Automating Feature Construction for Multi-View Time Series Data

Arne De Brabandere, Pieter Robberechts, Tim Op De Beéck, and Jesse Davis

Dept of Computer Science, KU Leuven, Belgium
{firstname.lastname}@cs.kuleuven.be

Abstract. Constructing features for classifying time series data is a challenging and time-consuming task. This has motivated the development of systems that aim to automate the feature construction process. These systems typically provide a library of feature extraction transformations along with a selection method to find the relevant features. However, the existing systems are designed for univariate time series data. In practice, time series are often generated by multiple sensors. It may be useful to define new series by fusing the series generated by these sensors. In this work, we propose a feature construction method that considers such transformations. In addition, we develop a Python package called `TSFuse` that implements our approach. To evaluate the system, we perform experiments on real-world time series datasets and evaluate classification models trained with the constructed features. The experimental results show that our feature-based approach can outperform a neural network approach in terms of both accuracy and time. Compared to univariate feature extraction, our system is able to find a better feature representation when time series fusion is beneficial. Our findings suggest that `TSFuse` is suitable for medium-sized datasets where neural networks fail to learn a good representation, but where the problem is too complex to construct the features manually.

Keywords: Feature construction · Sensor fusion · Time series analysis.

1 Introduction

Time series are collected in various applications such as human activity recognition [9], athlete monitoring [22], water consumption analysis [25], and condition monitoring of industrial systems [16]. When the series are annotated with labels (e.g., the activity performed by a person at each point in time), analyzing the data typically involves training a classification model to predict these labels.

There exist different approaches for classifying time series data. The feature-based approach [11] transforms the time series to a feature vector representation and then applies standard supervised learning techniques on top of this representation. Section 2 describes the state-of-the-art of this approach. The (deep) neural network approach learns time series classification models in an end-to-end fashion [10, 18]. Examples of such networks are multilayer perceptrons, recurrent neural networks, and convolutional neural networks. Other approaches include distance-based methods which employ a similarity metric (e.g., dynamic time warping), model-based methods such as hidden Markov models, and many more [3].

This work focuses on feature-based time series classification where there has been a growing realization that manually defining and extracting features repeatedly for each application is time consuming and error prone. Even though manual feature construction often works well for small datasets, larger datasets allow for a more complex feature representation, which is difficult for humans to come up with based on domain knowledge alone. This has spurred the development of approaches that automate the feature construction process by computing a set of predefined features from the time series and possibly selecting a subset of the most relevant ones for the prediction task at hand [8, 13]. Yet, the current approaches have several weaknesses, namely that they are designed for univariate time series. Many applications are characterized by the presence of multiple different sensors, e.g., multiple accelerometers and gyroscopes attached to different body parts of a person [22] or pressure, humidity, temperature and wind speed sensors placed at different locations of an industrial site [29]. Oftentimes it may be useful to define new series (e.g., deriving the total acceleration [22]) or to compute features by comparing values from different series (e.g., to measure the symmetry between the left and right side of the body [22]). Unfortunately, current approaches do not exhibit such functionality.

In this work, our goal is to fill this gap and develop an automated feature construction approach that works for data characterized by the presence of multiple time series. Technically, this is challenging because there are many more possible features to consider. We propose a search method for finding a set of relevant features. In addition to considering the original time series, the proposed method constructs new series by fusing multiple existing series. We implement our system as a Python package called `TSFuse`, which contains the search method along with a library of time series feature extraction transformations. Empirically, we find that our approach outperforms existing automated feature construction approaches and deep neural networks on three datasets.

We can summarize the contributions of this paper as follows:

1. We propose a feature construction method that automates the process of extracting and selecting features from multiple time series.
2. We implement a library of time series feature transformations which also includes pre-processing steps.
3. We develop `TSFuse`, a Python package that contains the library of feature transformations and an implementation of the proposed construction method.
4. We compare the performance of our approach to a state-of-the-art feature construction system as well as to an LSTM network on three real-world time series datasets.

2 Related Work

In this section, we describe the state-of-the-art of the feature-based approaches for time series classification. A popular approach is the *highly comparative* approach [12]. This approach constructs features by computing a large set of pre-defined features, typically hundreds or thousands of features, and selecting the relevant ones using a filter or wrapper selection method [14]. The set of features can include both time-domain features (mean, variance, number of peaks, etc.) and frequency-domain features (Fourier transform coefficients, power spectral density, etc.).

There exist different systems that implement the highly comparative approach. Currently, `tsfresh` [8] is one of the most widely used tools. This system extracts a set of 63 features with different parameters, resulting in a feature vector of length 794. The relevant features are selected using the FRESH algorithm (FeatuRe Extraction based on Scalable Hypothesis tests). This algorithm performs hypothesis tests to measure the dependency between the target labels and each feature's values, and selects a subset of the features based on the p-values computed by these tests. Another popular system is `hctsa` [13], which is implemented in Matlab. This system extracts a similar set of features as `tsfresh`, but uses a different selection approach. Instead of a statistical test, it applies forward selection with a linear model.

Another feature-based approach is genetic programming (GP) [15, 20, 24]. Different from the highly comparative methods, GP is not limited to a pre-defined set of features. Instead, this approach can invent new features by generating expression trees that compute features. However, this involves a high computational cost, since evaluating the fitness of the expression trees requires computing a large number of features. Moreover, as opposed to the different implementations that exist for the highly comparative approach, the systems developed for the GP approach (e.g., Autofead [15]) are not publicly available.

3 Preliminaries

Our goal is to automatically construct a set of features that are relevant for a time series classification task. This section formally defines this task and the representation of the input data.

The input of our system is a *multi-view time series dataset*. Such a dataset consists of time series generated by multiple views. A view typically corresponds to the data collected by a single sensor. Since sensors can measure more than one variable, the data of each view is represented as a multivariate time series. For example, a three-dimensional accelerometer collects three series: the x, y and z acceleration. We employ a similar notation as in [19] and specify the representation in the definitions below.

Definition 1. *A* ***multivariate time series*** *is an ordered sequence of m vectors $X = [x_1, \ldots, x_m] \in \mathbb{R}^{d \times m}$ where each x_t is a d-dimensional vector containing the values recorded at the t^{th} time stamp.*

Definition 2. *A **multi-view time series** is a set $\hat{X} = \{X_1, \ldots, X_V\}$ where each $X_v \in \mathbb{R}^{d_v \times m_v}$ is the multivariate time series of the v^{th} view.*

Our aim is to automatically transform $\hat{X}$ into a feature vector representation that can be used as the input of a classification model. Learning such a model requires a dataset where each instance represents a window labeled with a class. Formally, the datasets that we consider are defined as follows:

Definition 3. *A **multi-view time series dataset** is a set of N instances with views $\hat{\mathbf{X}} = \{X_{i,v} \mid i = 1, \ldots, N, v = 1, \ldots, V\}$ and labels $\mathbf{y} = \{y_i \mid i = 1, \ldots, N\}$.*

4 Feature Construction

Our problem can be defined as follows:

Given: A multi-view time series dataset $\{\hat{\mathbf{X}}, \mathbf{y}\}$.
Construct: A feature vector representation which is relevant with respect to the target data $\mathbf{y}$.

Our approach defines the feature vector representation using a computation graph G. This graph specifies all transformation steps that have to be applied on the input data in order to compute the features. A computation graph G can be seen as a function that can be applied to the input data $\hat{\mathbf{X}}$. The result $G(\hat{\mathbf{X}})$ is an attribute-value representation, which can be used directly as an input for a classification model. Consequently, the automated feature construction problem involves discovering a good computation graph. This can be thought of as a search problem which requires picking which transformations to apply as well as the order in which they have to be applied.

The remainder of this section describes our computation graph representation as well as our search procedure. Algorithm 1 shows the pseudo-code for our construction method.

4.1 Computation Graph

A computation graph is a directed acyclic graph (DAG) whose nodes are connected by edges that represent the flow of the data. There are three types of nodes: inputs, constants, and transformers. Inputs are nodes without incoming edges. They serve as placeholders for the given time series data. Constants also have no incoming edges, but as opposed to inputs they hold values that do not depend on the input data. Transformer nodes represent the computation steps. Each transformer creates new data from existing data, which is given by the nodes linked to its incoming edges. We consider two types of transformers:

Series-to-series transformers These create new time series from existing time series, e.g., the ratio between two series.
Series-to-attribute transformers These transform time series into an attribute-value representation, e.g., the mean of a series.

The outputs of a computation graph are the values computed by all transformers that either have no outgoing edges or are marked as an output node. Figure 1 shows an example of a computation graph. The graph has two inputs, representing two views with time series data. The `Ratio` transformer computes the ratio between these series. Then, the result of this transformer is passed on to two series-to-attribute transformers, whose results are the outputs of the graph.

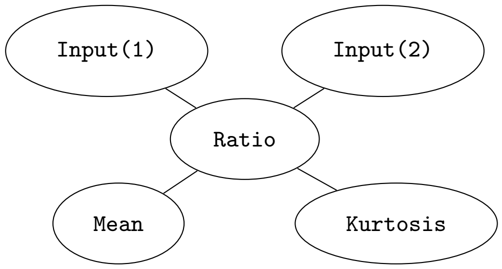

Fig. 1. Example of a computation graph.

4.2 Searching for a Good Computation Graph

When searching for a good computation graph, the central challenge that arises is that considering all possible transformations, particularly when deriving new series such as the total acceleration, can become prohibitively expensive due to the number of possible combinations that exist. Therefore, our approach separates this process into a sequence of distinct steps. Moreover, it attempts to limit the number of series that are considered by trying to quickly assess the redundancy among the set of series and ignoring those it deems to be redundant. Concretely, it performs the following three steps:

Step 1: Select a subset of the input views to add to G.
Step 2: Add series-to-series transformers to G to derive new views from the selected input views.
Step 3: Add series-to-attribute transformers to G to extract an attribute-value representation.

Step 1: Select Input Views Initially, the computation graph G is empty, meaning that it does not contain any inputs or transformer nodes. Step 1 adds input nodes to this graph one at a time where each input node corresponds to a view X_v of the given multi-view time series dataset. This step selects views according to the following redundancy test: we evaluate the correlation between the series and all other series in G on the basis of six statistical features that are fast to compute: min, max, mean, variance, skewness and kurtosis. If at least one of these statistics is not highly correlated to the statistics of the previously

Algorithm 1: Feature construction method

Input: Multi-view time series dataset $\hat{\mathbf{X}}$ with labels $\mathbf{y}$

Parameters:
- Series-to-series transformers S
- Maximal correlation coefficient c for the redundancy test
- Series-to-attribute transformers A
- Significance level α for the relevance test

Output: Computation graph G which computes the constructed features

```
INITIALIZATION
$G$ = empty computation graph
stats = $\varnothing$
```

```
STEP 1: SELECT INPUT VIEWS
for $v \in \{1, \ldots, V\}$ do
    stats$_v$ = compute_stats($X_v$)
    if non_redundant(stats$_v$, stats, $c$) then
        add Input($v$) to $G$
        stats = stats $\cup$ {stats$_v$}
    end
end
```

```
STEP 2: CONSTRUCT FUSED VIEWS
for Transformer $\in S$ do
    for *inputs $\in$ combinations(Transformer.n_inputs, $G$.inputs) do
        node = Transformer(*inputs)
        stats$_t$ = compute_stats(apply(node))
        if non_redundant(stats$_t$, stats, $c$) then
            add node to $G$
            stats = stats $\cup$ {stats$_t$}
        end
    end
end
```

```
STEP 3: EXTRACT ATTRIBUTES
for parent $\in G$.series do
    for Transformer $\in A$ do
        node = Transformer(parent)
        values$_t$ = apply(node)
        if relevant(values$_t$, $\mathbf{y}$, $\alpha$) then
            add node to $G$
        end
    end
end
```

selected series, then the view is added. We consider a variable highly correlated when its Pearson correlation coefficient exceeds a user-defined maximal correlation coefficient c.

Step 2: Construct Fused Views After adding the input nodes, we add series-to-series transformers selected from a given list S in order to construct new views. The inputs of the series-to-series transformers are combinations of the input nodes. These combinations can consist of multiple nodes. For example, the `Ratio` transformer fuses two views by computing the ratio between the series of these views at each point in time. When adding a transformer from S, we generate all combinations of n nodes selected among G's input nodes, where n is the number of required inputs of the transformer (e.g., $n = 2$ for `Ratio`). For each combination, we test the redundancy of the resulting view using the same test as in Step 1, and add a transformer node to G for each combination that results in a non-redundant series.

Step 3: Extract Attributes The final step starts from the series-to-series computation graph G created in Step 1 and 2, and adds series-to-attribute transformers selected from a given list A. These transformers are added for both the selected input views and the constructed views. This step selects the transformers that compute relevant attributes with respect to the target value $\mathbf{y}$ by performing an F-test to evaluate the linear dependency between the transformer's output and $\mathbf{y}$. We add the transformer when the p-value of this test is lower than a given significance level α. While other criteria can be used to measure non-linear dependencies (e.g., mutual information), we only consider linear dependencies to limit the computational cost of the test.

5 Experiments

To experimentally evaluate the proposed system, we address the following research questions:

Q1 How does our system compare to (1) existing feature-based methods, and (2) neural networks, in terms of accuracy and computational cost?
Q2 Can the fusion of multiple time series improve the feature representation of univariate feature construction?

5.1 Datasets

We selected three real-world multi-view time series datasets to evaluate our system. In all datasets, the time series are recorded using multiple sensors, where each sensor's data corresponds to one view. The sensors are different in terms of the measured variables as well as the sample rate at which they operate. We describe the datasets below and summarize their properties in Table 1.

mHealth The Mobile Health dataset is a human activity recognition dataset collected from 10 participants [4, 5]. The input data consists of measurements generated by eight sensors at a sample rate of 50Hz. The sensors include seven inertial sensors at three body parts (one accelerometer at the chest, and two accelerometers, gyroscopes and magnetometers at the left ankle and the right lower arm). Each of these sensors has three dimensions, i.e., one series for each axis. Apart from the inertial sensors, a 2-lead ECG sensor was attached to the chest. We split the sensor readings in sliding windows with a length of $5.12s$ (256 samples) and a step size of $1s$, which is the same windowing strategy used in previous studies [23]. The task is to predict for each window which activity is performed.

Skoda The Skoda mini checkpoint dataset is a gesture recognition dataset collected from a person working in a car quality control checkpoint [30]. The sensors consist of accelerometers positioned at different locations of the person's left and right arm, 10 at each side, resulting in a total of 20 views. Similar to the mHealth dataset, each accelerometer has three dimensions. Each repetition of a gesture is considered as one instance. Since some gestures take longer than others, the repetitions contain 69–1713 samples. With an approximate sample rate of 98Hz this corresponds to instances of 0.7–$17.5s$ in duration. The goal is to detect the gesture that the person performs.

UnoViS The Unobtrusive Vital Sign dataset is a medical monitoring database with unobtrusive vital sign data [27]. It consists of multiple datasets collected in different settings. Here, we use the dataset collected from 10 subjects lying on a bed [26]. The dataset is collected using six sensors, each generating a one-dimensional series at a sample rate of 200Hz. The sensors include three capacitive electrocardiography (cECG) sensors and three optical photoplethysmography (PPG) sensors. The goal is to detect the subject's heart beats as annotated using a reference ECG sensor. The series are split into windows of $0.5s$ (101 samples) in such a way that the center t of each window (the 51^{th} sample) corresponds to a heart beat. Additionally, for each such window there is a shifted window of the same length, starting at a time stamp randomly chosen from $[t - 100, t - 20] \cup [t + 20, t + 100]$. The target label is a binary variable where a positive example corresponds to a window with a heart beat in its center.

5.2 Approaches

This section lists the methods that we evaluate in our experiments.

Our Approach In order to evaluate the benefit of constructing fused views, we consider two versions of our system:

- $\texttt{TSFuse}_{\mathbf{R}}$ This version skips step 2 of Algorithm 1 and only uses the raw input views to construct the features.

Table 1. Dataset properties.

	mHealth	Skoda	UnoViS
Views	8	20	6
Dimensions	23	60	6
Sample rate (Hz)	50	98	200
Samples per instance	256	69–1713	101
Instances	6329	700	14918
Target	Activity	Gesture	Heart beat
Classes	12	10	2

`TSFuse`$_\mathtt{F}$ This version applies the complete construction method and hence uses both the raw input views and the constructed fused views.

Like `tsfresh`, `TSFuse` has three settings which employ different subsets of the series-to-attribute transformers:

– *minimal*: a small set of statistical transformations;
– *fast*: the transformations for which `TSFuse` has a fast C implementation;
– *full*: all transformers that are implemented in `TSFuse`.

Selecting one of these settings allows control of the computational cost of the construction method. However, it also involves making a trade-off between time and expressivity. Appendix A denotes which transformers apply in each setting.

Baselines We compare our system with two baselines:

`tsfresh` [8] This feature-based approach is currently a popular time series feature construction tool. It is designed for univariate time series, but can work for multiple series as well by considering each series separately. `tsfresh` is run under the same three sets of series-to-attribute transformers as `TSFuse`: *minimal*, *fast*, and *full*.

LSTM [17] Long short-term memory (LSTM) neural networks are a form of a recurrent deep neural network that are often applied to time series classification tasks. Even though there exist many other types of architectures [10], we restrict our comparison to one type.

The feature-based methods (`TSFuse` and `tsfresh`) only return a set of features. Therefore, we train a gradient boosted tree model on top of the features constructed by these methods.

5.3 Methodology

This section describes how we evaluate the different methods.

Evaluation Metrics We report two metrics. First, we report the accuracy of the methods. Second, we report their run time as the wall clock time. For the feature-based methods, this includes the time to construct the features and learn the model. All experiments are run on a single CPU.

Cross-Validation We employ 5-fold cross-validation. In the mHealth and UnoViS datasets, the instances represent trials collected from different subjects. Therefore, we split the data on the subject level to ensure that a subject does not appear in both the training and testing data. For the Skoda dataset, the trials are collected from a single subject. The instances of this dataset are split in a stratified way, i.e., making sure that the percentage of instances for each class is preserved.

Implementation Details As shown in Algorithm 1, `TSFuse` requires four parameters: a list of series-to-series transformers S and series-to-attribute transformers A, a maximal correlation coefficient c for the redundancy test, and a significance level α for the relevance test. The series-to-attribute transformers correspond to the three settings described above. For the series-to-series transformers, the system currently uses four transformers: the resultant of the series of a single view, and the ratio, absolute difference and relative difference between the series of a pair of views. We set the maximal correlation coefficient $c = 0.99$ in all experiments. For the relevance test, we set $\alpha = 0.05$ which is a commonly used default value in filter feature selection methods, for example, in the false positive rate test implemented in `scikit-learn`.

For the `tsfresh` baseline, we use the default parameter settings. That is, we use the default statistical tests for computing the p-values of the features, and use the default value of 0.05 for the false discovery rate of the Benjamini-Hochberg procedure that selects the features based on these p-values.

For training the gradient boosted tree models, we employ XGBoost [6] with `subsample` = 0.9 for subsampling the instances and `subsample_bynode` = 0.5 for subsampling the features, which reduces the risk for overfitting. We use the default values for all other parameters. Since the subsampling steps involve a random selection of instances and features, we repeat the evaluation three times and average the results.

For the LSTM network, we set up an architecture where each view's data (a multivariate time series) is given as an input to an LSTM layer with 128 hidden units. We concatenate the hidden units of all LSTM layers. To prevent overfitting, we use a dropout layer with a 0.5 dropout ratio. The output layer is a softmax layer with one output per class. To train the networks, we use Keras [7] with TensorFlow [1] as the backend engine. We use the Adam optimizer and set the number of epochs to 100.

5.4 Results

In this section, we evaluate our system by comparing it to feature-based and neural network baselines (**Q1**). Additionally, we evaluate the benefit of fusing multiple series (**Q2**) compared to a univariate approach which considers each series separately.

Results for Q1 Table 2 shows the accuracy for all approaches. `TSFuse` is always able to find a better or equivalently good feature representation as `tsfresh` on all three datasets. For the Skoda dataset, $\texttt{TSFuse}_{\mathrm{F}}$ improves the feature representation by selecting input views and constructing new views. Note that for the other two datasets, $\texttt{TSFuse}_{\mathrm{R}}$ selects all input views, and hence evaluates the same features as `tsfresh`. For these datasets, the difference in accuracy is due to using a different test for selecting the relevant attributes.

Interestingly, the feature-based approaches yield big wins compared to the LSTM network in terms of accuracy on the Skoda (all feature settings) and UnoViS (on the fast and full feature settings) datasets. They perform equivalently on the mHealth dataset. Note that the datasets used in our experiments can be considered medium-sized datasets. On the one hand, deep learning may work better for larger datasets, since deep neural networks have the potential to learn discriminative features that our feature construction may miss. On the other hand, for small datasets it may be hard to automatically select relevant features. A manual approach based on domain knowledge may be more suitable for such datasets.

Table 2. Accuracy of the models, averaged over the five cross-validation folds. For the feature-based methods, the table shows the results for three settings: minimal (min), fast, and full. On all datasets, `TSFuse` is able to find a better or equivalently good feature representation as `tsfresh`.

Method		mHealth	Skoda	UnoViS
`tsfresh`	min	0.942	0.979	0.808
$\texttt{TSFuse}_{\mathrm{R}}$		0.951	0.985	0.811
$\texttt{TSFuse}_{\mathrm{F}}$		0.952	0.985	0.806
`tsfresh`	fast	0.946	0.990	0.874
$\texttt{TSFuse}_{\mathrm{R}}$		**0.957**	0.989	0.874
$\texttt{TSFuse}_{\mathrm{F}}$		0.956	**0.996**	0.859
`tsfresh`	full	0.953	0.990	0.872
$\texttt{TSFuse}_{\mathrm{R}}$		**0.957**	0.992	**0.899**
$\texttt{TSFuse}_{\mathrm{F}}$		0.943	0.992	0.876
LSTM network		0.951	0.881	0.814

Table 3 shows the run times for all approaches. $\texttt{TSFuse}_{\mathrm{R}}$ outperforms `tsfresh` in all cases. For $\texttt{TSFuse}_{\mathrm{F}}$, the run times are lower than `tsfresh` in 8 out of 9 cases, even though it considers a much larger number of time series. The speed-up is due to three reasons. First, `TSFuse` selects a subset of the input views by removing redundant views. This accounts for some of the difference in the Skoda datasets where only 10 out 20 views (see Table 4) are selected, but `TSFuse` selects all views in the other two datasets. Second, our method uses a different test for selecting the relevant attributes. Different from `tsfresh`, this test does not control the false discovery rate, but instead selects features directly based on the p-values, which saves time. Third, the minimal and fast transformers are implemented more efficiently in `TSFuse` since they avoid looping over the data using Python.

The feature-based approaches are substantially faster to train than the LSTM network. Even though deep learning has been applied to each of these datasets in related work [2, 21, 28], our comparison shows that the feature-based methods can find more accurate models in a more computationally efficient way.

Whereas our experiments evaluate the system for different settings of S and A, we did not consider different values for the maximal correlation coefficient c and the significance level α. Similar to the false discovery rate parameter of `tsfresh`, c and α affect the accuracy and run times of our system. For the datasets considered in our experiments, the chosen parameter settings result in a reasonable trade-off between the computational cost of the construction method and the accuracy of the models trained with the constructed features. Other datasets may need different parameter settings, e.g., noisy datasets where all p-values are larger than 0.05. Future work can look into ways to set these parameters automatically based on the given time series data.

Table 3. Run time (in seconds), reported as mean ± standard deviation aggregated over the five cross-validation folds.

Method		mHealth	Skoda	UnoViS
`tsfresh`	min	158.9 ± 18.1	135.6 ± 7.9	95.2 ± 3.5
`TSFuse`$_{\mathrm{R}}$	min	23.3 ± 0.4	10.8 ± 0.7	2.4 ± 0.2
`TSFuse`$_{\mathrm{F}}$	min	113.4 ± 5.5	45.9 ± 3.3	20.8 ± 1.6
`tsfresh`	fast	2103.2 ± 249.0	1190.6 ± 104.1	1079.4 ± 234.1
`TSFuse`$_{\mathrm{R}}$	fast	317.1 ± 3.8	108.9 ± 5.5	104.0 ± 2.5
`TSFuse`$_{\mathrm{F}}$	fast	1151.6 ± 53.0	197.8 ± 12.0	518.7 ± 44.5
`tsfresh`	full	10 106.3 ± 507.9	6018.0 ± 125.8	7664.6 ± 719.3
`TSFuse`$_{\mathrm{R}}$	full	3917.6 ± 64.6	2367.4 ± 134.5	1039.0 ± 36.7
`TSFuse`$_{\mathrm{F}}$	full	12 821.6 ± 661.4	3688.3 ± 163.1	5267.5 ± 589.5
LSTM network		34 553.9 ± 947.3	59 266.0 ± 1117.5	23 753.7 ± 353.9

Results for Q2 To evaluate the benefit of fusing multiple series, Table 2 compares `TSFuse`$_{\mathrm{R}}$ (without fusion) with `TSFuse`$_{\mathrm{F}}$ (with fusion). For the Skoda dataset, the most accurate model is obtained with `TSFuse`$_{\mathrm{F}}$ using the fast transformers. This shows that fusing time series can be more beneficial than using a more extensive set of series-to-attribute transformers. On the other datasets and settings, sometimes fusing helps a little, and other times it slightly degrades the performance.

In terms of run time, `TSFuse`$_{\mathrm{F}}$ is computationally more expensive than `TSFuse`$_{\mathrm{R}}$ since it requires computing features for a larger number of views.

The run times heavily depend on the number of constructed views. With the series-to-series transformers currently implemented in the system, the number of views that can possibly be constructed is $O(V^2)$ where V is the number of input views.[1] Therefore, the redundancy test of steps 1 and 2 is an important check to keep the number of views low. To evaluate the effect of this test, Table 4 shows the total number of views that are selected (in step 1) and constructed (in step 2) for all datasets. As can be seen from this table, the number of constructed views is typically much lower than the total number of possibilities. This makes sure that the run times of $\texttt{TSFuse}_\mathrm{F}$ do not increase with a factor $O(V^2)$ compared to $\texttt{TSFuse}_\mathrm{R}$.

Table 4. Number of selected raw views V_R and fused views V_F, and the total number of views ($V_\mathrm{R} + V_\mathrm{F}$) generated by $\texttt{TSFuse}_\mathrm{F}$. The table also shows the total number of views that our method would construct without the redundancy test.

Views	mHealth	Skoda	UnoViS
V_R	8 out of 8	10 out of 20	6 out of 6
V_F	30 out of 176	53 out of 1160	43 out of 96
$V_\mathrm{R} + V_\mathrm{F}$	38 out of 184	63 out of 1180	49 out of 102

6 Conclusion

This paper presented `TSFuse`, a feature construction system designed for multi-view series data. The key contribution of the proposed construction method is the ability to fuse multiple series instead of considering the series independently from each other. We evaluated the system on three real-world datasets containing time series collected by different types of sensors. Empirically, we found that our system is able to improve the feature representation constructed by existing univariate approaches. In terms of run time, it is able to do so in a computationally efficient way compared to both feature-based and neural network baselines.

Acknowledgments

ADB and PR are supported by the EU Interreg VA project Nano4Sports. TODB is supported by the KU Leuven Research Fund (C22/15/015). JD is partially supported by the KU Leuven Research Fund (C14/17/07, C32/17/036) and the EU Interreg VA project Nano4Sports.

[1] Specifically, the number of views that can be constructed is $V + 3 \cdot V \cdot (V - 1)$. The system currently uses four series-to-series transformers: the resultant of the series of a single view, and the ratio, absolute difference and relative difference between pairs of views. The number of views that can be constructed using the resultant transformer is the number of input views V. Each of the other three transformers can construct $V \cdot (V - 1)$ views.

References

1. Abadi, M., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), http://tensorflow.org/
2. Antink, C.H., Breuer, E., Uguz, D.U., Leonhardt, S.: Signal-level fusion with convolutional neural networks for capacitively coupled ecg in the car. Computing **45**, 1 (2018)
3. Bagnall, A., Lines, J., Bostrom, A., Large, J., Keogh, E.: The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery **31**(3), 606–660 (2017)
4. Banos, O., Garcia, R., Holgado-Terriza, J.A., Damas, M., Pomares, H., Rojas, I., Saez, A., Villalonga, C.: mhealthdroid: a novel framework for agile development of mobile health applications. In: International Workshop on Ambient Assisted Living. pp. 91–98. Springer (2014)
5. Banos, O., Villalonga, C., Garcia, R., Saez, A., Damas, M., Holgado-Terriza, J.A., Lee, S., Pomares, H., Rojas, I.: Design, implementation and validation of a novel open framework for agile development of mobile health applications. Biomedical engineering online **14**(S2–S6), 1–20 (2015)
6. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794. KDD '16, ACM, New York, NY, USA (2016)
7. Chollet, F., et al.: Keras. https://keras.io (2015)
8. Christ, M., Braun, N., Neuffer, J., Kempa-Liehr, A.W.: Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package). Neurocomputing (2018)
9. Decroos, T., Schütte, K., De Beéck, T.O., Vanwanseele, B., Davis, J.: Amie: Automatic monitoring of indoor exercises. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 424–439. Springer (2018)
10. Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review. Data Mining and Knowledge Discovery **33**(4), 917–963 (2019)
11. Fulcher, B.D.: Feature-based time-series analysis. In: Feature Engineering for Machine Learning and Data Analytics, pp. 87–116. CRC Press (2018)
12. Fulcher, B.D., Jones, N.S.: Highly comparative feature-based time-series classification. IEEE Transactions on Knowledge and Data Engineering **26**(12), 3026–3037 (2014)
13. Fulcher, B.D., Jones, N.S.: hctsa: A computational framework for automated time-series phenotyping using massive feature extraction. Cell systems **5**(5), 527–531 (2017)
14. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. Journal of machine learning research **3**(Mar), 1157–1182 (2003)
15. Harvey, D.Y., Todd, M.D.: Automated feature design for numeric sequence classification by genetic programming. IEEE Transactions on Evolutionary Computation **19**(4), 474–489 (2015)
16. Helwig, N., Pignanelli, E., Schütze, A.: Condition monitoring of a complex hydraulic system using multivariate statistics. In: 2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings. pp. 210–215. IEEE (2015)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)

18. Längkvist, M., Karlsson, L., Loutfi, A.: A review of unsupervised feature learning and deep learning for time-series modeling. Pattern Recognition Letters **42**, 11–24 (2014)
19. Li, S., Li, Y., Fu, Y.: Multi-view time series classification: A discriminative bilinear projection approach. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. pp. 989–998. ACM (2016)
20. Mierswa, I.: Automatic feature extraction from large time series. In: Classification – the Ubiquitous Challenge, pp. 600–607. Springer (2005)
21. Mohammad, Y., Matsumoto, K., Hoashi, K.: Deep feature learning and selection for activity recognition. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing. pp. 930–939. ACM (2018)
22. Op De Beéck, T., Meert, W., Schütte, K., Vanwanseele, B., Davis, J.: Fatigue prediction in outdoor runners via machine learning and sensor fusion. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 606–615. ACM (2018)
23. Reiss, A., Stricker, D.: Introducing a new benchmarked dataset for activity monitoring. In: Wearable Computers (ISWC), 2012 16th International Symposium on. pp. 108–109. IEEE (2012)
24. Schuller, B., Reiter, S., Rigoll, G.: Evolutionary feature generation in speech emotion recognition. In: IEEE International Conference on Multimedia and Expo. pp. 5–8. IEEE (2006)
25. Vercruyssen, V., Meert, W., Verbruggen, G., Maes, K., Baumer, R., Davis, J.: Semi-supervised anomaly detection with an application to water analytics. In: ICDM. pp. 527–536 (2018)
26. Wartzek, T., Brüser, C., Schlebusch, T., Brendle, C., Santos, S., Kerekes, A., Gerlach-Hahn, K., Weyer, S., Lunze, K., Antink, C.H., et al.: Modeling of motion artifacts in contactless heart rate measurements. In: Computing in Cardiology 2013. pp. 931–934. IEEE (2013)
27. Wartzek, T., Czaplik, M., Antink, C.H., Eilebrecht, B., Walocha, R., Leonhardt, S.: Unovis: the medit public unobtrusive vital signs database. Health information science and systems **3**(1), 2 (2015)
28. Yuan, Y., Xun, G., Ma, F., Wang, Y., Du, N., Jia, K., Su, L., Zhang, A.: Muvan: A multi-view attention network for multivariate temporal data. In: 2018 IEEE International Conference on Data Mining (ICDM). pp. 717–726. IEEE (2018)
29. Zagorecki, A.: Prediction of methane outbreaks in coal mines from multivariate time series using random forest. In: Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, pp. 494–500. Springer (2015)
30. Zappi, P., Lombriser, C., Stiefmeier, T., Farella, E., Roggen, D., Benini, L., Tröster, G.: Activity recognition from on-body sensors: Accuracy-power trade-off by dynamic sensor selection. In: Verdone, R. (ed.) Wireless Sensor Networks. pp. 17–33 (2008)

A Series-to-Attribute Transformers

	Min	Fast	Full	Parameters
ArgMax		x	x	first ∈ {t, f}
ArgMin		x	x	first ∈ {t, f}
AutoCorrelation		x	x	
AutoRegressiveCoefficients			x	i ∈ {0, 1, . . . , 9}

	Min	Fast	Full	Parameters
BinnedEntropy		x	x	bins = 10
C3		x	x	lag ∈ {1, 2, 3}
CID		x	x	
CWT			x	i ∈ {0, 1, . . . , 9}
CountAboveMean		x	x	
CountBelowMean		x	x	
Energy		x	x	
EnergyRatio		x	x	chunks = 10
FFT			x	i ∈ {0, 1, . . . , 99}
FriedrichCoefficients			x	m=3; r=30; i ∈ {0, 1, 2, 3}
HasDuplicate			x	
HasDuplicateMax		x	x	
HasDuplicateMin		x	x	
HighStandardDeviation		x	x	r ∈ {.1, .2, .3, .4, .6, .7, .8, .9}
HighVariance		x	x	
IndexMassQuantile		x	x	q ∈ {.1, .2, .3, .4, .6, .7, .8, .9}
Kurtosis	x	x	x	
Length	x	x	x	
LinearTrend			x	
LongestStrikeAboveMean		x	x	
LongestStrikeBelowMean		x	x	
Max	x	x	x	
MaxLangevinFixedPoint			x	m=3; r=30
Mean	x	x	x	
MeanChange		x	x	abs ∈ {t, f}
MeanSecondDerivativeCentral		x	x	
Median	x	x	x	
Min	x	x	x	
NumberCrossings		x	x	threshold ∈ {-1, 0, 1}
NumberPeaksCWT			x	
NumberUniqueValues			x	
Outliers		x	x	r ∈ {1, 1.5, 2, 3, 4, 5}; rel ∈ {t, f}
PowerSpectralDensity			x	
Quantile			x	q ∈ {.1, .2, .3, .4, .6, .7, .8, .9}
RangeCount		x	x	(min, max) ∈ {(-1, 1), (-∞, 0), (0, ∞)}
Skewness	x	x	x	
SpectralKurtosis(FFT)			x	
SpectralMean(FFT)			x	
SpectralSkewness(FFT)			x	
SpectralVariance(FFT)			x	
StandardDeviation	x	x	x	
Sum	x	x	x	
SumChange		x	x	
SumReoccurringDataPoints			x	
SumReoccurringValues			x	
SymmetryLooking		x	x	r ∈ {.1, .2, .3, .4, .6, .7, .8, .9}
TimeReversalAsymmetryStatistic		x	x	lag ∈ {1, 2, 3}
ValueCount		x	x	value ∈ {-1, 0, 1}
Variance	x	x	x	