ith element $\mathbf{x}_t$ is set to $S(u_i, v)$. In other words, the set of similarity scores between each of the models in $\mathcal{M}$ and a new subsequence become the feature vector. The feature vector and class label pairs $(\mathbf{x}_t, y_t)$ are then used to train a classifier, exactly as in the SAMME.R algorithm. Note that the feature vector $\mathbf{x}_t$ is built from a window of observations starting at o_t; one could also consider using the mode of the class labels within this window, rather than the label of the first observation, y_t, as the target for training the classifier.

We use the notation $h_m^k(\mathbf{x})$ to denote the probability assigned to class k by the hypothesis built at round m when evaluated on input feature $\mathbf{x}$. Classifiers that output only a

class label can be used by modifying the SAMME algorithm in the same way that we modified SAMME.R. We also experimented with the SAMME variant, but found that modifying SAMME.R gives better performance on our data.

At each iteration, the set of models $\mathcal{M}$ is updated. This step can be application specific. The easiest approach is to generate a new set of models at each step, either by randomly sampling new segments or by a more informed method. Or, specific models can be replaced by new models. In our gait recognition application, discussed in more detailed in the following section, the set of models $\mathcal{M}$ contains one model for each class, and at each iteration one model is selected and replaced by a new model for the same class. The choice of model is informed by the current boosting weights: We select a segment of the data on which the boosting weights (i.e., the cumulative training error) is high. Based on the assumption that short subsequences of training points are likely to all have the same label, the intuition here is that by building models for regions of the time series in which performance is poor, the features will focus the ensemble hypothesis on these "harder" regions, and improve overall classification.

The algorithm differs from classical boosting because of this feature selection step that precedes the construction of the base classifiers. However, the theory of boosting still applies to our setting. To see this, consider base classifiers in which *all* possible models are included as features in a linear classifier. In this case, minimizing the error on the new distribution is equivalent to increasing the parameter (in the linear classifier) of the nearest examples to those with high weights in the boosted distribution. However, from a computational point of view, we cannot afford to include a large set of models in the classifier. What we do can be viewed as setting to 0 the weight of the existing model and greedily picking the area that most reduces training error. This is an approximation to the case when all models are included, but is still guaranteed to reduce error on the boosting distribution. Since this is the main requirement of boosting theory, existing results still apply.

5.2 Walking Dataset

The problem of identifying an individual using data collected from a wearable accelerometer sensor, called gait recognition, is a challenging and well-studied time series classification task. However, empirical evaluations consider data collected in very controlled environments. For example, Ailisto et al. [51] and Gafurov et al. [52] collected datasets composed of subjects walking a distance of approximately 20 m, in a straight line, on a level surface, with a sensor carefully attached to a specific body part. Bächlin et al. [53] collected data from subjects walking on a treadmill, so the speed could be controlled. Due to the personal nature of the data, it is typically not shared between groups, and while each of the proposed methods performs well on the data collected by the corresponding authors, thorough comparisons on the same dataset are rarely performed. Additionally, since these algorithms have potential for commercialization, authors are hesitant to share code, making empirical comparisons difficult.

We envision gait recognition as a useful practical technology. Mobile devices are ubiquitous, and frequently contain accelerometers. Gait recognition algorithms provide opportunities to improve both the security and the user experience of these devices. For example, if more than one person shares a mobile phone, the phone can be aware of its current user and personalize the contact list or e-mail accounts that are available to the user. As a security feature, mobile phones could detect if someone other than the owner is carrying the device and prevent access to private information. However, the conditions in which these algorithms have to operate are much noisier than existing benchmarking datasets.

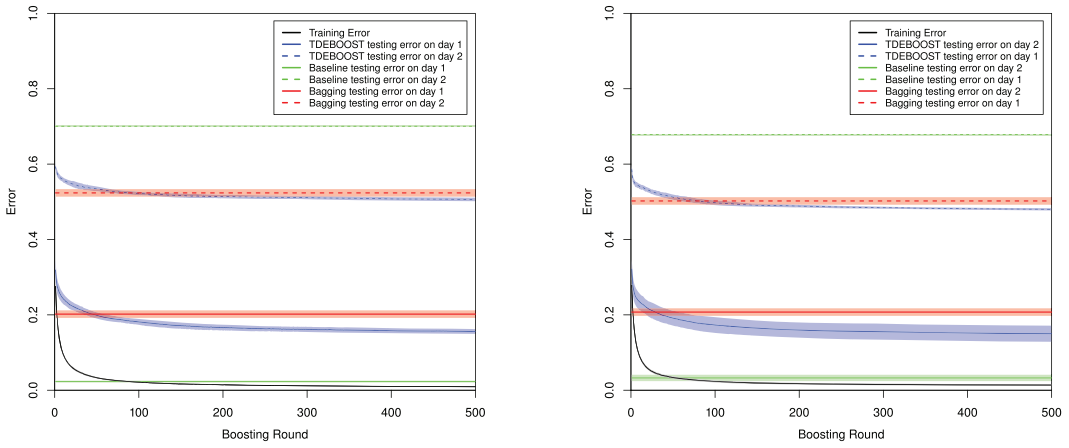
We present a dataset that was collected in real circumstances, analogous to the environment in which a practical gait recognition algorithm would operate. We are making the dataset publicly available,[5] and hope that it will provide a challenging benchmark for gait recognition as well as machine learning research. The data were collected from 20 subjects, 10 males and 10 females, ranging in age from 19 to 35, ranging in weights from 47 to 100 kg, and ranging in heights from 160 to 193 cm. Each subject was asked to carry an HTC Nexus One© mobile phone in their pocket and walk around outdoors for 15 minutes, then return on a subsequent day to perform another 15 minute walk. The subjects were allowed to change clothing and footwear between the two days at their discretion, and the phone was casually placed in the pocket, rather than being affixed at a specific position. Subjects were also asked to walk both on grass and concrete surfaces, up and down hills, and to pause occasionally. The device also logged GPS data to verify that the subjects followed instructions. Specific details, as well as descriptions of the footwear and clothing worn by the subjects, are included in the dataset.

The HTC Nexus One phone runs the Google Android operating system; we wrote a custom application that collects data from the sensors as quickly as possible. The sampling frequency is variable and depends on factors such as the processor load. The data are time stamped with millisecond precision. In the data we collected, the average frequency is approximately 28.57 Hz, with a standard deviation of approximately 4.17 Hz. The frequency is relatively low compared to the data analyzed in previous work; this is representative of what one would expect from a commercial mobile device.

5.3 Results

We use frame-based performance metrics [9] as the evaluation criterion. We resample the data to a fixed frequency of 25 Hz using linear interpolation, and compute a single magnitude measurement from the three acceleration directions, as in Section 4.3. The data contain segments of walking, as well as segments where the subjects were stationary. We associate a common label, *lingering*, to all portions of the data, for all subjects, that contain *nonwalking* data. We use a crude but effective method for labeling lingering data. We consider a window of 50 samples; if the sum of the absolute values of the acceleration magnitudes in this window was less than $125\ \mathrm{m/s^2}$, then the first sample in the window was labeled as lingering. These parameters were determined by trial-and-error, visually inspecting the resulting labels. Lingering data were assigned the class

5. Data available at http://www.cs.mcgill.ca/~jfrank8/gait.html.



(a) Results when classifiers are trained on day 1. (b) Results when classifiers are trained on day 2.

Fig. 7. Learning curves for the TDEBOOST, baseline, and bagging algorithms. Shaded regions represent 95 percent confidence intervals around the mean.

label 0, and the subjects were identified with labels 1 to 20. Every sample is assigned a label. After filtering, there are between 18,000 and 50,000 samples per walker, and roughly 87,000 samples for the lingering class.

Since every sample has an associated label and there is no significant class skew, we evaluate performance by computing the accuracy, or percentage of samples correctly labeled. When considering a single subject, we report both precision (i.e., the fraction of examples that our algorithm labels as that subject that are correct) and recall (i.e., the fraction of examples that should have been labeled as a certain subject that were correctly labeled).

We compare TDEBOOST with a baseline approach which extracts 159 features used in the activity recognition system of Lester et al. [31], comprised of Fourier coefficients, cepstral coefficients, and band-pass filter responses over windows ranging from 2 seconds to a minute. Features are associated with the first sample in the window. The features are used to train an AdaBoost.M1 classifier [49] with random forests composed of 100 trees, each trained on 10 randomly selected features as the base classifier. One hundred iterations of boosting were performed. We also compared against selecting models at random from the training data and using them as feature extractors to train a classifier, which we call the bagging approach. We were unable to compare against other gait recognition approaches as they involve segmenting the data into individual footsteps, and these techniques do not scale to large, noisy datasets.

For TDEBOOST, we maintain a set of 20 models, one for each subject. At each round of boosting, the set of models is updated by sampling a training example according to the boosting weights. If the sample corresponds to the lingering class, then we resample and repeat until the sample corresponds to a subject. Let i be the label associated with the sampled subject; a new model u is built from the window of data beginning with the selected sample, and it replaces the model corresponding to subject i in the current set. As in the baseline approach, we use a random forest with 100 trees, each built on 10 randomly selected features, as the base classifier. We collected an additional trace of two subjects that did not participate in the large dataset, walking for approximately one minute each; we used this small amount of data to select parameters. This additional data were split into two halves. The first half was used to tune the values of the time-delay embedding parameters, using initial guesses of $W = 50$ (2 seconds) and $s = 32$. Then, fixing the time-delay embedding parameters, the second half was used to tune the parameters W and s. The values selected by this procedure were $m = 7$, $\tau = 2$, $W = 125$ (5 seconds), and $s = 16$. The number of neighbors was chosen to be $k = 2$, based on the fact that the models were small, and thus contained few gait cycles.

We ran TDEBOOST for 500 iterations. Although the training error converged after approximately 100 iterations, we wanted to assess whether the algorithm would overfit the data. We considered two scenarios: training on a segment of the data from one day and testing on the remaining data from that same day, and training and testing on different days. When training and testing on the same day, we performed 10-fold cross validation, where each fold consisted of a contiguous segment of the data. When training and testing on different days, we used the 10 classifiers that were built in the previous scenario (one per round of cross validation) and evaluated them on all of the data from the day on which they were not trained. Since the baseline algorithm operates with a larger number of features, we subsample 10 percent of the training data when training the base classifier. To ensure that the results are not an artifact of the data sampled, we repeat this 10 times for each fold and average the results.

For the bagging algorithm, we use 100 randomly selected training segments (five per subject) for models and a random forest classifier with 500 trees. This was repeated 10 times with 10 different sets of models for each of the 10 rounds of cross validation, and we report the average accuracy (i.e., in total 100 classifiers were built and tested).

Fig. 7a shows the results when the classifiers are trained on data collected on the first day, and Fig. 7b shows the results when the classifiers are trained on data from the second day. The solid lines depict the testing error when trained on data from the same day and the dotted lines depict the testing error when trained on the other day. The

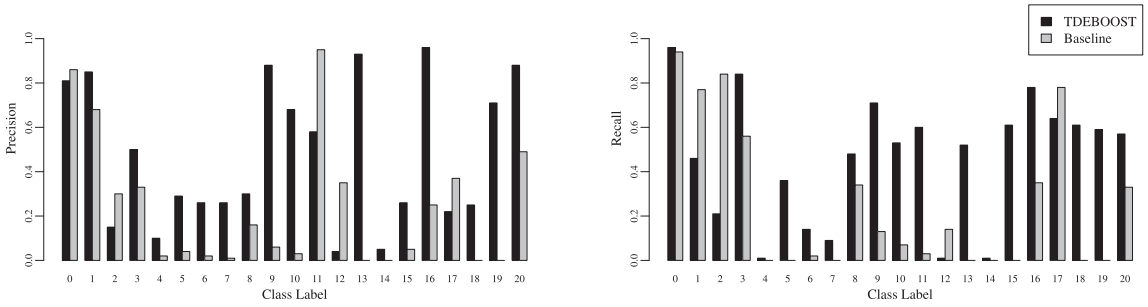


Fig. 8. Precision (left) and recall (right) when training on day 1 and testing on day 2. Class 0 is lingering; the other classes correspond to the subjects.

shaded regions represent 95 percent confidence interval around the mean. All three methods perform significantly worse when the training and testing sets are from different days, confirming the findings of Bächlin et al. [53]. If we do not consider the lingering data, the accuracy of TDEBOOST after 500 rounds is 0.42, while the accuracy of the baseline method is 0.20. Chance is 0.05, disregarding the lingering class.

When the classifiers were trained and tested on data from the first day, the average performance of bagging 100 models was roughly equivalent to the performance that TDEBOOST with 45 models and roughly equivalent to TDEBOOST with only 29 models for the second day. When training and testing on different days, the improvement over bagging was not as drastic, but still significant. When training on day 1 and testing on day 2, bagging with 100 models is roughly equivalent to TDEBOOST with 90 models, and when training on day 2 and testing on day 1, bagging is roughly equivalent to TDEBOOST with 78 models. In both cases, when training and testing on data from the same day, the baseline outperforms both the boosting and bagging approaches, and when training and testing on different days, both boosting and bagging classifiers built with the GeTeM-based features outperform the baseline.

The training error for TDEBOOST drops quickly and then continues to slowly decrease. However, the testing error also continues to decrease, which highlights an attractive property of boosting: It is typically not prone to overfitting. When trained and tested on the same day, the baseline algorithm outperforms TDEBOOST after 100 rounds by approximately 0.15. On the other hand, when the training and testing data come from different days, TDEBOOST outperforms the baseline by approximately 0.18. After 500 rounds, TDEBOOST is able to build a classifier from the data collected on one day and label over half of the data from the other day correctly.

We gain more insight into the performance differences by looking at the average precision and recall values for particular classes. For example, Fig. 8 shows these values when the training data are from day 1, and the testing data are from day 2. Assuming the task is to identify the user of the device, then, loosely speaking, precision gives a measure of how often the person that is identified as the user is in fact the real user, and recall gives a measure of how often the real user is correctly identified when they carry the phone.

There are 11 subjects on which the precision for the baseline is less than 0.1, but only three for which TDEBOOST has precision less than 0.1. TDEBOOST achieves higher scores than the baseline for both precision and recall on 15 of the 20 subjects, while the baseline achieves higher scores for both precision and recall on only three of the subjects. Subjects 4 and 14 were poorly classified by both algorithms; these are the two subjects for which the change in clothing between day 1 and day 2 was most drastic; subject 4 changed from a loose flowing dress to tight shorts, and subject 14 changed from jeans to baggy shorts. Collecting data over many days could substantially improve performance.

We performed an additional postprocessing step in order to smooth the labels. Our approach is simple and based on the assumption that walking segments are separated by segments of lingering. Therefore, whenever we encounter a segment of data that have been labeled as something other than lingering, we begin to accumulate counts for each label and reassign to the sample the label with the majority vote. We continue until we encounter a sample that is labeled as lingering, in which case we reset the counters. With this smoothing, if the classifier is trained on data from day 1 and tested on data from day 2, the average accuracy for TDEBOOST increases from approximately 49.4 to 63.0 percent, while the accuracy for the baseline decreases from approximately 29.5 percent to approximately 27.4 percent.

Finally, we compared against a nearest neighbor classifier. We built a classifier with 2,000 randomly selected windows of 70 samples. The number 2,000 was chosen so that the nearest neighbor classifier has roughly 50 times as many training points as the 120 time-delay models used for 100 rounds of TDEBOOST. Each window was associated with the class label of the majority of the samples in the window. The experiment was repeated 100 times and the accuracy for training on day 1 and testing on day 2 was approximately 42.4 percent and training on day 2 and testing on day 1 was approximately 41.5 percent. These values are roughly equivalent to the accuracy of TDEBOOST after the first round, with 20 models (each one roughly the size of four 70-sample windows) selected at random. It is interesting, however, that these accuracies are higher than the baseline accuracies of 32.2 and 29.9 percent in these two scenarios.

6 Conclusion and Future Work

In this paper, we presented the GeTeM algorithm and evaluated its use as a distance measures for time series in unsupervised, semi-supervised, and supervised learning. GeTeM compared very favorably to other commonly used time series distance measures, such as those based on DTW, DFT, and DWT. On the ECG dataset in Section 4.1, the

GeTeM-based distance is the only method able to correctly cluster the data. On the large benchmark dataset from the UCR time series repository [13], the improvements in accuracy of GeTeM over DTW and GeTeM over euclidean distance, when used in a nearest neighbor classifier, are both statistically significant ($\alpha = 0.05$). We also presented a boosting algorithm, TDEBOOST, that uses GeTeM for computing features from long time series. For larger datasets that are not segmented a priori, TDEBOOST is able to locate segments on which to build models for extracting GeTeM features. TDEBOOST outperforms two strong baseline algorithms on the difficult problem of gait recognition when trained and tested with clothing and footwear variations.

GeTeM is not meant to be a replacement for DTW or other approaches for comparing time series. It makes different assumptions about the process that generated the data and is therefore more suited for problems where the data correspond to measurements of a latent, periodic, nonlinear dynamical system. However, GeTeM is nonetheless robust to violations of these assumptions. GeTeM is also computationally efficient (for example, it takes less computation time than DTW) and is thus suitable for real-time analysis of streaming data. GeTeM should be considered a powerful addition to the suite of tools that a time series analyst has at their disposal.

As future work, we intend to develop algorithms that further exploit the cluster structure that GeTeM finds in unlabeled data. For example, the semi-supervised algorithm in Section 4.3 was intentionally simple in order to highlight the quality of the clusters. More powerful semi-supervised learning algorithms exist [12], and employing GeTeM in these frameworks would likely lead to better performance. We are also studying the use of GeTeM in an active learning framework for choosing when to solicit labels from an oracle. Finally, the models that GeTeM builds can be treated as predictive models for time series. In fact, the GeTeM score is essentially a prediction residual, computed by comparing the expected next state in the model to the observed next state. We intend to further investigate the use of GeTeM models for prediction and forecasting.

ACKNOWLEDGMENTS

This work was supported in part by NSERC, FQRNT, the Israel Science Foundation under contract 890015, and the European Union under the Harvest Program under the PASCAL2 Network of Excellence. The authors thank the anonymous reviewers for their insightful feedback.

REFERENCES

[1] G. Batista, X. Wang, and E. Keogh, "A Complexity-Invariant Distance Measure for Time Series," *Proc. 11th SIAM Int'l Conf. Data Mining,* pp. 699-710, 2011.

[2] K. Kalpakis, D. Gada, and V. Puttagunta, "Distance Measures for Effective Clustering of ARIMA Time-Series," *Proc. IEEE Int'l Conf. Data Mining,* pp. 273-280, 2001.

[3] D. Ge, N. Srinivasan, and S. Krishnan, "Cardiac Arrhythmia Classification Using Autoregressive Modeling," *BioMedical Eng. OnLine,* vol. 1, no. 1, p. 5, 2002.

[4] M. Corduas and D. Piccolo, "Time Series Clustering and Classification by the Autoregressive Metric," *Computational Statistics & Data Analysis,* vol. 52, no. 4, pp. 1860-1872, 2008.

[5] T. Sauer, J. Yorke, and M. Casdagli, "Embedology," *J. Statistical Physics,* vol. 65, no. 3, pp. 579-616, 1991.

[6] H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis.* Cambridge Univ. Press, 2004.

[7] M. Small, *Applied Nonlinear Time Series Analysis: Applications in Physics, Physiology and Finance.* World Scientific, 2005.

[8] F. Takens, "Detecting Strange Attractors in Turbulence," *Dynamical Systems and Turbulence,* vol. 898, no. 1, pp. 365-381, 1981.

[9] J.A. Ward, P. Lukowicz, and H.W. Gellersen, "Performance Metrics for Activity Recognition," *ACM Trans. Intelligent Systems Technology,* vol. 2, pp. 1-23, Jan. 2011.

[10] J.C. Dunn, "Well-Separated Clusters and Optimal Fuzzy Partitions," *J. Cybernetics,* vol. 4, no. 1, pp. 95-104, 1974.

[11] D.L. Davies and D.W. Bouldin, "A Cluster Separation Measure," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 1, no. 2, pp. 224 -227, Apr. 1979.

[12] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-Supervised Learning.* MIT Press, 2006.

[13] E. Keogh, X. Xi, L. Wei, and C.A. Ratanamahatana, "The UCR Time Series Classification/Clustering Homepage," www.cs.ucr.edu/~eamonn/time_series_data/, 2006.

[14] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. Ratanamahatana, "Fast Time Series Classification Using Numerosity Reduction," *Proc. Int'l Conf. Machine Learning,* pp. 1033-1040, 2006.

[15] H. Sakoe, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. 26, no. 1, pp. 43-49, Feb. 1978.

[16] C. Ratanamahatana and E. Keogh, "Making Time-Series Classification More Accurate Using Learned Constraints," *Proc. SIAM Int'l Conf. Data Mining,* pp. 11-22, 2004.

[17] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When Is 'Nearest Neighbor' Meaningful?" *Proc. Int'l Conf. Database Theory,* pp. 217-235, 1999.

[18] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient Similarity Search in Sequence Databases," *Proc. Fourth Int'l Conf. Foundations of Data Organization and Algorithms,* pp. 69-84, 1993.

[19] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 419-429, 1994.

[20] K. Chan and A. Fu, "Efficient Time Series Matching by Wavelets," *Proc. Int'l Conf. Data Eng.,* pp. 126-133, 1999.

[21] F. Korn, H.V. Jagadish, and C. Faloutsos, "Efficiently Supporting Ad Hoc Queries in Large Data Sets of Time Sequences," *ACM SIGMOD Record,* vol. 26, no. 2, pp. 289-300, 1997.

[22] C.S. Burrus, R.A. Gopinath, and H. Guo, *Introduction to Wavelets and Wavelet Transforms: A Primer.* Prentice Hall, 1998.

[23] J. Durbin, "The Fitting of Time-Series Models," *Rev. Int'l Statistical Inst.,* vol. 28, no. 3, pp. 233-244, 1960.

[24] L. Rabiner and B. Juang, "An Introduction to Hidden Markov Models," *IEEE ASSP Magazine,* vol. 3, no. 1, pp. 4-16, Jan. 1986.

[25] P. Smyth, "Clustering Sequences with Hidden Markov Models," *Proc. Advances in Neural Information Processing Systems,* pp. 648-654, 1997.

[26] B.H. Juang and L.R. Rabiner, "A Probabilistic Distance Measure for Hidden Markov Models," *AT&T Technical J.,* vol. 64, no. 2, pp. 391-408, 1985.

[27] L. Bao and S.S. Intille, "Activity Recognition from User-Annotated Acceleration Data," *Proc. Int'l Conf. Pervasive Computing,* pp. 1-17, 2004.

[28] E.A. Heinz, K.S. Kunze, S. Sulistyo, H. Junker, P. Lukowicz, and G. Trster, "Experimental Evaluation of Variations in Primary Features Used for Accelerometric Context Recognition," *Proc. First European Symp. Ambient Intelligence,* pp. 252-263, 2003.

[29] N. Ravi, N. Dandekar, P. Mysore, and M.L. Littman, "Activity Recognition from Accelerometer Data," *Proc. 17th Conf. Innovative Applications of Artificial Intelligence,* pp. 1541-1546, 2005.

[30] T. Huynh and B. Schiele, "Analyzing Features for Activity Recognition," *Proc. Joint Conf. Smart Objects and Ambient Intelligence,* pp. 159-163, 2005.

[31] J. Lester, T. Choudhury, and G. Borriello, "A Practical Approach to Recognizing Physical Activities," *Proc. Fourth Int'l Conf. Pervasive Computing,* pp. 1-16, 2006.

[32] T. Buzug and G. Pfister, "Optimal Delay Time and Embedding Dimension for Delay-Time Coordinates by Analysis of the Global Static and Local Dynamical Behavior of Strange Attractors," *Physics Rev. A,* vol. 45, no. 10, pp. 7073-7084, 1992.

[33] M. Kennel, R. Brown, and H. Abarbanel, "Determining Embedding Dimension for Phase Space Reconstruction Using the Method of False Nearest Neighbors," *Physics Rev. A*, vol. 45, no. 6, pp. 3403-3411, 1992.
[34] A. Galka, *Topics in Nonlinear Time Series Analysis: With Implications for EEG Analysis*. World Scientific, 2000.
[35] J. Röschke and J. Aldenhoff, "The Dimensionality of Human's Electroencephalogram During Sleep," *Biological Cybernetics*, vol. 64, no. 4, pp. 307-313, 1991.
[36] R. Esteller, G. Vachtsevanos, J. Echauz, T. Henry, P. Pennell, C. Epstein, R. Bakay, C. Bowen, and B. Litt, "Fractal Dimension Characterizes Seizure Onset in Epileptic Patients," *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing*, vol. 4, pp. 2343-2346, 1999.
[37] K. Bush and J. Pineau, "Manifold Embeddings for Model-Based Reinforcement Learning under Partial Observability," *Proc. Advances in Neural Information Processing Systems*, vol. 22, pp. 189-197, 2009.
[38] P. Indyk and R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality," *Proc. ACM Symp. Theory of Computing*, pp. 604-613, 1998.
[39] D. Lemire, "Faster Retrieval with a Two-Pass Dynamic-Time-Warping Lower Bound," *Pattern Recognition*, vol. 42, no. 9, pp. 2169-2180, Sept. 2009.
[40] A.L. Goldberger, L.A.N. Amaral, L. Glass, J.M. Hausdorff, P.C. Ivanov, R.G. Mark, J.E. Mietus, G.B. Moody, C.-K. Peng, and H.E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals," *Circulation*, vol. 101, no. 23, pp. e215-e220, 2000.
[41] S. Johnson, "Hierarchical Clustering Schemes," *Psychometrika*, vol. 32, no. 3, pp. 241-254, 1967.
[42] J. Ward Jr., "Hierarchical Grouping to Optimize an Objective Function," *J. Am. Statistical Assoc.*, vol. 58, pp. 236-244, 1963.
[43] M. Friedman, "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance," *J. Am. Statistical Assoc.*, vol. 32, no. 200, pp. 675-701, 1937.
[44] B. Bergmann and G. Hommel, "Improvements of General Multiple Test Procedures for Redundant Systems of Hypotheses," *Multiple Hypotheses Testing*, pp. 100-115, Springer, 1988.
[45] E. Keogh and S. Kasetty, "On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration," *Data Mining and Knowledge Discovery*, vol. 7, no. 4, pp. 349-371, 2003.
[46] D. Berndt and J. Clifford, "Using Dynamic Time Warping to Find Patterns in Time Series," *Proc. AAAI Workshop Knowledge Discovery in Databases*, pp. 229-248, 1994.
[47] A. Subramanya, A. Raj, J. Bilmes, and D. Fox, "Recognizing Activities and Spatial Context Using Wearable Sensors," *Proc. 22nd Conf. Uncertainty in Artificial Intelligence*, 2006.
[48] J. Zhu, S. Rosset, H. Zou, and T. Hastie, "Multi-Class AdaBoost," technical report, Dept. of Statistics, Univ. of Michigan, 2005.
[49] Y. Freund and R. Schapire, "A Decision Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Computer and System Sciences*, vol. 55, no. 1, pp. 119-139, 1997.
[50] I. Mukherjee and R. Schapire, "A Theory of Multi-Class Boosting," *Proc. Advances in Neural Information Processing Systems*, 2010.
[51] H. Ailisto, M. Lindholm, J. Mantyjarvi, E. Vildjiounaite, and S. Makela, "Identifying People from Gait Pattern with Accelerometers," *Proc. Soc. Photo-Optical Instrumentation Eng. Conf. Series*, vol. 5779, pp. 7-14, 2005.
[52] D. Gafurov, E. Snekkenes, and P. Bours, "Spoof Attacks on Gait Authentication System," *IEEE Trans. Information Forensics and Security*, vol. 2, no. 3, pp. 491-502, Sept. 2007.
[53] M. Bächlin, J. Schumm, D. Roggen, and G. Töster, "Quantifying Gait Similarity: User Authentication and Real-World Challenge," *Proc. Third Int'l Conf. Biometrics*, pp. 1040-1049, 2009.

Jordan Frank received the BSc degree in computer science from Simon Fraser University in 2006, and the MSc degree in computer science from McGill University in 2008. He is currently working toward the PhD degree in the School of Computer Science at McGill University, where he is supported by the NSERC Alexander Graham Bell PhD Canadian Graduate Scholarship. His research interests include machine learning and pattern recognition, reinforcement learning, and ubiquitous computing. He is a student member of the IEEE.

Shie Mannor received the BSc degree in electrical engineering, the BA degree in mathematics, and the PhD degree in electrical engineering from the Technion-Israel Institute of Technology, Haifa, Israel, in 1996, 1996, and 2002, respectively. From 2002 to 2004, he was a Fulbright scholar and a postdoctoral associate at MIT. He was with the Department of Electrical and Computer Engineering at McGill University from 2004 to 2010, where he held the Canada Research Chair in Machine Learning. He has been an associate professor at the Faculty of Electrical Engineering at the Technion since 2008. His research interests include machine learning and pattern recognition, planning and control, multi-agent systems, and communications. He is a senior member of the IEEE.

Joelle Pineau received the PhD degree from Carnegie Mellon University in 2004. She is an associate professor of computer science at McGill University. Her research interests are in artificial intelligence, probabilistic machine learning, and robotics.

Doina Precup received the BEng degree in computer science from the Technical University Cluj-Napoca, Romania, and the MSc and PhD degrees in computer science from the University of Massachusetts, Amherst, where she was a Fulbright fellow. She is an associate professor in the School of Computer Science at McGill University. Her current research interests include artificial intelligence, machine learning, and the application of these methods to real-world time series data.