Model-based Time Series Classification

Alexios Kotsifakos[1] and Panagiotis Papapetrou[2]

[1]Department of Computer Science and Engineering, University of Texas at Arlington, USA
[2]Department of Computer and Systems Sciences, Stockholm University, Sweden

Abstract. We propose `MTSC`, a filter-and-refine framework for time series Nearest Neighbor (NN) classification. Training time series belonging to certain classes are first modeled through Hidden Markov Models (HMMs). Given an unlabeled query, and at the filter step, we identify the top K models that have most likely produced the query. At the refine step, a distance measure is applied between the query and all training time series of the top K models. The query is then assigned with the class of the NN. In our experiments, we first evaluated the NN classification error rate of HMMs compared to three state-of-the-art distance measures on 45 time series datasets of the UCR archive, and showed that modeling time series with HMMs achieves lower error rates in 30 datasets and equal error rates in 4. Secondly, we compared `MTSC` with `Cross Validation` defined over the three measures on 33 datasets, and we observed that `MTSC` is at least as good as the competitor method in 23 datasets, while achieving competitive speedups, showing its effectiveness and efficiency.

1 Introduction

Time series data have become ubiquitous during the last decades. Sequences of numerical measurements are produced at regular or irregular time intervals in vast amounts in almost every application domain, such as stock markets, medicine, and sensor networks. Large databases of time series can be exploited so as to extract knowledge on what has happened in the past or to recognize what is happening in the present. More specifically, the task at hand is *classification*. Given an *unlabeled* time series, i.e., of which the *category/class* is *not* known, we wish to assign to it the *most* appropriate class, which corresponds to the class of the *Nearest Neighbor* (*NN*) time series. Consequently, there is a need for *searching* the time series database.

One way of implementing such search is, given a distance measure, to perform *whole sequence matching* between each time series in the database and the query, and finally select the closest time series, i.e., the one with the smallest distance to the query. Thus, it can be easily understood that the selection of the distance measure to be used for comparing time series is critical, as it essentially decides whether a time series is a good match for the query or not, influencing the NN classification accuracy (percentage of time series correctly classified).

Several distance measures have been proposed, which are often computed in a dynamic programming (DP) manner [6]. The most widely known and used measure is Dynamic Time Warping (DTW) [22]. Many variants of DTW have also been proposed, such as cDTW [30], EDR [8], and ERP [9]. All of these measures have the attractive

characteristic that they are robust to misalignments along the temporal axis. Moreover, they provide very good classification accuracy results [34].

Searching large time series databases with any of these DP-based distance measures can be computationally expensive due to their quadratic complexity with respect to the time series length, though speedups can be achieved for cDTW by applying a lower-bounding technique [16]. An alternative approach would be to first represent each class of the time series database with a *model*, such as a *Hidden Markov Model* (*HMM*) [5], and then perform searching based on the constructed models. HMMs are widely known and have been used in a variety of domains, such as speech recognition [29], and music retrieval [28]. In this paper, we deal with both effectiveness (accuracy) and efficiency (runtime) and we propose a novel approach, named `MTSC` (shorthand for Model-based Time Series Classification). Given sets of time series of certain classes, `MTSC` first models their underlying structure through the training of one HMM per class. An HMM is capable of identifying the relationships between the observations within the time series [18]. At runtime, given a query time series of unknown class, `MTSC` finds the top K models that have most likely produced the query. Then, it refines the search by applying an appropriate distance measure between the query and all the training time series that compose the K selected models. What remains to be answered is what distance measure to use during the refine step. Intuitively, given a collection of distance measures, we can choose the one providing the highest classification accuracy on the training set.

The **main contributions** of this paper include: 1) A novel way of representing time series of a specific class via an HMM, and a comparative evaluation of this representation against three distance measures (DTW, ERP, and MSM) in terms of classification accuracy on the training sets of 45 datasets [17]. The evaluation shows that HMMs can attain significantly higher accuracy in 18 datasets, relatively higher accuracy in 12, and equal accuracy in 4; hence better or equal accuracy in 34 datasets. 2) `MTSC`: an model-based framework for effective and efficient time series NN classification. The framework works in a filter-and-refine manner, by exploiting the novel model-based representation of time series belonging to the same class. 3) An extensive experimental evaluation on NN classification accuracy between `MTSC` and the `Cross Validation` method defined over DTW, ERP, and MSM, on 33 datasets. We observed that `MTSC` is at least as good as `Cross Validation` in 23 datasets, while achieving competitive speedups, showing *both* its effectiveness and efficiency.

2 Related Work

A plethora of distance/similarity methods for time series have been proposed during the last decades. As shown by Wang et al. [34], Dynamic Time Warping (DTW) [22] is not only a widely used distance measure for computing distances between time series, but also provides very good classification accuracy results. Hence, several lower bounds have been proposed to speed up its expensive computation [2,24]. Variants of DTW include constrained DTW (cDTW) [30], Edit Distance on Real sequence (EDR) [9], and Edit distance with Real Penalty (ERP) [8]. A common characteristic of these methods is that they allow aligning elements "warped" in time. ERP and EDR satisfy the triangle inequality, and ERP is found to be more robust to noise than EDR. Another

very recently proposed measure is called Move-Split-Merge (MSM) [31], which applies three types of operations (Move, Split, Merge) to transform one time series to another. MSM is metric and invariant to the choice of origin as opposed to ERP. Longest Common SubSequence (LCSS) [32] finds the maximum number of elements being common in the compared sequences allowing for gaps during the alignment. Other distance or similarity measures include DISSIM [12], TWED [26], and SMBGT [19,21].

Several techniques for time series representation have been proposed in the literature that capture global or local structural characteristics, e.g., SpaDe [10] and SAX [25]. Moreover, and *Shapelets* [35] focus on determining discriminant time series subsequence patterns, and have been used for classification. For a thorough comparison of different representations and measures please refer to Wang et al. [34], which demonstrates that there is little difference among them. Speeding up similarity search in large databases based on different time series summarizations, such as DFT [1] has also attracted the attention of researchers. Some recent approaches for faster whole sequence matching are embedding-based, which use a set of reference objects to transform similarity matching in a new vector space instead of the original one [3].

Although the aforementioned approaches are very promising, they focus on representing each time series by taking advantage of its structure. However, in this work we try to represent "groups" of time series that belong to the same class, which is orthogonal to the previously proposed techniques.

We focus on HMMs, which model the underlying structure of sequences determining the relationships between their observations. HMMs have been applied to speech recognition [29] and music retrieval [20]. Although training may be computationally expensive, once they are constructed they can be highly applicable to time series, as shown in this work through the classification task. Our approach differs from selecting the best model in Markovian Processes [15], since at each state we neither perform an action nor give a reward. Furthermore, model-based kernel for time series analysis requires significant amount of time [7]. Conditional Random Fields (CRFs) [23,33] can be used for modeling temporal patterns. Nonetheless, we do not target in finding and modeling patterns, rather to represent groups of "homogeneous" sequences by identifying the relationships among their observations. In addition, non-parametric techniques have also been proposed within the field of functional data analysis [11,14], though our approach is orthogonal to those. Finally, HMM-based approaches have also been used for the problem of time series clustering [13,27]; however, our focus here is classification and hence the proposed approach is customised for this task.

3 MTSC: Model-based Time Series Classification

Let $X = (x_1, \ldots, x_{|X|})$ and $Y = (y_1, \ldots, y_{|Y|})$ be two time series of length $|X|$ and $|Y|$, respectively, where $x_i, y_j \in \mathbb{R}$, $\forall (i = 1, \ldots, |X|; j = 1, \ldots, |Y|)$. Since we are interested in labeled time series, we denote the class of X as c_X. The pair (X, c_X) will correspond to X along with its label. Given a distance measure $dist_x$, the distance between X and Y is defined as a function $d_{\text{dist}_x}(X, Y)$. The problem we would like to solve is given next.

Nearest Neighbor Classification Given a collection of N training time series $\mathscr{D} = \{(X_1, c_{X_1}), \ldots, (X_N, c_{X_N})\}$, a distance measure $dist_x$, and an unlabeled query time series

Q, find the class c_Q of Q as follows:

$$c_Q = \{c_{X_j} | \arg\min_j (d_{dist_x}(Q, X_j)), \forall (j = 1, \ldots, N)\}$$

Next, we provide an overview of HMMs, how we can represent classes of time series with appropriate training of HMMs, and describe the `MTSC` framework, which takes advantage of the trained models for NN classification.

3.1 Hidden Markov Models

An HMM is a doubly stochastic process containing a finite set of states [29]. Formally, it is defined by M distinct states, L values that can be observed at each state (for time series any real number can be observed), the set $T = \{t_{uv}\}$ of transition probabilities, where $t_{uv} = P[s_t = v | s_{t-1} = u]$ with $1 \leq u, v \leq M$ and s_t being the state at time t (first order Markov chain), the set $E = \{e_v(k)\}$ of the probabilities of values at state v, where $e_v(k) = P[o_t = k | s_t = v]$ with o_t being the observed/emitted value at time t, and the set $\Pi = \{\pi_v\}$ of prior probabilities, where $\pi_v = P[s_1 = v], 1 \leq v \leq M$.

When a database consists of sets of time series belonging to certain classes, HMMs can be used to model the different classes after being trained on their respective time series. This lies on the fact that a trained HMM can reflect the probabilistic relations of the values within the sequences, and consequently represent their common structure. Thus, HMMs can be highly applicable for retrieval or classification [28]. Given a query Q, we can look for the model that maximizes the likelihood of having generated Q. With this direction in mind, the time series matching problem is transformed to probabilistic-based matching.

3.2 Training HMMs

Assume that we have a dataset $\mathscr{D}$ with z classes $C_1, \ldots, C_z$. Let $\mathscr{C}_i$ be the set of training time series that belong to class C_i, with $i = 1, \ldots, z$. The size of $\mathscr{C}_i$ is denoted as $|\mathscr{C}_i| = n_i$. The training phase of an HMM for each $\mathscr{C}_i$ is split to two phases, which are performed offline: a) initialization and b) iterative refinement.

Initialization Step For each time series X_j ($j = 1, \ldots, n_i$) of $\mathscr{C}_i$ ($i \in [1, z]$) we compute the average distance of all other time series $X_k \in \mathscr{C}_i$ ($j \neq k$) to X_j, which we denote as $a^j_{dist_x}$, i.e.,

$$a^j_{dist_x} = \frac{1}{n_i - 1} \sum_{\forall X_k \in \mathscr{C}_i, X_j \neq X_k} d_{dist_x}(X_j, X_k).$$

For the above computation we choose DTW to be the distance measure, i.e., $dist_x =$ DTW, since it has been shown to be one of the most competitive measures for time series matching [34]. In addition, we keep track of the *warping path* of all the pair-wise time series alignments involved in this process.

Next, we identify the *medoid* of $\mathscr{C}_i$, denoted as $X_{\mu_{C_i}}$, which is the time series with the minimum average distance to the rest of the training set, where

$$\mu_{C_i} = \arg\min_j (a^j_{dist_x}), \forall (j = 1, \ldots, n_i).$$

The medoid $X_{\mu_{C_i}}$ is broken into M equal-sized *segments*, where each segment $m \in [1,M]$ corresponds to one HMM state. Using these segments and the stored warping paths we can determine the observed values of each state. Specifically, for each state (that corresponds to a segment m) the observed values include all elements of $X_{\mu_{C_i}}$ in m, along with the elements of all time series in $\mathscr{C}_i$ that have been aligned to elements of m; the latter can be retrieved from the stored warping paths.

A common case in HMMs is to have for each state a Gaussian distribution for E, which is defined by the mean and standard deviation of the stored elements. To compute T, since we consider each segment as a state, at time t when an observation is emitted we can either stay at the same state or move forward to the next state. Let $|s_t|$ denote the total number of elements at time t of state s_t. The probability of jumping to the next state is $p = n_i/|s_t|$. This is quite straightforward: consider only one segment and one time series X_j ($j = 1,\ldots,n_i$), and suppose that X_j contributes y elements to that segment. Since only the last element in the segment can lead to a transition from s_t to s_{t+1}, the transition probability is $1/y$. Considering now that all n_i time series contribute to the segment, the probability of a state transition is $p = n_i/|s_t|$. Finally, the probability of staying at the same state is $(1-p) = (|s_t| - n_i)/|s_t|$, while for the last state it is 1. In total, the complexity of this step is $O((n_i^2 max_{j\in[1,n_i]}|X_j|)^2)$.

Iterative Refinement Step In this step, we refine the z HMMs constructed during initialization. For a specific class C_i ($i \in [1,z]$), for each $X_j \in \mathscr{C}_i$, we compute the Viterbi algorithm [29] to find its best state sequence. Specifically, let us denote as δ the Mx$|X_j|$ probability matrix. Each X_j always starts from state 1 (thus $\pi_1 = 1$), and in the initialization phase the log-likelihood of its first element is computed according to the Gaussian distribution. The remaining elements of the first column of δ are set to $-\infty$. Since to get an observation we have either stayed at the same state or have performed a transition from the previous state, in the recursion phase we consider only the values of δ representing the probabilities of the previous element for the previous and the current state. For cell (u,v) these values are $\delta(u,v-1)$ and $\delta(u-1,v-1)$, which were computed in the initialization step. Hence, we first find the most probable transition by computing $m = max(\delta(u,v-1) + \log(t_{uu}), \delta(u-1,v-1) + \log(t_{u-1u}))$, and then $\delta(u,v) = m + \log(e_u(k))$. Finally, we backtrack from $\delta(M,|X_j|)$ and store the elements of X_j falling within each state. Having done this step for all $X_j \in \mathscr{C}_i$, the mean and standard deviation for E of each state, and also T are updated. The complexity of the aforementioned procedure is $O(M\sum_{j=1}^{n_i}|X_j|)$.

The refinement step is performed for the z HMMs and is repeated until a stopping criterion is met, e.g., the classification accuracy on the N training time series composing $\mathscr{D}$ cannot be further improved (Section 4.1). The final outcome of this step is a set of z HMMs, denoted as $\mathscr{H} = \{H_1,\ldots,H_z\}$, where each $H_i \in \mathscr{H}$ defines a probability distribution for $\mathscr{C}_i$, which essentially describes the likelihood of observing any time series of class C_i.

3.3 Filter-and-Refine Framework

Given $\mathscr{D}$, the `MTSC` framework consists of three steps: offline, filter, and refine.

Offline Step First, we construct $\mathscr{H}$ as described in Section 3.2. To make our framework more generic, assume that we have available a set of l distance measures $\{dist_1,\ldots,dist_l\}$.

For each $dist_x$ ($x \in [1,l]$) we compute the NN classification accuracy on the N training time series using leave-one-out cross validation.

Filter Step Since we have created a "new" probabilistic space for time series similarity matching, we should define a way of measuring how "good" each HMM model $H_i \in \mathscr{H}$ is for a query Q. This can be achieved by applying the Forward algorithm [29], which computes the likelihood of Q having been produced by H_i. Thus, the "goodness" of H_i is the likelihood estimate given by the algorithm. The complexity of the Forward algorithm is $O(|Q|M^2)$. Note that this step involves only z computations of the Forward algorithm and it is significantly fast, given that in practice z is rarely higher than 50. This is based on the fact that in the 45 datasets of the UCR archive [17], which cover real application domains, z is at most 50. After computing the likelihood of each H_i for Q, we identify the K models with the highest likelihood of producing Q.

Refine Step Next, the training time series that comprise each of the top K selected models are evaluated with Q. This evaluation is performed using the distance measure that achieved the highest classification accuracy on the training set during the offline step. Finally, Q is assigned with the class of the closest time series. The complexity of this step is $O(K'comp(dist_x))$, where K' is the total number of training time series corresponding to the K selected models, and $comp(dist_x)$ is the complexity of computing the distance between two time series using $dist_x$ (selected in the offline step).
It has to be mentioned that the smaller the K the faster our approach is. However, since each HMM is a very compact representation of all time series of a certain class, reducing the number of models selected at the filter step may greatly reduce accuracy, as the time series that will be evaluated at the refine step may not include those of the correct class. On the contrary, as K increases towards z, more training time series will be evaluated, resulting in the brute-force approach evaluating Q with all time series when $K = z$, which is certainly undesirable. Hence, a good value for K is needed to achieve a good tradeoff between effectiveness and efficiency. The choice of distance measure also influences accuracy, but it is beyond our scope to select the "best" measure for *each* Q.

4 Experiments

In this section, we present the setup and the experimental evaluation for HMM-based representation and `MTSC`.

4.1 Experimental Setup

We experimented on the 45 time series datasets available from the UCR archive [17]. A summary of the dataset statistics is included in Table 1. We first evaluated the performance of HMMs (Section 3.2) against DTW, ERP, and MSM. Secondly, we compared `MTSC` with `Cross Validation` using the same three measures.

Although any distance measure can be used to perform NN classification, an exhaustive consideration of all of distance measures is beyond the scope of this paper. The rationales behind selecting these measures are the following: (1) DTW is extensively used in time series and has been shown to provide excellent classification accuracy results [34], (2) ERP is a variant of DTW and Edit Distance fixing the non-metric property

of DTW, and (3) MSM is also metric and has been shown to outperform ERP and DTW in terms of NN classification accuracy on several datasets. Their time complexity is quadratic with respect to the time series length.

The `Cross Validation` method works as follows: (1) for each dataset we computed the classification accuracy for DTW, ERP, and MSM on the training set using leave-one-out cross validation, and (2) the method outputs the classification accuracy on the test set of the measure with the best accuracy on the training set. If more than one measures provide the same highest classification accuracy on the training set, then the accuracy of `Cross Validation` is the accuracy on the test set of the measure that outperforms the other tied measure(s) on most datasets (on their training sets).

Table 1. NN classification error rates attained by MSM, DTW, ERP, and HMMs on the training set of 45 datasets from the UCR repository. The table shows for each dataset: the number of training and test objects, the length of each time series in the dataset, the number of classes, the value of parameter c used by MSM on that dataset that yielded the lowest error rate on the training set (when two or three values are given, the one in italics was randomly chosen), the number of states as a percentage of the time series length and the number of iterations for which the HMMs achieved the lowest error rate on the training set. Numbers in bold indicate the smallest error rate.

ID	Dataset	train error rate (%)				train	test	length	class	parameter	state	iter.
		MSM	DTW	ERP	HMMs	size	size	$\|X\|$	num. z	c (MSM)	perc.	num.
1	*Synthetic*	1.33	1.00	0.67	**0.33**	300	300	60	6	0.1	0.4	3
2	*CBF*	**0.00**	**0.00**	**0.00**	**0.00**	30	900	128	3	0.1	0.5	2
3	*FaceAll*	**1.07**	6.79	2.50	1.25	560	1,690	131	14	1	0.5	11
4	*OSU*	19.50	33.00	30.50	**17.00**	200	242	427	6	0.1	0.3	11
5	*SwedishLeaf*	12.40	24.60	13.40	**12.20**	500	625	128	15	1	0.9	6
6	*50Words*	21.11	33.11	28.22	**8.89**	450	455	270	50	1	0.5	1
7	*Trace*	1.00	**0.00**	9.00	**0.00**	100	100	275	4	0.01	0.3	2
8	*TwoPatterns*	**0.00**	**0.00**	**0.00**	**0.00**	1,000	4,000	128	4	1	0.1	1
9	*FaceFour*	8.33	25.00	12.50	**0.00**	24	88	350	4	1	0.1	1
10	*Lightning-7*	27.14	32.86	28.57	**7.14**	70	73	319	7	1	0.2	1
11	*Adiac*	38.97	40.51	39.49	**15.90**	390	391	176	37	1	0.5	8
12	*Fish*	13.71	26.29	17.14	**7.43**	175	175	463	7	0.1	0.5	4
13	*Beef*	66.67	53.33	66.67	**23.33**	30	30	470	5	0.1	0.4	2
14	*OliveOil*	16.67	13.33	16.67	**3.33**	30	30	570	4	0.01	0.3	2
15	*ChlorineConc.*	38.97	38.97	**38.76**	56.32	467	3,840	166	3	1	0.7	4
16	*ECG_torso*	12.50	32.50	25.00	**2.50**	40	1,380	1,639	4	1	0.4	3
17	*Cricket_X*	**18.46**	20.26	21.54	25.38	390	390	300	12	1	0.5	14
18	*Cricket_Y*	24.10	20.51	23.33	**19.23**	390	390	300	12	0.1, *1*	0.4	9
19	*Cricket_Z*	24.10	**22.56**	24.87	23.08	390	390	300	12	1	0.4	14
20	*Diatom Red.*	6.25	6.25	6.25	**0.00**	16	306	345	4	0.01, 0.1, *1*	0.1	4
21	*FacesUCR*	2.50	10.00	5.50	**0.50**	200	2,050	131	14	1	0.8	9
22	*Haptics*	49.68	58.71	54.19	**29.68**	155	308	1,092	5	1	0.1	15
23	*InlineSkate*	50.00	59.00	49.00	**43.00**	100	550	1,882	7	1	0.5	3
24	*MALLAT*	5.45	5.45	5.45	**0.00**	55	2,345	1,024	8	1	0.1	1
25	*MedicalImages*	27.82	27.56	**26.51**	34.91	381	760	99	10	0.1	0.4	13
26	*StarLightC.*	10.70	9.60	13.80	**8.60**	1,000	8,236	1,024	3	0.1	0.5	14
27	*Symbols*	**0.00**	4.00	8.00	**0.00**	25	995	398	6	0.1	0.1	1
28	*uWaveGest_X*	25.78	29.35	26.67	**25.22**	896	3,582	315	8	0.1, *1*	0.5	11
29	*uWaveGest_Y*	**28.24**	37.05	33.93	34.60	896	3,582	315	8	1	0.7	13
30	*uWaveGest_Z*	29.24	33.59	31.25	**27.46**	896	3,582	315	8	1	0.2	12
31	*WordsSynon.*	22.10	36.33	28.46	**13.86**	267	638	270	25	1	0.8	13
32	*ECGThorax1*	18.17	20.11	17.83	**7.17**	1,800	1,965	750	42	1	0.5	15
33	*ECGThorax2*	10.83	14.17	11.72	**7.67**	1,800	1,965	750	42	1	0.5	14
34	*Gun Point*	**4.00**	18.00	8.00	8.00	50	150	150	2	0.01	0.3	2
35	*Wafer*	**0.10**	1.40	**0.10**	1.70	1,000	6,164	152	2	1	0.9	4
36	*Lightning-2*	16.67	13.33	13.33	**5.00**	60	61	637	2	0.01	0.1	15
37	*ECG*	14.00	23.00	18.00	**12.00**	100	100	96	2	1	0.8	2
38	*Yoga*	**12.00**	18.33	17.33	22.33	300	3,000	426	2	0.1	0.2	15
39	*Coffee*	25.00	**14.29**	25.00	21.43	28	28	286	2	0.01	0.3	1
40	*ECGFiveDays*	26.09	43.48	26.09	**0.00**	23	861	136	2	1	0.2	4
41	*ItalyPowerDemand*	**4.48**	**4.48**	5.97	5.97	67	1,029	24	2	0.1, *1*	0.9	2
42	*MoteStrain*	15.00	25.00	25.00	**0.00**	20	1,252	84	2	0.1	0.5	7
43	*SonySurfaceI*	10.00	20.00	15.00	**0.00**	20	601	70	2	1	0.1	1
44	*SonySurfaceII*	11.11	14.81	18.52	**3.70**	27	953	65	2	0.1	0.3	1
45	*TwoLeadECG*	4.35	8.70	4.35	**0.00**	23	1,139	82	2	*0.01*, 0.1	0.1	4

For each dataset, parameter c of MSM was selected from $\{0.01, 0.1, 1\}$ [31]. For each value the classification accuracy was found using leave-one-out cross-validation

on the training set, and the value yielding the highest accuracy was selected. For the training of HMMs, for each of the 45 datasets, we varied M from 0.1 to $0.9 * |X|$ (step 0.1) and applied 15 refinement iterations (135 combinations). For each combination we measured the percentage of training time series for which the model producing the highest likelihood through the Forward algorithm was the correct one. Then, the combination leading to the highest accuracy was selected. If more than one combinations provided the same highest accuracy we chose the smallest M (for further ties smallest number of iterations).

Evaluation Measures We first evaluated the performance of HMMs against DTW, ERP, and MSM on the training sets, and between `MTSC` and `Cross Validation` on the test sets in terms of *classification error rate*. This rate is defined as the percentage of time series misclassified using the NN classifier. Secondly, we evaluated the *efficiency* of `MTSC` and `Cross Validation`. Nonetheless, runtime measurements may depend on particular aspects of the hardware, implementation details, compiler optimizations, and programming language. To overcome these limitations, we also present per dataset the percentage of classes selected at the filter step ($(K/z) * 100$), and, more importantly, the percentage of training time series that are finally evaluated by `MTSC`, as the number of time series may (sometimes greatly) deviate among classes in the same dataset. `MTSC` was implemented in Matlab, while, for efficiency, DTW, MSM, ERP, and the Forward algorithm were implemented in Java. Experiments were performed on a PC running Linux, with Intel Xeon Processor at 2.8GHz.

4.2 Experimental Results

Next, we present our experimental findings for the methods and evaluation measures.

Classification accuracy of HMMs For each of the 45 datasets, we compared the classification error rates on the training set for MSM, DTW, ERP, and HMMs. The results are shown in Table 1. We observe that HMMs achieve better or equal error rate than that of the competitor distance measures in 34 datasets, out of which they outperform them in 30. The performance of HMMs is in many cases significantly better than all competitors. For example, for *ECGFiveDays* HMMs achieve an error rate of 0% as opposed to the next best which is 26.09% (achieved by both ERP and MSM), while for 18 datasets (e.g., *50Words*, *Lightning-7*, *Adiac*, *Beef*, *OliveOil*, and *ECG_torso*) the error rate of HMMs is at least two times lower than that of the competitors. These numbers show that modeling time series classes with HMMs is highly competitive and promising for NN classification.

Classification accuracy of MTSC We studied the classification error rates of `MTSC` against `Cross Validation` on the test sets of the 33 datasets with $z > 2$. The results are shown in Table 2. Note that if $z = 2$ with `MTSC` we can either select one or two models for the refine step. However, $K = 1$ would essentially exclude the refine step, since time series of only one class would be evaluated, which is meaningless. Hence, the classification error rate would depend solely on how well the HMMs represent and discriminate the classes of the dataset, which is not always the case due to their very compact representation of (sometimes large) classes. In addition, $K = 2$ would make our approach perform brute-force search, which is undesirable. In columns "top K", "% classes", "avg train num.", "% train obj." we show the K value used at the filter step of

Table 2. NN classification error rates attained by `MTSI` and `Cross Validation` on the test set of 33 datasets from the UCR repository. The table also shows for each dataset: the classification error rate of MSM, DTW, and ERP on the test set, the number of HMM models used at the refine step of `MTSI` and the respective percentage of classes it corresponds to, the average number of training objects evaluated at the refine step per test object, and the percentage of training objects this average corresponds to. Numbers in bold indicate the smallest error rate.

ID	Dataset	MTSI	Cross Valid.	error rate (%) MSM	DTW	ERP	top K	% classes	avg train num.	% train obj.
1	*Synthetic*	**3.33**	3.70	2.67	0.70	3.70	2	33.33	100.00	33.33
2	*CBF*	3.67	**1.22**	1.22	0.30	0.30	2	66.67	20.45	68.16
3	*FaceAll*	18.99	**18.88**	18.88	19.20	20.20	5	35.71	200.00	35.71
4	*OSU*	22.73	**19.83**	19.83	40.90	39.70	5	83.33	169.00	84.50
5	*SwedishLeaf*	**9.60**	10.40	10.40	21.00	12.00	3	20.00	100.87	20.17
6	*50Words*	**19.56**	**19.56**	19.56	31.00	28.10	40	80.00	412.26	91.61
7	*Trace*	**0.00**	**0.00**	7.00	0.00	17.00	2	50.00	49.82	49.82
8	*TwoPatterns*	**0.05**	0.08	0.08	0.00	0.00	2	50.00	498.47	49.85
9	*FaceFour*	**4.55**	5.68	5.68	17.00	10.20	2	50.00	11.44	47.68
10	*Lightning-7*	**21.92**	23.29	23.29	27.40	30.10	6	85.71	64.18	91.68
11	*Adiac*	**33.76**	38.36	38.36	39.60	37.90	2	5.41	20.96	5.37
12	*Fish*	**7.43**	8.00	8.00	16.70	12.00	6	85.71	149.87	85.64
13	*Beef*	**46.67**	50.00	50.00	50.00	50.00	3	60.00	18.00	60.00
14	*OliveOil*	16.67	**13.33**	16.67	13.33	16.67	3	75.00	21.80	72.67
15	*ChlorineConc.*	40.42	**37.40**	37.27	35.20	37.40	2	66.67	362.14	77.55
16	*ECG_torso*	15.07	**10.29**	10.29	34.90	25.00	3	75.00	30.36	75.91
17	*Cricket_X*	**25.90**	27.18	27.18	22.30	29.23	5	41.67	162.54	41.68
18	*Cricket_Y*	**20.00**	20.80	16.67	20.80	21.28	5	41.67	162.38	41.64
19	*Cricket_Z*	**20.77**	**20.77**	21.54	20.77	24.36	10	83.33	330.00	84.62
20	*Diatom Red.*	**4.58**	**4.58**	4.58	3.30	5.23	3	75.00	14.67	91.67
21	*FacesUCR*	**3.20**	3.27	3.27	9.51	4.24	13	92.86	191.02	95.51
22	*Haptics*	**57.47**	59.42	59.42	62.30	57.47	3	60.00	98.00	63.22
23	*InlineSkate*	57.45	**56.91**	55.64	61.60	56.91	6	85.71	87.25	87.25
24	*MALLAT*	**6.74**	**6.74**	6.74	6.60	7.46	6	75.00	41.98	76.33
25	*MedicalImages*	**27.89**	**27.89**	24.74	26.30	27.89	7	70.00	334.98	87.92
26	*StarLightC.*	9.35	**9.30**	11.72	9.30	13.62	2	66.67	759.62	75.96
27	*Symbols*	3.12	**3.02**	3.02	5.00	5.83	5	83.33	21.00	83.98
28	*uWaveGest_X*	**22.28**	22.36	22.36	27.30	25.71	5	62.50	574.04	64.07
29	*uWaveGest_Y*	**30.35**	30.37	30.37	36.60	33.61	5	62.50	553.80	61.81
30	*uWaveGest_Z*	**29.12**	31.07	31.07	34.20	32.97	2	25.00	222.04	24.78
31	*WordsSynon.*	23.67	**23.51**	23.51	35.10	32.13	24	96.00	263.92	98.85
32	*ECGThorax1*	**18.37**	19.29	18.27	20.90	19.29	2	4.76	85.99	4.78
33	*ECGThorax2*	**10.89**	11.25	11.25	13.50	10.74	7	16.67	300.03	16.67

`MTSC` (due to space limitations we present the smallest $K < z$ for which the accuracy could not be further improved or provided a competitive error rate), the ratio $(K/z) * 100$, the average number of training time series evaluated per query at the refine step, and the percentage of the "train size" to which this average corresponds to, respectively.

We observe that `MTSC` achieves at least as good or better error rates than `Cross Validation` in 23 datasets; it is better in 17 datasets and equal in 6. There are two reasons for such competitive performance of `MTSC`: a) the correct class of the test time series is among the K models selected at the filter step, and the distance measure applied at the refine step is able to better differentiate the correct class from the rest $K-1$, since there are less training objects to throw away as being "bad" matches compared to brute-force search, and b) the HMMs of these 23 datasets are constructed exploiting a sufficient number of training time series comprising their classes, making the probability distribution of such classes effectively represent these (and similar) time series.

Carefully analyzing our experimental findings, we concluded that 16 training time series is a sufficient number to provide a good model representing a class of objects. This claim is supported by the following examples, where `MTSC` yields worse error rates than `Cross Validation`. Datasets with *ID* 14, 16, and 20 consist of 4 classes, but no class of the first two has more than 15 training time series, while the classes of the latter include only 1, 6, 5, and 4 time series. Moreover, none of the 6 classes of dataset with *ID* 27 has more than 8 time series, *WordsSynon.* (*ID* 31) with $z = 25$ has only 4 classes with more than 15 time series, as happens with 3 out of the 7 classes of *InlineSkate* (*ID*

23). The error rates for datasets *ChlorineConc.* and *StarLightC.* (*ID* 15 and 26) can be attributed to overfitting, since for the first no class comprises of less than 91 time series, while for the second all classes have more than 152 time series. In addition, building a histogram over the number of classes that include specific numbers of training time series, we observed that 167 out of the 399 classes (comprising the 33 datasets) have up to 15 training time series. As a result, we would like to emphasize the need for a sufficient number of training time series per class.

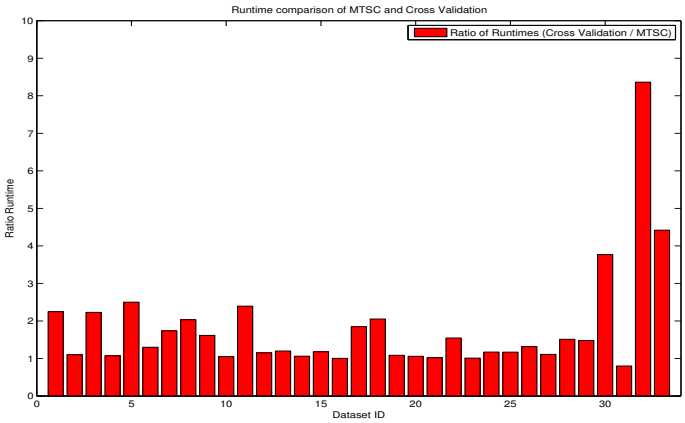

Fig. 1. Speedup of `MTSC` vs. `Cross Validation` for 33 datasets. Each bar represents the ratio of the `Cross Validation` avg. total runtime to that of `MTSC`, for NN classification of a test object.

Efficiency In Figure 1 we present the average *speedup* per test time series when using `MTSC` instead of `Cross Validation` for 33 datasets. The speedup is the runtime ratio of `Cross Validation` over `MTSC`, and it intuitively depends on K. For example, we gain up to a factor of 9 in terms of runtime for dataset with *ID* 32, since `MTSC` selects only 2 out of 42 classes at the filter step, and its error rate is lower than that of `Cross Validation`. We observe that there are several datasets for which there is no significant speedup. This is mainly because, for these datasets, `MTSC` could not achieve a competitive error rate for large K, even for $z-1$ in some cases. Thus, for such values its runtime converged to that of brute-force using the appropriate distance measure. Additionally, there may be cases where the length of the time series is not huge enough to provide a noticeable difference in the runtimes of the two competitors (*ID* 25), resulting in a slight speedup. The latter result may also happen when "train size" is small and/or the average number of training time series for the K selected models is much higher than that of the non-selected ones. The last claim holds, e.g., for datasets with *ID* 6, 15, 20, 25, 26, where the value of "% train obj." is significantly higher than that of "% classes", showing that the training time series are not equally distributed to all classes. Additionally, the percentage of training time series that were evaluated ranges from just 4.78% (*ID* 32) to 98.85% (*ID* 31), which is the worst possible case since no smaller K could provide better accuracy. We have to point out, though, that out of the 23 datasets for which `MTSC` is better than or equal to `Cross Validation` there are 11 datasets for which less than 50% of their training set is evaluated.

Based on these results, we can argue that `MTSC` outperforms `Cross Validation` in classification error rate more often than not, while allowing for a speedup of up to a

factor of 9. An acute reader may argue that the runtime comparison of the two methods is unfair since we could alternatively have used existing speedup methods for DTW[4], or even faster techniques such as cDTW with LB_Keogh [16]. Nonetheless, we argue that any speedup achieved by each method used by `Cross Validation` is also equally beneficial for `MTSC`. This is due to the fact that `MTSC` is using the exact same set of methods for the refine step, and thus any speedup obtained by `Cross Validation` is essentially exploited by `MTSC` as well (the filter step cost is negligible compared to that of the refine step).

5 Conclusions and Future Work

We presented an effective way of modeling classes of time series using HMMs and proposed `MTSC`, a filter-and-refine framework for NN classification of time series. Experimenting with 45 widely known time series datasets and three distance measures we observed that HMMs provide better or equal classification accuracies than the competitors on the training set in 34 datasets. `MTSC` has equal or better accuracy than `Cross Validation` in 23 out of 33 datasets, while achieving a speedup of up to a factor of 9. We plan to test `MTSC` on larger datasets with more classes, where we expect its performance to further improve.

References

1. R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *FODO*, pages 69–84. Springer Verlag, 1993.
2. I. Assent, M. Wichterich, R. Krieger, H. Kremer, and T. Seidl. Anticipatory dtw for efficient similarity search in time series databases. *PVLDB*, 2(1):826–837, 2009.
3. V. Athitsos, M. Hadjieleftheriou, G. Kollios, and S. Sclaroff. Query-sensitive embeddings. In *SIGMOD*, pages 706–717, 2005.
4. V. Athitsos, P. Papapetrou, M. Potamias, G. Kollios, and D. Gunopulos. Approximate embedding-based subsequence matching of time series. In *SIGMOD*, pages 365–378, 2008.
5. L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.
6. R. Bellman. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60(6):503–515, 1954.
7. H. Chen, F. Tang, P. Tino, and X. Yao. Model-based kernel for efficient time series analysis. In *SIGKDD*, pages 392–400, 2013.
8. L. Chen and R. Ng. On the marriage of l_p-norms and edit distance. In *VLDB*, pages 792–803, 2004.
9. L. Chen and M. T. Özsu. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.
10. Y. Chen, M. A. Nascimento, B. Chin, O. Anthony, and K. H. Tung. Spade: On shape-based pattern detection in streaming time series. In *ICDE*, pages 786–795, 2007.
11. F. Ferraty and P. Vieu. Curves discrimination: a nonparametric functional approach. *Computational Statistics and Data Analysis*, 44(1-2):161–173, 2003.
12. E. Frentzos, K. Gratsias, and Y. Theodoridis. Index-based most similar trajectory search. In *ICDE*, pages 816–825, 2007.

13. S. Ghassempour, F. Girosi, and A. Maeder. Clustering multivariate time series using hidden markov models. *International Journal of Environmental Research and Public Health*, 11(3):2741–2763, 2014.
14. J. Gonzalez and A. Munoz. Representing functional data using support vector machines. 5197:332–339, 2008.
15. A. Hallak, D. Di-Castro, and S. Mannor. Model selection in markovian processes. In *ICML*, 2013.
16. E. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.
17. E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei, and C. Ratanamahatana. The UCR time series classification/clustering homepage: www.cs.ucr.edu/ eamonn/time_series_data/, 2011.
18. A. Kotsifakos, V. Athitsos, P. Papapetrou, J. Hollmén, and D. Gunopulos. Model-based search in large time series databases. In *PETRA*, 2011.
19. A. Kotsifakos, P. Papapetrou, J. Hollmén, and D. Gunopulos. A subsequence matching with gaps-range-tolerances framework: A query-by-humming application. *PVLDB*, 4(11):761–771, 2011.
20. A. Kotsifakos, P. Papapetrou, J. Hollmén, D. Gunopulos, and V. Athitsos. A survey of query-by-humming similarity methods. In *PETRA*, pages 5:1–5:4, 2012.
21. A. Kotsifakos, P. Papapetrou, J. Hollmén, D. Gunopulos, V. Athitsos, and G. Kollios. Huma-song: a subsequence matching with gaps-range-tolerances query-by-humming system. *PVLDB*, 5(12):1930–1933, 2012.
22. J. B. Kruskall and M. Liberman. The symmetric time warping algorithm: From continuous to discrete. In *Time Warps*. Addison-Wesley, 1983.
23. J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.
24. D. Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern recognition*, 42(9):2169–2180, 2009.
25. J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *SIGMOD workshop DMKD*, pages 2–11, 2003.
26. P.-F. Marteau. Time warp edit distance with stiffness adjustment for time series matching. *Pattern Analysis and Machine Intelligence*, 31(2):306–318, 2009.
27. T. Oates, L. Firoiu, and P. R. Cohen. Clustering time series with hidden markov models and dynamic time warping. In *In Proceedings of the IJCAI*, pages 17–21, 1999.
28. A. Pikrakis, S. Theodoridis, and D. Kamarotos. Classification of musical patterns using variable duration hidden Markov models. *Transactions on Audio, Speech, and Language Processing*, 14(5):1795–1807, 2006.
29. L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
30. H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Transactions on Acoustics, Speech and Signal Processing*, 26:43–49, 1978.
31. A. Stefan, V. Athitsos, and G. Das. The move-split-merge metric for time series. *Transactions on Knowledge and Data Engineering*, 2012.
32. M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.
33. S. B. Wang, A. Quattoni, L.-P. Morency, D. Demirdjian, and T. Darrell. Hidden conditional random fields for gesture recognition. In *CVPR*, pages 1521–1527, 2006.
34. X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. J. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Minining and Knowledge Discovery*, 26(2):275–309, 2013.
35. L. Ye and E. Keogh. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data Mining and Knowledge Discovery*, 22(1-2):149–182, 2011.