Interpretable Time Series Classification with Interval-Based Forests

From Random Sampling to Stratified Selection in TSF and STSF

Lars Tönsing
University of Würzburg

ABSTRACT

Time series classification (TSC) presents significant challenges in domains requiring both high efficiency and interpretability, particularly for long and high-frequency sensor data. Existing state-of-the-art models achieve strong performance but often suffer from prohibitive computational costs and opaque decision processes. This paper introduces the Supervised Time Series Forest (STSF), a novel interval-based TSC method that directly addresses these limitations. STSF employs a supervised binary search strategy across three complementary time series representations—original, derivative, and frequency domain—guided by feature ranking metrics such as the Fisher score. Unlike the Time Series Forest (TSF), which relies on random interval sampling, STSF drastically reduces the feature space from quadratic to logarithmic scale, enabling orders-of-magnitude faster training times. Experimental results on 85 benchmark datasets demonstrate that STSF achieves classification accuracy on par with state-of-the-art methods like HIVE-COTE and TS-CHIEF, while maintaining linear complexity in practice. Additionally, STSF enables interpretable classification via regions of interest derived from discriminatory interval overlaps. The proposed method bridges the gap between performance, scalability, and interpretability, offering a viable TSC solution for modern data-intensive applications.

1 INTRODUCTION

Time series classification is a supervised learning task of assigning labels to sequences of observations. There are two basic approaches to this work.

Instance-based methods use regular classifiers that treat each time point as a feature. An example for that is One-Nearest-Neighbor With Dynamic Time Warping (NNDTW), which is robust to the distortion of the time axis and exceptionally difficult to beat. This method however offers little interpretability regarding which temporal regions or patterns differentiate the classes.

In contrast feature-based methods use temporal features calculated over time series intervals. These are called interval features and they capture localized patterns that may be highly discriminative, even when the same patterns occur at slightly different times. They provide more interpretable models, because they allow to identify specific regions that contribute to classification decisions. These classification structures are then called decision trees

Previous work has constructed decision trees using class-based measures such as entropy gain, evaluating numerous candidate splits derived from interval-based features. However, many of these splits exhibit similar class-separation ability. To resolve such ambiguities, more refined measures are needed to differentiate between equally effective splits. Additionally, there is demand for a classifier that is not only accurate and efficient but also capable of revealing which temporal characteristics drive classification decisions.

Seminar: Ausgewählte Themen des Machine Learning, SS 2025

To fulfill this demand TSF introduces a new split criterion called Entrance Gain. This measure has been shown to outperform Entropy Gain and also NNDTW algorithms. Computational complexity remains efficient and scalable, because of a random sampling strategy.

TSF also has its shortcomings however. Because of the random sampling strategy of the intervals, alignment with meaningful temporal patterns is reduced, leading to suboptimal accuracy and interpretability. To alleviate this problem STSF and r-STSF are introduced. STSF addresses this by generating a fixed pool of intervals prior to training, thus leading to higher alignment to discriminative temporal segments. r-STSF builds on this by stratifying interval sampling by class, allowing to capture class-specific temporal patterns further boosting performance and robustness.

2 BASICS

2.1 Interval Features

On a time series interval starting at t_{start} and ending at t_{end} an interval feature is a statistical measure on the values between those two points. Let K be the number of feature types and $f_k((t_{\text{start}}, t_{\text{end}}))(k = 1, 2, \ldots, K)$ be the k-th type.

For TSF the three types: $f_1 = mean$, $f_2 = standard\ deviation$ and $f_3 = slope$ are chosen and defined as:

$$f_1(t_{\text{start}}, t_{\text{end}}) = \frac{\sum_{i=t_{\text{start}}}^{t_{\text{end}}} v_i}{t_{\text{end}} - t_{\text{start}} + 1}$$

$$f_2(t_{\text{start}}, t_{\text{end}}) = \begin{cases} \sqrt{\frac{\sum_{i=t_{\text{start}}}^{t_{\text{end}}} (v_i - f_1(t_{\text{start}}, t_{\text{end}}))^2}{t_{\text{end}} - t_{\text{start}}}} & t_{\text{end}} > t_{\text{start}} \\ 0 & t_{\text{end}} = t_{\text{start}} \end{cases}$$

$$f_3(t_{\text{start}}, t_{\text{end}}) = \begin{cases} \hat{\beta} & t_{\text{end}} > t_{\text{start}} \\ 0 & t_{\text{end}} = t_{\text{start}} \end{cases}$$

where $\hat{\beta}$ is the slope of the least squares regression line of the training set $\{(t_{\text{start}}, v_{t_{\text{start}}}), (t_{\text{start}} + 1, v_{t_{\text{start}}+1}), \ldots, (t_{\text{end}}, v_{t_{\text{end}}})\}$.

In the here considered algorithm TSF only these three interval features are considered. This is because they offer high interpretability and low computational costs. STSF uses additionally the statistical features: $f_4 = median$, $f_5 = interquartileRange$, $f_6 = min$ and $f_7 = max$.

However using a larger set of feature space, using more complex and potentially more discriminative interval features is possible. The TS-CHIEF classifier, for example, uses more features . Incorporating such features would substantially increase complexity and reduce the transparency of the model, diverging from the design priorities of TSF and r-STSF and are therefore not considered.

2.2 Decision Trees and Random Forests

Definition

Decision trees are binary trees used for classification. Each tree consists of two types of components: internal nodes and leaves.

Each internal node has two children and contains a splitting function that determines whether incoming data is passed to the left or right child. The splitting function follows the pattern:

$$f_*({t_{\text{start}}}^*, {t_{\text{end}}}^*) \leq \tau^*$$

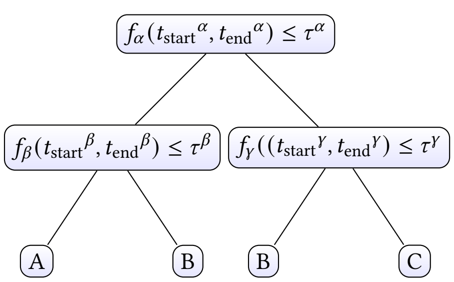
where f_* is a chosen interval feature, $({t_{\text{start}}}^*, {t_{\text{end}}}^*)$ is the selected time interval, and τ^* is a threshold.

In practice, this means each input is evaluated on a specific interval using a particular feature. If the result is less than or equal to the threshold, the data is passed to the left child; otherwise, it is passed to the right child.

Leaves, the second component type, do not perform any splits. They assign a class label to any data that reaches them. Nodes may have either other nodes or leaves as children.

Let A, B, and C be class labels, and α, β, γ denote the chosen parameters at each internal node. A decision tree can be visualized as follows:

Visualized Decision Tree



Random Forests

To now classify a timeseries not just one tree is used. Multiple trees give their class prediciton, which is then aggregated and shown as a class probabilty. For this process to make sense each tree should look at a diverse set of features. This is done by using a random sampling strategy in the training of each tree.

The combination of many relatively uncorrelated trees can produce a forest whose prediction is more accurate than that of any of the individual trees.

2.3 Entropy and Entropy Gain

Entropy gain is a widely used criterion to evaluate a split in a tree node. It quantifices the reduction in class impurity after a given split. The higher the entropy gain, the better the seperation.

Let $\{1, 2, ..., C\} := \mathfrak{C}$ be all class labels and γ_c with $c \in \mathfrak{C}$ the proportion of the class c in the current dataset. Then entropy is defined as

$$\text{Entropy}(\text{data}) := -\sum_{c=1}^{C} \gamma_c \log \gamma_c$$

At a node the dataset is split into two and passed to the nodes children. Those new datasets have a different entropy than the parents dataset, with a lower entropy meaning a better purity. Let ds_p with $n_p = |\text{ds}_p|$ be the parents dataset and ds_l, ds_r with $n_l, n_r = |\text{ds}_l|, |\text{ds}_r|$ the childrens. To now measure the entropy gain following formular is used.

$$\Delta\,\text{Entropy} = \text{Entropy}(\text{ds}_p) - \left(\frac{n_l}{n_p}\,\text{Entropy}(\text{ds}_l) + \frac{n_r}{n_p}\,\text{Entropy}(\text{ds}_r)\right)$$

So lower entropy meaning higher purity in the children adjustet for the sizes of the datasets will lead to higher entropy gain.

2.4 Supervised Interval Selection

2.5 Time Series Representation

3 TIME SERIES FOREST CLASSIFIER (TSF)

3.1 Entrance Gain

Entropy gain however has its limits, because "the number of candidate splits can be large, and there are often cases where multiple candidate splits have the same Δ Entropy.

The paper introduced the splitting criterion entrance gain. For this the difference between the the threshold τ and the value of the interval feature of all timeseries in the dataset is considered. The minimum of all those values is called the Margin.

For a split $f_k^n(t_{\text{start}}, t_{\text{end}}) \leq \tau$ for the timeseries n

$$\text{Margin} = \min_{n=1,2,\ldots,N} \left| f_k^n(t_{\text{start}}, t_{\text{end}}) - \tau \right|$$

Entrance gain E then is the combination of this Margin and ΔEntropy with a a scalar α small enough to make the entrance gain a tiebreaker and nothing more.

$$E = \Delta\,\text{Entropy} + \alpha \cdot \text{Margin}$$

3.2 Main algorithm

To build now a tree a recursive function is used. In each recursion step a single node of the tree is created. Each node holds an entrance gain E^*, an entropy gain $\Delta Entropy^*$, the interval $(t_{\text{start}}, t_{\text{end}})^*$, a threshold τ^* and an interval feature f_*.

Random Sampling Strategy

The feature space of all possible intervals for a timeseries the length of M is $O(M^2)$. To reduce this feature space an algorithm $SampleIntervals()$ is introduced. It first randomly samples $\sqrt{M}$ interval sizes. Then for each interval size w it selects a set of starting points from $\{1, \ldots, M - w + 1\}$. The size of the set is $\sqrt{M - w + 1}$, so that there are fewer starting points for larger interval sizes. For all these starting points t_{start} the point $t_{end} = t_{start} + w - 1$ is added to the set of all intervals, so that the feature space gets reduced to $O(\sqrt{M} \cdot \sqrt{M}) = O(M)$.

Algorithm 1 SAMPLEINTERVALS(M), The function RandSampNoRep(set, samplesize) randomly selects samplesize elements from set without replacement.

$T_{start} \leftarrow \emptyset$, $T_{end} \leftarrow \emptyset$
$W \leftarrow$ RANDSAMPNOREP($\{1, \ldots, M\}, \sqrt{M}$)
for all $w \in W$ **do**
  $S \leftarrow$ RANDSAMPNOREP($\{1, \ldots, M - w + 1\}, \sqrt{M - w + 1}$)
  for all $t_{start} \in S$ **do**
    $T_{start} \leftarrow T_{start} \cup \{t_{start}\}$
    $T_{end} \leftarrow T_{end} \cup \{t_{start} + w - 1\}$
  end for
end for
return $\langle T_1, T_2 \rangle$

Threshold Set Calculation

To identify suitable split points for each feature type f_k, TSF constructs a fixed-size set of candidate thresholds. Let $f_k(t_{\text{start}}, t_{\text{end}})$ denote the k-th interval feature computed over the interval $(t_{\text{start}}, t_{\text{end}})$ for all training instances at the current node. Let $V = \{f_k^{(i)}(t_{\text{start}}, t_{\text{end}}) \mid i = 1, \ldots, N\}$ be the set of resulting feature values across all N instances.

TSF then assigns a fixed number of κ threshold per feature. This is achieved by first computing the minimum and maximum values in V, denoted $a = \min V$ and $b = \max V$, respectively. The interval $[a, b]$ is then divided into $\kappa+1$ equal-width segments. The κ internal division points are used as candidate thresholds:

$$\tau_j = a + j \cdot \frac{b-a}{\kappa+1}, \quad j = 1, 2, \ldots, \kappa$$

This method ensures that the thresholds are uniformly spaced across the feature value range and that their number remains constant regardless of data distribution or sample size.

Recursion and Termination

After getting the interval set from the $SampleIntervals()$ algorithm and computing the threshold-set for each feature type, TSF loops over every possible interval, every possible threshold and every possible interval feature. It then calculates the entropy and entrance gain for each combination and maximizes the entrance gain and saves the split data. Afterwards it checks if the entropy gain is zero. This is the termination condition for the recursion and the node is marked as a leaf. Then the data is split according to the best features, interval and threshold and the function is called on the left and right dataset to build the children of the node.

Algorithm 2 BUILDTSFTREE($\mathcal{D}$)

$\langle T_1, T_2 \rangle \leftarrow$ SAMPLEINTERVALS(M)
Compute candidate thresholds Threshold$_k$ for each feature type k
Initialize $E^* \leftarrow 0$, $\Delta H^* \leftarrow 0$, $t_1^* \leftarrow 0$, $t_2^* \leftarrow 0$, $\tau^* \leftarrow 0$, $f^* \leftarrow \emptyset$
for all $(t_{start}, t_{end}) \in \langle T_{start}, T_{end} \rangle$ **do**
  for $k = 1$ to K **do**
    for all $\tau \in$ Threshold$_k$ **do**
      Compute $\Delta Entropy$ and E for $f_k(t_{start}, t_{end}) \leq \tau$
      if $E > E^*$ **then**
        $E^* \leftarrow E$,
        $\Delta Entropy^* \leftarrow \Delta Entropy$,
        $t_{start}^* \leftarrow t_{start}$,
        $t_{end}^* \leftarrow t_{end}$,
        $\tau^* \leftarrow \tau$,
        $f^* \leftarrow f_k$
      end if
    end for
  end for
end for
if $\Delta Entropy^* = 0$ **then**
  return leaf node with majority class of $\mathcal{D}$
end if
$\mathcal{D}_{\text{left}} \leftarrow \{(X, y) \in \mathcal{D} \mid f^*(X[t_{start}^* : t_{end}^*]) \leq \tau^*\}$
$\mathcal{D}_{\text{right}} \leftarrow \{(X, y) \in \mathcal{D} \mid f^*(X[t_{start}^* : t_{end}^*]) > \tau^*\}$
node.left $\leftarrow$ BUILDTSFTREE($\mathcal{D}_{\text{left}}$)
node.right $\leftarrow$ BUILDTSFTREE($\mathcal{D}_{\text{right}}$)
return internal node with $(f^*, t_{start}^*, t_{end}^*, \tau^*, \text{Left}, \text{Right})$

3.3 Complexity

3.4 Temporal Importance Curve

"TSF consists of multiple trees and is difficult to understand", so there is need for a method to extract interpretability from a forest. The temporal importance curve is introduced to reveal which parts of the time series contribute most to the classification decision. For that am importance function is built for each feature type and used for all time points of the dataset. The Curve then is built by graphing those results out.

The importance function is based on the entropy gain of each split node v in the forest SN for the given feature type. All the nodes which intervals time-point t are considered and their entropy gain for the given feature type is summed up. If the feature is not used in the split of the node, the entropy gain is set to zero.

$$\text{Imp}_k(t) = \sum_{t_{start} \leq t \leq t_{end}, v \in SN}$$

The method reveals temporal patterns in the data relevant for classification. Peaks in the curve indicate time regions used by strong splits. Because points in the center of the series appear in more intervals, there is a natural bias towards them. To counteract this, the authors compare uninformative datasets with the calculated importance curve.

In practice, the curves highlight discriminative intervals successfully As an example the temporal importance curves peak precisely in regions with specific intervals

Using this method of observing the strength of the splits, it can be shown that entrance gain gives sharper and more localized peaks, thus improving the interpretability and specificity of the importance curves.

4 METHOD 2

4.1 Approach

STSF uses a similar algorithm as TSF to build decision trees. However the difference lies in the feature space used, the time series representation and the sampling strategy. The goal is to reduce the interval space size to $O(\log M)$ and to find more discriminate intervals to achieve better classification. Instead of relying on random intervals, it employs a supervised search to identify informative intervals across multiple representations.

4.2 Time Series Representations

In TSF only the original/time-domain representation of a time series is looked at. To capture a broader range of discriminations STSF uses multiple representations.

Periodogram

Using a discrete Fourier transform the time series is mapped into the frequency domain. This allows STSF to detect intervals with discriminative periodic patterns, even if they occur at varying time points across instances. "A side benefit of this representation is that it helps to indirectly detect phase-independent discriminatory intervals, i.e.m discriminatory features located at different time regions of the original series as described in Fig."

Derivative representation

STSF also adopts the derivative representation as one of three complementary views on the input time series. The derivative captures local trends. This view emphasizes hte shape or dynamic behavior of the series over absolute levels. "The discrete derivative of a time series f with length n is defined by

$$f'(i) = f(i) - f(i-1) \forall\ i \in \{2, 3, \ldots, n\}$$

4.3 Fisher Score

4.4 Supervised Search

The algorithm loops over all 3 representation and gives back samples denoted as A_O, A_F and A_D respectively. All together then are used as the samples for the TSF algorithm $A^* = A_O \cup A_F \cup A_D$ Each Set A_R is generated by recursively searching for class discriminative interval features across multiple random partitions of the input data. Let $X \in \mathbb{R}^{n \times m}$ be the dataset of n univariate time series of length m and let $y \in \{1, \ldots, c\}^n$ be the vector for class labels. For every interval feature f_k the algorithm select a random split index $u \in \{1, \ldots, m-1\}$ to divide each time series into a left and right sub-series. $X_L \in \mathbb{R}^{n \times u}, X_R \in \mathbb{R}^{n \times (m-u)}$ This randomized partitioning is repeated a fixed number of times to increase diversity of the search space. For each representation and aggregation function, the algorithm then applies a recursive supervised function on both halves via SupervisedSearch(X_L, f_k, y, f_r = Fisher score, $A =$) and SupervisedSearch(X_L, f_k, y, f_r = Fisher score, $A =$)

This function works as follows: it recursively bisects the interval, evaluates the two resulting sub-intervals using the aggregation function f_k, and scores the resulting feature vector with the ranking metric f_r. The half with the higher score is retained in A and further partitioned, while the other is discarded. The termination of the function is when the interval length falls below 2.

Each path gives one interval feature, which is defined by its start and end indices and the function used. Since the loops are constant and the procedure follows a binary partitioning pattern, it returns $O(\log m)$ features per call. These features are collected into A_R and ultimately passed into the tree classifier, which selects final splits during training.

Algorithm 3 SupervisedSearch

Require: $X' \in \mathbb{R}^{n \times m'}$: time series set, $y \in \{1, \ldots, c\}^n$: class labels,
f: aggregation function, f_r: feature ranking metric, $\bar{A}$: set of candidate intervals
Ensure: Updated set $\bar{A}$ of discriminatory interval features
if $m' < 2$ **then**
  return $\bar{A}$
else
  $a_L \leftarrow f(X', 1, m'/2)$
  $a_R \leftarrow f(X', m'/2, m')$
  $score_L \leftarrow f_r(a_L, y)$
  $score_R \leftarrow f_r(a_R, y)$
  if $score_L \geq score_R$ **then**
    $\bar{A} \leftarrow \bar{A} \cup \{a_L\}$
    SupervisedSearch($X'[:, 1 : m'/2]$, y, f, f_r, $\bar{A}$)
  else
    $\bar{A} \leftarrow \bar{A} \cup \{a_R\}$
    SupervisedSearch($X'[:, m'/2 : m']$, y, f, f_r, $\bar{A}$)
  end if
end if
return $\bar{A}$

4.5 Tree Construction

To now construct trees a modified TSF is used. Instead of getting random samples the feature set is passed through the construction method. Then as in TSF entrance gain is maximized. This approach has a faster complexity, because the sample size is reduced to $O(\log m)$, so that the complexity drops to $O(nTree \log mN \log N)$, where N is the maximum depth of the tree and $nTree$ is the amount of trees created.

5 CONCLUSION

Time Series Forest (TSF) nad its successor, the Randomized-Supervised Time Series Forest(r-STSF), show both complementary advances in classifying time series. The goal is the same: Achieve interpretable, accurate classification in linear time complexity. They manage to use simple changes to existing algorithms to built decision trees without resorting to overly complex feature engineering. TSF introduces the core insight of using randomly sampled interval values with a custom splitting criterion, which outperforms previous algorithms built on entropy gain. This simply adjustment enables TSF to outperform strong baselines like NN-DTW on a wide range of benchmarks, while preserving the key advantage of interpretability via temporal importance curves.

The reliance on purely random feature generation, limits TSF in settings where the relevant temporal structure is sparse or subtle Precision is sacrificed for robustness Also while being conceptually elegant, the temporal importance curve is biased toward central indices due to interval frequency effects. TSF does not introduce a mechanism for normalizing or filtering out these uninformative peaks.

These issues are addressed in the construction of STSF, by replacing node-level feature randomness with tree-level interval supervision. A pre-search of the entire time series using a Fisher score-based strategy is used to identify candidate discriminatory intervals It also further enhances coverage with through multiple time series representation. These mechanisms increase both precision and efficiency resulting in STSF achieving better accuracy than TSF while being an order of magnitude faster on long series.

Despite these improvements, STSF comes with new costs: interpretability suffers and especially t

In sum, TSF and STSF show that it is possible to build time series classifiers that combine competitiveness with interpretability. TSF leverages randomness to ensure robustness and simplicity and r-STSF introduces supervision to increase focus and precision. They establish a foundation for interval-based methods that are both practical and transparent. Their success suggests that refining how and where information is extracted within the time series is preferable to increasing complexity.

6 LITERATURE

You should have 10 citations from peer reviewed papers and/or books.
Length estimate: 1/2 column