Classification of Multi-Dimensional Streaming Time Series by Weighting each Classifier's Track Record

Bing Hu, Yanping Chen, Jesin Zakaria, Liudmila Ulanova, Eamonn Keogh
Department of Computer Science and Engineering
University of California, Riverside
{bhu002, ychen053, jzaka001, lulan001,eamonn}@cs.ucr.edu

***Abstract*-Extensive research on time series classification in the last decade has produced fast and accurate algorithms for the *single*-dimensional case. However, the increasing prevalence of inexpensive sensors has reinforced the need for algorithms to handle *multi*-dimensional time series. For example, modern smartphones have at least a dozen sensors capable of producing streaming time series, and hospital-based (and increasingly, *home*-based) medical devices can produce time series streams from more than twenty sensors. The two most common ways to generalize from single to multi-dimensional data are to use *all* the streams or just the *single* best stream as determined at training time. However, as we show here, both approaches can be very brittle. Moreover, neither approach exploits the observation that different sensors may be considered "experts" on different classes. In this work, we introduce a novel framework for multi-dimensional time series classification that weights the class prediction from each time series stream. These weights are based not only on each stream's previous track record on the class it is currently predicting, but also on the distance from the unlabeled object. As we demonstrate with extensive experiments on real data, our method is more accurate than current approaches and particularly robust in the face of concept drift or sensor noise.**

Keywords - multi-dimensional time series*; *classification

I. INTRODUCTION

Many physiological, medical, and scientific processes produce copious amounts of Multi-Dimensional Time series (*MDT*) data [20][26][36][41]. If we need to classify patterns manifest on just a *single* (independent) stream from an *MDT*, there is strong evidence that the simple nearest neighbor algorithm should be the algorithm of choice [9][15][19]. However, in many cases, the *m* individual time series in the *MDT* may reflect different views of the same underlying phenomena we want to classify. For example, we may have two different leads recording an ECG (Figure 1) or several gyroscopes on a Body Area Network (*BAN*) (Figure 2). In such a case, how should we use information from multiple sensors? The obvious choices are:

- **ALL**: Use all *m* time series [36]. In this category, we include efforts that transform all *m* time series into a new space, using SVD [40] or Markov models [42], etc.
- **BEST**: Use only the *single* best time series, which is either found empirically or suggested by domain knowledge [17]. In many research efforts the latter is probably done as a matter of course and reported *fait accompli* without discussion.
- **SUB**: Use the *best* subset of the time series that is either found empirically or suggested by domain knowledge [14][20][31][34][40].

Note that while **SUB** includes **ALL** and **BEST** as special cases, the latter two choices are usually made without an effort to evaluate other possible subsets.

There are two reasons why we believe that none of the above is the ideal solution for the task at hand.

First, consider the two-lead ECG snippets shown in Figure 1 below. Here, we want to classify *myocardial ischemias* in this patient to correlate them with (independently recorded) sleep states. While the example shown in Figure 1.*left* could be classified from *either* the V_5 or V_{5R} lead, other examples are much more subtle and benefit from using *both* leads. However, suppose we use **ALL**, pooling evidence from both leads, then later on if *either* of them becomes noisy or disconnected (a very common occurrence [6][17]), we will do very poorly.

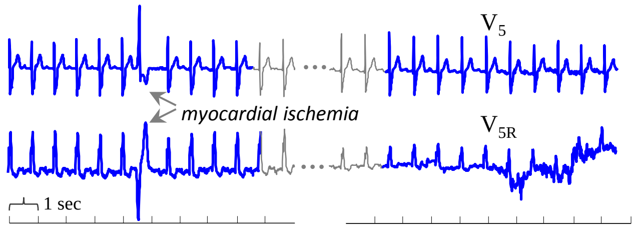

Figure 1. *left*) A snippet from a two-lead polysomnogram. *right*) At certain times, V_{5R} becomes noisy while V_5 remains almost unaffected. At other times (not shown), we see these roles reversed.

The second reason why most of the current approaches are sub-optimal is even more intuitive. The best subset of time series to use is almost always *class*-dependent. To see this, consider the *BAN* data shown in Figure 2. As we might expect, `rope-jumping` activities can be more easily classified using data from a sensor on the wrist than using data from a sensor on the shoe. Conversely, to classify `ascending-stairs` behavior, using data from a sensor on the shoe is more accurate than using data from a sensor on the wrist. This can be easily explained if we imagine how the body moves during these behaviors.

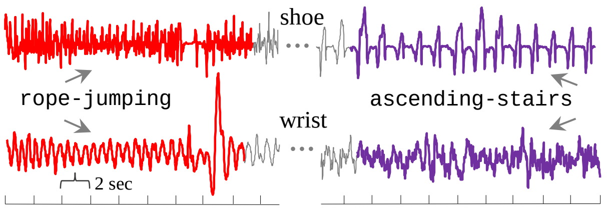

Figure 2. Two snippets of gyroscope data (110Hz) from a physical activity dataset [26]. Activities denoted `rope-jumping` (red/left) and `ascending-stairs` (purple/right) are more obvious from the wrist and shoe sensors, respectively.

In this work, we introduce a novel framework to address these two observations. At classification time, each sensor is polled for its vote on the class label. However, the vote is weighted by the sensor's self-reported confidence in its prediction. This self-reported confidence is based on two factors:

- **Confidence-based** classification: the sensor's expertise on the class in question. This element is independent of the object to be classified. The expertise simply reflects that a sensor should not be confident in predicting one class if it was mostly wrong when it predicted this class during the training phase.
- **Distance-based** classification: the similarity of the object to be classified and the examples seen during the training phase should be considered. This element reflects the fact that a sensor should not be confident in predicting *any* class if the object to be classified is significantly different than exemplars encountered during training.

As we shall demonstrate, by taking into account these two factors, we can make *MDT* classification both more accurate and more robust.

The rest of this paper is organized as follows. We first introduce the notations and intuition behind our framework in Section II. We will defer the discussion of related work in Section III, when the reader's intuition for the domain has been developed. Section IV explains how our novel voting framework works. In Section V, we provide an extensive evaluation of our ideas with several real-world datasets from diverse domains. Finally, we offer conclusions and directions for future work in Section VI.

II. NOTATION AND BACKGROUND

In this section, we describe the definitions and intuition of our framework. We begin with the basic definitions.

A. Basic Time Series Definitions

We begin with the definition of a *time series*:

Definition 1: A *time series* $T = \{t_1,t_2,...,t_n\}$ is a continuous sequence of *n* real-valued numbers.

The recent ubiquity of inexpensive sensors, for example, in smartphones or medical devices, has led to greater interest in *multi-dimensional* time series [27][36][41]. We define *multi-dimensional time series* (*MDT*) as follows:

Definition 2: A *multi-dimensional time series MDT* = $\{T_1;\ T_2;\ldots T_m\}$consists of *m* time series T_i, which are synchronously recorded streams.

For convenience in this work, we refer to each dimension in *MDT* as a *stream* or a *sensor*, where there is no ambiguity.

There is near unanimous consensus that the *nearest neighbor (NN)* classifier is the best option for time series data [9][15][19]. Thus, this is our classifier of choice. In order to use the *nearest neighbor classifier* in classification of *MDT*, we must slightly generalize from ubiquitous *single* time version [9][15][19]. We define the *nearest neighbor classifier* in the classification of *MDT* as follows:

Definition 3: The *nearest neighbor classifier* for an *MDT* is an algorithm that for each dimension $\mathtt{q}_i$ in an incoming *MDT* query $\mathtt{q} = \{\mathtt{q}_1;\mathtt{q}_2;\ldots\mathtt{q}_m\}$ finds its nearest neighbor only in the corresponding dimension T_i from the *MDT* training data $\{T_1;\ T_2;\ldots T_m\}$. The class label of $\mathtt{q}$ is determined by a combination of the nearest neighbor results for $\mathtt{q}_i$.

Hereafter, when we refer to a *classifier*, we mean a single *nearest neighbor classifier* operating on a *single* dimension in *MDT*.

As shown in Figure 3, the query $\mathtt{q}_i$ from a given dimension only finds its nearest neighbor in the respective dimension T_i in training data; the query $\mathtt{q}_i$ does not find its nearest neighbor in any other dimension T_j.

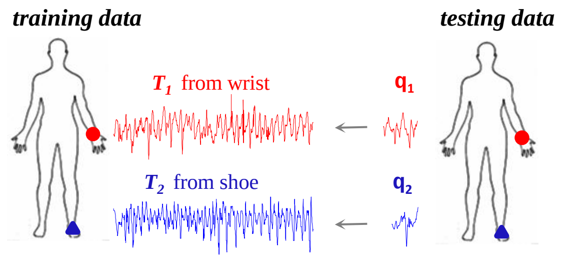

Figure 3. The red dot/blue triangle represent sensors mounted in wrist/shoe, respectively. *left*) A two dimensional time series (T_1 , T_2), T_1 from a sensor on the wrist and T_2 from a sensor on the shoe. *right*) A query $\mathtt{q}$ with two dimensions ($\mathtt{q}_1$ and $\mathtt{q}_2$), will find their nearest neighbors in T_1 and T_2 , respectively.

B. Supporting Confidence-Based Classification

As noted above, rather than using an approach that uses the **ALL**, **BEST,** or **SUB** streams of an *MDT*, we propose to evaluate and exploit the expertise of each data stream. In other words, for each time series stream in *MDT*, we have an individual nearest neighbor classifier, and a (dynamically determined) combination of classifier's predictions is used as the overall class prediction.

At query time, each classifier tells us not only *what* it predicts for the class label, but also how *confident* it is in its prediction. Our central claim in this work is that by judiciously considering these confidence-annotated predictions, we can outperform all the obvious rival methods. While similar ideas (*weighted voting* [7][13][18][45], *Bayesian classification* [7]) exist in the literature for general classification (cf. Section III), the application and adaption to the unique structure of time series data we present in this work is novel.

Our technique opens several questions, the most immediate of which is how to learn each classifiers' expertise?

The expertise of each classifier could be labeled by domain experts. For example, a clinician may know that an ECG from electrodes placed on the right of the sternum (S_5) are generally better for recognizing *atrial flutter*, whereas data from the patient's back (V_7, V_8, V_9) tends to be better for detecting *myocardial infraction* [6][12]. However, experts are expensive. Thus, we will create a framework that automatically *learns* the expertise of each classifier directly from the training data. As our framework requires that each classifier must report a score indicating how confident it is for its predicted class label, we define *confidence score* as follows:

Definition 4: A *confidence score C* with range [0, 1] is a self-reported confidence of a classifier on its prediction result. Numbers closer to 1 indicate higher confidence in prediction.

Before we demonstrate how we learn and use the confidence scores in Section IV, we show an intuitive example to demonstrate that the expertise of classifiers *does* vary. In Figure 4, we show the confidence score of classifiers learned for various human activities (more details in Section V) in a heavily cited benchmark dataset for human activity recognition [26].

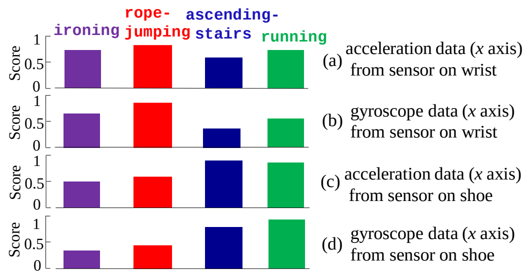

Figure 4. The performance of four classifiers (a), (b), (c), and (d) on four activities. In each classifier, the height of the bar is the confidence score for each activity.

Note that classifier (b) which tracks a sensor on the wrist has high confidence in the upper body activities `ironing` and `rope-jumping`[1], but has relatively low confidence in `ascending-stairs` and `running`, which are clearly lower body activities. Conversely, classifiers (c) and (d), which are embedded in the participant's shoes, have the opposite expertise.

If there are *p* concepts to be learned, then we must learn a *confidence score vector* $C\_vector = [C_1, C_2, \ldots C_p]$ for *each* stream in *MDT*. Each element C_i represents how confident the classifier is when predicting the i^{th} class label.

Accordingly, for an *MDT* comprised of *m* streams, our framework learns a *confidence score matrix* $C\_matrix = \{C\_vector_1;\ C\_vector_2;\ \ldots C\_vector_m\}$ with row number *m* and column number *p* for the *m* classifiers. This, in essence, is what Figure 4 illustrates.

C. Supporting Distance-Based Classification

The confidence scores in Figure 4 are learned offline in *training* phase. However, as noted in the introduction, we have an additional observation we plan to exploit, and this observation requires adjustment of confidence in the *testing* (or *deployment*) stage. Our observation is that an individual stream classifier should not be confident predicting *any* class if the object being classified is significantly different than the exemplars encountered during training. This problem was hinted at in Figure 1 and was observed in nearly all of the case studies in Section V. A common trivial reason for this occurring is that a battery dies on one sensor, and thus the time series to be classified is just a constant line. This effect is very commonly seen in medicine when one lead is unplugged or falls off the patient. Moreover, the sensor failure problem has been frequently observed in the literature. For example, a recent paper states: "*…part of the sensed data is missed due to battery failure…*" [44].

Furthermore, there are other possible reasons why the testing data might differ from the training data. If one of the concepts we learned with high confidence is `ascending-stairs`, we may find the new behaviors to be classified range from near identical time series patterns to more and more distorted patterns. This is because we may encounter data from a user that is tired, or wearing new shoes, or carrying heavy groceries, or encountering fresh snow etc. In these cases, even if the time series is still closer to `ascending-stairs` than any other class, the relevant classifier should signal a more tentative class prediction.

In Figure 5, we show a concrete example to demonstrate the importance of integrating the nearest neighbor distance with the confidence score. This is real-world data which we have slightly contrived for clarity. For simplicity, assume that there is an *MDT* with two dimensions. Further assume that at query time there is an incoming query `q` with two dimensions ($\mathtt{q}_1$ and $\mathtt{q}_2$). We want to determine the class membership of `q` using the confidence score approach.

Consider a case when we discover that among a dozen possible classes, $\mathtt{q}_1$ and $\mathtt{q}_2$ report that their nearest neighbors are different, say `running` and `rope-jumping`, respectively. (If they had agreed on a class label, then our prediction would have just been the agreed upon that label.) Given our observation about the confidence scores, we can break the tied vote by trusting the more confident classifier, which in this case was `running` with a confidence score 0.82.

However, as shown in Figure 5, this may not be the optimal decision. While $\mathtt{q}_1$ is a little closer to `running` than $\mathtt{q}_2$ is to `rope-jumping`, neither is particularly similar to its class prototypes. We simply do not have much experience with such objects. Nevertheless, if we take into account the learned distributions of nearest neighbor distances for the two classifiers, we find that the probability of being a true positive for $\mathtt{q}_2$ is much higher than $\mathtt{q}_1$. In Section IV.*C*, we formalize this visual intuition of how we adjust the prior knowledge –the confidence score − to a posterior probability by integrating the nearest neighbor distance as the new evidence. We discover that the prediction `rope-jumping` from the second classifier has a higher confidence score after this adjustment, which is the correct answer in this example.

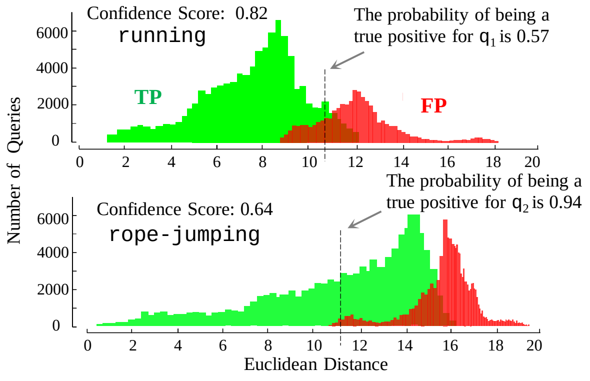

Figure 5. The distributions of nearest neighbor distances for true positives (*green*/left) and false positives (*red*/right) in the classification of activity `running` using data from wrist (*top*) and activity `rope-jumping` using data from shoe (*bottom*).

Note that the above observation only makes an overall difference in accuracy if there is variability in the

[1] We classify `rope-jumping` as *upper body* because the participant may have variable footwork, skipping on left, right or both legs; however their wrist action has very low variability.

distributions observed in each class. Empirically, we find that this is almost always the case for real-world problems. Some classes are intrinsically simple; for example there is only so much variability possible in say, `running`. However, some behaviors such as `ironing` are much more amiable to individual idiosyncrasies. Moreover, variability in equipment or clothing being ironed will also tend to produce distributions with greater means and larger standard deviations.

In summary, simply voting with the confidence score learned in the training phase may be sub-optimal, unless the testing data is *exactly* like the training data, an unlikely eventuality.

To take into account the above observation, we define the *adjusted* confidence score as follows:

Definition 5: The *adjusted* Confidence score (*adC)* with range [0, 1] is a score that subsumes the confidence score (c.f. Definition 4) by incorporating information about the distance between testing objects and the training objects as measured at query time.

In Section IV.*C*, we show how we adjust the confidence score in a principled way by combining the nearest neighbor distance at query time using Bayes theorem [7]. If the query is not similar to *any* class that the classifier learned, the *adC* for predicting the label of this query should be very low.

We use *ACV* as the abbreviation for our algorithm *Adjusted Confidence Vote*, which incorporates these observations.

As we have shown in Figure 5, the nearest neighbor distance distributions of true positives and false positives for each class play an important role in adjusting the confidence score at query time. We define *distributions of nearest neighbor distances* as:

Definition 6: The *distributions of nearest neighbor distances* (*DN*) are two distributions; one is the distribution for nearest neighbor distances of the true positives and another one is for the false positives. For each concept that a classifier learns during the training phase has two such *distributions*.

In Definition 4, we showed that our algorithm learns *C* in the training phase by evaluating the classification performances for each classifier. During this process, we also store the *distributions of nearest neighbor distances*. For each classifier, we have a vector of distributions $DN\_vector = [DN_1, DN_2, \ldots DN_p]$ with length p.

D. Allowing Real World Deployment

Recently, it has been noted that much of the literature on time series classification implicitly or explicitly makes unjustified assumptions that limit the applicability of the proposed algorithms to real-world scenarios [15][16]. These assumptions are:

Large amounts of perfectly aligned *atomic* patterns can be obtained [8][15][19]. That is to say, the algorithms assume they will only be given whole and complete heartbeats/gait cycles/atomic behaviors, with no extra spurious leading or trailing data.

The patterns are all of *equal length* [17][19][32][37]. For example, in the world's largest collection of time series datasets, the UCR classification archive, all forty-five time series datasets contain *only* equal-length data [19].

All patterns presented to the classifier will belong to one of two or more well-defined classes, that is to say, there is no possibility for the classifier to label an object as `unknown` [9][19].

These unrealistic assumptions are violated by most real-world datasets. In particular at least one assumption is violated by *all* five datasets we consider in Section V. Thus, while we know a lot about the ability of various classification paradigms on the datasets found in the UCR archive [19], based on the hundred-plus research efforts that report results on it [37], we know a lot less about how well these ideas will perform in a real-world deployment.

Thus, while it is not strictly necessary to demonstrate our novel observations, in this work we will follow the lead of [15] and introduce our framework in a way that does not make such unwarranted assumptions about the data. The next two definitions are required to remove these assumptions.

We define the *weakly-labeled training data* as follows:

Definition 7: *weakly-labeled training data* (*WT)* is a collection of the *weakly-labeled* time series annotated by behavior/state or some other mapping to the ground truth.

This is best understood by contrast to strongly-labeled training data (i.e. *all* of the UCR datasets [19]). Strongly-labeled data presents objects with explicitly labeled sections. For example, in the Kitchen Activity Dataset we consider in Section V.*E*, someone has taken the effort to annotate the precise moment that the various atomic food preparation activities begin and end. In contrast, in *WT* data, we are given data labeled like this: "*in these two minutes of data there are some examples of* `chopping`." This is clearly a more realistic and scalable way to annotate data and our efforts are made with these more assumptions in mind.

There are two important properties of *WT* that we must consider and which are illustrated in Figure 6.*a*.

First, *WT* will generally contain *extraneous/irrelevant* sections. For example, when recoding ECG data, a section of recoding is clearly *extraneous* when the machine was not plugged in, as shown by the "flatline" in Figure 6.*a*. Similar phenomena occur in all the datasets we examined. Second, *WT* will almost certainly contain significant *redundancies*. Consider Figure 6.*a* again. Once we have a single normal heartbeat, say pattern $\mathbf{N_1}$ (**N**ormal beat), then there is little utility in adding additional examples of the same type of ECG in the training data. Rather, what we should add into the training data are representative examples of other types of heartbeats, in this case, one example of the pattern **S** (**S**upraventricular Ectopic Atrial) and one example of the pattern $\mathbf{V_1}$ or $\mathbf{V_2}$ (Premature **V**entricular Contraction).

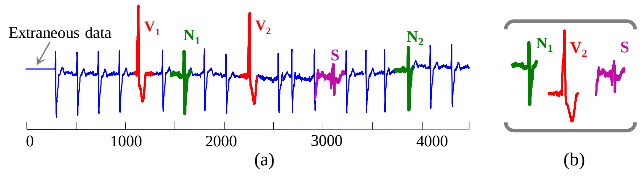

Figure 6. A snippet of BIDMC Congestive Heart Failure Database ECG, Record-03. (a) *WT*, which exhibits both extraneous and redundant data.

There are two types of anomalous heartbeats (V, S) and normal beat (N) in *WT*. (b) A minimally redundant set of representative heartbeats (a data dictionary) could be used as training data.

Rather than having the redundant data in *WT*, we desire a smaller but smarter training subset that does not have the spurious data, while still covering the target concepts. For example, Figure 6.*b* consists of just one example of **N**, **V** and **S**. We define the minimally redundant training subset as a *data dictionary*:

Definition 8: A data dictionary *D* is a (potentially very small) "smart" subset of *WT*, while covering all the *p* concepts in *WT*.

Note that in our simple example in Figure 6.*b*, it happens to be the case that one example from each of {**N**, **S**, **V**} suffices to cover the concept space. This does not have to be the case; for example, the class **V** could be polymorphic and we may need to have multiple examples of it in order to represent its variability.

While *D* could be manually built by domain experts, again we note that human domain expertise is expensive. The framework in [15] demonstrates how to build *D* automatically using a simple data editing technique, which removes data redundancy while retaining just enough examples of the concept to cover the space of its "variability."

There are two important properties of *D*. First, the classification accuracy obtained from using just *D* is generally much higher than that obtained from using all *WT* [15]. This may be a little surprising, as we generally think *more is better* when it comes to data. Recall that *D* is designed so that it does not contain spurious data. If we have voluminous spurious data, then there is a high probability that some of it will be close to an exemplar from a different class, reducing classification accuracy.

The second important property of *D* is its size. In most real-world settings, *D* is a very small fraction of *WT*, perhaps only one-hundredth its size. This allows real-time deployment on resource limited devices (embedded devices, smartphones, etc. [5]).

For an *MDT* with *m* dimensions, our framework must learn *m* data dictionaries for the respective dimension.

III. Related Work

Since the adjusted confidence score is at the heart of our contribution, we will take the time to discuss its relationship with the related work.

A. *Relationship to Ensemble Methods*

We are finally in a position to clarify the relationship between our algorithm *ACV* and the ensemble methods that it superficially resembles, for example, Boosting or Bagging [3][13][18][22][28][30][45].

In brief, our approach is different from such ensemble methods in the sense that we do *not* generate redundant classifiers that later combine for prediction [45].

The common approach in the first step of Boosting and Bagging is that they both generate multiple base classifiers in order to produce diverse "views" of the data [45]. However, we do not generate classifiers. Instead, we perform the classic *nearest neighbor classifier* on each *single* data stream as an individual classifier. For example, the most famous algorithm in Boosting is AdaBoost [45]. In order to focus more on the training examples that are "difficult" to classify, AdaBoost iteratively generates different classifiers to focus on the training examples that are incorrectly classified. However, our *ACV* framework does not generate classifiers to adapt the data.

The last step of ensemble methods performs a combination of the votes from the base classifiers [45]. Our contribution of the novel voting scheme using *adC* was informed by this combination step. In particular, using *just C* is similar to the weight in weight voting. (There is still a difference between using just *C* and the weight in weighted voting, which we clarify in Section III.*B*). However, we augment weighted voting by adjusting the weight with the similarity measure at query time.

In Section V, we show using several large datasets from diverse domains that *ACV* framework beat the most popular voting methods: majority weight and weighted voting [45].

B. *The Adjusted Confidence vs. the Weight in Weighted Voting*

In general, our voting framework is similar to weighted voting [7][13][18][29][45]. Since *adC* is an augmentation of *C*, we first clarify the difference between *C* (Definition 4) and the "weight" in weighted voting.

In weighted voting, the key decision is how to choose the weight [45]. To obtain the best performance, the general intuition is that the weights should be chosen in proportion to the performance of individual classifiers [45]. Our confidence score *C* has a similar intuition in the sense that both of them are chosen based on the performance of the classifiers in the *training* phase. However, unlike the weights in weighted voting, which are chosen based on the *overall* performance of the classifier, *C* is calculated based on the performance of the classifier for each *individual* class in the classifier.

In other words, for all the classes considered by a classifier, there is a corresponding *C* for each class. As shown in Section II.*B*, instead of having a single weight for each classifier as the weighted voting does, we have a $C\_vector = [C_1, C_2, \ldots C_p]$ for each classifier.

As *adC* is an augmentation of *C*, it can also be seen as an extension of the strategy of the weighted voting algorithm [7][13][18][45]. The modification lies in the fact that *adC* is the *posterior* probability by combining the new evidence (nearest neighbor distance at query time) and a *prior* knowledge (the confidence score) using Bayes theorem [10]. In contrast, the weighted voting only uses *prior* knowledge, in particular the past performance of the classifiers [45].

As we will show in Table I, our framework takes the predicted label with the highest posteriori probability, the highest sum of *adC*. The optimality of using the maximum posteriori estimation together with Naïve Bayes over other approaches has already been proven [1][10][43]. These optimality results require that we treat the multiple data streams as independent of each other. This may be an unrealistic assumption, but it has been shown that Naïve Bayes is surprisingly robust to violations of this assumption [10][43]. The experimental results in Section V will show that our *ACV* approach is more accurate and robust than all the rival methods.

IV. ALGORITHMS

In order to best explain our framework, we first explain how our classification model works *given* that the confidence scores of an *MDT* *C_matrix*, the distributions of nearest neighbor distances *DN_matrix*, and the data dictionaries *D_matrix* for *WT* have already been created. Later, in Section IV.*B* and C, we revisit the task of *learning* them.

A. Classification of Multi-Dimensional Time Series using the Adjusted Confidence Scores

For an incoming *m*-dimensional unlabeled object `q`, we classify each dimension with the corresponding classifier in *D_matrix* using the classic *one nearest neighbor* algorithm [19]. For each class p_j, we sum the *adC* of each classifier that assigned class p_j to query `q`. The class with the highest sum is returned as the class prediction for `q`.

In Table I, we explain the algorithm in more detail. We begin in line 1 by initializing all of the *m* *adC* from the *m* classifiers to zeros. In line 3, we calculate the nearest neighbor for each dimension of `q` with the corresponding classifier in *D_matrix*. To be clear, each dimension is considered completely independently of all others.

The function `One_NN_search` is simply the classic one nearest neighbor algorithm discussed in Definition 3 [19]. We omit details of the function `One_NN_search`, as it is well known [9][36]. Note that while the distance measure used in line 3 could be any measure [9], we only consider the Euclidean distance, as it has been shown to be an extremely competitive measure [15][19][39]. In line 4, the algorithm computes the adjusted confidence score calculated by equation (1) in Section IV.*C*.

Note that if we only use the confidence score retrieved from *C_matrix* without any adjustment, *ACV* degenerates to the weighted voting algorithm scheme [45]. (Although to our knowledge, this has never been done for *time series* before.) However, as we argued in Section II, we need to augment this confidence score with the observed nearest neighbor distances.

We take the class label that has the highest sum of adjusted confidence scores.

Table I. ADJUSTED CONFIDENCE CLASSIFICATION ALGORITHM

Input	***C_matrix***, a confidence score matrix that contains *p* columns and *m* rows(from *m* classifiers) ***DN_matrix***, distributions of nearest neighbor distances ***D_matrix***, a matrix that has *m* data dictionaries ***q***, a query with *m* dimensions
Output	***The class membership of q*** ***score***, the total confidence for the prediction
1	`adC_vector ← zeros(1,m);`
2	***for*** `i ← 1 to m`
3	`[NN_labels(i),NN_dist] ← One_NN_search(q{i},`*D_matrix*`{i});` // NN_labels is a vector with m elements for the m classifiers
4	`adC_vector(m) ← calculate_adC(NN_labels(i),NN_dist);`
5	***endfor***
6	`[class_label,score]← class_with_highest_sum(adC_vector,NN_labels);`

Having demonstrated how the classification model works in conjunction with *C_matrix* and *DN_matrix*, we are now in a position to illustrate how to *learn* them.

B. Learning the Confidence Score

We learn the confidence scores by evaluating the classification performance for each classifier during the training phase. As a byproduct of this, we also obtain *DN* for every class in all the classifiers, which we use to calculate the probability of being a true positive given the nearest neighbor distance at query time. To be concrete, for each class, we use the precision [35] of the classification as the confidence score.

In Table II, we show how we learn *C_matrix* and *DN_matrix*. Note that we randomly split the *weakly-labeled* training data into two parts. We learn the data dictionaries from one fold using the framework in [15] and treat another fold as holdout data.

We first randomly sample a large number of queries from the holdout data in line 1. From lines 2 to 10, we calculate the *C_vector* and *DN_vector* for each classifier. In line 3, we calculate the classification result for queries in the ith classifier. Then the algorithm retrieves the *DN* (i.e. `NN_true` and `NN_false`) from `NN_dists` in lines 5 to 6. Line 7 shows that the algorithm adds *DN* to *DN_matrix*. In line 8 the algorithm calculates the precision for the classification result as the confidence score for the jth class in the ith classifier.

Table II. LEARNING THE CONFIDENCE SCORE

Input	***D_matrix***, The number of classes in each *D* is *p*; ***Holdout_WT***, holdout data in the *WT*
Output	***C_matrix***, confidence score matrix contains *m* confidence vectors for the *m* classifiers; ***DN_matrix***, the distributions *NN* distances
1	`qs ← a large number of multi-dimensional queries randomly sampled from `*Holdout_WT*
2	***for*** `i ← 1 to `*m*
3	`[NN_labels,NN_dists]← One_NN_search(qs(i),MD(i));` // perform classification for the ith classifier
4	***for*** `j ← 1 to `*p*
5	`NN_true ← NN_dists for true positives in j`th` class`
6	`NN_false ← NN_dists for false positives in j`th` class` // DN is NN_true and NN_false
7	`DN_matrix(i,j) ← [NN_true,NN_false];`
8	`C_matrix(i,j)← calculate_precision(NN_true,NN_false)`
9	***endfor***
10	***endfor***

In the next section, we illustrate how we adjust the confidence score at query time by combining the nearest neighbor distance and the confidence score in a principled manner.

C. Learning the Adjusted Confidence Score

The adjusted confidence score is the confidence score augmented by integrating the nearest neighbor distance at query time.

The Bayes theorem is the optimal model to learn the adjusted confidence score [10]. This is because the adjusted confidence score is the posterior probability by combining the new evidence (nearest neighbor distance at query time) and the prior knowledge (the confidence score). We denote the following:

pl : predicted nearest neighbor label

tl: true label

dist: the nearest neighbor distance calculated at query time

The adjusted confidence score is calculated as follows:

$$P(pl = tl \mid dist) = \frac{P(dist \mid pl = tl) \times P(pl = tl)}{P(dist)}$$

$$= \frac{P(dist \mid pl = tl) \times P(pl = tl)}{P(dist \mid pl = tl) \times P(pl = tl) + P(dist \mid pl \neq tl) \times P(pl \neq tl)} \quad (1)$$

In the above equation, $P(pl = tl)$ is the confidence score that we have learned using algorithm in Table II. We can easily calculate $P(dist \mid pl = tl)$ given *DN* and the *dist* with density estimation.

V. EXPERIMENTS

We begin by stating our experimental philosophy. To ensure that our experiments are easily reproducible, we have built a website which contains all the datasets and code [46]. In addition, this webpage contains further experiments which are omitted here for brevity.

Before listing the seven straw men that we compare to, we note that in addition we have compared our approach with many other widely-used rival classification frameworks, in particular SVM, boosted decision trees and the C4.5 decision tree [26][42][45]. The best result among these is achieved by C4.5 decision tree; however it is still not competitive with results produced by our algorithm *ACV*. Thus for clarity and brevity we relegate these results to our website [46]. We do not claim this as a novel finding, the superiority of nearest neighbor methods over eager leaning methods for time series has been noted by many others [9][37].

We test on five datasets (*plus* another three we relegated to [46]), which we believe is the largest set of *MDT* datasets ever considered in a single work. In particular, more than 90% of the papers on this problem test on exactly *one* dataset [14][17][26][34] [41][44].

For the purpose of comparison, we list the seven straw men we use. Note that each straw man has been used in at least one recent paper. We begin by explaining **ALL**, **BEST**, and S**UB** in more detail.

- For **ALL**, we calculate the sum of the distance[2] between query `q` with *m* dimensions and the *m* classifiers and then find the one with the minimum distance as the nearest neighbor of `q`.
- For **BEST**, at query time, we use only the *one* classifier that has the best performance in the training phase [34].
- For **SUB**, in the training phase, we perform a heuristic greedy search over all the classifiers until the accuracy starts to decrease [14][20][31][40].

In addition to **ALL**, **BEST**, and **SUB**, there are four other obvious rival approaches in the literature that we need to compare:

- Minimum Distance Vote: choose the class label of the classifier that reported the minimum distance among the nearest distances from all the classifiers [36].
- Majority Vote: choose the most commonly predicted class label [45]. (Technically, this is a "plurality" and not a "majority," but we will use the common term).
- Random Vote: at classification time, randomly choose a classifier and take its class prediction [45].
- Weighted Vote: choose the class label with the highest sum of the weights from the classifiers that agreed on that class label. The weight is learned purely based on the past performance of the classifiers on the training data [45].

[2] We considered other variants, including summing the *squared* distances, etc. Our chosen variant was empirically the best method that used all dimensions.

A. *Physical Activity Data*

We consider a physical activity dataset containing 36 axis synchronous measurements from three Inertial Measurement Units (IMUs) located on the wrist, chest and ankle. This dataset has eight subjects performing activities such as: `ironing`, `rope-jumping`, `running`, `folding laundry`, `ascending-stairs`, etc [26]. Approximately eight hours of data at 110Hz was collected.

We performed the following experimental procedure. We randomly chose 40% of the dataset as training data, and treat the rest as testing data. A data dictionary matrix *D_matrix* that contains less than ten percent of all the training data is learned using the framework in [15].

Note that in all of our case studies, our experiment are *subject independent* evaluation, which is considered much harder than subject *dependent* evaluation [23] [26][34].

As shown in Table III, our *ACV* approach achieved a classification error rate of 0.05. In contrast, the original authors of the data reported an overall classification error rate at 0.10 [26][34]. While these two results are not exactly commensurate, the evaluation procedure in [26][34] would be expected to produce *higher* accuracy based on their split sizes. Their method extracts signal features from sliding windows and reports the *best* result after testing the feature vectors with all the popular classification algorithms using Weka [34].

Table III. Classification Results on the Physical Activity Data for *ACV* and Seven Straw Men

Algorithms	Accuracy: Original Data	Accuracy: Occluded Data
ALL [5][20]	0.19	0.16
BEST [23][26]	0.72	0.63
SUB [2][20][31]	0.78	0.64
Minimum NN dists[36]	0.59	0.58
Random[45]	0.51	0.47
Majority Vote [45]	0.84	0.76
Weighted Vote [45]	0.89	0.77
Adjusted Confidence Vote	0.95	0.94

Moreover, Table III shows that the *ACV* method beats all seven straw men by a significant margin.

Recall that the strongest motivation for our ideas is to produce a framework that is robust for missing (or "occluded") data. Our claim is that such missing data is very common, but researchers often "clean" datasets before publicly releasing them. This is a noble idea, but one that perhaps shields the community from the realities of real-world deployments. Indeed, authors have been critiqued for releasing less than idealized data; For example, authors in [44] criticize the UC-Berkeley WARD Dataset [41] by noting "*part of the sensed data is missed due to battery failure*".

While we have seen multiple *real* examples of occluded data, to allow systematic testing we must *synthetically* occlude data. Let us revisit this widely studied dataset [26] as an example (Later datasets had a similar treatment.) There are 36 data streams, arranged in 12 triplets. For example, there are *x*, *y*, *z* axes for the acceleration data, and *roll*, *pitch*, and *yaw* for gyroscope data. We perform our occlusion experiments by simulating sensor failure of one triplet at a time. For each of the three streams, in a randomly chosen triple, we toss a fair coin to decide if we should replace it with either a straight line or a sine wave. We report the average performance by testing all the 12 cases. In Table III, *rightmost* column, we show the classification result for these data occlusion experiments.

As we can observe, *ACV* also achieves the highest accuracy for occluded data. Among the seven straw men, the Majority Vote and Weighted Vote methods return competitive results in using *original* data. However, when it comes to data with occlusion, the performance of these two algorithms drops precipitously. This is because data in the testing phase is different from data used in the training phase. While only one tenth of the data is different (by definition), this is enough to make a drastic difference in their performance.

In contrast, our *ACV* approach is relatively robust for data with occlusion, since *ACV* carefully factors in the nearest neighbor distances in the testing phase.

Given the relatively poor performance of the five straw men on both the original and occluded data (shown in gray in Table III), we omitted the results for these approaches in the rest of this work. Instead, we put the results of a complete comparison on the supporting webpage [46].

B. *Avian Audio Data*

Audio classification typically begins by extracting acoustic features such as Mel-Frequency Cepstral Coefficients (MFCCs) from audio signals [4][25]. MFCCs represent the speech amplitude spectrum in a compact form by transforming the audio data into thirteen coefficients[3].

In most algorithms that use the MFCCs for speech recognition, researchers either use one coefficient or use all the coefficients [4][25][38]. As noted above, it is our claim that *both* these choices may result in poor performance. To see this, we consider two species, *East Brazilian Pygmy Owl* (*Glaucidium minutissimum*) and *Common Potoo* (*Nyctibius griseus*) as examples. As shown in Figure 7, it is clear that for the *Owl*, the patterns (***green***/bold) exhibited in the third and fourth coefficients (*red*) are much clearer than the ones in the second and fifth coefficients. While for *Potoo*, the patterns in the third and fourth coefficients seem random.

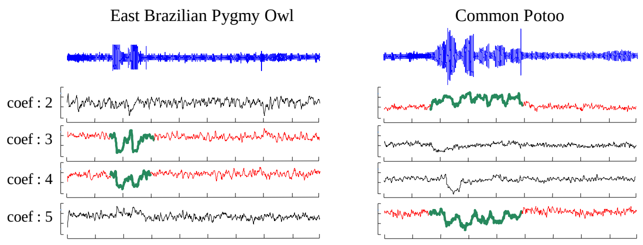

Figure 7. *left*) A snippet of sound spectrum and MFCCs from 2 to 5 for the *East Brazilian Pygmy Owl*. *right*) A snippet of sound spectrum and MFCCs from 2 to 5 for the *Common Potoo*.

Clearly, it is not a trivial task to automatically identify which coefficients are most useful for which species, even for experienced avian bio-acousticians. Moreover, even within a single species, the bird calls in the testing phase may be subtly different from in the training phase, as the inevitable background noise may affect different coefficients in different ways.

Thus, we see this domain as an ideal candidate for our ideas and treat the twelve coefficients as an *MDT*.

Xeno-canto is a large data pool of bird sound files with the aim of sharing bird sounds. Avian audio files are uploaded by volunteers from all over the world [38]. We randomly chose four hours of audio data from four species of birds to perform a classification experiment. The four species are *East Brazilian Pygmy Owl*, *Common Potoo*, *Dusky Capped Flycatcher* (*Myiarchus tuberculi*fer), and *Acadian Flycatcher* (*Empidonax virescens*). We have placed the original audio files and the extracted MFCCs time series on the supporting webpage [46].

In the bird sound datasets, we do not need to explicitly perform experiment with occlusion because of the natural variability of the bird sounds, recorded—in some cases—years and hundreds of miles apart [38].

Our dataset consists of approximately eighteen hundred bird calls. We randomly chose 40% of the data as training data and treated the rest as testing data. The classification accuracy using our *ACV* approach is 0.87, while for Majority Vote and Weighted Vote, the classification accuracy is 0.66 and 0.79, respectively.

C. *Recognition of Cricket Umpire Signals*

Cricket is a very popular game in British Commonwealth countries. An umpire signals different events in the game to a distant scorer/book-keeper. The signals are communicated with motions of the hands. For example, `No Ball` is signaled by touching each shoulder with the opposite hand. A complete list of signals can be found in [24].

The dataset in [21] consists of four umpires performing twelve signals. There are four umpires performing each signal ten times. The data with frequency 184Hz was collected by placing two accelerometers on the wrists of the umpires. Each accelerometer has three synchronous measures for three axes (*x*, *y* and *z*). Thus, we have a six dimension *MDT* from the two accelerometers. Figure 8 shows the data for two example signals, `Six` and `Leg Bye`. To signal `Six`, the umpire raises both hands above his head. `Leg Bye` is signaled with a hand touching the umpire's raised knee three times.

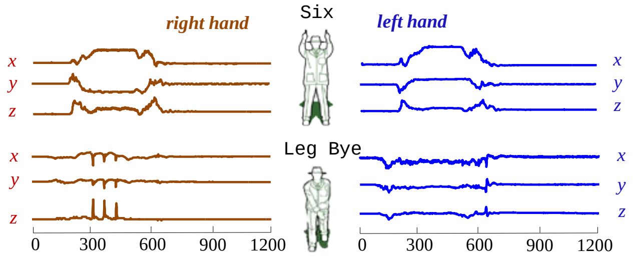

Figure 8. *x*, *y*, *z* acceleration data from right hand (*brown*) and left hand (*blue*) for two signals `Six` and `Leg Bye`.

We randomly chose 40% of the data as the training data and treated the rest as testing data. The classification results are shown in Table IV. As noted in Section V. *A*, since the Majority Vote and Weighted Vote methods return the most competitive results among the seven straw men, we only list the results using these two straw men due to space

[3] Usually the top thirteen coefficients are used for audio analysis. The first coefficient is a normalized energy parameter, which is not used for speech recognition [25].

limitations. However, we reiterate that we have put the full result on the supporting webpage [46].

To produce *real-world* occluded data, we had two experienced officials perform the twelve cricket signals under the same experimental conditions as in [21]. By contriving a battery failure, we arranged that one subject had a sensor failure on the left hand and the other subject had a sensor failure on the right hand. We added this data to the original data in [21].

As we can see in the result for occluded data in the rightmost column of Table IV, our *ACV* approach is significantly more robust to sensor failure than Majority Vote or Weighted Vote. Moreover, for original data [21], *ACV* once again achieves the highest accuracy.

Table IV. Classification Results on the Cricket Data

Algorithms	Accuracy Original Data	Accuracy Occluded Data
Majority Vote [45]	0.88	0.71
Weighted Vote [45]	0.92	0.78
Adjusted Confidence Vote	0.96	0.93

D. Gesture Recognition

Almost all modern smartphones are equipped with multiple sensors (i.e. acceleration sensors, gyroscopes, etc.). This has inspired dozens of research efforts on creating gesture recognizers for mobile devices [23].

The dataset provided in [23] is rapidly becoming a benchmark in this domain. The data was created by fifteen subjects wearing iPhones on their wrists to create six hand gestures as shown in Figure 9. Each participant provided each gesture fifteen times. There are six dimensions comprised of 3-axis acceleration data and 3-axis gyroscope data recorded at a frequency of 80Hz.

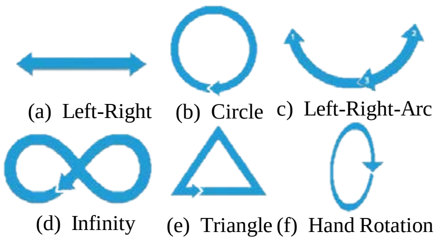

Figure 9. Visualization of the six gesture classes. This figure from [23] is used with permission.

We randomly chose 40% of the data as the training data and treated the rest as the testing data. Using the same method discussed in Section V.*A*, we randomly choose half of the testing subjects for the occluded data experiment. The comparison in Table V shows that our *ACV* approach obtains the highest accuracy in both cases of using the original data and data with occlusion.

Table V. Classification Results on the Gesture Data

Algorithms	Accuracy Original Data	Accuracy Occluded Data
Majority Vote [45]	0.86	0.67
Weighted Vote [45]	0.89	0.74
Adjusted Confidence Vote	0.97	0.93

E. Kitchen Activity Data

A recent European effort in assisting elderly people to live more independently [33] has investigated technology to support activities in the kitchen, including automatic guidance while cooking and cleaning. Sensors embedded into kitchen utensils provide continuous data streams while being used. This provides an ideal test bed to demonstrate our framework. The first major dataset released [11][33] has four Wii-remote instrumented utensils to collect acceleration data, as shown in Figure 10. Twenty subjects performed seven hours of a recipe for a mixed salad preparation [11]. There are eleven classes, including `peeling`, `slicing`, `scraping`, `chopping`, etc. The data was recorded at a frequency of 40Hz.

Figure 10. *a*) Modified Wii Remotes embedded in specially designed utensils. *b*) A subject is preparing salad. This figure is used with permission from [33].

Since in this dataset there are only three axes, we cannot use the same method with occluded data. Instead, we perform our occlusion experiments by simulating sensor failure of one axis at a time. By randomly choosing 40% data as training data and the rest as testing, we obtain the classification result as shown in Table VI. Once again, our *ACV* approach obtains the highest accuracy in both cases.

Table VI. Classification Results on the Kitchen Data

Algorithms	Accuracy Original Data	Accuracy Occluded Data
Majority Vote [45]	0.74	0.54
Weighted Vote [45]	0.84	0.76
Adjusted Confidence Vote	0.92	0.88

F. Robustness to Irrelevant Features

To demonstrate the robustness of our approach, we repeated the experiments above with an interesting modification. We added time series with no relation to the class into the data.

Let us consider the cricket dataset as an example. Originally, the dataset is an *MDT* with six dimensions. However, we added another six dimensions of random walk data to the original data. To be clear, none of the explicit algorithms "know" which, if any, dimensions are irrelevant.

We repeated the experiment shown in Table IV with the modified dataset. Both the Majority Vote and Weighted Vote are quite brittle to the additional irrelevant data, as their classification accuracies drop steeply to 0.69 and 0.78, respectively. However, our *ACV* approach obtains an accuracy of 0.95, barely affected by the irrelevant features. This is very important advantage when exploiting new domains in which we may have poor intuitions as to *which* features are useful.

VI. Conclusion and Future Work

Building on the general techniques of weighted voting [7][45] and Bayesian classification [7], and extending the techniques of "realistic assumptions" dictionary-based classification [15], we have introduced a novel voting framework for accurate classification of multi-dimensional time series. We demonstrated on several large, real-world datasets from diverse domains that our approach is significantly more accurate and robust than rival approaches. In particular, we have shown that our framework is very robust to missing data and irrelevant, a problem that frequently occurs in the real world [41]. In future work, we plan to investigate the theoretical foundations of our observations and implement the minor extensions needed to support Dynamic Time Warping [37] and Uniform Scaling [15]. Finally, we have given away *all* code and data to allow others to confirm, extend and use our work [46].

ACKNOWLEDGMENTS

We would like to acknowledge the financial support for our research provided by NSF IIS-1161997II. We thank all the donors of datasets. We especially thank Denisa Duma for fruitful discussion.

REFERENCES

[1] J. Aldrich, R.A. Fisher and the making of maximum likelihood 1912-1922, *Statistical Science*, 12(3), 1922.

[2] P. Bartlett, Y. Freund, W. Lee and R. Schapire, Boosting the Margin: A New Explanation for the Effective of Voting methods, *The Annals of Statistics*, vol(26),1998.

[3] E. Bauer and R. Kohavi, An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting and Variants, *Journal of Machine Learning*, 1998.

[4] F. Briggs, R. Raich and X. Fern, Audio Classification of Bird Species: a Statistical Manifold Approach, *ICDM*, 2009.

[5] L. Bao and S.S. Intille, Activity Recognition from User-Annotated Acceleration Data, *2nd International Conference on Pervasive Computing*, 2004.

[6] E. Braunwald, Heart Disease: A Textbook of Cardiovascular Medicine, Ninth Edition, 2011.

[7] C.M. Bishop, M. Svensén, Bayesian Hierarchical Mixtures of Experts, *Procs of 19th Conference on Uncertainty in Artificial Intelligence*, 2003.

[8] Y. Chen, B. Hu, E. Keogh and G. E.A.P.A Batista, DTW-D, Time Series Semi-Supervised Learning from a Single Example, *KDD*, 2013

[9] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang and E. Keogh, Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures, *PVLDB* 1(2): 1542-1552 , 2008.

[10] P. Domingos and M. Pazzani, Beyond Independence: Condition for the Optimality of thhe Simple Bayesian Classifier, *Machine Learning*, vol(29),p(103-137), 1997.

[11] Digital Interaction at Culture Lab, *di.ncl.ac.uk/publicweb/AmbientKitchen/*, accessed on Jan 9, 2013.

[12] Electrocardiography, *en.wikipedia.org/wiki/Electrocardiography*

[13] Y. Freund and R. Schapire, A Short Introduction to Boosting, *Journal of Japanese Society for Artificial Intelligence*, vol(14), 1999.

[14] S. Günnemann, I. Färber, K. Virochsiri, and T. Seidl, Subspace Correlation Clustering: Finding Locally Correlated Dimensions in Subspace Projections of the Data, *KDD*, 2012.

[15] B. Hu, Y. Chen and E. Keogh, Time Series Classification under More Realistic Assumptions, *SDM*, 2013.

[16] B. Hu, T. Rakthanmanon, Y. Hao, S. Evans, S. Lonardi, E. Keogh, Discovering the Intrinsic Cardinality and Dimensionality of Time Series using MDL, *ICDM*, 2011

[17] Y. Hu, S. Palreddy and W. Tompkins, A Patient-Adaptable ECG Beat Classifier using a Mixture of Experts Approach, *IEEE Transactions on Biomedical Engineering*, vol(44),2007.

[18] M. Jordan and R. Jacobs, Hierarchical Mixtures of Experts and the EM Algorithm, *A.I.Memo* No.1440, *C.B.C.L.Memo*. No.83, 1993.

[19] E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei and C.A. Ratanamahatana. The UCR Time Series Classification/Clustering Homepage: *www.cs..ucr.edu/~eamonn/time_series_data/*, 2006.

[20] H. Kremer, S. Günnemann, A. Held and T. Seidl, Mining of Temporal Coherent Subspace Clusters in Multivariate Time Series Databases, *PAKDD*, 2012.

[21] M.H. Ko, G. West, S.Venkatesh and M. Kumar. Online context recognition in multisensory system using dynamic time warping. *In Intelligent Sensors, Sensor Networks and Information Processing Conference*, 2005.

[22] J. Kolter and M. Maloof, Dynamic Weighted Majority: A New Ensemble Method for Tracking Concept Drift, *ICDM*, 2003.

[23] S. Kratz and M. Rohs, A $3 Gesture Recognizer – Simple Gesture Recognition for Devices Equipped with 3D Acceleration Sensors, *IUI*, 2010.

[24] Lord's, The Home of Cricket, *www.lords.org/laws-and-spirit/laws-of-cricket/laws/*, accessed on Feb 5th, 2013.

[25] P. Mermelstein, Distance measures for speech recognition, psychological and instrumental, *Pattern Recognition and Artificial Intelligence*, 1976.

[26] PAMAP, Physical Activity Monitoring for Aging People, *www.pamap.org/demo.html*, retrieved 2012-05-12

[27] J. Pärkkä, M. Ermes, P. Korpipää, J. Mäntyjärvi, J. Peltola, and I. K Korhonen, Activity classification using realistic data from wearable sensors, *IEEE Trans. Inf. Tech. Biomed*., vol. 10, pp. 119-28, 2006.

[28] R.E. Schapire, and Y. Singer, Improved Boosting Algorithms using Confidence-rated Predictions, *Journal of Machine Learning*, 1999.

[29] W. Street and Y. Kim, A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification, *KDD*, 2001.

[30] D. Optiz and R. Maclin, Popular Ensemble Methods: An Empirical Study, *Journal of Artificial Intelligence Research, vol(11)*,1999.

[31] D. Optitz, Feature Selection for Ensembles, *AAAI, 1999*

[32] M. Radovanović,A. Nanopoulos and M. Ivanović, Time Series Classification in Many Intrinsic Dimensions, *SDM*, 2010.

[33] C. Pham and P. Olivier, Slice & Dice: Recognizing Food Preparation Activities using Embedded Accelerometers, *Procs of the European Conference on Ambient Intelligence*, 2009.

[34] A. Reiss and D. Stricker, Introducing a Modular Activity Monitoring System, *33rd IEEE EMBS*，2011.

[35] C.J. van Rijsbergen, Information Retrieval, London, GV, 2nd Edition, 1979, ISBN 0-408-70929-4

[36] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos and E. Keogh, Indexing Multi-Dimensional Time Series with Support for Multiple Distance Measures, *KDD*, 2003.

[37] X. Xi, E. Keogh, C. Shelton, L. Wei and C. Ratanamahatana, Fast Time Series Classification Using Numerosity Reduction, *ICML*, 2006.

[38] Xeno-canto, Sharing Bird Sounds from around the World, *www.xeno-canto.org/*, accessed on Feb 6, 2013.

[39] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, E. Keogh, Experimental comparison of representation methods and distance measures for time series data. *DMKD*, vol26(2), 2013.

[40] H. Yoon, K. Yang, C. Shahabi, Feature Subset Selection and Feature Ranking for Multivariate Time Series, *IEEE Trans. Knowl. Data Eng.* 17(9): 1186-1198 , 2005.

[41] A.Yang, A. Giani, R. Giannatonio, K. Gilani, Distributed Human Action Recognition via Wearable Motion Sensor Networks, *Journal of Ambient Intelligence and Smart Environments*, 2009.

[42] J.Yin and Q. Yang, Integrating Hidden Markov Models and Spectral Analysis for Sensory Time Series Clustering, *ICDM*, 2005.

[43] H. Zhang, The Optimality of Naïve Bayes, *AAAI, FLAIRS Conference*, 2004.

[44] M. Zhang and A.A. Sawchuk, *USC-HAD:* A Daily Activity Dataset for Ubiquitous Activity Recognition Using Wearable Sensors, *UbiComp*, 2012.

[45] Z. Zhou, Ensemble Methods: Foundations and Algorithm, Chapman and Hall/CRC, 1st edition, 2012.

[46] Project webpage: sites.google.com/site/mtdtsadc/