TECHNICAL REPORT

TR-CS-NMSU-2019-07-14

Yifan Hao
Huiping Cao
Abdullah Mueen*
Sukumar Brahma

Department of Computer Science
New Mexico State University
* University of New Mexico

July 14, 2019

Identify Significant Phenomenon-specific Variables for Multivariate Time Series

Yifan Hao[1] Huiping Cao[2] Abdullah Mueen[3] Sukumar Brahma[4]
yifan@nmsu.edu, hcao@cs.nmsu.edu, mueen@cs.unm.edu, sbrahma@clemson.edu
[1,2]Computer Science, New Mexico State University, U.S.A.
[3]Computer Science, University of New Mexico, U.S.A.
[4]Electrical and Computer Engineering, New Mexico State University, U.S.A.

Abstract—Multivariate time series (MTS) are collected for different variables in studying scientific phenomena or monitoring system health where one time series records the values of one variable for a time period. Among the different variables, it is common that only a few variables contribute significantly to a specific phenomenon. Furthermore, the variables contributing significantly to different phenomena are often different. We denote the different variables that contribute to the occurrences of different phenomena as *Phenomenon-specific Variables (PVs)*. In this paper, we formulate a *novel problem* of identifying significant PVs from MTS datasets. To analyze MTS data, feature extraction techniques have been extensively studied. However, most of them identify important *global* features for one dataset and do not utilize the temporal order of time series. To solve the newly introduced problem, we propose a solution framework, CNN_{mts}-*X*, which is a new variant of the Convolutional Neural Networks (*CNN*) and can embed other feature extraction techniques (as *X*). Furthermore, we design a CNN_{mts}-*LR* method that implements a new feature identification approach (*LR*) as *X* in the CNN_{mts}-*X* framework. The *LR* method leverages both Linear Discriminant Analysis (*LDA*) and Random Forest (*RF*). Our extensive experiments on five real datasets show that the CNN_{mts}-*LR* method has exhibited much better performance than several other baseline methods. Using 30% of the PVs discovered from the CNN_{mts}-*LR*, classifications can achieve better or similar performance than using all the variables.

Index Terms—Multivariate Time Series (MTS), Convolutional Neural Network (*CNN*), Linear Discriminant Analysis (*LDA*), Random forest (*RF*), Imbalanced Data

1 Introduction

Many applications collect multivariate time series (MTS) for different variables where one variable's time series records the values of this variable for a time period. For example, in tracking human-body movement, multiple sensors (which are treated as variables) are attached to different parts of a body to collect their location information; In environmental sciences, different sensors are used to track environmental information such as temperature and soil moisture. MTS data are typically associated with corresponding phenomena labeled as classes (e.g., walking, sitting, budding). Utilizing both the MTS data and their corresponding class labels, scientists can conduct predictions or classifications. Very often, it is desired to make as accurate predictions or classifications as possible.

However, generating highly accurate predictions is not sufficient. In many situations, it is even more important to understand variables that are most critical for phenomena interpretation or decision making. We observe that, among all the variables, it is common that only a few variables contribute significantly to a specific phenomenon. Variable selection can help reduce storage and computational cost, improve classification performance, or achieve a better understanding of the data [1]. Furthermore, the variables contributing significantly to different phenomena are different. For example, in tracking human body movement, we observe that sensors attached to lower legs can help better identify walking activities than sensors attached to upper arms. Thus, it is more useful to monitor different sets of sensors when a person is conducting different activities (sitting or walking). Another example is that different PM2.5 composition particles (up to hundreds) may contribute to different types of diabetes [2]; identifying which PM2.5 particles contribute to a specific diabete (e.g., Type-2 diabetes) will help reduce such diseases through air pollution control. Our observation of different variables contributing to different phenomena is also utilized in clustering analysis where projected clustering (PC) [3], [4] obtains groups of points that are close in *different subsets* of dimensions. However, typical PC does not work well with variable selection on MTS data. PC treats all the values in a time series as independent dimensions (i.e., each time point is a dimension); thus, the clusters are time-point specific, instead of variable specific.

We denote the different variables that contribute significantly to different phenomena as *Phenomenon-specific Variables* (PVs). PVs carry the most critical information for a specific phenomenon. We *formulate a novel problem of identifying significant PVs from multivariate time series*. Note that the solution to this problem is not finding the different features for better predictions. Instead, we are interested in finding variables that make critical contributions to the explanation of specific phenomena (or events).

The proposed problem is different from existing efforts that analyze time series data. Most existing techniques identify global features for one dataset (e.g., [5], [6], [7], [8], [9], [10], [11], [12]). Such global features are used together to analyze the different events in one dataset. The PVs are different from global features because they are specific to different phenomena. Due to such differences, most existing techniques cannot be directly utilized to solve our proposed problem.

Two major challenges need to be addressed to solve the proposed problem. The first challenge comes from the large amount of computation from a huge search space. Assume that the MTS datasets are collected for A variables and the time series instances correspond to E different event types, then the possible number of variable subsets is $E\cdot(2^A-1)$, which is the search space of the PVs. The second challenge comes from the nature of time series, which has values recorded in a temporal order. Treating these values with or without temporal order may generate very different results. A successful example of utilizing the temporal order of the values is the Shapelets approach [9]. Shapelets approaches are orthogonal to our methods because Shapelets approaches identify the important subsequences (for multiple or all variables) in MTS data, while our work detects the important variables among all the variables. In the calculation of PVs, we desire to consider the temporal order of values in each variable's time series.

This paper proposes a new solution framework, *CNN*$_{mts}$*-X* to solve the problem. This framework designs a variant of Convolutional Neural Networks (*CNN*), denoted as *CNN*$_{mts}$, and allows flexible utilization of other feature extraction techniques as X. We also present a new PV identification algorithm *LR*, which takes advantage of both Linear Discriminant Analysis (*LDA*) [13] and Random Forest (*RF*) [14]. *CNN*$_{mts}$ can capture the *temporal order* of values in a time series and *LR* identifies the PV sets while reducing the search space. The contributions of this paper are as follows.

- We formulate the novel problem of discovering significant PVs from MTS data.
- We propose a solution framework *CNN*$_{mts}$*-X* to solve the problem. The *CNN*$_{mts}$*-X* framework includes a new variant of *CNN* model, *CNN*$_{mts}$, to deal with multivariate time series data. As a side effect, *CNN*$_{mts}$ can also be used to classify MTS data with multiple class labels.
- We implement one newly designed oversampling batch generation strategy in *CNN*$_{mts}$ to process imbalanced datasets.
- We present a new PV identification algorithm (*LR*) that leverages *LDA* and *RF*. And, we implement *CNN*$_{mts}$*-LR* which embeds *LR* in *CNN*$_{mts}$*-X* framework to identify the most important variables.
- We have conducted a deep analysis and mining of the intermediate results from a *CNN*$_{mts}$ model.
- We have implemented several baseline approaches and evaluated the effectiveness and efficiency of our proposed techniques by using five real datasets in different sizes. The experiments show that *CNN*$_{mts}$*-LR* outperforms other methods.

The paper is organized as follows. Section 2 formally defines the problem and related terminology. Section 3 presents our proposed *CNN*$_{mts}$*-X* framework and the new *LR* method. Section 4 experimentally demonstrates the effectiveness and efficiency of our proposed approaches using real datasets. Section 5 discusses the literature. Finally, Section 6 concludes our work.

2 Problem Formulation and Terminology

This section introduces the terminology used to formally formulate the problem that we are going to solve.

Definition 1. A **variable** for a multivariate time series is a factor in the time series. If a multivariate time series consists of observations for A variables, these variables are denoted as $a_1, a_2, \cdots, a_A$.

In different applications that collect multivariate time series data, variables represent different meanings. E.g., in human body movement, a variable can be a sensor that is attached to a specific part of a human body.

For each variable, values at different times can be recorded. Such values form a sequence (or time series). Formally,

Definition 2. An **m-sequence** S is in the form of (v_1, t_1), (v_2, t_2), $\cdots$, (v_m, t_m) where $t_i < t_j$ for $1 \leq i < j \leq m$, v_i is either a categorical or a numerical value recorded for one variable at time point t_i, and m is the length (or the number of temporal points) of the variable sequence. When the time intervals between consecutive t_is are fixed, this sequence can be simplified to v_1, v_2, $\cdots$, v_m. Each sequence is for one variable.

Definition 3. An **event type**, denoted as et, is the phenomenon that a study is interested in. Let E denote the total number of event types. One event type can have many corresponding instances. An event instance is represented as et_i.

In the study of human body movement, there can be 10-20 different event types for people's activities (e.g., sitting, running). For each specific event type (e.g., sitting), there can be hundreds or thousands of instances. Event types and event instances in our problem are analogous to class labels and instances in classification problems.

Definition 4. A **multivariate time series** (MTS) contains A m-sequences. Formally, one MTS can be represented as

$$\begin{pmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,m} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,m} \\ \cdots & \cdots & \cdots & \cdots \\ v_{A,1} & v_{A,2} & \cdots & v_{A,m} \end{pmatrix}.$$

Each MTS corresponds to an event type (e.g., a person is running) and records the values for all the variables that contribute to the occurrence of one event.

To study what variables contribute more to an event, all the variables for which an MTS is collected need to be investigated. However, as discussed before, among all the variables, different variables may contribute significantly to different phenomena.

Definition 5. **Phenomena-specific variables** (PVs) for an event type are the variables that contribute significantly to the occurring of that event type.

Definition 6. The **problem of identifying phenomena-specific variables** from MTS data takes as input (i) a set of MTS associated with event types, and (ii) a number $\sigma(\in (0, 1])$, and finds the top $\lfloor \sigma \times A \rfloor$ variables

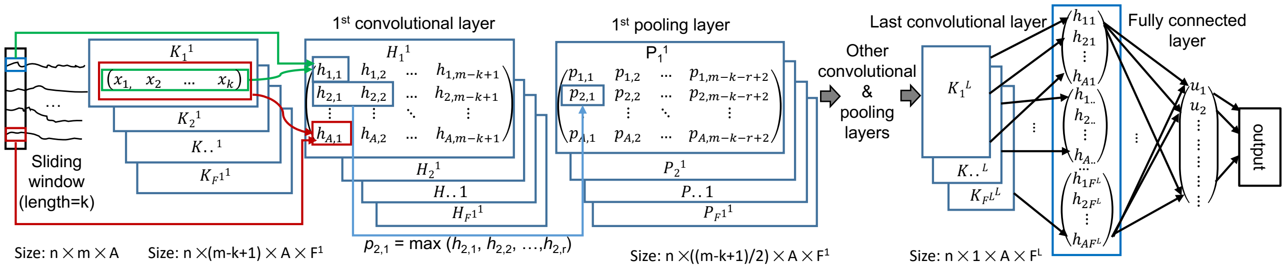


Fig. 1: CNN_{mts} model for MTS (L convolutional layers and $L-1$ pooling layers)

$\{a_{i,1}, \cdots, a_{i,\lfloor \sigma \times A \rfloor}\}$ for each event type et_i such that the chosen variables contribute the most to characterize the given event type.

3 Convolutional Neural Networks Based Approach

This section presents a new framework CNN_{mts}-X to identify PVs from multivariate time series.

Convolutional Neural Networks (*CNN*) are a special type of neural networks (*NN*). A *CNN* has special hidden layers, convolutional layers. Different from the hidden layers in regular *NN*, the nodes in convolutional layers are only connected to a small region, which is called *receptive field*, of the previous layer. The receptive fields are spatially connected to capture the local spatial connectivity when a *CNN* is utilized in image classification. This idea can be utilized to capture the local temporal connectivity of time series in MTS analysis.

The *CNN* model is adopted in our proposed framework to address the major challenges that are discussed in Section 1 because of two major reasons. First, in the analysis of MTS, it is very necessary to capture the local temporal connectivity in a time series [10], [15], [16], [17], [18], [19]. I.e., letting a subsequence contribute to one node in the next layer. Convolutional layers with properly designed kernels can help us achieve this. Second, *CNN* has shown good performance in classifying large amount of data in very high dimensional space [20], [21]; thus adopting *CNN* can help reduce the computational complexity.

The *CNN* approach is capable of automatically extracting features from the training datasets and utilizing such features to recognize different phenomena. Note that these features are combinations of different variables in the original MTS. This work, however, does not target at *purely recognizing the different phenomena utilizing the combined features*. The purpose of this work, as discussed in Section 1, is to identify the variables (not combined features) that contribute the most to specific phenomena. Thus, the original *CNN* method cannot directly work to solve this PV identification problem.

The CNN_{mts}-X framework works in two steps: (i) the first step (Section 3) is to construct and train a CNN_{mts} model, and (ii) the second step (Section 3.2) is to design a PV Identification (PVI) algorithm to extract significant PVs from the intermediate results of the CNN_{mts} models. To verify the effect of PVs, classifications can be utilized. Section 3.3 introduces the classification algorithm using the PVs identified by CNN_{mts}-X.

Phenomenon	Variables	Time sequences
Playing Basketball (PB)	LA LL	20, 40, 60, 80, 60, 40, 20 4, 6, 5, 6, 5, 5, 6
Playing Basketball (PB)	LA LL	10, 30, 50, 70, 50, 30, 10 3, 5, 4, 4, 5, 4, 3
Rowing Machine (RM)	LA LL	10, 15, 20, 25, 20, 15, 10 4, 8, 12, 16, 12, 8, 4
Elevator UP (EU)	LA LL	20, 70, 120, 170, 220, 270 0, 50, 100, 150, 200, 250

TABLE 1: Toy dataset: LA represents the y-coordinate of the left arm sensor and LL is the y-coordinate of the left leg sensor

3.1 Proposed CNN_{mts} model

The first step of the CNN_{mts}-X framework is to train a variant of the traditional *CNN* model (CNN_{mts}) for MTS data. To explain the concepts and the algorithms, we will use a running example with the toy dataset in Example 1.

Example 1 (MTS toy data). Table 1 shows a toy dataset with three real phenomena: playing basketball, rowing machine, and Elevator UP. Assume that there are two variables representing the height of the sensors attached to the left arm (LA) and the left leg (LL).

3.1.1 *Structure of* CNN_{mts}

The CNN_{mts} model is based on and improves the model in [16]. Given an MTS training instance (Def. 4) $\begin{pmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,m} \\ \cdots & \cdots & \cdots & \\ v_{A,1} & v_{A,2} & \cdots & v_{A,m} \end{pmatrix}$, Fig. 1 shows the structure of our CNN_{mts} model. This model contains L convolutional layers, $L-1$ pooling layers, and one fully connected layer. In the first convolutional layer, we apply F^1 filters with kernels $K_1^1, \cdots, K_{F^1}^1$ of size $1 \times k$ $(1 < k < m)$ to the subsequences gotten by sliding a window (whose length is also k) over an MTS instance. In particular, a node $h_{i,j}$ in the first convolutional layer H_1^1 is calculated as $h_{i,j} = \sum_{l=j}^{j+k-1} v_{i,l} \cdot x_{l-j+1}$. The different kernels differ in their initial values and are utilized to remove the randomness caused by the kernel initialization. The first convolutional layer has $F^1 \times A \times (m-k+1)$ nodes because each time series in an MTS instance has length m and the number of subsequences gotten from sliding a length-k window for each variable is $m-k+1$.

Our CNN_{mts} model applies downsampling to get pooling layers after each convolutional layer. The first pooling layer is obtained by applying F^1 max pooling filters with size $1 \times r$ to the first convolutional layer. In particular, a node $p_{i,j}$ in the pooling layer P_1^1 is the maximum value of r corresponding consecutive nodes in the immediate previous convolutional layer H_1^1. I.e., $p_{i,j} = max_{l=j}^{j+r-1}\{h_{i,l}\}$. The number of nodes in the first pooling layer is $F^1 \times A \times (m-k-r+2)$. Our *CNN*$_{mts}$ model is different from the model in [16] in that we use sliding windows to get the pooling layers, while the model in [16] utilizes non-overlapping windows. We take the sliding window strategy as we observe that *CNN* models using sliding windows can achieve more stable performance in each iteration.

Other convolutional and pooling layers are constructed in a similar manner although the number of convolutional kernels, the kernel sizes for different convolutional layers, and the sizes of pooling filters can be different. The kernel size of the last convolutional layer is set to be the same as the length of the time series output from the previous pooling layer. The last convolutional layer is not followed by any pooling layer. This is because both the convolutional kernels and the pooling filters are not mixing values from different variables, thus the time series of each variable has been abstracted to exactly one corresponding node in the last convolutional layer. Suppose that the last convolutional layer is calculated using F^L kernels, then each MTS training instance is abstracted as $F^L \times A$ nodes. For n instances, this layer has $n \times F^L \times A$ nodes. The last convolutional layer connects to a fully connected layer which generates the output. The bottom of Fig. 1 shows the size of the matrixes at the different layers of this *CNN*$_{mts}$ structure. Table 2 summarizes the meaning of the major symbols in *CNN*$_{mts}$.

Example 2. For the dataset in Example 1, $E = 3$, $A = 2$, $n = 4$, and $m = 7$. Assume that we set the number of kernels for the different convolutional layers in a *CNN*$_{mts}$ model to be F^1=50, F^2=40, and F^3=30. When "playing basketball" is the positive class, the first two instances are positive instances and the last two instances are negative instances. The input to this *CNN*$_{mts}$ is $4 \times 7 \times 2$ ($n \times m \times A$) and the output of the last convolutional layer is $4 \times 30 \times 2$ ($n \times F^3 \times A$). Similarly, for the other two phenomena, each phenomenon has an output object of size $4 \times 30 \times 2$. Then, the total number of output objects is $3 \times (4 \times 30 \times 2)$ for all the 3 phenomena.

Symbol	Meaning
E	# of distinct event types
A	# of variables for an MTS dataset
n	# of instances for an MTS dataset
m	length of one time series in an MTS dataset
F^i	# of kernels in the ith convolutional layer of *CNN*$_{mts}$

TABLE 2: Symbols

3.1.2 CNN$_{mts}$ *for multiple event types*

Different from existing methods (e.g., [16]), which generally train one *CNN* model for all the event types. Our framework constructs and trains a *CNN*$_{mts}$ model for each event type et with the above described structure by treating the dataset having only two event types (one has et and the other one has $\neg et$). For all the E event types, we train E models in total. The last convolutional layers of all these *CNN*$_{mts}$ models contain $E \times (n \times F^L \times A)$ nodes. These nodes represent each variable as different numbers (instead of subsequences) while encoding the temporal order of the sequences for this variable. The numbers representing the variables may have dependency relationships. However, there is no temporal order among these numbers. Thus, they can be used to extract PVs without considering the temporal dependency relationships among values in sequences. Let us use $\mathcal{L}$ to denote these nodes. The next step in Section 3.2 uses $\mathcal{L}$ to extract PVs.

3.1.3 *Process imbalanced data*

The data for the proposed PV identification problem are generally very imbalanced (one vs rest), simply applying existing feature extraction approaches may not work well in this case. We introduce a new strategy to process imbalanced data when training the proposed *CNN*$_{mts}$.

A *CNN*$_{mts}$ model is trained with multiple epochs [22] and its training terminates when it meets certain criteria such as the model accuracy is good enough. Each epoch consists of $\lceil n/B \rceil$ iterations (or steps) where B is the number of instances used in one iteration. In each iteration, the sampled instances are fed to the model to adjust the model parameters. The B instances used in one iteration is called a *batch*. The batches of each epoch are typically generated in a random manner: the first batch contains B (out of n) randomly selected instances. This random-batch generation strategy generally works well when the data have balanced event types.

Random batch generation with adjusted coefficients. When the data is imbalanced, one major issue with the default batch generation is that the sampled instances in one batch are imbalanced. A widely utilized strategy to alleviate this issue is to give different coefficients to different event types. Instances with rare event types are given higher coefficients so that they can contribute more in deciding the output. For example, if a batch contains 10 and 1000 instances from two event types et_1 and et_2 respectively, then the instance coefficients for et_1 and et_2 can be set to 100 and 1 respectively.

Batch generation with oversampling. We observe that the strategy of adjusting coefficients may still not work well when a batch has extremely unbalanced data. At the same time, we observe that one batch may not utilize all the necessary instances from rare event types because one batch only consists of a subset of instances. Given these two observations, we propose an oversampling strategy, which has been utilized in processing imbalanced data [23]. This oversampling strategy works as follows. After getting the B instances for each batch, we calculate the ratio of instances in different event types. If the ratio is low (e.g., less than 1/3 for a dataset with two event types), we sample more instances from the rare event types to this batch to make the instances for different event types close-to-be balanced. Then, using the actual number of instances of different event types in a batch, we adjust the coefficients of the event types. The sizes of batches generated by this strategy are bigger than B and some instances are utilized several times in different batches for one epoch.

3.2 Extract PVs from intermediate results of *CNN*$_{mts}$ model

The second step of the *CNN*$_{mts}$-X framework extracts significant PVs from $\mathcal{L}$ with $E \times (n \times F^L \times A)$ nodes. We propose Algorithm PVI (representing *PV Identification*, shown in Fig. 2) for this step. This algorithm can use different feature extraction techniques in Step 2(a)iii. Algorithm PVI calculates an important score that each variable contributes to every event type by aggregating the variable importance from all the n instances and F^L kernels.

Algorithm: PVI ($\mathcal{L}$, Y, σ, A)
Input:
(1) $\mathcal{L}$: $E \times n \times F^L \times A$ array from CNN_{mts},
(2) Y: the event-type vector for n instances,
(3) σ and A: see problem definition.
Output: PV_{set}: $\{PV_1, PV_2, \cdots, PV_E\}$ where PV_{et} consists of $\lfloor \sigma \cdot A \rfloor$ PVs for the event type et

1) Initialize an $E \times F^L \times A$ array ω with score zero;
2) For each event type et (et=1$\cdots E$)
   a) For each kernel f (f=1$\cdots F^L$)
      i) Let an $n \times A$ matrix $M_{et,f}$ = $\mathcal{L}[et, 1...n, f, 1...A]$;
      ii) Normalize the values of each variable in $M_{et,f}$;
      iii) $\omega[et, f, 1...A]$ = *aggregateInstance*($M_{et,f}$, Y, et); /*For a fixed event type et and a kernel f, aggregate the importance of each variable from all instances*/
3) $\Gamma[1\cdots E, 1\cdots A]$= *aggregateKernel*($\omega$, σ, E, A, F^L); /*Calculate the importance of each variable by combining the effect of the F^L kernels*/
4) For each event type et
   a) PV_{et} = $\lfloor \sigma \cdot A \rfloor$ variables with top ranks in $\Gamma[et, 1\cdots A]$;
5) Return PV_{set}:$\{PV_1, PV_2, \cdots, PV_E\}$;

Fig. 2: The framework of PV identification

Specifically, PVI works as follows. It first adds up the importance scores of each variable from n instances and saves the scores to an $E \times F^L \times A$ array ω (Step 2, details see below). The score $\omega[et, f, a_i]$ denotes the importance of the i-th variable a_i to the event type et when using kernel f by considering all the instances. Then, it combines the effect of F^L kernels (Step 3). Next, it extracts the PVs for each event type from the combined ranks Γ (Step 4).

PVI calculates the importance scores ω (Step 2) using three steps. First, for each distinct event type and each of the F^L kernels, it gets the node values from $\mathcal{L}$, which form an $n \times A$ matrix $M_{et,f}$ (Step 2(a)i). This matrix is for all the n instances and A variables. Then, from matrix $M_{et,f}$, it calculates the importance of each variable to et by aggregating scores for all the instances (Step 2(a)iii). Before this step, we conduct column-wise normalization for all the values in $M_{et,f}$ using the L^∞-norm so that all the values for one variable (in one column) are comparable.

Example 3. Given the data in Example 1, the size of $\mathcal{L}$ (the input for the PVI Algorithm 2) is $3 \times (4 \times 30 \times 2)$. Step 2 aggregates the features learned from $\mathcal{L}$ using F^3 (which is 30) kernels. The size of $M_{et,f}$ is (4×2). *aggregateInstance* returns the variable importance vector $\omega[et, f, 2]$ (A=2) in Line 2(a)iii and *aggregateKernel* combines the importance scores from each kernel. The final PV_{set} is $3 \times (2 \times 50\%)$ if σ is set to be 50% (E is 3 and A is 2).

To further illustrate the procedure of the algorithm and show how different data structures are changed, Fig. 3 shows a high-level data flow of this algorithm.

3.2.1 A new algorithm LR to calculate variable importance

In the CNN_{mts}-X framework, X can be any feature extraction technique. We propose a new approach that leverages both Linear Discriminant Analysis (*LDA*) [13] and Random

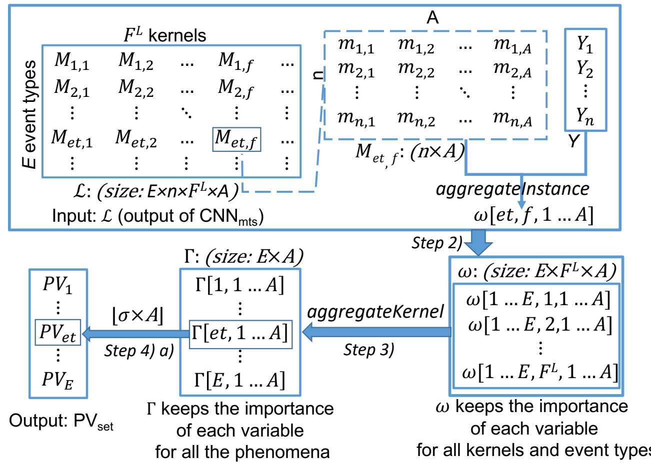


Fig. 3: Flow chart of PVI

Function: aggregateInstanceLR ($M_{et,f}$, Y, et)
Input: (1) $M_{et,f}$: $n \times A$ matrix, (2) Y: event vector for n instances, (3) et: a fixed event type
Output: ω_{et}: a length-A score vector

1) Initialize a length-A vector ω_{et} with returned scores;
2) Create a new length-n vector Y'
3) For the j-th instance in Y
   a) if ($Y[j]$ == et) $Y'[j]$ = 1
   b) else $Y'[j]$ = 0
4) $Model^{LDA}$, ACC^{LDA}=LDA($M_{et,f}$, Y')
5) ω^{LDA}_{et}=$Model^{LDA}$.coefficient
6) $Model^{RF}$, ACC^{RF}=RF($M_{et,f}$, Y')
7) ω^{RF}_{et}=$Model^{RF}$.important_score
8) ω^{LR}_{et}=ω^{LDA}_{et} $\times$ ACC^{LDA} + ω^{RF}_{et} $\times$ ACC^{RF}
9) Return ω^{LR}_{et};

Fig. 4: Calculate variable importance using *LR*

Forest (*RF*) [14]. *LDA* identifies linear combinations of variables as features. Such features can explicitly model the difference between different classes [24]. However, *LDA* cannot directly return the variable importance. We use the weight values to estimate variable importance since a variable with higher weight means it contributes more to the combined feature. *RF* is another widely used technique to rank the importance of variables in a regression or classification problem [25]. *RF* can directly return the variable importance but *RF* focuses on each individual variable instead of the variable combinations.

To make use of good characteristics of *LDA* and *RF*, we propose a new approach *LR* to learn a combined variable importance. Fig. 4 shows this approach as Function *aggregateInstanceLR*. This function calculates the importance of all the A variables for a given event type et and keeps them in a length-A vector ω_{et} (defined at Step 1 in Fig. 4).

More specifically, it creates a new vector Y' whose element values are either zero or one denoting two distinct event types. Here, only two distinct event types are used because PVs are used to distinguish one event type from all the other event types. The value is one when the corresponding actual event type is et and is zero otherwise. *LDA* is conducted using $M_{et,f}$ and the new event vector Y' (shown from Line 4 to Line 5 in Fig. 4). This procedure can be formally represented as shown below.

$$\left(\underbrace{\begin{matrix} M[1,1] & M[1,2] & \cdots & M[1,A] \\ M[2,1] & M[2,2] & \cdots & M[2,A] \\ \cdots & \cdots & \cdots & \\ M[n,1] & M[n,2] & \cdots & M[n,A] \end{matrix}}_{M_{et,f}} \underbrace{\begin{matrix} y'_1 \\ y'_2 \\ \cdots \\ y'_n \end{matrix}}_{Y' \text{ for } et}\right) \rightarrow \underbrace{\begin{pmatrix} c_{et,1} \cdots c_{et,A} \\ c_{\neg et,1} \cdots c_{\neg et,A} \end{pmatrix}}_{\omega_{et}^{LDA}}$$

Note that the values of the first row of ω_{et}^{LDA} is the same as the second row. This is because the first row consists of coefficients that differentiate et and all the other event types (only $\neg et$), and the second row has coefficients to differentiate $\neg et$ from all the other types (only et).

Line 6 to Line 7 in Fig. 4 utilize *RF* to evaluate the variable importance. As shown from Lines 4 and 7, the training accuracies from *LDA* and *RF* are both returned. The training accuracy for each approach is used to weigh the important scores of the variables. The final variable importance is the weighted summation of the variable weights returned from *LDA* and *RF* where the weights are the training accuracies in *LDA* and *RF* (Line 8).

Example 4. Let us follow the previous example to explain Algorithm 4. $M_{et,f}$ is of size 4×2 $(n \times A)$ and the size of Y is 4×1. Line 4 and Line 7 evaluate the variable importance using LDA and RF respectively. For example, the output of Line 4 and Line 5 is $\omega_{et}^{LDA} = \{0.68, 0.32\}$, and $ACC^{LDA} = 1.0$. The output of Line 6 and Line 7 is $\omega_{et}^{RF} = \{0.55, 0.45\}$, and $ACC^{RF} = 0.75$. Line 8 combines the two results and gets the final importance score vector: $\omega_{et}^{LR} = \{0.68, 0.32\} \times 1.0 + \{0.55, 0.45\} \times 0.75 = \{1.09, 0.66\}$. It means that the first variable carries more information than the second variable. Note that the numbers are not exactly the same with those calculated from the model, which show very similar results. We use these numbers here merely for explanation purpose.

3.2.2 Ensemble variable importance

The last step (step 3) of the Algorithm PVI (Fig. 2) is to ensemble variable importance for all the kernels based on the calculated importance scores ω from all the n instances and F^L different kernels. Fig. 5 shows the details of this step.

Function: ***aggregateKernel*** **(**$\omega, \sigma, E, A, F^L$**)**
Output: an $E \times A$ matrix Γ denoting the importance rank of every variable to all event types.

1) Initialize Γ to be an $E \times A$ matrix;
2) For each distinct event type et
   a) Initialize an $F^L \times A$ rank matrix γ with value zero;
   b) For each kernel f and each variable a_i,
      i) Let $\gamma[f, a_i]$ = the rank (in descending order) of $\omega[et, f, a_i]$ among the A elements in $\omega[et, f, 1 \cdots A]$
   c) For each variable a_i, $\Gamma[et, a_i] = agg_{f=1}^{F^L} \gamma[f, a_i]$
3) Return Γ;

Fig. 5: Calculate variable importance by combining results from different kernels

This function ensembles the importance scores for each event type. For an event type et, it first ranks the importance of all the variables for each kernel (Step 2b). The ranking results are kept in an $F^L \times A$ rank matrix γ. Then, it calculates the overall importance of each variable a_i for a fixed event type by aggregating the importance ranks from all the kernels (Step 2c). The importance ranks of all the variables to the different kernels Γ are returned to PVI to extract PVs. Note that we do not directly utilize the importance scores in ω to extract the significant PVs. Instead, we utilize the importance ranks. This strategy is to remove the effect of unbalanced importance scores.

Time: The running time of the CNN_{mts}-*X* PVI approach consists of two stages: learning CNN_{mts} and conducting *X*. Given a dataset with E events, in the worst case we need to learn E CNN_{mts} models. We also note that in many real cases, the number of CNN_{mts} models that we need to train depends on the number of phenomena that people are interested in. We may not need to get the important variables for all the E phenomena. For example, one scientist may only be interested in two phenomena (among hundreds), then our method just needs to train two CNN_{mts} models (instead of hundreds of models) to identify those variables. The exact time complexity of learning the *CNN* model is beyond our control. Thus, we empirically calculate the running time of the PVI algorithms. The results and analysis can be found in Section 4.5.

3.3 Classification using PVs

PVs are identified to differentiate different phenomena. PVs' differentiating effect cannot be tested by directly applying existing classification algorithms because PVs are specific to different phenomena. In order to examine the effect of PVs, we design a PV-based classification algorithm.

Algorithm: PVC ($\mathcal{L}$, Y, A**)**
Input:
(1) X_{tr}: training data: $n_{tr} \times A \times m$,
(2) Y_{tr}: training labels with length n_{tr}
(3) X_{te}: testing data: $n_{te} \times A \times m$,
(4) Y_{te}: testing labels with length n_{te}
(5) PV_{set}: Set of PVs from PVI algorithm in Fig. 2
Output: F_1 vector and overall $Accuracy$

1) Initialize a vector $Prob$: $n_{te} \times E$ with zeros;
2) Initialize a vector F_1: $E \times 1$ with zeros;
3) For each event type et (et=$1 \cdots E$)
   a) Generate training sub-matrix X'_{tr}: $n_{tr} \times |PV_{et}| \times m$ where each instance only contains the time series from PV_{et} variables
   b) Generate testing sub-matrix X'_{te}: $n_{te} \times |PV_{et}| \times m$ where each instance only contains the time series from PV_{et} variables
   c) Create a vector Y'_{tr} with length n_{tr} and a vector Y'_{te} with n_{te} to hold the binary class labels for training and testing data respectively
   d) For the j-th instance in Y_{tr}
      i) if ($Y_{tr}[j] == et$) $Y'_{tr}[j]$ = 1 else $Y'_{tr}[j]$ = 0
   e) For the j-th instance in Y_{te}
      i) if ($Y_{te}[j] == et$) $Y'_{te}[j]$ = 1 else $Y'_{te}[j]$ = 0
   f) Train classification model PVM_{et} using X'_{tr} and Y'_{tr}
   g) Apply PVM_{et} to testing X'_{te} and get prediction Y'_{pred} and assign the prediction probability to Prob[$0 \ldots n_{te}, et$]
   h) $F_1[et]$ = F_1 score calculated from the prediction Y'_{pred} and Y'_{te}
4) Y_{pred}=$argmax(Prob)$
5) $Accuracy$ is calculated based on Y_{te} and Y_{pred}
6) Return F_1 vector and $Accuracy$

Fig. 6: Classifications using PVs

PV classification (PVC) algorithm is used to run classifications based on PVs. Fig. 6 shows the detailed procedure of this algorithm. This algorithm returns a vector of F_1 values for all the E classes (or event types) and the overall testing $Accuracy$. In particular, this algorithm tests the PVs' effect for each et (Line 3). For a given et, it truncates the training and testing data to contain only time series related to this event type's related PVs (Lines 3a-3b). Also, it updates the training and testing labels to contain only et (denoted as one) and $\neg et$ (denoted as zero) (Lines 3c-3e). Then, it trains a classification model to get the classification F_1 value and the prediction probability (Lines 3f-3h) over classes et and $\neg et$. The final prediction is the class label with the highest probability (Line 4). The overall $Accuracy$ is calculated from the final prediction (Line 5).

Function: aggregateInstancePCA ($M_{et,f}, Y, et$)
/*The parameters have the same meaning as those in *aggregateInstanceLR**/
1) Initialize a length-A vector ω_{et} with returned scores;
2) Create a new empty matrix $M'_{et,f}$;
3) For the j-th instance in Y
   a) if ($Y[j]$ == et) Append $M_{et,f}[j]$ to $M'_{et,f}$ as a new row;
4) W = Run *PCA* over $M'_{et,f}$ and get a $A \times A$ matrix;
5) ω_{et}^{PCA} = sum the absolute weight values from all the columns (principle components) in W for each row (variable) and get a length-A vector;
6) Return ω_{et}^{PCA};

Fig. 7: Calculate variable importance using *PCA*

4 Experiments

All the methods are implemented using $Python$ 2.7, and tested on a server with i7-2600 CPU cores @ 3.40GHz and 256GB RAM. TensorFlow (www.tensorflow.org) is used to build our neural network framework.

4.1 Methods to compare

Method	PVI
CNN_{mts}-*LR*	*LR* is used to identify PVs in CNN_{mts}-*X*
CNN_{mts}-*LDA*	*LDA* is used to identify PVs in CNN_{mts}-*X*
CNN_{mts}-*RF*	*RF* is used to identify PVs in CNN_{mts}-*X*
CNN_{mts}-*PCA*	*PCA* is used to identify PVs in CNN_{mts}-*X*
CNN_{mts}-*CPCA*	*CPCA* [5] is used to identify PVs in CNN_{mts}-*X*
LR	*LR* is used to identify PVs without CNN_{mts}-*X*
LDA	*LDA* is used to identify PVs without CNN_{mts}-*X*
RF	*RF* is used to identify PVs without CNN_{mts}-*X*
PCA	*PCA* is used to identify PVs without CNN_{mts}-*X*
CPCA	*CPCA* is used to identify PVs without CNN_{mts}-*X*
BVS-RF	Backward Variable Selection with *RF*
FVS-RF	Forward Variable Selection with *RF*
BVS-CNN	Backward Variable Selection with CNN_{mts}
FVS-CNN	Forward Variable Selection with CNN_{mts}
SFS-FW-CNN	Sequential Forward Selection with CNN_{mts}

TABLE 3: PV selection methods to compare

To better understand the advantages/disadvantages of different PV identification methods, we compare the effect of the PVs selected by the proposed method and several other baseline methods. All the methods are listed in Table 3.

Our proposed method is denoted as CNN_{mts}-*LR*. We also adopt *LDA* and *RF* alone in the CNN_{mts}-*X* framework and get two baseline methods CNN_{mts}-*LDA* and CNN_{mts}-*RF*. In particular, CNN_{mts}-*LDA* and CNN_{mts}-*RF* return w_{et}^{LDA} and w_{et}^{RF} respectively in Fig. 4. Furthermore, since Principal Component Analysis (*PCA*) [26], [27] is another well-recognized classical feature extraction technique, we adopt *PCA* in the CNN_{mts}-*X* framework and get CNN_{mts}-*PCA*. Another approach based on Common *PCA* (*CPCA*) [5] can identify important global variables, we adopt *CPCA* in our framework and get CNN_{mts}-*CPCA*. CNN_{mst}- *PCA* (or CNN_{mts}-*CPCA*) calls *PCA* (or *CPCA*) only on the instances from class et instead of on all the instances. The details of CNN_{mts}-*PCA* can be found from Fig. 7. For CNN_{mts}-*CPCA*, *CPCA* is used at Line 4.

We also compare our proposed method with other techniques that does not employ our proposed CNN_{mts}-*X* framework. Corresponding to the five methods that utilize CNN_{mts}-X framework, the five baseline approaches are *LR*, *LDA*, *RF*, *PCA*, and *CPCA*. These five baseline methods learn importance scores of each variable for different event types and select the variables with the top $\lfloor \sigma \cdot A \rfloor$ absolute importance scores as PVs. For the *LR* method, the importance scores of all the variables to an event type et are the weighted summation of the variable scores returned from *LDA* and *RF*. The variable scores from *LDA* are calculated based on the coefficients ($c_{et,1}, \cdots, c_{et,A}$), while the variable scores from *RF* directly come from the trained *RF* model. For the *PCA* (or *CPCA*) methods, the importance scores are learned as follows. For each event type, we first conduct *PCA* (or *CPCA*) on all the training instances with event type et. Then, we calculate each variable's importance by adding the absolute weight values of the PCs for this variable. Higher weight values carry more importance.

PVs can also be identified by using other existing variable selection approaches with slight changes. We compare our methods with three other such approaches, Forward wrapper Variable Selection (*FVS*), Backward wrapper Variable Selection (*BVS*), and Sequential Forward Selection with Fixed Width (*SFS-FW*) [11]. *FVS* and *BVS* are the most basic representatives. *SFS-FW* is the newest recommended paper before 2013 [12], which is a very effective approach with similar performance to *FVS* [12]. More recent works either focus on specific data domains (e.g., [28], [29], [30]) or specific classifiers (e.g., the work in [31] improves the classification performance of *KNN*, but not other classifiers). Some recent approaches may gain better classification performance but the time cost is still higher than *SFS-FW* [32], [33]. All these methods are wrapper methods, which need to include a classifier in the evaluation step. The code for FVS and BVS is from [34], which use *decision tree* and *SVM* as the classifier to evaluate the performance while we utilize CNN_{mts} and *RF* as the evaluation classifiers because CNN_{mts} has shown the best classification performance and *RF* is the most efficient classifier in our problem (Testing time in Table 23). We denote these methods as *FVS-RF*, *FVS-CNN*, *BVS-RF*, *BVS-CNN*. For *SFS-FW*, we only apply CNN_{mts} as the evaluation classifier because of its better classification performance than RF. This method is denoted as *SFS-FW-CNN*.

The effect of the proposed PVs are also compared with the effect of all the variables (denoted as ***All-variables***) and top global variables (denoted as $\boldsymbol{CNN_{mts}}$***-LR-GV***). The *All-variables* method directly feeds all the values v_{ij} in an MTS to E CNN_{mts} classifiers for the E event types. CNN_{mts}-*LR-GV* is designed based on CNN_{mts}-*LR* method. It utilizes the intermediate results Γ (Step 3 in the PVI algorithm in Fig. 2) from CNN_{mts}-*LR*. From $\Gamma = \begin{pmatrix} \Gamma[1,1] & \cdots & \Gamma[1,A] \\ \Gamma[2,1] & \cdots & \Gamma[2,A] \\ \cdots & \cdots & \cdots \\ \Gamma[E,1] & \cdots & \Gamma[E,A] \end{pmatrix}$, each column's values (the importance rank of each variable for different event types) are added to get the overall importance rank of the variables. The $\lfloor \sigma{\cdot}A \rfloor$ variables with the top overall ranks are chosen as significant global variables.

Dataset	n	E	A	m
DSA	9120	19	45	125
RAR	35350	33	117	20
ARC	78051	18	107	30
ARC_{fixed}	78051	18	107	30
ASL	2565	95	22	90

TABLE 4: Dataset statistics

4.2 Experimental settings

(1) Datasets: We use five real datasets to test the performance of our approaches. The first dataset is the Daily and Sports Activities data (denoted as DSA) [35]. The second dataset is extracted from the ideal-placement scenario in the REALDISP Activity Recognition data (denoted as RAR) [36]. The third and the fourth datasets are the Activity Recognition Challenge data from opportunistic activity recognition systems for subject 1 (denoted as ARC) [37]. The fourth dataset also comes from the ARC dataset, but it has fixed training and testing portion used in [16]. This dataset is denoted as ARC_{fixed} and utilized for comparison with [16]. The last dataset is for Australian Sign Language (ASL) [38]. The detailed statistics for the datasets are shown in Table 4.

For DSA, RAR, and ARC datasets, we run ten-fold cross-validation to get stable results. For ASL, we run three-fold cross validation. We did not use ten folds because the number of instances in each class is not as many as in the other datasets.

(2) Evaluation methods: We utilize two ways to evaluate the effectiveness of the selected PVs: (a) conducting classification using the selected PVs, significant global variables, and all the variables (Sections 4.4.1-4.4.3) and (b) manually examining the meaning of the extracted PVs through surveys (Section 4.4.4). Please note that the purpose of classification is mainly to evaluate the selected PVs, instead of merely achieving better classification performance.

The PVs are one type of features. They are identified to differentiate different phenomena. PVs are identified for each phenomenon and can be used in binary classification (when a user is only interested in one phenomenon) or Multi-Phenomena Classification (*MPC*) when a user is interested in e ($1 < e \leq E$) phenomena.

Binary classification: When PVs are used for binary classification for an event type et that a user is interested in, the binary classification strategy truncates the training and testing data to contain only time series related to this event type's PVs. Also, it updates the training and testing labels to contain only et and $\neg et$. Then, it trains a binary classification model to get the classification F_1 value and the prediction probability over classes et and $\neg et$. The final prediction is the class label with the highest probability. The overall *Accuracy* is calculated from the final prediction. The more detailed pseudo-code can be found from Fig. 6.

MPC: When a user is interested in e ($1 < e \leq E$) event types, we design two PV-based multi-phenomena classification methods.

- *MPC-ALL-PV* for classifying e ($e \leq E$) phenomena: This approach trains E classifiers for all the E phenomena. Given a testing instance, the prediction from one classifier (for event type et) is the probability that the testing instance is predicted as et. The final event-type prediction of this instance is the type with the highest probability. Even when $e < E$, this method still need to run E classifiers.
- *MPC-PV*: This method is different from *MPC-ALL-PV* in that it only trains e CNN_{mts} to capture the corresponding PVs for the e ($e < E$) phenomena. Given a testing sample, we first calculate the probabilities that this sample belongs to the e phenomena. Then, we either assign the sample to the phenomenon with the highest probability (bigger than 0.5) or assign it to none of those e phenomena if all the probabilities are smaller than 0.5.

For comparison purpose, we further implement two other baselines for *MPC*.

- *MPC-basic*: a very basic Multi-Phenomena CNN_{mts} model without using PVs. It just needs to train one CNN_{mts} model to directly classify one instance to one of the E phenomena.
- *MPC-AV*: *MPC-AV* trains e classifiers, which is similar to *MPC-PV*. Different from *MPC-PV*, it does not use PVs to train the classifiers. Instead, it uses all the variables to train the e classifiers.

To eliminate the bias of classification techniques, we utilize four widely adopted classification methods, convolutional neural network (*CNN*) [39], k-nearest neighbors (*KNN*) [40], support vector machine (*SVM*) [41] and random forest (*RF*) [14].

(3) Evaluation measurements: We report the F_1 and *Accuracy*, to show the classification performance. Note that the traditional F_1 is used to measure the performance of binary classifiers. In our experiments, each dataset has more than two event types. We calculate F_1 for each event type by treating all the instances belonging to this type as positive and all the other instances as negative.

(4) Parameter setting: The parameters used to train the CNN_{mts} models for both the PV selection and for the classification task are the same. The numbers of convolutional layers and pooling layers are set to be 3 and 2 respectively. For the convolutional layers, the kernel sizes k are 50, 30, and 20. For the pooling layers, the filter size r is 2. The maximum number of epochs is 5, and the batch size B is 100. For the classifiers, *KNN* sets the parameter K to be 1. *LibSVM* uses balanced class weights and sets Radial Basis Function (RBF) as the kernel. We are aware that setting different parameter values to achieve good classification performance is still an open problem and that is not the focus of this paper. Meanwhile, we also run experiments with different parameter values to justify our parameter setting (Section 4.4.6).

4.3 Compare the proposed CNN model with others

This set of experiments compares the proposed CNN_{mts} model with another *CNN* baseline, Fully Convolutional Networks (*FCN*) [15]. *FCN* is proposed as a strong baseline for image classification. *FCN* has only a global pooling layer before the final output layer (instead of a pooling layer after every convolutional layer). *FCN* may not be a good choice in MTS feature selection because the pooling layer after each convolutional layer helps identify the similar features in a time range. For example, two people are conducting the same activity, hand up-down movement, with different speed: the first person moves his/her hand slowly, with 20 cm up/down per second, and the second person move his/her hand faster, with 40 cm up/down per second. Table 5 (a) shows the location of the y sensor on one hand for five time stamps. Table 5 (b) shows the results after a max pooling with size 1×2. It is clear that this pooling layer amplifies the similarity between these two time sequences. The next convolutional layer can utilize the amplified similarity. However, in *FCN* (without a pooling layer after each convolutional layer), the next convolutional layer cannot utilize any amplified similarity.

time stamp	1	2	3	4	5
Person 1	30 cm	50 cm	70 cm	50 cm	30 cm
Person 2	0	40 cm	80 cm	40 cm	0

(a) Before pooling layer

time stamp	1	2	3	4
Person 1	50 cm	70 cm	70 cm	50 cm
Person 2	40 cm	80 cm	80 cm	40 cm

(b) After pooling layer (size = 2)

TABLE 5: Example: Locations of sensor y on a hand

Method	DSA	RAR	ARC	ARC_{fixed}	ASL
CNN_{mts}-*LR*	**0.928**	**0.946**	**0.962**	**0.628**	**0.788**
FCN-LR	0.889	0.941	0.955	0.611	0.769

(a) CNN_{mts} classifier

Method	DSA	RAR	ARC	ARC_{fixed}	ASL
CNN_{mts}-*LR*	**0.872**	**0.897**	**0.962**	**0.530**	**0.634**
FCN-LR	0.761	0.892	0.955	0.500	0.604

(b) *KNN* classifier

Method	DSA	RAR	ARC	ARC_{fixed}	ASL
CNN_{mts}-*LR*	**0.721**	**0.645**	**0.730**	**0.516**	**0.407**
FCN-LR	0.685	0.547	0.712	0.520	0.397

(c) *LibSVM* classifier

Method	DSA	RAR	ARC	ARC_{fixed}	ASL
CNN_{mts}-*LR*	0.786	**0.710**	**0.767**	**0.355**	**0.553**
FCN-LR	**0.802**	0.669	0.764	0.347	0.510

(d) *RF* classifier

TABLE 6: *Comparison of* CNN_{mts}-*LR* and *FCN-LR* (F_1 using the top 30% PVs)

Method	CNN	KNN	LibSVM	RF
CNN_{mts}-*LR*	**0.850**	**0.779**	**0.604**	**0.634**
FCN-LR	0.833	0.742	0.572	0.618

TABLE 7: *Comparison of* CNN_{mts}-*LR* and *FCN-LR* (averaged F_1 using the top 30% PVs)

We use the *FCN* model to replace the CNN_{mts} model in our proposed CNN_{mts}-*LR* method and get a *FCN-LR* method. Table 6 and Table 7 show the detail and averaged F_1 results over all five datasets of comparing *FCN-LR* and CNN_{mts}-*LR*. It can be observed that features learned from CNN_{mts}-*LR* outperforms the features from *FCN-LR* in almost all cases.

Method	DSA	RAR	ARC	ARC_{fixed}	ASL
CNN_{mts}-*LR*	**0.961**	**0.971**	**0.982**	**0.889**	0.797
FCN-LR	0.893	0.965	0.953	0.884	0.778

(a) CNN_{mts} classifier (CNN_{mts}-*LR* always ranks top 1)

Method	DSA	RAR	ARC	ARC_{fixed}	ASL
CNN_{mts}-*LR*	0.898	0.900	0.981	0.856	0.562
FCN-LR	0.739	0.875	0.979	0.845	0.511

(b) *KNN* classifier (CNN_{mts}-*LR* always ranks top 3)

Method	DSA	RAR	ARC	ARC_{fixed}	ASL
CNN_{mts}-*LR*	**0.768**	0.649	**0.910**	0.840	0.207
FCN-LR	0.512	0.524	0.913	0.833	0.200

(c) *LibSVM* classifier (CNN_{mts}-*LR* always ranks top 3)

Method	DSA	RAR	ARC	ARC_{fixed}	ASL
CNN_{mts}-*LR*	0.913	0.897	0.949	**0.889**	0.667
FCN-LR	0.915	0.884	0.949	0.878	0.633

(d) *RF* classifier (CNN_{mts}-*LR* always ranks top 2)

TABLE 8: *Comparison of* CNN_{mts}-*LR* and *FCN-LR* ($Accuracy$ using the top 30% PVs)

The overall $accuracy$ results in Table 8 are similar to the F_1 results. Therefore, we use CNN_{mts} instead of *FCN* as the *CNN* classifier.

4.4 Effectiveness analysis.

This section hows the identified PVs can be used to differentiate different phenomena for binary classification (Section 4.4.1), MPC (Section 4.4.3), and survey (Section 4.4.4).

4.4.1 Compare the effect of PVs using different PV selection approaches

This section compares the effect of PVs selected by the PV selection approaches listed in Table 3.

Method	CNN	KNN	LibSVM	RF
CNN_{mts}-*LR*	**0.850**	**0.779**	**0.604**	**0.634**
CNN_{mts}-*LDA*	0.759	0.691	0.427	0.559
CNN_{mts}-*PCA*	0.740	0.685	0.336	0.454
CNN_{mts}-*CPCA*	0.700	0.635	0.478	0.502
CNN_{mts}-*RF*	0.803	0.731	0.505	0.609
LR	0.810	0.726	0.561	0.612
LDA	0.718	0.652	0.448	0.501
PCA	0.701	0.641	0.361	0.493
CPCA	0.721	0.617	0.464	0.475
RF	0.814	0.740	0.478	0.618

TABLE 9: Comparison of 10 PV identification methods (averaged F_1 over all five datasets)

Method	CNN	KNN	LibSVM	RF
CNN_{mts}-*LR*	**0.920**	**0.839**	**0.675**	**0.863**
CNN_{mts}-*LDA*	0.823	0.763	0.546	0.826
CNN_{mts}-*PCA*	0.832	0.747	0.471	0.755
CNN_{mts}-*CPCA*	0.789	0.710	0.544	0.722
CNN_{mts}-*RF*	0.867	0.790	0.601	0.853
LR	0.896	0.799	0.653	0.858
LDA	0.786	0.744	0.551	0.803
PCA	0.816	0.728	0.528	0.746
CPCA	0.827	0.639	0.559	0.741
RF	0.899	0.706	0.594	0.853

TABLE 10: Comparison of 10 PV identification methods (averaged $Accuracy$ over all five datasets)

For the first *ten approaches*, we report the averaged F_1 and averaged *Accuracy* over all the five datasets in Table 9 and

Table 10 respectively. These tables show that the proposed approach achieves the best averaged F_1 and averaged *Accuracy*. The detailed F_1 and *Accuracy* results for each dataset and the individual F_1 values for all phenomena in different datasets can be found in Appendix A. The results showing the similar performance.

Method	DSA	ASL
CNN_{mts}-LR	**0.928**	**0.788**
BVS-RF	0.891	0.647
FVS-RF	0.908	0.630
FVS-CNN	-	0.673
BVS-CNN	-	0.651
SFS-FW-CNN	0.910	0.662

(a) F_1

Time (hours)	DSA	ASL
CNN_{mts}-LR	**2.2**	**8.9**
BVS-RF	97	3.7
FVS-RF	17.0	0.8
FVS-CNN	-	33
BVS-CNN	-	50
SFS-FW-CNN	33	22

(b) PV identification time (sec)

TABLE 11: Comparison of 6 PV identification methods

The other *five* methods are only run over the DSA and ASL datasets, representing human activities and the sign language, because they are extremely time consuming (Table 11(b)). These wrapper methods need to evaluate all the variables in each iteration for each phenomenon. They may not be suitable for high-dimensional dataset due to the high time cost [42]. The results for *FVS-CNN* and *BVS-CNN* on the DSA dataset have not been filled because the running time is unreasonably long (more than 7 days). The results in Table 11 show that the proposed *CNN_{mts}-LR* approach achieves better F_1 values than *FVS*, *BVS* and *SFS-FW-CNN*. There are two major reasons for this result. First, thee methods choose one variable in each iteration. When the variable is not chosen correctly (because of the bias of the evaluation classifier), the method has no way to correct its wrong choice. Second, these methods do not consider the combined effect of the chosen variables because they add/remove one variable each time. However, our method combines the chosen PVs in the last fully connected layer of *CNN_{mts}* to implicitly leverage the combined effect of the PVs.

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
All-variables	**0.956**	**0.953**	0.975	0.592	**0.802**
CNN_{mts}-LR	0.928	0.946	**0.962**	**0.628**	0.788
CNN_{mts}-LR-GV	0.895	0.817	0.859	0.295	0.591

(a) *CNN_{mts}* classifier

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
All-variables	0.778	**0.900**	0.920	0.440	**0.654**
CNN_{mts}-LR	**0.872**	0.897	**0.962**	**0.530**	0.634
CNN_{mts}-LR-GV	0.742	0.715	0.658	0.162	0.448

(b) *KNN* classifier

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
All-variables	0.604	0.593	0.695	**0.532**	0.392
CNN_{mts}-LR	**0.721**	**0.645**	**0.730**	0.516	**0.407**
CNN_{mts}-LR-GV	0.555	0.502	0.282	0.177	0.202

(c) *LibSVM* classifier

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
All-variables	**0.789**	0.616	0.756	0.299	**0.559**
CNN_{mts}-LR	0.786	**0.710**	**0.767**	**0.355**	0.553
CNN_{mts}-LR-GV	0.731	0.458	0.484	0.094	0.339

(d) *RF* classifier

TABLE 12: F_1 comparison using all the variables, top 30% PVs, and top 30% of GVs

Method	CNN	KNN	LibSVM	RF
All-variables	**0.856**	0.738	0.563	0.604
CNN_{mts}-LR	0.850	**0.779**	**0.604**	**0.634**
CNN_{mts}-LR-GV	0.691	0.545	0.344	0.421

TABLE 13: Averaged F_1 (over all five datasets) using all the variables, top 30% PVs, and top 30% of GVs

4.4.2 *Compare the effect of PVs, selected global variables, and all the variables*

This section evaluates the performance of (i) the PVs found using the proposed *CNN_{mts}-LR* method, (ii) the global variables (GVs) discovered using *CNN_{mts}-LR-GV*, which is based on *CNN_{mts}-LR*, as well as (iii) all the variables (denoted as *All-variables*). *CNN_{mts}-LR-GV* is based on *CNN_{mts}-LR* because the experimental results in Section 4.4.1 demonstrate that classifications using PVs from *CNN_{mts}-LR* return the best performance. For this set of experiments, the *All-variables* approach uses all the variables to run classification, while *CNN_{mts}-LR* and *CNN_{mts}-LR-GV* select approximately 30% of all the variables. Different classification algorithms are applied in oder to get unbiased results.

The F_1 (by treating each phenomenon as a positive class and running binary classification) using different classifiers are collected and the detailed/averaged F_1 results over five datasets are shown in Table 12/Table 13, respectively. The results show that classification using the top 30% of PVs from *CNN_{mts}-LR* achieves similar or even better F_1 values compared with the classification results using all the variables. Note that our method does not perform better than the *All-variables* method. It is mainly because of the characteristics of the data. When the dataset has noisy variables, our PV selection approach is able to identify the important non-noisy variables and utilize them for classification and such classification generally has better performance than the method using all the variables. On the other hand, when all the variables in a dataset are very useful (i.e., no noisy variables), the PV selection approach then misses some variable information and gets slightly worse performance than the method using all the variables.

These results indicate that the 30% PVs identified from *CNN_{mts}-LR* are able to capture the significant variables and discard other noisy variables. The results also demonstrate that classifications using PVs generate much better F_1 values than classifications using GVs from *CNN_{mts}-LR-GV*. This is consistent with our expectation and intuition since GVs are important variables for all the class labels and PVs are important variables for different class labels. We also get the overall $Accuracy$ values (Appendix B), which show similar results as F_1, and the individual F_1 values for all phenomena in different datasets (Appendix B), which are consistent with the rank of the overall F_1 values.

Note that all the above results are obtained by conducting binary classifications using *CNN_{mts}-LR*, *CNN_{mts}-LR-GV*, and *All-variables*. These methods can be easily adapted to conduct MPC. The different MPC strategies (e.g., *MPC-PV*, *MPC-ALL-PV*) using PVs are described in Section 4.4.3. Such strategies can be applied to *CNN_{mts}-LR-GV* by changing PVs to GVs. We did not conduct further experiments using GVs because GVs are not effective as PVs and are not the focus of this work.

4.4.3 Effect of PVs on multi-phenomena classification

This section discusses the effect of using the identified PVs for MPC.

Method	*MPC-PV*	*MPC-ALL-PV*
DSA_5	0.967	0.969
ASL_5	0.891	0.887

(a) Overall Accuracy

Method	*MPC-PV*		*MPC-AV*	
Time (sec.)	T_{PV}	T_{MPC}	T_{PV}	T_{MPC}
DSA_5	1483	1277	7772	4943
ASL_5	1612	631	32151	11420

(b) Running time (sec)

TABLE 14: Results for e phenomena classification ($e < E$)

It is common that a scientist may only be interested in e ($e < E$) phenomena instead of all the E phenomena. This set of experiments test the *strategies of MPC using PVs over* e *phenomena* by comparing *MPC-PV* and *MPC-ALL-PV*, which builds e and E classifiers respectively. We randomly pick e ($e = 5$) phenomena from DSA and ASL datasets and repeat this selection ten times to get generalized results. The overall accuracy and running time are presented on Table 14. T_{PV} represents the PV identification time and T_{MPC} denotes the MPC training time. It shows that *MPC-PV* achieves similar (slightly better) results with much less PV identification time (T_{PV}) and MPC training time.

Method	*MPC-basic*	*MPC-PV*	*MPC-AV*
DSA	0.59	0.91	0.96
RAR	0.67	0.93	0.97
ASL	0.54	0.81	0.84

(a) Overall Accuracy

Method	*MPC-basic*		*MPC-PV*		*MPC-AV*	
Time (sec.)	Train	Test	Train	Test	Train	Test
DSA	1060	5	4943	35	14559	81
RAR	1586	5	5361	14	37713	166
ASL	284	1	11420	33	28037	110

(b) Running time (sec)

TABLE 15: Results for E phenomena classification

Then, we examine whether using PVs can help achieve better classification results compared with classifiers without using PVs (*MPC-basic* and *MPC-AV*). The results over three datasets (DSA, RAR, and ASL) are shown in Table 15. It is very clear that the classification performance of *MPC-PV* is better than *MPC-basic*. Note that *MPC-basic* builds only one classifier, thus its time is least. The accuracy of *MPC-PV* is slightly worse than *MPC-AV* because of the same reason for that the *All-variables* method slightly outperforms CNN_{mts}*-LR*. We also note that *MPC-PV* uses much less time than *MPC-AV*, which is an advantage of *MPC-PV*.

Note that CNN_{mts}*-LR-GV* can be easily adapted to conduct MPC. The different MPC strategies (e.g., *MPC-PV*, *MPC-ALL-PV*) using PVs can be applied to CNN_{mts}*-LR-GV* by changing PVs to GVs. We did not conduct further experiments using GVs because GVs are not as effective as PVs and are not the focus of this work.

4.4.4 User studies of the effectiveness of PVs

In this section, we conduct user studies to examine the capability of PVs to differentiate different phenomena. .

Survey setting: We randomly pick 10 phenomena from the DSA dataset (representing human activities) and 10 phenomena from the ASL dataset (representing sign language). For each phenomenon, we collect the top-5 PVs returned by our method CNN_{mts}*-LR* and the second best method. CNN_{mts}*-LDA* and *LR* are the second best method for the DSA and ASL datasets respectively based on the results of Table 25 in Appendix A. For 6 phenomena, on which the returned two PV sets differ by at most one PV, we do not ask users to input their preference because of their trivial differences. For the remaining 14 phenomena, we present the two PV sets to users. To avoid bias, we change the display order of the two PV sets for different phenomena. For the phenomena related to ASL, we put a short video for each sign (i.e., phenomenon) to educate users because they may not be familiar with sign language. A user is asked to choose from one of the three options (a) PV set 1 is better, (b) PV set 2 is better, and (c) the two PV sets are similar (tie).

We recruited 14 (undergraduate and graduate) student volunteers from different disciplines to work on the survey to avoid biased judgement with the same background.

# of phenomena	# of votes for preferred PV sets returned by method		
Total (14)	CNN_{mts}-LR	2nd best method	Tie
1	14	0	0
1	13	0	1
3	13	1	0
2	12	1	1
3	11	3	0
1	10	4	0
1	9	5	0
1	7	5	2
1	6	7	1

TABLE 16: Summary of survey results

Table 16 presents the survey results. The first column counts the number of phenomena with the corresponding voting results listed in the following columns at the same row. The last three columns show the # of volunteers voting for the preferred methods. On most phenomena, volunteers agree that PVs from the CNN_{mts}*-LR* can better differentiate the corresponding phenomenon.

We also ask the volunteers to select the ground truth PVs for each phenomena. The survey results show that different volunteers seldom agree on the same (or even similar) variables to be the ground truth for one phenomenon. This indicates that creating ground truth for such datasets is not easy. Methods like ours can at least provide users reasonable candidates.

Manual Checking setting: Furthermore, we manually verify the usefulness of the extracted features. We show a few number of PVs learned from our CNN_{mts}*-LR* model for the different classes in the DSA and ASL datasets where DSA represents the human activity domain and ASL represents the sign language domain. In the DSA dataset, there are three groups of variables, accelerometers, gyroscopes, and magnetometers. An accelerometer records the tilt relative to the earth's surface, a magnetometer keeps the heading direction if a person holds the sensor that is parallel to the ground, and a gyroscope sensor keeps rotational velocity without any absolute reference. Those sensors are placed on the torso, left arm, right arm, left leg, and right leg. Table 17(a) shows the top 6 PVs learned on

Phenomena	*Playing Basketball*	*Rowing machine*
Top 1 PV	Y gyroscopes (left arm)	X magnetometers (left leg)
Top 2 PV	X gyroscopes (left arm)	X magnetometers (right leg)
Top 3 PV	Y gyroscopes (right arm)	X magnetometers (torso)
Top 4 PV	X gyroscopes (right arm)	X accelerometers (left arm)
Top 5 PV	Y accelerometers (left arm)	X accelerometers (right arm)
Top 6 PV	Y accelerometers (right arm)	X accelerometers (left leg)

(a) DSA Dataset

Phenomena	*Please*	*stubborn*
Top 1 PV	Roll (right hand)	Middle finger bend (left hand)
Top 2 PV	Y position (right hand)	Middle finger bend (right hand)
Top 3 PV	Forefinger bend (right hand)	Little finger bend (left hand)
Top 4 PV	Middle finger bend (right hand)	Forefinger bend (right hand)
Top 5 PV	Little finger bend (right hand)	Forefinger bend (left hand)
Top 6 PV	Yaw (right hand)	Pitch (right hand)

(b) ASL Dataset

TABLE 17: Top 6 PVs selected for two phenomena

DSA. The first phenomenon is "playing basketball", which is an activity of bouncing the basketball repeatedly using two arms. We can see that all the top 6 PVs for this activity are related to left arm and right arm. In addition, the x and y gyroscopes for both two arms are the top 4 attributes because the arms are rotating while bouncing a ball. The 5th and 6th attributes are the y accelerometers for two arms. They are picked to capture the up-down movement while playing. These PVs are reasonable to identify the "playing basketball" activity. Consider another activity in the DSA dataset, "Rowing machine", which is an activity requiring the whole body to move, we show its top six variables in the third column of Table 17(a). These 6 PVs represent the sensors from torso, arms, and legs. Magnetometers and accelerometers are more helpful to identify this activity because rowing is a forward-backward activity.

Table 17(b) presents the analysis of two phenomena from the ASL dataset. The first phenomenon is "Please" [43], which bends all four fingers without the thumb finger of the right hand. Meanwhile, the right hand needs to move upward-downward. The corresponding top 6 PVs are all from the right hand. The top PVs contain the roll, y position, yaw sensors (which are gyroscopic devices) from the right hand and the bend sensors from three fingers (not the thumb finger). The second phenomenon is "Stubborn" [44], for which both hands bend four fingers without the middle finger, and two hands move both vertically and horizontally. The selected top 6 PVs are bend sensors and the x accelerometers from both hands, which are consistent with the sign language.

4.4.5 Compare the effect of PVs and existing work

This set of experiments compares the classification performance using the PVs selected by *CNN$_{mts}$-LR* and two other state-of-the-art approaches: the *CNN* model in [16] and multivariate shapelet in [6]. The PVs from *CNN$_{mts}$-LR* cannot be directly applied to the *CNN* [16] model since the PVs found in our work are phenomenon specific. However, we still report the F_1 and overall accuracy for reference. We directly get the F_1 and accuracy from [16] (without smoothing) using ARC_{fixed} (*RF* is not used in [16]).

Table 18 (a) and (b) report the F_1 and the overall accuracy respectively. The results show that classification using the PVs gets better F_1 and overall accuracy. This is due to the new variant of the *CNN* model, *CNN$_{mts}$*, and the PV based classification algorithm, PVC.

Method	*CNN$_{mts}$*	*KNN*	*LibSVM*
CNN$_{mts}$-LR	**0.628**	**0.530**	**0.516**
CNN in [16]	0.555	0.427	0.456

(a) F_1

Method	*CNN$_{mts}$*	*KNN*	*LibSVM*
CNN$_{mts}$-LR	**0.889**	**0.856**	**0.840**
CNN in [16]	0.870	0.793	0.838

(b) $Accuracy$

TABLE 18: Overall comparison of *CNN$_{mts}$-LR* and *CNN in [16] (ARC$_{fixed}$)*

Next, we compare the classification performance using shapelets. Note that the focus of shapelet extraction is different from PV identification: shapelets are the important subsequences in the sequences of multiple variables, while PVs are the important variables. I.e., they are orthogonal and complement with each other. Given these differences, we compare the classification performance using shapelets that are generated from the overall MTS and from the sequences for PVs. We have implemented two versions of the shapelet generation. The first version directly extracts shapelets from the overall MTS (denoted as *Shapelet$_{all}$*). The second version extracts shapelets from the sequences whose corresponding variables are identified as PVs (denoted as *Shapelet$_{PV}$*).

KNN	F_1	$Accuracy$	Time (sec.)
Shapelet$_{all}$	0.881	**0.917**	8^5
Shapelet$_{PV}$	**0.888**	0.914	$\mathbf{1.7^5}$

TABLE 19: Performance comparison of shapelets extracted from the overall MTS and from PV sequences (DSA)

Table 19 presents the classification results using the shaplet features of the two versions for the DSA dataset. Shapelet generation is known to be time-consuming [45]. Therefore, the DSA dataset is used because it has fewer instances than RAR and ARC datasets and has a much smaller number of classes than ASL. The results show that the shaplets generated using PVs can achieve similar accuracy as the shapelets identified from all the variables, while the *Shapelet$_{PV}$* uses only $\sim 20\%$ of the time used for *Shapelet$_{all}$*.

MASK	shapNum	shapMin	shapMax	Time (Sec)
ASL	10	3	5	$> 1.4 \times 10^4$ (4 hours)

(a) Running Time

Method	*CNN$_{mts}$*	*KNN*	*LibSVM*	*RF*
CNN$_{mts}$-LR	**0.788**	**0.634**	**0.407**	**0.553**
MASK in [46]	0.473	0.382	0.214	0.347

(b) F_1

TABLE 20: Performance comparison with *CNN$_{mts}$-LR* and MASK on ASL

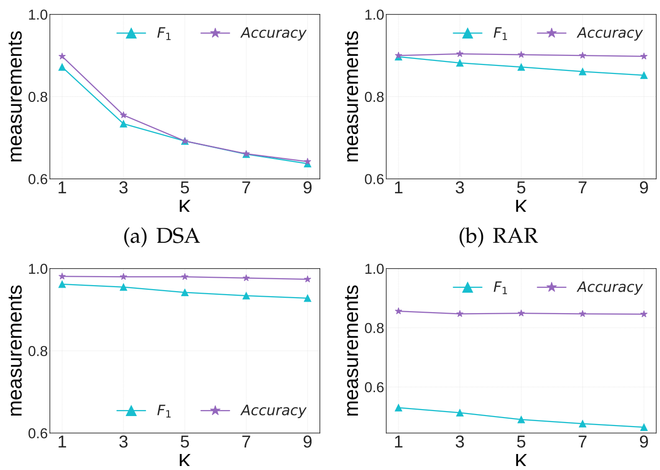
Table 20 compares our proposed approach with another recent approach, MASK [46]. MASK identifies the shapelet from time-series sequences and returns a mask to evaluate the importance of different variables. We note that MASK is very time consuming and performs poorly on imbalanced data. For the smallest data set (ASL), MASK runs around 4 hours for *one* class even when the parameter values are set to be small (for larger parameter values, the algorithms runs much longer time). The setting details and running time are shown in Table 20(a). Furthermore, Table 20(b) shows that the CNN_{mts} achieves ∼20% better F_1 scores than MASK.

4.4.6 Effect of parameters

We first show how the number of PVs affect the classification performance using the RAR dataset.

σ	F_1	Accuracy
10%	0.912	0.925
20%	0.933	0.962
30%	0.946	0.971
50%	0.948	0.970

TABLE 21: Classification performance (F_1, *Accuracy*) using PVs selected by CNN_{mts}-*LR* (RAR dataset, CNN_{mts} classifier, ten-fold)



(a) DSA (b) RAR (c) ARC (d) ARC$_{fixed}$

Fig. 8: *KNN* with varying K

Table 21 reports the F_1 and *Accuracy* from the ten-fold cross validation and for all the event types. It can be observed that the performance improves with the increase of σ. However, when σ is more than 30%, the performance increase does not grow much.

Then we evaluates the performance of the *KNN* classifier with varying K values (Fig. 8). In this part of the experiment, we pick the odd numbers of a concatenate sequence from 1 to 9. The results show that *KNN* with $K = 1$ returns the best averaged F_1 for all four datasets and the averaged F_1 reduces with the increase of K. So we set $K = 1$ for our *KNN* classifier.

4.5 Efficiency Analysis

This section shows (a) the running time of different PV identification methods, (b) the time to train different classifiers using sequences for PVs, and (c) the time to predict the event type of one testing instance for different datasets.

Table 22 presents the running time for building the CNN_{mts}-*X* framework using nine folds (with ten-fold cross validation) of the dataset (two folds for ASL). The running time for CNN_{mts}-*X* on ARC$_{fixed}$ is not included in Table 22 and Table 23 due to the table width limitation. This time consists of the running time for (i) constructing the

Time (sec.)	DSA		RAR		ARC		ASL	
	CNN_{mts}	PVI	CNN_{mts}	PVI	CNN_{mts}	PVI	CNN_{mts}	PVI
CNN_{mts}-*LR*	7733	39	18490	375	26624	398	32110	41
FCN-LR	8973	33	19217	298	25761	417	30144	47
CNN_{mts}-*LDA*	7733	9	18490	136	26624	153	32110	36
CNN_{mts}-*PCA*	7733	5	18490	24	26624	51	32110	7
CNN_{mts}-*CPCA*	7733	9	18490	31	26624	59	32110	14
CNN_{mts}-*RF*	7733	11	18490	130	26624	145	32110	29
LR	0	1049	0	832	0	1211	0	128
LDA	0	788	0	404	0	625	0	101
PCA	0	54	0	44	0	130	0	21
CPCA	0	61	0	47	0	141	0	28
RF	0	180	0	363	0	429	0	33

TABLE 22: Averaged time to build CNN_{mts}-*X* framework using nine folds (2 folds for ASL) of the data

Time (sec.)	DSA	RAR	ARC	ASL
CNN_{mts}	4643	5361	13288	3135
KNN	0	0	0	0
LibSVM	687	3623	26636	376
RF	36	36	199	7

(a) Training time using nine fold (2 folds for ASL) of the data

Time (sec.)	DSA	RAR	ARC	ASL
CNN_{mts}	0.010	0.004	0.003	0.040
KNN	0.367	0.956	1.881	0.203
LibSVM	0.078	0.087	0.058	0.065
RF	2.266×10^{-5}	5.660×10^{-5}	6.536×10^{-5}	2.78×10^{-4}

(b) Testing time for one instance

TABLE 23: Running time of PVC algorithm

CNN_{mts} model using all the attributes (Section 3) and (ii) executing the PVI algorithm (Section 3.2). The results show that the CNN_{mts} model construction utilizes the majority of the time due to their known long training time. The *LR* method's running time is approximately the summation of the running time of *LDA* and *RF*. *LDA* and *RF* use more time than *PCA* because *LDA* and *RF* need to be conducted on all the instances for E times while the *PCA* methods are only applied to a subset of instances E times. Table 23 reports the training and testing time of all E binary classifications over different datasets. Note that the training time is the running time using nine folds of the dataset (two folds for ASL) and the testing time is for one instance. The prediction/testing time per instance is almost ignorable compared to the PV identification time. As expected, the *KNN* methods use more testing time than other methods. Please note that the training time is the running time for all the phenomena (evens) and this training typically happens offline.

4.6 Compare batch processing strategies of CNN$_{mts}$ for imbalanced data

This set of experiments tests the effect of batch processing strategies. The RAR dataset is used because it contains imbalanced data. CNN_{mts}-*LR* is used to select the top significant 30% PVs and CNN_{mts} classifiers are used to conduct ten-fold cross validation.

Batch processing strategy		F_1	Accuracy
CNN_{mts}-*LR*	Classifier CNN_{mts}		
without oversampling	without oversampling	0.813	0.866
without oversampling	with oversampling	0.902	0.910
with oversampling	with oversampling	**0.946**	**0.971**

TABLE 24: Effect of different batch processing strategies (CNN_{mts}-*LR*, RAR dataset, ten-fold)

Table 24 shows the results. The first two rows of the results show that when the PVs are fixed, the classifier with oversampling can improve both the F_1 and *Accuracy* about 9% and 4% respectively. Comparing the last two rows, we can see that, when the classifier is fixed, PV selection with oversampling can improve both the F_1 and *Accuracy* about 4% and 6%.

All these show that CNN_{mts} with the oversampling batch processing strategy works better than the default *CNN* models.

5 Related Works

Identifying significant variables is highly related to feature extraction. The problem of feature extraction has been extensively investigated in the past several decades. For example, Principal Component Analysis (*PCA*) [26] [47] and Linear Discriminant Analysis [13] are among the commonly used feature extraction techniques proposed in earlier days. However, both methods cannot be directly utilized to identify significant PVs because they cannot treat one time series as a variable directly.

More recent techniques of identifying features from sequence data (e.g., [6], [7], [8], [9]) generally convert the sequence to a set of features and analyze the data in the feature space. Most of the identified features cannot preserve the temporal continuity information that is explicit in the original sequence data. Among the works of extracting features from sequence data, the Shapelets feature, introduced in [9], can preserve the temporal order of points in a time series. Shapelets discovery has gained exploding interest from independent research groups (e.g., [6], [9], [45], [46], [48], [49], [50], [51]) to analyze time series data. The methods that extract Shapelet features cannot be directly used to solve our problem either because the purpose of Shapelet extraction is to get global Shapelet features that can help achieve high accuracy of classification tasks, while our problem is to find variable subsets that can contribute the most to specific event types. Furthermore, the extraction of shapelets from multiple sequences dramatically complicates the Shapelet extraction algorithms which are already very complex even on single-sequence instances.

Techniques that classify multi-class datasets (e.g., [52]) typically focus on improving classification accuracy and do not study the importance of different variables for different classes.

Subspace clustering such as projected clustering [3] has been studied based on the similar rationale of PV identification. It identifies clusters from a dataset such that the points in one cluster are close regarding a subset of dimensions. The dimension subsets are generally different for different clusters. Although having similar intention, the results of projected clustering do not keep the temporal order of the selected dimensions, which cannot be used to identify PVs.

Recent works (e.g., [16], [17], [18], [19], [53]) have utilized convolutional neural networks (*CNN*) in the analysis of MTS data. Most of these methods focus on improving classification accuracy or learning the *CNN* structure. Thus, they cannot be directly utilized to solve our problem.

6 Conclusions

In this paper, we introduced a new problem of identifying significant Phenomena-specific variables (PVs) from MTS data. This problem selects significant variables that are important to different event types of the data. To solve this problem, we proposed a novel CNN_{mts}-*X* framework. In this framework, a new variant of convolutional neural networks, CNN_{mts}, is designed to convert each variable's corresponding sequence to independent features. The *X* in this framework can be other feature detection technology. We also designed a new *LR* approach to be used in this CNN_{mts}-*X* framework for the identification of important PVs. The results from extensive experiments on four real datasets by comparing CNN_{mts}-*LR* with seven baseline methods show that (i) our CNN_{mts}-*LR* method can identify more useful PVs than other methods, (ii) 30% of the PVs found from CNN_{mts}-*LR* are able to carry almost all import information as all the variables, and (iii) the CNN_{mts} with a new batch processing strategy outperforms typical *CNN* models when classifying imbalanced multi-class MTS data.

Acknowledgment

This work is supported by NSF #1633330, #1345232, and #1757207.

References

[1] J. Miao and L. Niu, "A survey on feature selection," *Procedia Computer Science*, vol. 91, pp. 919–926, 12 2016.

[2] F. Liang, X. Yang, F. Liu, J. Li, Q. Xiao, J. Chen, X. Liu, J. Cao, C. Shen, L. Yu, F. Lu, X. Wu, L. Zhao, X. Wu, Y. Li, D. Hu, J. Huang, Y. Liu, X. Lu, and D. Gu, "Long-term exposure to ambient fine particulate matter and incidence of diabetes in china: A cohort study," *Environment International*, vol. 126, pp. 568–575, 05 2019.

[3] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park, "Fast algorithms for projected clustering," in *Proceedings of ACM SIGMOD Intl. Conference on Management of Data*, 1999, pp. 61–72. [Online]. Available: http://doi.acm.org/10.1145/304182.304188

[4] F. Chen, B. Zhou, A. Alim, and L. Zhao, "A generic framework for interesting subspace cluster detection in multi-attributed networks," in *2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017*, 2017, pp. 41–50. [Online]. Available: https://doi.org/10.1109/ICDM.2017.13

[5] H. Yoon, K. Yang, and C. Shahabi, "Feature subset selection and feature ranking for multivariate time series," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 9, pp. 1186–1198, Sep. 2005.

[6] J. Grabocka, M. Wistuba, and L. Schmidt-Thieme, "Fast classification of univariate and multivariate time series through shapelet discovery," *Knowl. Inf. Syst.*, vol. 49, no. 2, pp. 429–454, 2016. [Online]. Available: https://doi.org/10.1007/s10115-015-0905-9

[7] S. Li, Y. Li, and Y. Fu, "Multi-view time series classification: A discriminative bilinear projection approach," in *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM*, 2016, pp. 989–998. [Online]. Available: http://doi.acm.org/10.1145/2983323.2983780

[8] J. Lin, E. J. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: a novel symbolic representation of time series," *Data Min. Knowl. Discov.*, vol. 15, no. 2, pp. 107–144, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10618-007-0064-z

[9] L. Ye and E. J. Keogh, "Time series shapelets: a new primitive for data mining," in *SIGKDD*, 2009, pp. 947–956. [Online]. Available: http://doi.acm.org/10.1145/1557019.1557122

[10] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *Web-Age Information Management: 15th International Conference*, 2014, pp. 298–310. [Online]. Available: https://doi.org/10.1007/978-3-319-08010-9_33

[11] M. Gütlein, E. Frank, M. A. Hall, and A. Karwath, "Large-scale attribute selection using wrappers," in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2009, part of the IEEE Symposium Series on Computational Intelligence 2009, Nashville, TN, USA, March 30, 2009 - April 2, 2009*, 2009, pp. 332–339. [Online]. Available: https://doi.org/10.1109/CIDM.2009.4938668

[12] G. Wang, Q. Song, H. Sun, X. Zhang, B. Xu, and Y. Zhou, "A feature subset selection algorithm automatic recommendation method," *J. Artif. Int. Res.*, vol. 47, no. 1, pp. 1–34, May 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2566972.2566973

[13] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müllers, "Fisher discriminant analysis with kernels," in *IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing IX*, 1999, pp. 41–48. [Online]. Available: http://ieeexplore.ieee.org/document/788121/

[14] T. K. Ho, "Random decision forests," in *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ser. ICDAR '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 278–. [Online]. Available: http://dl.acm.org/citation.cfm?id=844379.844681

[15] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 1578–1585.

[16] J. Yang, M. N. Nguyen, P. P. San, X. Li, and S. Krishnaswamy, "Deep convolutional neural networks on multichannel time series for human activity recognition," in *Intl. Joint Conference on Artificial Intelligence, IJCAI*, 2015, pp. 3995–4001. [Online]. Available: http://ijcai.org/Abstract/15/561

[17] M. Qiu, P. Zhao, K. Zhang, J. Huang, X. Shi, X. Wang, and W. Chu, "A short-term rainfall prediction model using multi-task convolutional neural networks," in *2017 IEEE International Conference on Data Mining (ICDM)*, Nov 2017, pp. 395–404.

[18] X. Li and J. Lin, "Linear time complexity time series classification with bag-of-pattern-features," in *2017 IEEE International Conference on Data Mining (ICDM)*, Nov 2017, pp. 277–286.

[19] S. Yi, J. Ju, M. Yoon, and J. Choi, "Grouped convolutional neural networks for multivariate time series," *CoRR*, vol. abs/1703.09938, 2017. [Online]. Available: http://arxiv.org/abs/1703.09938

[20] W. Jiang and Z. Yin, "Human activity recognition using wearable sensors by deep convolutional neural networks," in *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference, MM '15, Brisbane, Australia, October 26 - 30, 2015*, 2015, pp. 1307–1310. [Online]. Available: http://doi.acm.org/10.1145/2733373.2806333

[21] F. J. Ordóñez and D. Roggen, "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, p. 115, 2016. [Online]. Available: http://www.mdpi.com/1424-8220/16/1/115

[22] D. Kriesel, *A Brief Introduction to Neural Networks*, 2007. [Online]. Available: availableathttp://www.dkriesel.com

[23] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009. [Online]. Available: https://doi.org/10.1109/TKDE.2008.239

[24] A. M. Martinez and A. C. Kak, "Pca versus lda," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, pp. 228–233, 2001.

[25] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[26] H. Hotelling, "Relations between two sets of variates," *Biometrika*, vol. 28, no. 3/4, pp. 321–377, 1936. [Online]. Available: http://www.jstor.org/stable/2333955

[27] A. Kane and N. Shiri, "Multivariate time series representation and similarity search using pca," in *Advances in Data Mining. Applications and Theoretical Aspects*, P. Perner, Ed. Cham: Springer International Publishing, 2017, pp. 122–136.

[28] N. Cilia, C. De Stefano, F. Fontanella, S. Raimondo, and A. Scotto di Freca, "An experimental comparison of feature-selection and classification methods for microarray datasets," *Information*, vol. 10, p. 109, 03 2019.

[29] O. S Powar, K. Chemmangat Manakkal Cheriya, and S. Figarado, "A novel pre-processing procedure for enhanced feature extraction and characterization of electromyogram signals," *Biomedical Signal Processing and Control*, vol. 42, 02 2018.

[30] C. D'Agostino, A. Saidi, G. Scouarnec, and L. Chen, "Learning-based driving events recognition and its application to digital roads," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 2155–2166, Aug 2015.

[31] A. Wang, N. An, G. Chen, L. Li, and G. Alterovitz, "Accelerating wrapper-based feature selection with k-nearest-neighbor," *Know.-Based Syst.*, vol. 83, no. C, pp. 81–91, Jul. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2015.03.009

[32] M. Hammami, S. Bechikh, C.-C. Hung, and L. Ben Said, "A multi-objective hybrid filter-wrapper evolutionary approach for feature construction on high-dimensional data," 07 2018.

[33] F. Moslehi and A. Haeri, "A novel hybrid wrapper–filter approach based on geneticalgorithm, particle swarm optimization for feature subset selection," *Journal of Ambient Intelligence and Humanized Computing*, Jun 2019. [Online]. Available: https://doi.org/10.1007/s12652-019-01364-5

[34] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 94, 2018.

[35] "UCI Machine Leaning Repository, daily and sports activities data set. http://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities."

[36] "UCI Machine Leaning Repository, REALDISP activity recognition data set. https://archive.ics.uci.edu/ml/datasets/REALDISP+Activity+Recognition+Dataset."

[37] "Opportunistic Activity Recognition Systems, activity recognition challenge data set. http://www.opportunity-project.eu/system/files/Challenge/OpportunityChallengeLabeled.zip."

[38] "UCI Machine Leaning Repository, australian sign language signs (high quality) data set . https://archive.ics.uci.edu/ml/datasets/Australian+Sign+Language+signs+(High+Quality)."

[39] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[40] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, January 1967.

[41] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT '92. New York, NY, USA: ACM, 1992, pp. 144–152. [Online]. Available: http://doi.acm.org/10.1145/130385.130401

[42] A. A. Danasingh, S. Balamurugan, and J. L. EPIPHANY, "Literature review on feature selection methods for high-dimensional data," *International Journal of Computer Applications*, vol. 136, 02 2016.

[43] "ASL, *please* from auslan signbank http://www.auslan.org.au/dictionary/words/please-1.html."

[44] "ASL, *stubborn* from auslan signbank http://www.auslan.org.au/dictionary/words/stubborn-1.html."

[45] E. J. Keogh and T. Rakthanmanon, "Fast shapelets: A scalable algorithm for discovering time series shapelets," in *SDM*, 2013, pp. 668–676. [Online]. Available: http://dx.doi.org/10.1137/1.9781611972832.74

[46] D. S. Raychaudhuri, J. Grabocka, and L. Schmidt-Thieme, "Channel masking for multivariate time series shapelets," *CoRR*, vol. abs/1711.00812, 2017. [Online]. Available: http://arxiv.org/abs/1711.00812

[47] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Inc., 2006.

[48] K. Chang, B. Deka, W. W. Hwu, and D. Roth, "Efficient pattern-based time series classification on GPU," in *ICDM*, 2012, pp. 131–140. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2012.132

[49] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning time-series shapelets," in *SIGKDD*, 2014, pp. 392–401. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623613

[50] J. Lines, L. M. Davis, J. Hills, and A. Bagnall, "A shapelet transform for time series classification," in *SIGKDD*, 2012, pp. 289–297. [Online]. Available: http://doi.acm.org/10.1145/2339530.2339579

[51] A. Mueen, E. J. Keogh, and N. Young, "Logical-shapelets: an expressive primitive for time series classification," in *SIGKDD*, 2011, pp. 1154–1162. [Online]. Available: http://doi.acm.org/10.1145/2020408.2020587

[52] L. Chen, J. Tang, and B. Li, "Embedded supervised feature selection for multi-class data," in *Proceedings of the 2017 SIAM*

International Conference on Data Mining, 2017, pp. 516–524. [Online]. Available: https://doi.org/10.1137/1.9781611974973.58

[53] H. Gao and S. Ji, "Efficient and invariant convolutional neural networks for dense prediction," in *2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017*, 2017, pp. 871–876. [Online]. Available: https://doi.org/10.1109/ICDM.2017.107

Appendix A
Additional tables and figures for Section 4.4.1

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
CNN_{mts}-LR	**0.928 [1]**	**0.946 [1]**	**0.962 [1]**	**0.628 [1]**	**0.788 [1]**
CNN_{mts}-LDA	0.906 [2]	0.934 [2]	0.937 [7]	0.600 [3]	0.418 [9]
CNN_{mts}-PCA	0.831 [10]	0.926 [3]	0.956 [2]	0.489 [8]	0.499 [6]
CNN_{mts}-CPCA	0.895 [6]	0.876 [6]	0.921 [9]	0.441 [9]	0.369 [10]
CNN_{mts}-RF	0.897 [4]	0.902 [4]	0.950 [4]	0.596 [5]	0.669 [4]
LR	0.903 [3]	0.852 [8]	0.940 [6]	0.597 [4]	0.761 [2]
LDA	0.897 [4]	0.698 [10]	0.937 [7]	0.557 [7]	0.503 [5]
PCA	0.873 [9]	0.841 [9]	0.946 [5]	0.418 [10]	0.429 [8]
CPCA	0.887 [7]	0.858 [7]	0.919 [10]	0.492 [6]	0.448 [7]
RF	0.887 [7]	0.899 [5]	0.952 [3]	0.624 [2]	0.706 [3]

(a) CNN_{mts} classifier (CNN_{mts}-LR always ranks top 1)

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
CNN_{mts}-LR	0.872 [2]	0.897 [2]	0.962 [3]	**0.530 [1]**	0.634 [2]
CNN_{mts}-LDA	0.727 [9]	**0.907 [1]**	0.922 [5]	0.483 [4]	0.418 [6]
CNN_{mts}-PCA	0.751 [4]	0.883 [3]	0.902 [7]	0.381 [10]	0.510 [5]
CNN_{mts}-CPCA	0.743 [7]	0.837 [5]	0.896 [9]	0.383 [9]	0.314 [8]
CNN_{mts}-RF	0.666 [10]	0.833 [6]	0.960 [4]	0.528 [2]	**0.669 [1]**
LR	0.854 [3]	0.771 [9]	**0.966 [1]**	0.452 [6]	0.588 [4]
LDA	**0.903 [1]**	0.641 [10]	0.912 [6]	0.444 [7]	0.360 [7]
PCA	0.738 [8]	0.830 [7]	0.898 [8]	0.430 [8]	0.309 [10]
CPCA	0.744 [6]	0.806 [8]	0.853 [10]	0.372 [5]	0.312 [9]
RF	0.746 [5]	0.863 [4]	0.965 [2]	0.516 [3]	0.610 [3]

(b) KNN classifier (CNN_{mts}-LR always ranks top 3)

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
CNN_{mts}-LR	0.721 [2]	0.645 [2]	**0.730 [1]**	**0.516 [1]**	0.407 [2]
CNN_{mts}-LDA	0.518 [8]	0.461 [7]	0.594 [7]	0.325 [7]	0.237 [6]
CNN_{mts}-PCA	0.339 [10]	0.249 [10]	0.672 [5]	0.130 [9]	0.291 [5]
CNN_{mts}-CPCA	0.582 [5]	0.542 [4]	0.684 [4]	0.411 [8]	0.173 [8]
CNN_{mst}- RF	0.552 [6]	0.517 [6]	0.693 [3]	0.349 [5]	**0.413 [1]**
LR	0.611 [4]	**0.650 [1]**	0.711 [2]	0.452 [2]	0.380 [4]
LDA	0.643 [3]	0.402 [8]	0.570 [10]	0.435 [3]	0.192 [7]
PCA	0.441 [9]	0.522 [5]	0.573 [9]	0.110 [10]	0.157 [10]
CPCA	0.547 [7]	0.547[3]	0.589[8]	0.472 [4]	0.167 [9]
RF	**0.723 [1]**	0.275 [9]	0.662 [6]	0.337 [6]	0.393 [3]

(c) LibSVM classifier (CNN_{mts}-LR always ranks top 2)

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
CNN_{mts}-LR	**0.786 [1]**	0.710 [2]	0.767 [2]	0.355 [2]	**0.553 [1]**
CNN_{mts}-LDA	0.782 [3]	**0.712 [1]**	0.636 [7]	0.264 [9]	0.401 [5]
CNN_{mts}-PCA	0.577 [10]	0.706 [3]	0.359 [10]	0.232 [10]	0.398 [6]
CNN_{mts}-CPCA	0.775 [4]	0.685 [4]	0.406 [9]	0.351 [4]	0.293 [8]
CNN_{mst}- RF	0.772 [7]	0.673 [6]	0.743 [4]	0.347 [5]	0.513 [3]
LR	0.773 [5]	0.643 [7]	**0.786 [1]**	0.352 [3]	0.508 [4]
LDA	0.731 [8]	0.385 [10]	0.719 [5]	0.307 [8]	0.361 [7]
PCA	0.785 [2]	0.481 [8]	0.653 [6]	0.332 [7]	0.216 [10]
CPCA	0.699 [9]	0.448 [9]	0.612 [8]	0.339 [6]	0.278 [9]
RF	0.773 [5]	0.681 [5]	0.761 [3]	**0.360 [1]**	0.516 [2]

(d) RF classifier (CNN_{mts}-LR always ranks top 2)

TABLE 25: F_1 for different variable selection methods (Top 30% of PVs are selected). The values in [] denote the ranks of the classifier in a row to classify the dataset in a column.

Table 25 and Table 26 show the averaged F_1 and accuracy results for evaluating the effect of PVs selected by the ten PV selection approaches in Section 4.3.1. Fig. 9 to Fig. 13 show the details F_1 values for all event types by the top 4 PVI methods.

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
CNN_{mts}-LR	**0.961 [1]**	**0.971 [1]**	**0.982 [1]**	**0.889 [1]**	**0.797 [1]**
CNN_{mts}-LDA	0.942 [3]	0.969 [2]	0.979 [7]	0.880 [5]	0.347[9]
CNN_{mts}-PCA	0.879 [10]	0.956 [3]	**0.982 [1]**	0.832 [9]	0.511[6]
CNN_{mts}-CPCA	0.907 [7]	0.906 [7]	0.980 [4]	0.833 [8]	0.319 [10]
CNN_{mts}-RF	0.933 [4]	0.946 [4]	0.980 [4]	0.885 [4]	0.591 [4]
LR	0.944 [2]	0.907 [6]	0.980 [4]	0.888 [2]	0.762[3]
LDA	0.926 [5]	0.717 [10]	0.977 [8]	0.887 [3]	0.423[8]
PCA	0.882 [9]	0.905 [8]	0.976 [9]	0.818 [10]	0.498[7]
CPCA	0.889[8]	0.902[9]	0.950 [10]	0.879[6]	0.515 [5]
RF	0.924 [6]	0.942 [5]	0.981 [3]	0.876 [7]	0.773 [2]

(a) CNN_{mts} classifier (CNN_{mts}-LR always ranks top 1)

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
CNN_{mts}-LR	0.898 [2]	0.900 [2]	0.981 [3]	0.856 [3]	0.562 [2]
CNN_{mts}-LDA	0.738 [7]	**0.909 [1]**	0.967 [5]	0.853 [4]	0.347[5]
CNN_{mts}-PCA	0.774 [4]	0.894 [3]	0.959 [7]	0.809 [7]	0.297 [7]
CNN_{mts}-CPCA	0.720 [8]	0.813 [7]	0.955 [9]	0.802 [9]	0.258 [10]
CNN_{mts}-RF	0.684 [10]	0.837 [6]	0.979 [4]	0.858 [2]	0.591 [1]
LR	0.884 [3]	0.780 [9]	**0.984 [1]**	0.846 [5]	0.501 [4]
LDA	**0.917 [1]**	0.658 [10]	0.967 [5]	**0.866 [1]**	0.313 [6]
PCA	0.744 [6]	0.845 [5]	0.956 [8]	0.809 [7]	0.288 [9]
CPCA	0.709 [8]	0.797[8]	0.940 [10]	0.794 [10]	0.292 [8]
RF	0.764 [5]	0.870 [4]	0.983 [2]	0.844 [6]	0.529 [3]

(b) KNN classifier (CNN_{mts}-LR always ranks top 3)

Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
CNN_{mts}-LR	**0.768 [1]**	0.649 [2]	**0.910 [1]**	0.840 [3]	0.207 [2]
CNN_{mts}-LDA	0.529 [7]	0.469 [6]	0.854 [8]	0.805 [7]	0.071 [7]
CNN_{mts}-PCA	0.353 [10]	0.267 [10]	0.883 [6]	0.794 [9]	0.060 [8]
CNN_{mts}-CPCA	0.547 [6]	0.376 [8]	0.891 [4]	0.833 [4]	0.075 [6]
CNN_{mts}-RF	0.561 [4]	0.522 [4]	0.898 [3]	0.805 [7]	0.219 [1]
LR	0.683 [3]	**0.662 [1]**	0.903 [2]	0.852 [2]	0.163 [4]
LDA	0.548 [5]	0.412 [7]	0.853 [9]	**0.854 [1]**	0.090 [5]
PCA	0.466 [9]	0.529 [3]	0.848 [10]	0.790 [10]	0.005 [10]
CPCA	0.501 [8]	0.521[5]	0.883 [6]	0.830 [5]	0.060 [8]
RF	0.757 [2]	0.332 [9]	0.891 [4]	0.808 [6]	0.182 [3]

(c) LibSVM classifier (CNN_{mts}-LR always ranks top 3)

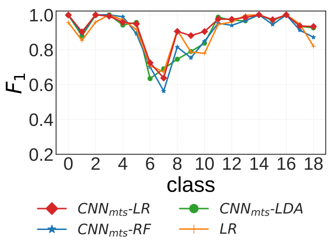
Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
CNN_{mts}-LR	0.913 [2]	0.897 [2]	0.949 [2]	**0.889 [1]**	**0.667 [1]**
CNN_{mts}-LDA	0.906 [4]	**0.920 [1]**	0.923 [6]	0.880 [5]	0.503 [5]
CNN_{mts}-PCA	0.841 [10]	0.894 [3]	0.868 [9]	0.832 [8]	0.339 [8]
CNN_{mts}-CPCA	0.852 [9]	0.760 [10]	0.860 [10]	0.790 [10]	0.347 [7]
CNN_{mst}- RF	0.909 [3]	0.868 [6]	0.949 [2]	0.885 [4]	0.656 [3]
LR	**0.915 [1]**	0.887 [4]	**0.953 [1]**	0.888 [2]	0.645 [4]
LDA	0.899 [6]	0.833 [7]	0.941 [5]	0.887 [3]	0.453 [6]
PCA	0.899 [6]	0.784 [8]	0.922 [7]	0.818 [9]	0.307 [10]
CPCA	0.872 [8]	0.766 [9]	0.914 [8]	0.840 [7]	0.314 [9]
RF	0.906 [4]	0.874 [5]	0.946 [4]	0.876 [6]	0.665 [2]

(d) RF classifier (CNN_{mts}-LR always ranks top 2)

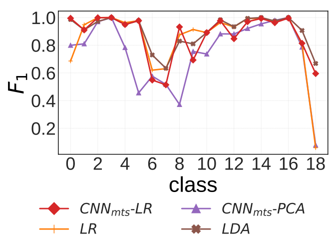
TABLE 26: Overall *Accuracy* for different variable selection methods (Top 30% of PVs are selected). The values in [] denote the ranks of the classifier in a row to classify the dataset in a column.

Appendix B
Additional tables and figures for Section 4.4.2

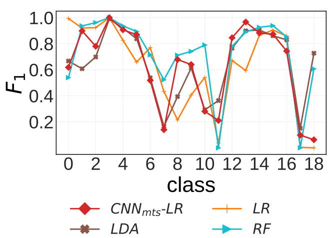
Table 27 shows the accuracy results for evaluating the performance of the PVs found using the proposed CNN_{mts}-*LR* method, CNN_{mts}-*LR-GV* and *All-variables*. Fig. 14 to Figre 18 shows the detail F_1 values for all event types using all the variables, top 30% of PVs, and top 30% of GVs.



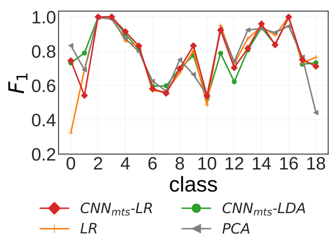
(a) CNN_{mts} with CNN_{mts}-X



(b) KNN with CNN_{mts}-X

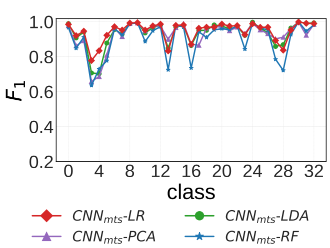


(c) $LibSVM$ with CNN_{mts}-X

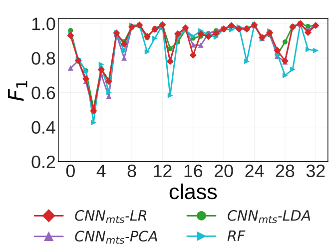


(d) RF with CNN_{mts}-X

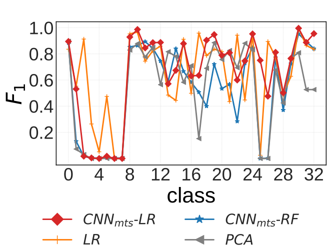
Fig. 9: F_1 for all event types on DSA (Top 30% of the PVs are selected by the top 4 PVI approaches)



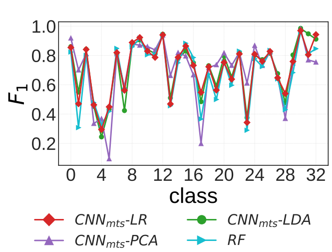
(a) CNN_{mts} with CNN_{mts}-X



(b) KNN with CNN_{mts}-X

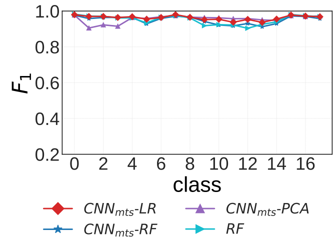


(c) $LibSVM$ with CNN_{mts}-X

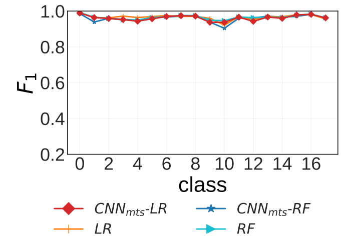


(d) RF with CNN_{mts}-X

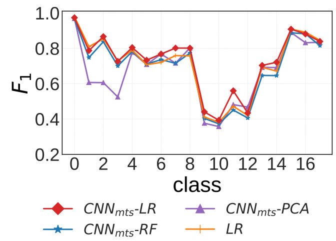
Fig. 10: F_1 for all event types on RAR (Top 30% of the PVs are selected by the top 4 PVI methods)



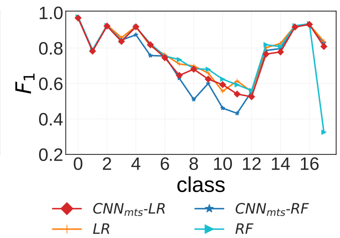
(a) CNN_{mts} with CNN_{mts}-X



(b) KNN with CNN_{mts}-X

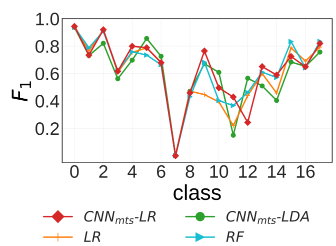


(c) $LibSVM$ with CNN_{mts}-X

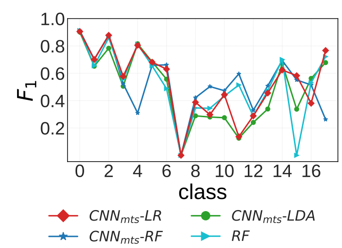


(d) RF with CNN_{mts}-X

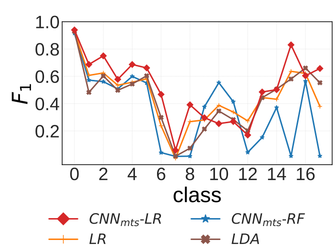
Fig. 11: F_1 for all event types on ARC (Top 30% of the PVs are selected by the top 4 PVI approaches)



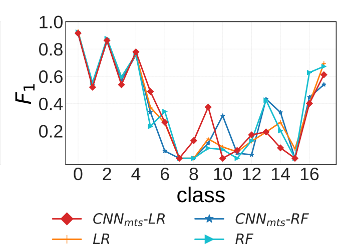
(a) CNN_{mts} with CNN_{mts}-X



(b) KNN with CNN_{mts}-X

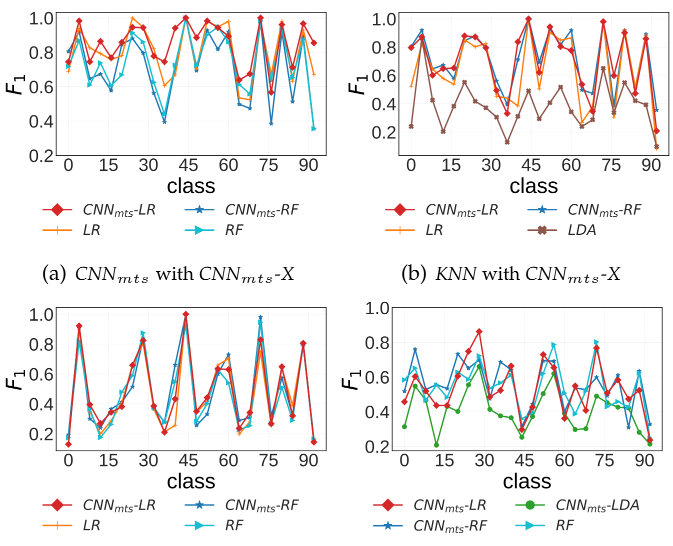


(c) $LibSVM$ with CNN_{mts}-X



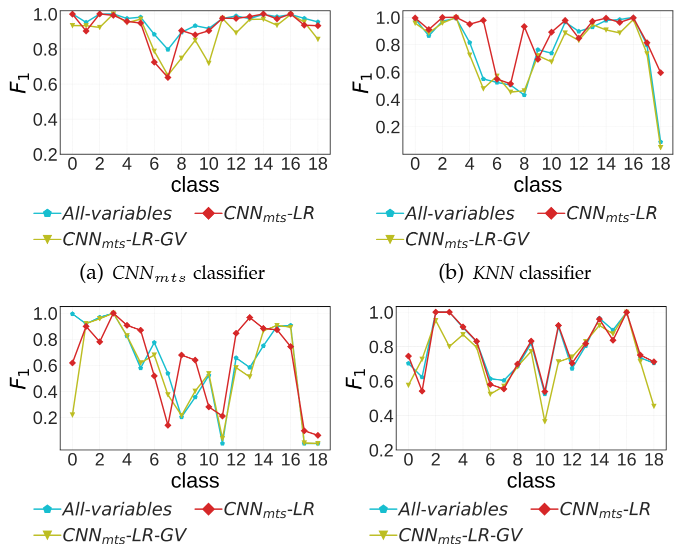
(d) RF with CNN_{mts}-X

Fig. 12: F_1 for all event types on ARC_{fixed} (Top 30% of the PVs are selected by the top 4 PVI methods)



(a) CNN_{mts} with CNN_{mts}-X (b) *KNN* with CNN_{mts}-X (c) *LibSVM* with CNN_{mts}-X (d) *RF* with CNN_{mts}-X

Fig. 13: F_1 for all event types on ASL (Top 30% of the PVs are selected by the top 4 PVI methods)

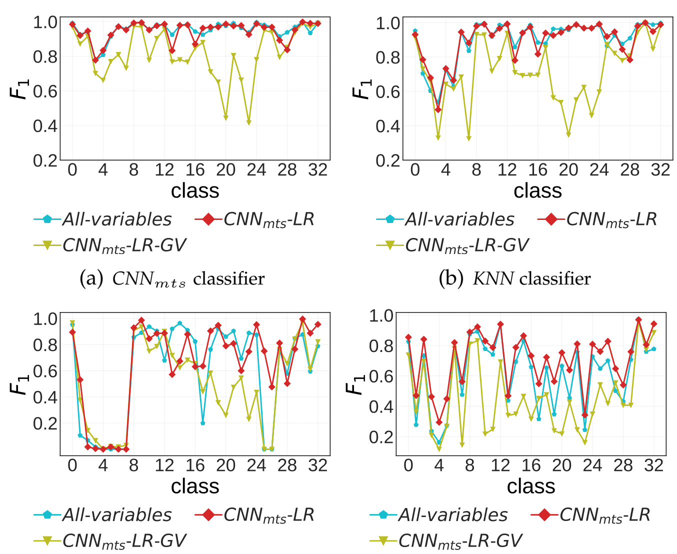


(a) CNN_{mts} classifier (b) *KNN* classifier (c) *LibSVM* classifier (d) *RF* classifier

Fig. 14: F_1 for different event types on DSA (Classification using all the variables, top 30% of PVs, and top 30% of GVs)

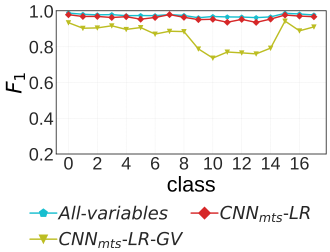
Method	DSA	RAR	ARC	ARC$_{fixed}$	ASL
(a) CNN_{mts} classifier					
All-variables	**0.978**	**0.975**	**0.988**	0.859	**0.811**
CNN_{mts}-*LR*	0.961	0.971	0.982	**0.889**	0.797
CNN_{mts}-*LR-GV*	0.922	0.878	0.947	0.660	0.603
(b) *KNN* classifier					
All-variables	0.791	**0.903**	0.969	0.807	**0.654**
CNN_{mts}-*LR*	**0.898**	0.900	**0.981**	**0.856**	0.562
CNN_{mts}-*LR-GV*	0.759	0.759	0.863	0.681	0.371
(c) *LibSVM* classifier					
All-variables	0.547	0.579	0.906	0.804	**0.231**
CNN_{mts}-*LR*	**0.768**	**0.649**	**0.910**	**0.840**	0.207
CNN_{mts}-*LR-GV*	0.556	0.527	0.696	0.709	0.118
(d) *RF* classifier					
All-variables	**0.936**	0.860	**0.950**	0.859	**0.668**
CNN_{mts}-*LR*	0.913	**0.897**	0.949	**0.889**	0.667
CNN_{mts}-*LR-GV*	0.881	0.755	0.878	0.660	0.419

TABLE 27: *Accuracy* comparison using all the variables, top 30% PVs, and top 30% of GVs

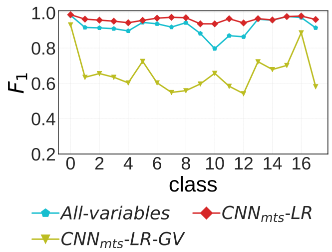


(a) CNN_{mts} classifier (b) *KNN* classifier (c) *LibSVM* classifier (d) *RF* classifier

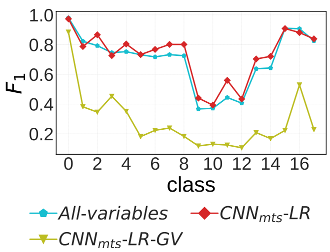
Fig. 15: F_1 for different event types on RAR (Classification using all the variables, top 30% of PVs, and top 30% of GVs)



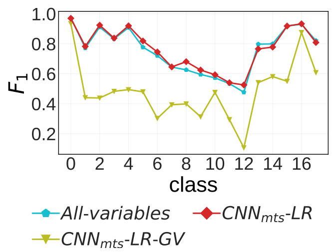
(a) CNN_{mts} classifier



(b) *KNN* classifier

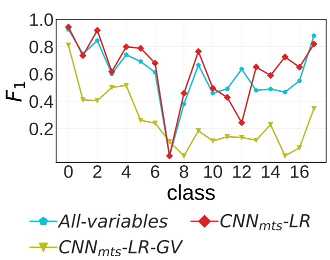


(c) *LibSVM* classifier

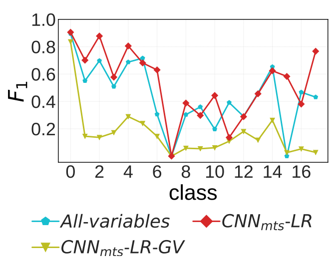


(d) *RF* classifier

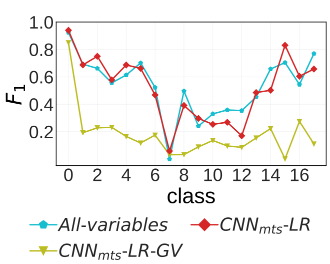
Fig. 16: F_1 for different event types on ARC (Classification using all the variables, top 30% of PVs, and top 30% of GVs)



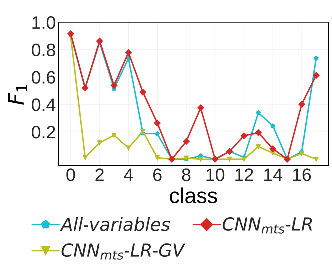
(a) CNN_{mts} classifier



(b) *KNN* classifier

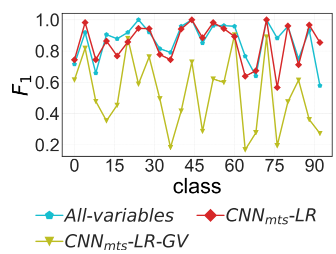


(c) *LibSVM* classifier

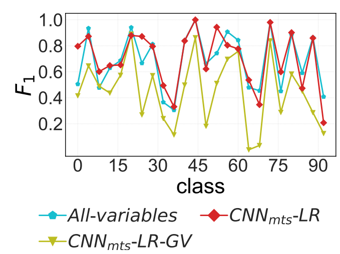


(d) *RF* classifier

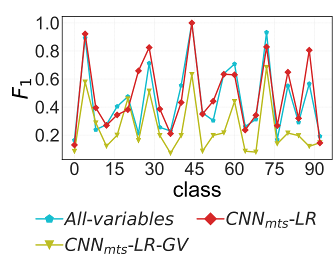
Fig. 17: F_1 for different event types on ARC$_{fixed}$ (Classification using all the variables, top 30% of PVs, and top 30% of GVs)



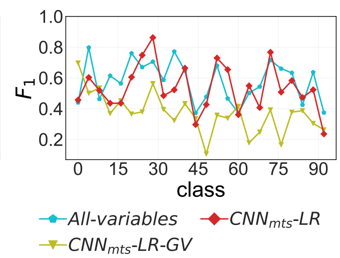
(a) CNN_{mts} classifier



(b) *KNN* classifier



(c) *LibSVM* classifier



(d) *RF* classifier

Fig. 18: F_1 for different event types on ASL (Classification using all the variables, top 30% of PVs, and top 30% of GVs)