Query Suggestion to allow Intuitive Interactive Search in Multidimensional Time Series

Yifei Ding
Department of Computer Science
University of California, Riverside
yding007@ucr.edu

Eamonn Keogh
Department of Computer Science
University of California, Riverside
eamonn@cs.ucr.edu

ABSTRACT

In recent years, the research community, inspired by its success in dealing with single-dimensional time series, has turned its attention to dealing with multidimensional time series. There are now a plethora of techniques for indexing, classification, and clustering of multidimensional time series. However, we argue that the difficulty of *exploratory* search in large multidimensional time series remains underappreciated. In essence, the problem reduces to the "chicken-and-egg" paradox that it is difficult to produce a meaningful query without knowing the best subset of dimensions to use, but finding the best subset of dimensions is *itself* query dependent. In this work we propose a solution to this problem. We introduce an algorithm that runs in the background, observing the user's search interactions. When appropriate, our algorithm suggests to the user a dimension that could be added or deleted to improve the user's satisfaction with the query. These query dependent suggestions may be useful to the user, even if she does not act on them (by reissuing the query), as they can hint at unexpected relationships or redundancies between the dimensions of the data. We evaluate our algorithm on several real-world datasets in medical, human activity, and industrial domains, showing that it produces subjectively sensible and objectively superior results.

CCS CONCEPTS

• **Information systems → Query suggestion**

KEYWORDS

Query Suggestion, Multidimensional Time Series

ACM Reference format:

Y. Ding and E. Keogh, 2017. Query Suggestion to allow Intuitive Interactive Search in Multidimensional Time Series. In *Proceedings of SSDBM, Chicago, Illinois, USA, June 2017 (SSDBM'17)*, 11 pages.
DOI: 10.1145/123 4

SSDBM '17, June 27-29, 2017, Chicago, IL, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5282-6/17/06…$15.00
http://dx.doi.org/10.1145/3085504.3085522

1 INTRODUCTION

The time series data mining/information retrieval research community has spent the last two decades addressing single-dimensional time series with considerable success [22]. For example, creating representations and algorithms that allow us to answer nearest-neighbor queries in billion-plus object datasets in real time [2], and producing classifiers for data-center chillers that allow "*$40,000 in annual savings*" by recognizing time series patterns that are indicative of undesirable and preventable behaviors [16]. More recently the community has begun to address multidimensional time series (**MTS**) [8]. Recent works have proposed techniques for indexing, classification and clustering of multidimensional time series [1][20][24].

However, we argue that the critical task of *interactive* and *exploratory* search in large multidimensional time series remains to be addressed. Here, the problem is the "chicken-and-egg" paradox that it is difficult to produce a meaningful query without knowing which subset of dimensions to use, but that finding a good subset of dimensions is itself not only *data* dependent, but even *query* dependent. In this work we offer a solution to mitigate this problem. In particular, we propose a framework that supports two interactive search operators:

- **Query Contraction**: Here the system suggests one or more dimensions that can be dropped from the query to make it more *meaningful*.
- **Query Expansion**: Here the system suggests one or more dimensions that can be *meaningfully* added to the query, without negatively affecting the result. In essence, here we are informing the user that a (perhaps unexpected) dimension is related to the original query.

For clarity, we will illustrate these ideas and better qualify our use of "meaningful" by their (loose) analogues in the more familiar text retrieval domain [17].

If we searched for the words **`machine learning culicidae`**, Google's top-ten results includes examples annotated as "**`Missing: ~~culicidae~~`**"[1]. Here the search engine uses some algorithm to predict that we would not be as satisfied with the single result returned by the conjunction of these three terms, and further predicts that dropping the most

[1] These examples were created just before this paper was submitted, and are likely to change over time, for example when *this* paper is indexed.

obscure term will improve the user satisfaction. This is a similar sense to our *query contraction*; dropping one or more dimensions from a multidimensional time series query may greatly improve user satisfaction.

Likewise, if we searched for **`machine learning bug`**, Google's search results include some results which have the word "`insect`" highlighted. Here the search engine inferred that the original query might benefit from being augmented with a related word. This is a similar sense to our *query expansion*, adding one or more dimension from a multidimensional time series query may allow the user to learn that, *given* a certain query, two dimensions are related.

This last point is worth restating. It may be that two words are only related *given* a particular query context. For example, is the word `cat` related to the word `jaguar`? *Given* the query **`jaguar zoo`**, almost certainly, but *given* the query **`jaguar hybrid-electric`**, almost certainly not. The time series analogue of this observation, that dimensions may or may not be related depending on the context, has only recently received attention in the supervised case [8]. To our knowledge, it has not been considered in the unsupervised case we consider here.

We implemented our algorithm as an unobtrusive agent that can be added to any existing **MTS** query system [5][6], running in the background. On many queries it may do nothing (an implicit third option missing from the bulleted list above), but on some queries it can pop an open window that offers the suggested action, such as "*You may wish to consider dropping dimension 12 (patient-temperature) from your query*".

The remainder of this paper is organized as follows. In Section 2, we demonstrate the importance of dimension choice in time series; then go on to review related work in Section 3. In Section 4, we provide a background of definitions before explaining our methodology in-depth in Section 5. Section 6 presents and discusses experimental results. We offer conclusions in Section 7.

2 IMPORTANCE OF DIMENSION CHOICE

Before proceeding, we will take the time to develop the readers' intuition for the importance of dimension choice in time series domains. While our ultimate goal is to aid in exploratory search in *unlabeled* data, here we consider a *labeled* data for which we can provide objective answers.

The PAMAP [14] dataset comprises of eight subjects performing activities such as *standing*, *sitting*, *walking-very-slowly*, *normal-walking*, *Nordic-walking*, *running*, *playing-soccer*, and *ascending/descending stairs*. Each subject wore a heart rate monitor and inertial measurement unit (IMU) on their hand, chest, and shoe. Each IMU consists of a 3D-accelerometer (acc), 3D-gyroscope, and 3D-magnetometer.

Suppose we query the database with the top-twenty nearest neighbor query of *normal-walking* shown in Figure 1.*top*. When we issue the query using the two dimensions {hand.acc.x, shoe.acc.x}, the precision of the query is 1.0. However, if we had performed the query with just one of the two dimensions, we would have achieved a precision of 0.5 and 0.8, respectively, demonstrating the utility of **MTS** search.

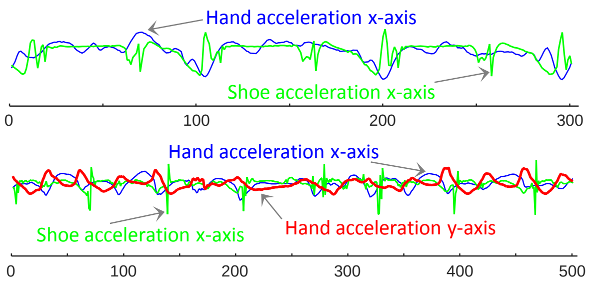

Figure 1: *top*) A query of a person performing normal-walking where utilizing both dimensions shown gives us perfect precision, but not when just utilizing either one. *bottom*) When we switch the query to a person playing soccer, a different set of dimensions gives us perfect precision, namely {hand.acc.x, hand.acc.y}.

This finding is intuitive as there is a correlation in the swinging of the arms and the legs when the subject is walking with normal gait (evolved to reduce the angular momentum of the body, balancing the rotational motion produced during walking [3]). Thus querying with two dimensions which complement each other helps us to obtain perfect precision.

Note that if we perform a top-twenty nearest neighbor query of a subject playing soccer shown in Figure 1.*bottom*, again using the same two dimensions as in Figure 1.*top*, the precision is just 0.9. But had we somehow known to use {hand.acc.x, hand.acc.y}, we could have achieved a precision of 1.0. This is because there is less correlation in the arm and the leg movements when a person is playing soccer compared to when a person is walking. So, issuing the query using two dimensions of the hand, which are more correlated during the soccer activity, gives us the higher precision. This tells us that the best set of dimensions to use is *query dependent*, an observation more forcedly made in [8].

Next, consider the query of *normal-walking* shown in Figure 2.*top*. Here we initially conduct a top-twenty nearest neighbor search query using {hand.acc.x, shoe.acc.x, shoe.acc.z}. The precision was 0.7. However, if we simply dropped {shoe.acc.z}, then the precision rises to 1.0. Note that dropping either of the other two dimensions *hurts* the precision. Had we dropped {hand.acc.x} or {shoe.acc.x}, the precision drops to 0.6 or 0.65 respectively.

The naive intuition that *any* information from the shoe must help for a *walking* query leads us astray here. The z-axis is recording motion orthogonal to the direction of travel, containing no exploitable information, but a lot of "noise". Therefore, when we include this "extraneous" dimension in our query search, the precision drops significantly. In the presence of labeled data we could use cross validation to determine which dimension (if any) could be dropped to improve our satisfaction with the query. How can we do this with unlabeled data? This is the question we address with our *query contraction* operator.

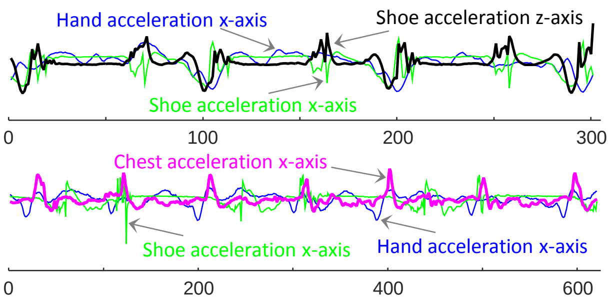

Figure 2: *top*) A query of a person performing normal-walking where utilizing all three dimensions does not give us perfect precision. However, using only {hand.acc.x, shoe.acc.x} does give perfect precision. This suggests the usefulness of query contraction. *bottom*) A query of a person performing walking-very-slowly where utilizing {hand.acc.x, shoe.acc.x} gives us perfect precision, and furthermore adding a related {chest.acc.x} maintains perfect precision, suggesting the usefulness of query expansion.

In a similar vein, consider the top-hundred nearest neighbor query of a subject performing *walking-very-slowly* shown in Figure 2.*bottom*. Here using the two dimensions {hand.acc.x, shoe.acc.x} gives us a precision of 1.0. However, for this query, if we add {chest.acc.x} we *still* get precision of 1.0. Moreover, we do *not* maintain perfect precision if we add any other unrelated dimensions, such as the heart beats per minute or chest, hand, or shoe temperature. At first blush it may be a little surprising that the chest acceleration provides useful information about gait, but as [19] notes "*empirical observation suggests that the (chest) plays an important dynamic role in balance control and gait stability*" [9][19]. It is important to note that this feature set {hand.acc.x, shoe.acc.x, chest.acc.x} is not good on *average*, for all queries, just for this one particular query.

Before moving on, we note that this exercise also suggests an objective technique to evaluate our ideas. On datasets for which we *do* have class labels, we can "hide" the class labels from our query mechanism, and do a post-hoc analysis of the precision before and after our query suggestions. We will revisit this objective technique in more detail in Section 6.1.

3 RELATED WORK

Our contributions span two related areas in the time series domain, *dimensionality reduction* and *multidimensional time series retrieval.*

There has been extensive work on dimensionality reduction for the purpose of mitigating the "curse of dimensionality" problem [11][20][22]. There are many techniques for the dimensionality reduction of interest. One of these is Singular Value Decomposition (SVD) [11], which converts the data such that the new *derived* dimensions are ordered from highest to lowest variance. As a result, the characteristics of the dataset is expressed in a reduced number of dimensions.

Our work does not apply any transformations or reductions directly to the dataset because our envisioned visual search task requires we maintain the original intuitive meaning of each dimension (e.g., acceleration of the x-axis in the hand).

Another technique of dimensionality reduction is feature subset selection [24]. Here, the purpose is to find and remove some irrelevant and redundant dimensions in order to obtain a higher accuracy in various tasks such as classification, clustering, nearest neighbor search, or querying the data. However, (with rare exceptions) the features are selected *globally*. That is, once the best subset of features is chosen, it is applied globally to the task-at-hand. In contrast, the main contribution of our work is emphasizing the need for *locality* when selecting the best subset of dimensions. That is, we show that the best subset of dimensions to produce meaningful results is dependent on a case-by-case basis for each particular **MTS** query in execution. There may be no intrinsically irrelevant or redundant dimensions, just dimensions which are irrelevant and redundant *given* a particular query.

A related technique with an emphasis on locality is *subspace clustering* in the time series domain [1], where the goal is to discover clusters in various subsets of dimensions generally using methods of top-down or bottom-up searching [15]. These clusters would not have been discovered had we performed clustering on *all* of the dimensions.

Another related area is querying of time series data, also known as *query-by-content*, or *similarity search* [22]. The single-dimensional problem has been of great interest over the past two decades due to its general wide applicability and usage [4][11][22], and in the past decade there has been growing interest in multidimensional case, engendered by the increasing prevalence of multidimensional time series produced by personal electronic devices and the Internet of Things (IoT).

Many research efforts focus on efficient indexing techniques [20]. In our work, we do not address efficiency directly, although we do use the efficient Euclidean Distance metric which is amiable to indexing [4][22].

4 BACKGROUND

In this section, we introduce the necessary definitions and terminology, beginning with the definition of a *time series*:

Definition 1: A time series **T** = $t_1, t_2, t_3, \ldots, t_n$ is a continuous ordered sequence of real values in equally spaced time intervals of length *n*.

With the emergence of low cost sensors in a wide-spectrum of areas such as personal electronic devices, household appliances, medical devices, and industrial equipment, there has been a surge of *multidimensional time series* data from which we can extract useful information.

Definition 2: A multidimensional time series **MTS** consists of m number of time series **T** of length n, where $m \geq 2$.
$\mathbf{MTS} = \{\mathbf{T_1}, \mathbf{T_2}, \mathbf{T_3}, \ldots, \mathbf{T_m}\}$,
where $\mathbf{T_1} = t_{11}, t_{12}, t_{13}, \ldots, t_{1n}$
$\mathbf{T_2} = t_{21}, t_{22}, t_{23}, \ldots, t_{2n}$
…
$\mathbf{T_m} = t_{m1}, t_{m2}, t_{m3}, \ldots, t_{mn}$
t_{ji} and t_{ki} are two points which occur simultaneously, and $j \neq k$.

The **MTS** datasets we focus on contain *exemplars*, ***E***.

Definition 3: An exemplar ***E*** is one time series data instance in an *m*-dimensional **MTS** dataset, and $\boldsymbol{E} = \{ e_1, e_2, \ldots, e_m\}$ where each e_i is a time series data corresponding to a dimension d_i, and each e_i is of the same length in the entire dataset.

One obvious way to glean insights from such data is to perform *query-by-content*, searching for a *query q*.

Definition 4: A single-dimension query q is an e_i from ***E*** corresponding to dimension d_i from a **MTS** dataset.

Our queries are taken from the data itself (*here is an interesting pattern on May 1st, let me see if it happened at another time*) commonly known as *query-by-example* [6], but a query may also be created by the user from first principle knowledge of the domain (*here is what I think the data would look like if there was a flaw in the condenser, let us see if this has ever occurred*), commonly known as *query-by-sketching* [7]. For simplicity we consider only on the *query-by-example* case, and all our queries originated from **MTS** datasets.

We can extend the previous definition to the **MTS** Q case.

Definition 5: A multidimensional query is $Q = \{q_1, q_2, \ldots, qk\}$ of an *m*-dimensional **MTS** where $m \geq k \geq 2$. Each q_i is from a different dimension in **MTS**, and Q is a subset of an *m*-dimensional ***E*** from a **MTS** dataset.

Note that a multidimensional Q need not and (generally) should not query all the available dimensions in a time series dataset as there may be many dimensions available, but only a subset is "meaningful" to the task-at-hand. For supervised tasks this subset can be learned [8], however it usually is not obvious which subset to use in the unsupervised case. This is in fact the essence of the problem this paper addresses.

An effective and efficient way to measure differences between time series is the ubiquitous *Euclidean Distance Metric* [4].

Definition 6: To obtain the top-K Nearest Neighbors (NN) of a single-dimensional query q (Def. 4), we first calculate the ED between q and every other time series data e_i in q's respective dimension, resulting in a list of *distances* = [ED(q, e_{i1}), ED(q, e_{i2}), … , ED(q, e_{is})], where s is the number of other time series exemplars. We sort *distances* in ascending order to retrieve q's top-K NN exemplars. This process is referred to as the *execution of a single-dimensional query q*.

Note that the execution of query q only involves searching for q's NNs in its own respective dimension, and no other dimensions. We can extend to the top-K NN of a multidimensional query Q. Similarly, we refer to this process as the *execution of a multidimensional query Q*.

Definition 7: To obtain the top-K NN of a multidimensional query $Q = \{q_1, q_2, \ldots, q_k\}$, we calculate a $distances_i$ list for each q_i, and obtain k *distances* lists. An element-wise summation of the k lists results in one aggregate list, which we sort in ascending order to retrieve Q's top-K NN.

In the previous definition, it is important to note the element-wise summation of the k lists is done before any sorting of the k lists. This is to ensure each element in the aggregate list is the sum of EDs belonging to the same Q and $\boldsymbol{E_i}$ pair.

With the background information defined, we are ready to introduce our framework.

5 METHODOLOGY

We begin with an outline of the representational power of our model. We envision our model as embedded into a **MTS** *query-by-content* system [5]. After a query is answered, our system makes one of the following suggestions:

- Add a single dimension to the query.
- Remove a single dimension from the query.
- No change to the query (referred to as *status quo* case).

Recall that these suggestions may be informative and useful to the user, even if she does not act upon them by reissuing the updated query. She may learn an unexpected relationship or redundancy between the dimensions.

For simplicity our system does not suggest adding or removing more than one dimension at a time. Furthermore, our system cannot simultaneously suggest removing dimension(s) and adding dimension(s). However, it is clear that in principle, any subspace of dimensions is reachable with multiple interactions. Therefore, our interaction model offers simplicity and intuitiveness along with the full expressiveness required by the problem we are addressing.

We outline our formal algorithms, *QueryContract* and *QueryExpand* in Table 2 and Table 3 of Sections 5.2 and 5.3, respectively. Here we provide an intuitive preview. The user executes an N-dimensional query (where $N \geq 1$ for query expansion, and $N \geq 2$ for query contraction), retrieving the top-K ($K \geq 5$) NN exemplars. In the background, the algorithm *QueryContract* and *QueryExpand* calculates a "dimension score" for each dimension and examines the results, potentially suggesting one dimension that could be removed from or added to the query. Both algorithms invoke a subroutine *getDimScore* for scoring each dimension. We explain this subroutine next.

5.1 Dimension Score

In Table 1, we introduce *getDimScore*, which computes a score that reflects a dimension's predicted "utility" or "meaningfulness" for a particular query, with a higher value being better. The inputs to *getDimScore* are dimension *d*, and the *topKNN* indices, which are the top-K indices of the NN from the execution of query *q* on its respective dimension *d*. The output is a real number dimension score which reflects *d*'s predicted "utility" or "meaningfulness" for query *q*. In line 1, *getDimScore* constructs a K-by-K distance matrix containing the pairwise Euclidean distances of the top-K matches. Then in line 2, the algorithm calculates *topKAvg*, which is the mean of the pairwise distances of the values in the upper triangle of the matrix.

Table 1: getDimScore algorithm

```
Procedure getDimScore( d, topKNN)
Input: A dimension d to score.
topKNN contains the indices of the top-K NN for query q on d.
Output: dimScore, a dimension score of meaningfulness
1  distTopK = K-by-K pairwise ED matrix of the top-K NN
2  topKAvg = mean(upper triangle(distTopK))
3  for j=1:100
4      distRandK = K-by-K pairwise ED matrix of the random-K NN
5      distrRandAvg(j) = mean(upper triangle(distRandK))
6  end
7  dimScore = | mean(distrRandAvg) − topKAvg |
8  return dimScore
```

In the for loop block spanning from lines 3 to 6, *getDimScore* employs random sampling by selecting any K exemplars from the entire aggregate list (Def. 7), and similarly obtains a mean of the upper triangle from the construction of a K-by-K distance matrix for each dimension. Repeating this process multiple times, we use one-hundred times, allows us to obtain a distribution of means, which we call *distrRandAvg*. As we will illustrate in the next section, our key idea is a "meaningful" and "relevant" dimension to the query should have a *topKAvg* lying further away from the mean of *distrRandAvg* when compared to an "meaningless" and "irrelevant" dimension. Based on this idea, for each dimension, we compute a score:

$$dimScore = |\, mean(\, distrRandAvg) - topKAvg\, |$$

This is reflected in line 7. With our dimension score methodology explained, we are now ready to discuss our first operation, which is *query contraction*.

5.2 Query Contraction

In Table 2, we introduce the *QueryContract* operator, which can suggest a single dimension to delete from a query.

The *QueryContract* algorithm takes as input a **MTS** *Q* and its corresponding dimensions *D* along with *topK*, which is the number of NN to retrieve. The output is a dimension to suggest removing (if any). The algorithm begins by retrieving the top-K NN indices after executing *Q* on *D* in line 1. Then *QueryContract* calls *getDimScore* in lines 2 to 4 to obtain a *dimScore* for each dimension. The program execution continues at line 5, where *QueryContract* ranks the scores in ascending order, and suggests a deletion of the lowest scoring dimension, if it is less than or equal to the threshold (reflected in lines 6 to 8). We defer the details of the threshold until Section 5.4.

Table 2: QueryContract algorithm

```
Procedure QueryContract(Q, D, topK)
Input: A MTS Q = {$q_1$, $q_2$, …, $q_N$} from the corresponding dimensions
D = {$d_1$, $d_2$, …, $d_N$}.
topK is the number of closest NN to retrieve.
Output: A dimension to remove (if any).
1  topKNN = Execute Q on dimensions D to retrieve top-K NN indices
2  for each $d_i$
3      dimScoreList(i) = getDimScore( $d_i$, topKNN)
4  end
5  lowScore = min(dimScoreList)
6  threshold = sqrt(sum(dimScoreList.^2))/N
7  if lowScore ≤ threshold return dimension with lowScore
8  else return none
```

To concretely illustrate the *QueryContract* algorithm, we consider the case where there is an objective truth, because we contrived it to exist in a semi-synthetic dataset.

Imagine a user who executes a three-dimensional query, taken from a snippet of a person performing *normal-walking*. Two of the dimensions are from the PAMAP dataset (which the reader may recall is from the same dataset as our introductory examples in Section 2), the x-axis acceleration of the hand and shoe. We created a third dimension consisting of random synthetic patterns, shown in Figure 3, such as sine waves, Gaussian pulses, and triangular pulses, among others. We added a slight amount of warping to each synthetic pattern so no two are exactly the same. This random synthetic dimension is clearly not a "meaningful" dimension for any particular query. In this case, the top-K nearest neighbors returned by the query of the synthetic dimension are "random" in relation to the query. Thus, we would hope and expect that the *QueryContract* algorithm would score this particular dimension the lowest out of the three, and suggest deleting it.

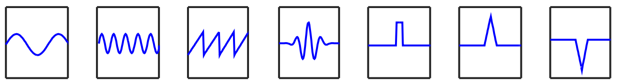

Figure 3: Our synthetic dimension is created with the seven patterns shown above.

As we can see in Figure 4, this is exactly what happens. Of the three dimensions the synthetic dimension in Figure 4.*bottom* is clearly the outsider with nearly identical values for the mean of both the top-K pairwise distances and the one-hundred random-K pairwise distances.

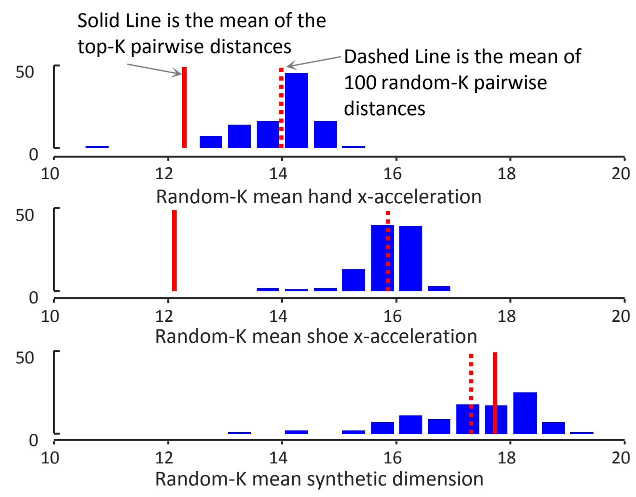

Figure 4: Our intuition for scoring three dimensions when executing a query of normal-walking. The solid red line is the mean of the top-K NN, while the dashed red line is the mean of 100 random-K NN. *top*) In the hand x-acceleration dimension, the solid line and dashed line are apart by at least one standard deviation. *middle*) Similarly, in the shoe x-acceleration dimension, the solid line and dashed line are also far apart. *bottom*) In the synthetic dimension, the solid line and dashed line almost coincide.

Having explained *query contraction*, we will next explain its complementary operation, which is *query expansion*.

5.3 Query Expansion

Here, we explore the case of suggesting a dimension to add, which is our operation of *query expansion*. In Table 3, *QueryExpand* takes in as input an initial **MTS** query *Q* from the corresponding dimensions *D* along with a list of potential candidate dimensions *Dc*, and its corresponding candidate queries *Qc*. Recall (Def. 5) every **MTS** query *Q* is part of an exemplar ***E*** in a **MTS** dataset, and is usually a subset of all the available dimensions. We can choose some (or all) of the other dimensions as our candidate dimensions *Dc*. For each dimension *dc* in *Dc*, we extract the candidate query *qc* from the same ***E*** as *Q*. These candidate queries are stored in *Qc*. The algorithm *QueryExpand* outputs a candidate dimension to add (if any).

Similar to *QueryContract*, in line 1 *QueryExpand* also retrieves the top-K NN indices after the execution of *Q* on *D*, and also calculates the *dimScores* for the query dimensions in lines 2 to 4. Next, lines 5 and 6 calculate the threshold. The highest scoring candidate dimension can only be added if it is greater than or equal to the threshold. Details of our threshold will be discussed in Section 5.4. Then, *QueryExpand* computes a *dimScore* for each potential candidate dimension (in lines 7 to 12). This is achieved by first defining a new query *Q*' and a new corresponding dimensions *D*' containing the original *Q* with a candidate query qc_i appended and the original *D* with a candidate dimension dc_i appended, respectively (lines 8 and 9). The algorithm executes *Q*' on *D*' to retrieve the top-K NN indices and obtains the *dimScores* (lines 10 and 11). *QueryExpand* ranks the candidate dimension scores in descending order, and suggests an addition of the highest scoring dimension if it greater than or equal to the threshold (lines 13 to 15).

Table 3: QueryExpand Algorithm

Procedure QueryExpand(*Q*, *D*, *Qc*, *Dc*, *topK*)
Input: A MTS $Q = \{q_1, q_2, \ldots, q_N\}$ from the corresponding dimensions $D = \{d_1, d_2, \ldots, d_N\}$. $Dc = \{dc_1, dc_2, \ldots, dc_M\}$ is a list of candidate dimensions with corresponding queries $Qc = \{qc_1, qc_2, \ldots, qc_M\}$.
topK is the number of closest NN to retrieve.
Output: A dimension to add (if any)

```
1   topKNN = Execute Q on dimensions D to retrieve top-K NN indices
2   for each $d_i$
3       dimScoreList(i) = getDimScore( $d_i$, topKNN)
4   end
5   virDimScore = mean(dimScoreList)
6   threshold = sqrt (sum (dimScoreList.^2 + virDimScore^2)) / (N+1)
7   for each $qc_i$ and $dc_i$
8       Q' = {$q_1$, $q_2$, ..., $q_N$, $qc_i$}
9       D' = {$d_1$, $d_2$, ..., $d_N$, $dc_i$}
10      topKNN = Execute Q' on D' to retrieve top-K NN indices
11      canddimScoreList(i) = getDimScore( $dc_i$, topKNN)
12  end
13  highScore = max(canddimScoreList)
14  if highScore ≥ threshold  return dimension with highScore
15  else return none
```

We have now explained our two main operators. However, up to this point we have glossed over how we compute the threshold. We repair this omission in the next section.

5.4 Dynamic Thresholding

Our threshold, which is dynamically determined for each **MTS** query, is crucial in determining when we should add or drop a dimension or make no change (status quo case).

The threshold in the query contraction case determines if the lowest scoring dimension in an N-dimensional query lies further away from the majority of the other dimensions in the query. More precisely, we define our formula as follows:

$$threshold_c = \frac{\sqrt{\sum_{i=1}^{N} dimScore_i^2}}{N}$$

We defined our threshold in *QueryContract* as the mean root sum of squared *dimScores*, where each $dimScore_i$ is the *i*th dimension score in an N-dimensional query. The purpose is to amplify higher *dimScores*. It is similar in spirit to measures such as Root Mean Square Error (RMSE) [23]. However, the main difference is that we "reward" high *dimScores* instead of "penalizing". If the lowest scoring dimension is below this threshold, our framework recommends a deletion of that dimension. Otherwise, no suggestion is made, which is our status quo case.

In the complementary operation, query expansion, we also use a threshold to determine if a candidate dimension is "meaningful" enough to add to the particular query at hand. But how do we calculate an appropriate threshold when we are in a sense "missing" a *dimScore*? That is, how do we determine the *dimScore* of a "meaningful" candidate dimension?

We solve the problem of a "missing" dimension by creating a "virtual" dimension score based on the mean of the *dimScores* in the initial query. The mean of the initial dimension scores suggests an expected value of the *dimScore* for a candidate dimension which is commensurate with the dimension scores of the initial query in execution. We call this "virtual" dimension score *virDimScore*. With this score, we are now able to define a threshold in *QueryExpand*. More concretely, if a user is executing an N-dimensional query, the threshold would be:

$$threshold_e = \frac{\sqrt{\sum_{i=1}^{N}(dimScore_i^2) + virDimScore^2}}{N+1}$$

If the highest scoring candidate dimension exceeds the threshold, our framework recommends an addition of that dimension. Otherwise, no suggestion is made. Having explained our subroutines for *query contraction*, *query expansion*, and dynamic thresholding, we are ready to explain how we put all the elements together for the general case of query suggestion.

For ease of use, a user is only required to specify her initial **MTS** query, and a list of candidate dimensions. Our framework recommends adding or removing a dimension (if any). However, the careful reader may have noticed there are cases where it is possible to suggest both adding and removing a dimension. In those cases, our framework will recommend the operation where its absolute difference of the *dimScore* and its threshold is greater.

6 EXPERIMENTAL EVALUATION

To allow reproducible research, we archived all code and data along with detailed instructions for every experiment and the introductory examples at [18].

6.1 Measures of Success

Recall in that Section 2, we have shown the importance of dimension choice and how choosing the right subset of dimensions for a particular query can dramatically impact the precision of the top-K NN. Those anecdotal examples suggest two measures of success. For simplicity, we evaluate the two operations separately. The first measure is whether *QueryContract* or *QueryExpand* made the "correct" suggestion. The second measure is the change of precision in the top-K NN after the choice of *QueryContract* or *QueryExpand* is taken into effect.

Our first measure, which we refer to as *choiceAcc*, is computed as follows:

$$choiceAcc = \frac{number\ of\ \text{"correct"}\ suggestions}{total\ number\ of\ suggestions\ made}$$

The values of *choiceAcc* range from 0 to 1, where a higher value means a better score. Recall from Section 5, there are three possible suggestions: add or remove a dimension, or no change.

In general, we do not have absolute ground truth in most domains. We mitigate this in two ways. In our first sequence of experiments, we contrive the data by adding synthetic data shown in Figure 3 such that there *is* an objectively correct answer. In the second sequence of experiments, we consider datasets for which we have strong intuitions as to what the ground truth should be. As an example, it is intuitive to suspect that the body temperature dimension is not relevant to queries involving gait.

Our second measure was hinted at in Section 2, where we mentioned an objective evaluation technique. *QueryContract* and *QueryExpand* is designed for unlabeled time series data, however we can evaluate how well our algorithm performs through the use of labeled data in a post-hoc analysis. We emphasize that *QueryContract* and *QueryExpand* does *not* look at or rely on the class labels during its execution; the class labels are completely "hidden" from our algorithms.

We will refer to our second measure as *changePrecision*, which is computed as the change in precision from the initial **MTS** query to the modified **MTS** query after the suggestion from *QueryContract* or *QueryExpand* has been taken into effect. The precision formula is the same as the one used in the classification context:

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

To be specific, we calculate *precision* from our top-K NN list returned upon the execution of the query. The NN list may contain exemplars which are in the same class as the query, also known as true positives. On the contrary, the list may also contain exemplars which are not in the same class as the query (i.e., false positives).

Below we define *changePrecision*:

$$changePrecision = precision_{After} - precision_{Before}$$

Where $precision_{Before}$ and $precision_{After}$ are the *precision* in the initial query and after any changes are in effect, respectively. The values of *changePrecision* range from -1 to 1. A positive value of *changePrecision* indicates an increase in precision after query is updated with the suggestion of *QueryContract* or *QueryExpand*. Nevertheless, it is possible for *changePrecision* to be 0 or even negative, after the query is updated.

Our framework has a quadratic time complexity, which is determined as follows. The time complexity is $O(\ d|q|(\ K^2 + E_n)\ )$, where d is a constant number of dimensions in the initial query and as candidates for query expansion, $|q|$ is the length of the time series query, K is a constant number of top-NN to retrieve (e.g., top-5, top-20, top-100), and E_n is the number of exemplars.

The query length $|q|$ and the number of exemplars E_n contribute to a quadratic time complexity.

6.2 Results

We evaluate on four **MTS** datasets in Matlab R2015a on an Intel Core 2 Duo 2.00 GHz CPU with 3.00 GB RAM. For simplicity and clarity, we evaluate *QueryContract* and *QueryExpand* separately. In datasets involving human activity, we consider *user-dependent* accuracy and precision [12].

In the following subsections, we briefly describe each dataset and then present our results for some subsets of the possible initial dimensions in Table 4 - Table 11. To measure the *choiceAcc* and *changePrecision* defined in an earlier section, for each dataset, we execute each exemplar as an **MTS** query against all the other exemplars as our data, in a leave-one-out fashion. We present *choiceAcc* and the average $precision_{Before}$, $precision_{After}$, and *changePrecision* for each selected starting dimensions.

6.2.1 PAMAP (human activities). Our first dataset is the PAMAP [14] that we have used throughout this paper as our anecdotal examples. We extracted fifteen exemplars each of three different outdoor activities: *normal-walking, running*, and *Nordic-walking*.

In Table 4, we present our results for *QueryContract*. In the first column, we list the initial dimensions for our **MTS** query, with synthetic or irrelevant dimensions in bold and colored font. We list our first measure of success, *choiceAcc*, in the second column. The third column indicates the average precision before and after any suggestions are taken into effect, and the average *changePrecision* (our second measure of success), which is denoted by **Δ**.

Table 4: PAMAP *QueryContract* results, a bold/colored dimension in the first column indicates it *should* be dropped.

Initial Dimensions	*choiceAcc*	Avg. Precision Before–After–Δ
hand.acc.x, shoe.acc.x, **synthetic**	43/45=0.95	0.86–0.98–0.12
hand.acc.x, **chest.mag.x**	45/45=1.00	0.91–1.00–0.09
hand.acc.x, chest.acc.x, shoe.acc.x	45/45=1.00	1.00–1.00–0.00

In the first row of Table 4, we ran our queries with a contrieved synthetic dimension which we can be 100% sure is irrelevant, thus establishing the ground truth. We show *QueryContract* correctly suggested dropping this synthetic dimension and upon acting on this recommendation, the average precision increased by 0.12.

In the second row of Table 4, we suspect the {chest.mag.x} is unmeaningful as the magnetometer is used for measuring direction with respect to the Earth's magnetic fields, which should not be as relevant as the various activities were performed without a fixed bearing. Our *QueryContract* operation correctly suggested dropping the chest, and applying that suggestion resulted in an increase of 0.09 in the average precision.

Finally, we also tested the effectiveness of our dynamic threshold in detecting *status quo* cases with an initial query of three relevant dimensions in the third row of Table 4. The three dimensions are relevant due to the average precision of 1.00 to begin with. Our *QueryContract* correctly did not recommend dropping any dimension, and the precision remained perfect.

The average running time for the query exemplars in the first, second, and third row of Table 4 is 0.59, 0.35, and 0.52 seconds, respectively.

In Table 5, we show the results of our *QueryExpand* operation. It is similar to the *QueryContract* table, but with an additional column which contains the list of candidate dimensions.

Table 5: PAMAP *QueryExpand* results, a bolded/colored candidate dimension in the second column indicates it is a correct choice.

Initial Dimensions	Candidate Dimensions	*choiceAcc*	Avg. Precision Before–After–Δ
shoe.acc.x, shoe.acc.y, shoe.acc.z	**hand.acc.z**, **chest.acc.z**, chest.mag.z, synthetic	45/45=1.00	1.00—1.00—0.00
hand.acc.z, chest.acc.z, shoe.acc.z	chest.mag.x, synthetic	45/45=1.00	1.00—1.00—0.00
chest.mag.x	**hand.acc.x**, **shoe.acc.x**	45/45 =1.00	0.31—0.95—0.64

For all such tables, the bolded dimensions in that list are *some* (but not *all* due to space constraints) of the meaningful dimensions from the dataset, and any bolded dimension is a "correct" one for adding. If the list does not contain any bolded dimensions, then the recommendation of maintaining *status quo* would be the "correct" suggestion. As the reader may notice from the first and second rows of Table 5, *QueryExpand* was able to correctly distinguish the case where a dimension can be added and where *status quo* should be maintained, respectively.

Our *QueryExpand* operation was also capable of significantly improving an initial query with an irrelevant dimension as indicated in the third row of Table 5. Imagine the case where a novice user executes a query with the {chest.mag.x} dimension, which has an average precision of only 0.31. Our system can help the user by suggesting an addition of either {hand.acc.x} or {shoe.acc.x}, both of which are relevant dimensions in this case. A user accepting the suggestions benefits from an average *changePrecision* of 0.64.

The average running time for the query exemplars in the first, second, and third row of Table 5 is 0.42, 0.35, and 0.24 seconds, respectively.

6.2.2 uWave Gesture (human gestures). Another human activity dataset we considered is uWave [12]. As shown in Figure 5 it consists of eight hand gestures identified in [10], and performed by users holding a Wii Remote.

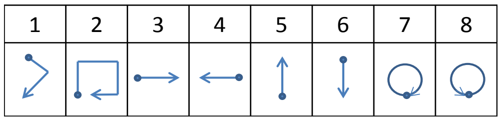

Figure 5: The eight hand gestures in the uWave dataset.

There are some query dependencies for the gestures shown in Figure 5. For example, the x-axis acceleration of the vertical line gestures 5 and 6 would intuitively be an irrelevant dimension. Similarly, the z-axis of the horizontal line gestures 3 and 4 is presumably also an irrelevant dimension. These query dependencies are indicated in parenthesis in the initial dimensions column. In contrast, gestures 1, 2, 7, and 8 involves all three axes as it would be difficult for the user to freely and naturally perform these gestures while restricted to movement in only two of the three dimensions.

The results shown in Table 6 indicate *QueryContract* did correctly suggest dropping the synthetic dimension resulting in an increase of 0.04 in the average *changePrecision* (reflected in first row). Furthermore, *QueryContract* was able to correctly identify which dimension to drop (if any) depending on the gesture. In rows two and three, the "correct" choice is to drop {hand.acc.z} when the query is gesture 3 or 4 and to drop {hand.acc.x} when the query is gesture 5 or 6.

The average running time for the query exemplars in the first, second, and third row of Table 6 is 0.98, 0.72, and 0.48 seconds, respectively.

Table 6: uWave QueryContract results

Initial Dimensions	*choiceAcc*	Avg. Precision Before–After–Δ
hand.acc.x, hand.acc.y, hand.acc.z,**synthetic**	57/80= 0.71	0.87–0.91–0.04
hand.acc.x (for gest. 5,6 only), hand.acc.y **hand.acc.z (for gest. 3,4 only)**	79/80=0.99	0.98–0.97– -0.01
hand.acc.x (for gest. 5,6 only), **hand.acc.z (for gest. 3,4 only)**	76/80=0.95	0.93–0.95–0.02

For our complementary operation, *QueryExpand*, the results in Table 7 show it is capable of mostly suggesting the relevant {hand.acc.y} dimension instead of the synthetic dimension (row one). In the second row, we show the operation improved an initial query of {synthetic, hand.acc.x}. The {synthetic} dimension is meaningless, and causes the Before Average Precision to have a low value of 0.34. Upon taking *QueryExpand*'s suggestion by either correctly adding {hand.acc.y} or {hand.acc.z}, the precision doubled to 0.68. Finally, in row three, the precision of an initial single-dimensional query improved by 0.18 by either correctly adding the {hand.acc.x} or {hand.acc.z} dimension.

Table 7: uWave QueryExpand results

Initial Dimensions	Candidate Dimensions	*choiceAcc*	Avg. Precision Before–After–Δ
hand.acc.x, hand.acc.z	**hand.acc.y**, synthetic	75/80=0.94	0.93–0.96–0.03
synthetic, hand.acc.x	**hand.acc.y,hand.acc.z**	80/80= 1.00	0.34–0.68–0.34
hand.acc.y	**hand.acc.x,hand.acc.z**	80/80 =1.00	0.78–0.96–0.18

The average running time for the query exemplars in the first, second, and third row of Table 7 is 1.05, 1.10, and 0.82 seconds, respectively.

6.2.3 Articulatory Word (medical). We also consider the Articulatory Word dataset [21], which is comprised of the 3D movements of twenty-five word sequences repeated multiple times. An ElectroMagnetic Articulograph (EMA) recorded these movements via four tongue sensors on tongue tip (T1), blade (T2), body front (T3), and body back (T4), and two lip sensors on the upper (UL) and lower (LL) lips. Data collected with the EMA is used to help people with speech impairments [21].

Below in Table 8, we again show *QueryContract* was able to drop the synthetic dimension with *choiceAcc* of 0.80 which resulted in a 0.05 increase in *changePrecision* (denoted in the first row). Furthermore, *QueryContract* was again capable of using the dynamic threshold to detect two *status quo* cases we chose arbitrarily (reflected in the second and third rows).

The average running time for the query exemplars in the first, second, and third row of Table 8 is 29.14, 23.72, and 10.98 seconds, respectively.

Table 8: Articulatory Word *QueryContract* results

Initial Dimensions	*choiceAcc*	Avg. Precision Before–After–Δ
T1y, T1z, T4y, T4z, **Synthetic**	458/575=0.80	0.63–0.68– 0.05
T1y, T1z, T2y, T2z	573/575=0.99	0.63–0.63–0.00
T2y and T2z	559/575=0.97	0.60–0.60–0.00

Our results for the complementary operation, *QueryExpand*, are shown in Table 9. To summarize, we show the operation correctly suggested related tongue dimensions {T4y} or {T4z} in the first row, and avoided suggesting the synthetic dimension in the second row. In the third row, our operation mostly avoided an irrelevant dimension, {T1x}. The x-axis measures side-to-side movement, which is almost negligible during speech [21].

Table 9: Articulatory Word *QueryExpand* results

Initial Dimensions	Candidate Dimensions	*choiceAcc*	Avg. Precision Before–After–Δ
T3y, T3z	**T4y,T4z**, synthetic	505/575=0.88	0.56–0.61– 0.05
T4y,T4z,T3y, T3z	synthetic	468/575=0.81	0.70–0.68– -0.02
T1y	**T1z** ,T1x	471/575=0.82	0.44–0.57– 0.13

The average running time for the query exemplars in the first, second, and third row of Table 9 is 26.45, 26.32, and 17.65 seconds, respectively.

6.2.4 Wafer (industrial). Our final data set is Wafer, a much-studied industrial data set [13], which consists of measurements from sensors during the etching process of a silicon wafer. At the end of the etching process, each wafer is classified as normal or abnormal. There are a total of six sensors determined by domain experts as most relevant: radio frequency forward power (Rffp), radio frequency reflected power (Rfrp), chamber pressure sensor (Cpr), 405 nm emission (405e), 520 nm emission (520e), and direct current bias (Dcb) [13].

In Table 10, we once again demonstrated *QueryContract* was able to suggest removing the synthetic dimension with *choiceAcc* of 0.70, increasing the precision by 0.02 (row one). In addition, *QueryContract* was able to detect *status quo* cases (rows two and three).

The average running time for the query exemplars in the first, second, and third row of Table 10 is 1.40, 1.78, and 0.78 seconds, respectively.

Table 10: Wafer *QueryContract* results

Initial Dimensions	***choiceAcc***	**Avg. Precision Before—After—Δ**
Rffp,Rrfp,Cpr,405e,520e,**synthetic**	35/50=0.70	0.74—0.76—0.02
Rffp,Rrfp,Cpr,405e, 520e, Dcb	50/50=1.00	0.77—0.77—0.00
Rffp,Rrfp,Cpr	50/50=1.00	0.78—0.78—0.00

Finally, in Table 11, we demonstrated *QueryExpand*'s capability to detect the *status quo* case with 0.72 *choiceAcc*. Since there were fifteen times the synthetic dimension was erroneously suggested for adding, the average *changePrecision* was negatively affected and did decrease by 0.02.

In rows two and three of Table 11, *QueryExpand* suggested adding the relevant dimension from the list. While this did not increase the average *changePrecision* in both cases, it does provide additional new information to the user which may not have been previously known before. For example, in the second row, in all fifty times, the chamber pressure sensor (Cpr) dimension was recommended by *QueryExpand* for adding. This informs a user (who may not have been aware previously) of the relevance of the pressure sensor for that particular initial query.

Table 11: Wafer *QueryExpand* results

Initial Dimensions	**Candidate Dimensions**	***choiceAcc***	**Avg. Precision Before—After–Δ**
Rffp,Rrfp,Cpr,405e,520e	synthetic	36/50= 0.72	0.77—0.75– -0.02
Rffp,Rrfp	**Cpr,405e, 520e**	50/50=1.00	0.77– 0.77– 0.00
405e	**Rffp,Rrfp,Cpr,520e, Dcb**	50/50=1.00	0.75—0.75—0.00

The average running time for the query exemplars in the first, second, and third row of Table 11 is 1.13, 1.23, and 1.81 seconds, respectively.

6.2.5 Putting it all together. The reader may recall in Section 5.4, we mentioned our framework can handle the case where the user does not need to know ahead of time which operation to use. Instead, the user simply needs to specify her initial **MTS** query and dimensions, and a list of candidate queries and dimensions. Furthermore, through multiple iterations of query execution in our framework, the user will have the possibility to converge on the ideal subset of dimensions for a particular query once the framework recommends the *status quo* case.

We illustrate this with an example from our UWave dataset with a focus on the vertical line hand gestures (5th and 6th gestures from Figure 5). Intuitively, the {hand.acc.x} is almost certainly irrelevant as we would expect very little side-to-side movement. Figure 6 illustrates an exemplar of a vertical line gesture #5 with all the available dimensions shown.

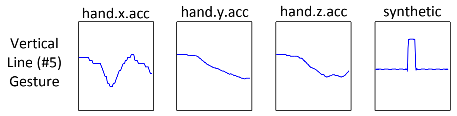

Figure 6: An exemplar of a vertical line gesture query with all the available dimensions shown.

However, let us consider the case where the user did not realize {hand.acc.x} was an irrelevant dimension and executes an initial query of {hand.acc.x, hand.acc.y} with the candidate dimensions {hand.acc.z} and {synthetic}. Our framework recommends an addition of {hand.acc.z} (which helps increase the average precision from 0.83 to 0.99).

With this suggestion, the user can continue in the next iteration executing the query of {hand.acc.x, hand.acc.y, hand.acc.z} along with a {synthetic} dimension as our candidate. Our framework recommends dropping the irrelevant {hand.acc.x} dimension (which helps increase the average precision from 0.99 to 1.00).

Finally, the user continues with the query dimensions of {hand.acc.y, hand.acc.z} with the synthetic dimension as our candidate dimension. Our framework recommends the *status quo* case, and at that point the user has found the optimal subset for the vertical line hand gestures.

7 CONCLUSIONS

We have introduced a novel and simple framework to support exploratory search in large multidimensional time series by suggesting dimensions that can be added or dropped, on a query-by-query basis. We have demonstrated on diverse datasets that our system can improve precision, and can offer insights into the data. Future work includes supporting DTW and other distance measures [22] and a detailed human subjects study.

REFERENCES

[1] Bahadori, M. T., Kale, D., Fan, Y., Liu, Y. Functional Subspace Clustering with Application to Time Series. *Proceedings of the 32nd ICML*, 228-237, 2015.
[2] Camerra, A., Shieh, J., Palpanas, T., Rakthanmanon, T., Keogh, E. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *Knowledge and information systems, 39(1)*, 123-151, 2014.
[3] Collins, S.H., Adamczyk, P.G., Kuo, A. D. Dynamic arm swinging in human walking. *Proceedings of the Royal Society, Biological Sciences, 276*, 3679–3688, 2009.
[4] Faloutsos, C., Ranganathan, M., Manolopoulos, Y. Fast Subsequence Matching in Time-Series Databases. *In Proceedings of the 1994 ACM SIGMOD*, 419-429, 1994.
[5] Hochheiser, H., Shneiderman, B. Dynamic query tools for time series data sets: Timebox widgets for interactive exploration. *Inf Vis Information Visualization. 3*, 1–18, 2004.
[6] Hochheiser, H., Shneiderman, B. Interactive Exploration of Time Series Data. *Discovery Science*, 441-446, 2001.
[7] Holz, C., Feiner, S. Relaxed selection techniques for querying time-series graphs. *Proc'the ACM Symposium on User Interface Software and Technology*, 213-222, 2009.
[8] Hu, B., Chen, Y., Zakaria, J., Ulanova, L., and Keogh, E. J. Classification of Multi-dimensional Streaming Time Series by Weighting Each Classifier's Track Record. *2013 IEEE 13th International Conference on Data Mining*, 281-290.
[9] Kang, H.G., Dingwell, J.B. Dynamic stability of superior vs. inferior segments during walking in young and older adults. *Gait & Posture 30*, 260–263, 2009.
[10] Kela, J., Korpipää, P., Mäntyjärvi, J., Kallio, S., Savino, G., Jozzo, L., Marca, D. Accelerometer-based gesture control for a design environment. *Personal and Ubiquitous Computing, 10(5)*, 285-299, 2006.

[11] Korn, F., Jagadish, H.V., Faloutsos, C. Efficiently supporting ad hoc queries in large datasets of time sequences. *Proceedings of the 1997 ACM SIGMOD*, 289-300, 1997.

[12] Liu, J., Wang, Z., Zhong, L., Wickramasuriya, J., Vasudevan, V. uWave: Accelerometer-based personalized gesture recognition and its applications. *IEEE International Conference on Pervasive Computing and Communications*, 657-675, 2009.

[13] Olszewski, R. T. Generalized feature extraction for structural pattern recognition in time-series data (No. CMU-CS-01-108). *Carnegie Mellon University School of Computer Science*, 2001.

[14] PAMAP, Physical Activity Monitoring for Aging People, http://www.pamap.org/demo.shtml, retrieved 2015-11-08.

[15] Parsons, L., Haque, E., Liu, H. Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, *6(1)*, 90-105, 2004.

[16] Patnaik, D., Marwah, M., Sharma, R., Ramakrishnan, N. Sustainable operation and management of data center chillers using temporal data mining. *Proceedings of the 15th ACM SIGKDD*, 1305-1314, 2009.

[17] Qiu Y., Frei, H. Concept Based Query Expansion. In *Proceedings of SIGIR-93*, 160-169, 1993.

[18] Reproducibility Web page. https://sites.google.com/site/ssdbm2017/

[19] Summa, A., Vannozzi, G., Bergamini, E., Iosa, M., Morelli, D. Multilevel Upper Body Movement Control during Gait in Children with Cerebral Palsy. *PLOS ONE 11*, 2016.

[20] Vlachos, M., Hadjieleftheriou, M., Gunopulos, D., Keogh, E. Indexing multi-dimensional time-series with support for multiple distance measures. *Proceedings of the ninth ACM SIGKDD*, 216-225, 2003.

[21] Wang, J., Samal, A., Green, J. R., Rudzicz, F. Whole-Word Recognition from Articulatory Movements for Silent Speech Interfaces. *In Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[22] Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery Data Min Knowl Disc. 26*, 275–309, 2012.

[23] Weatherburn, C.E. A first course in mathematical statistics. Cambridge University Press, Cambridge, 1961.

[24] Yoon, H., Yang, K., Shahabi, C. Feature subset selection and feature ranking for multivariate time series. *IEEE Transactions on Knowledge and Data Engineering. 17*, 1186–1198, 2005.