POLITECNICO DI TORINO
Repository ISTITUZIONALE

Multiclass Sparse Centroids With Application to Fast Time Series Classification

*Original*
Multiclass Sparse Centroids With Application to Fast Time Series Classification / Bradde, T.; Fracastoro, G.; Calafiore, G. C.. - In: IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS. - ISSN 2162-237X. - STAMPA. - (2021). [10.1109/TNNLS.2021.3124300]

*Availability:*
This version is available at: 11583/2957249 since: 2022-03-09T11:49:24Z

*Publisher:*
Institute of Electrical and Electronics Engineers Inc.

*Published*
DOI:10.1109/TNNLS.2021.3124300

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*
IEEE postprint/Author's Accepted Manuscript

©2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

12 October 2024

# Multi-class Sparse Centroids with Application to Fast Time-Series Classification

Tommaso Bradde, *Student Member, IEEE*, Giulia Fracastoro, *Member, IEEE*, Giuseppe C. Calafiore, *Fellow, IEEE*

***Abstract*—In this paper, we propose an efficient multi-class classification scheme based on sparse centroids classifiers. The proposed strategy exhibits linear complexity with respect to both the number of classes and the cardinality of the feature space. The classifier we introduce is based on binary space partitioning, performed by a decision tree where the assignation law at each node is defined via a sparse centroid classifier. We apply the presented strategy to the time series classification problem, showing by experimental evidence that it achieves performance comparable to that of state-of-the-art methods, but with a significantly lower classification time. The proposed technique can be an effective option in resource-constrained environments where the classification time and the computational cost are critical, or in scenarios where real-time classification is necessary.**

## I. Introduction

In the last years, the increasing pervasiveness of sensing devices and monitoring services allowed machine learning technologies to be successfully exploited in a growing number of different fields. Boosted by the efforts of both industry and academia, this trend is likely to be maintained in the near future. From a technological standpoint, the scarcity of computational resources represents today a serious limitation for the applicability of such techniques in areas where their potential could be exploited.

In this perspective, this work presents a computationally efficient classifier which may represent a convenient choice whenever the classification time or the computational cost are critical for the application. The proposed classifier is a feature-based method that relies on a binary tree classifier, where each node discerns between two sets of classes to which the upcoming sample might belong. Based on the recent developments of [1], the binary classification rule performed at each node is driven by a sparse nearest-centroid classifier, which relies on an optimal subset of available features, specifically selected during the training phase.

Since the feature selection is embedded in the tree construction, the upcoming samples can be classified by means of only low-dimensional euclidean distances, whose computation requires a negligible amount of time. The classification complexity of the proposed classifier is linear in both the number of classes of the problem and in the number of features retained after the selection stage. Besides the desirable reduction of the classification complexity, the sparsification process allows neglecting the features that retain no discriminating properties and that could lead to misclassifications, thus improving the accuracy of the classifier.

As a reference field of application for the proposed approach, we shall here refer to the problem of time series classification, which is a staple of the research in data mining and machine learning. The classification of time series performed in real-time or in resource-constrained environments is highly limited by the scalability of the employed technology since, in these cases, guaranteeing a high classification accuracy and computational efficiency, even with very high dimensional samples, is of primary importance. Applications where these constraints are critical include power grid data [2], predictive maintenance [3], and human activity recognition through portable devices [4].

In the available literature, virtually all of the most important general purpose classifiers, such as Support Vector Machines, K-Nearest Neighbors, Gaussian Mixture Models, and Random Forests, have been applied to time series classification [5]. Moreover, deep learning methods have also been tested, [6], a multitude of possible ad-hoc solutions have been proposed to tackle this problem, and the topic has been covered extensively in a number of reviews, see, e.g., [7], [8]. These tailored methods perform the classification on the basis of different criteria, ranging from the exploitation of a suitably defined distance metric computed over the raw time series, [9]–[12], or over a suitably defined approximation of the raw data [13]–[15]; others have been specifically designed to operate on discriminatory sub-sequences of the whole time series, referred to as shapelets [16].

Such a large number of proposed solutions suggests that it is not possible to identify a unique method that can be a valid choice for all the various application scenarios of time series classification. Contrary, the choice of the right technique must be evaluated on the basis of the specific application and the related technological constraints, thus making this problem a meaningful test bench for the strategy we are proposing.

The performance of the proposed classifier has been experimentally evaluated both on well-known benchmark data sets and on the specific problem of human activity recognition based on smartphone-acquired data. The obtained results show that the proposed classifier is on par with state-of-the-art time series classification methods from the point of view of classification performance, while it requires only a fraction of their computational cost.

The paper is organized as follows. In Section II we set the notation and we recall some background notions. In Section III we introduce and discuss the proposed classifier, which is

T. Bradde, G. Fracastoro and G.C. Calafiore, are with the Dept. of Electronics and Telecommunications, Politecnico di Torino, Torino 10129, Italy (e-mail: tommaso.bradde@polito.it, giulia.fracastoro@polito.it, giuseppe.calafiore@polito.it)

experimentally tested in Section IV. Conclusions are finally drawn in Section V.

## II. Background and Notation

The goal of a classification task is to define a map that links observed feature data to a class membership label. Let $\boldsymbol{x}_i \in \mathbb{R}^K$ be a vector of features representative of an observation; we define one element of our dataset $\mathcal{D}$ as the couple

$$d_i = (\boldsymbol{x}_i, c_i), \tag{1}$$

where $c_i \in \{1, 2, .., C\}$ is the class label of the $i$-th observation. Let $\mathcal{D}_T$ and $\mathcal{D}_V$ be the training and the validation set, respectively, where $\mathcal{D}_T \cup \mathcal{D}_V = \mathcal{D}$ and $\mathcal{D}_T \cap \mathcal{D}_V = \emptyset$. The core of the proposed classification algorithm is based on the nearest-centroid classifier. For a binary classification problem, i.e., $C = 2$, two centroids can be computed on the basis of the training data $\mathcal{D}_T$ as

$$\bar{\boldsymbol{x}}_1 = \frac{1}{n_1} \sum_{i \in \mathcal{I}_1} \boldsymbol{x}_i, \qquad \bar{\boldsymbol{x}}_2 = \frac{1}{n_2} \sum_{i \in \mathcal{I}_2} \boldsymbol{x}_i \tag{2}$$

where $\mathcal{I}_1 \doteq \{i : c_i = 1\}$, $\mathcal{I}_2 \doteq \{i : c_i = 2\}$, and $n_1$ and $n_2$ are the cardinalities of the sets $\mathcal{I}_1$ an $\mathcal{I}_2$, respectively.

The centroids can be used to define a classification rule. The class predicted for an upcoming sample $\boldsymbol{x}$ is the output of the function $g : \mathbb{R}^K \to \{1, 2\}$ defined as

$$g(\boldsymbol{x}) = \begin{cases} 1 & \text{if } ||\boldsymbol{x} - \bar{\boldsymbol{x}}_2||_2^2 - ||\boldsymbol{x} - \bar{\boldsymbol{x}}_1||_2^2 \geq 0 \\ 2 & \text{if } ||\boldsymbol{x} - \bar{\boldsymbol{x}}_2||_2^2 - ||\boldsymbol{x} - \bar{\boldsymbol{x}}_1||_2^2 < 0 \end{cases} . \tag{3}$$

When the number $K$ of available features is very large, a feature selection stage is commonly employed for reducing the complexity of the problem and for retaining only the information that is useful to effectively perform the classification, see e.g. [17]–[20]. Recent results in [1] proved that, in the case of binary classification, it is possible to define a sparse nearest-centroid classifier, which performs efficient *simultaneous* feature selection and classification by detecting the optimal subset of features for the classification task.

For a given sparsity level $m \leq K$, we define the sparse centroids as follows:

1) Compute the standard class centroids $\bar{\boldsymbol{x}}_1$ and $\bar{\boldsymbol{x}}_2$;
2) Compute the difference vector $\delta = \bar{\boldsymbol{x}}_2 - \bar{\boldsymbol{x}}_1$ and the centroids midpoint $\tilde{\boldsymbol{x}}$;
3) Collect in the set $\mathcal{R}$ the indices associated with the $m$ largest elements of $|\delta|$. Denote the complementary set $\mathcal{E} = \{1, ..., K\} \setminus \mathcal{R}$;
4) The sparse centroids are given by $\bar{\boldsymbol{\theta}}_1^m = \boldsymbol{x}_1^{\mathcal{R}} + \tilde{\boldsymbol{x}}^{\mathcal{E}}$ and $\bar{\boldsymbol{\theta}}_2^m = \boldsymbol{x}_2^{\mathcal{R}} + \tilde{\boldsymbol{x}}^{\mathcal{E}}$, where we use the notation $\boldsymbol{x}^{\mathcal{R}}$ ($\boldsymbol{x}^{\mathcal{E}}$) to denote a vector of the same dimension as $\boldsymbol{x}$ which coincides with $\boldsymbol{x}$ at the locations in $\mathcal{R}$ ($\mathcal{E}$) and is zero elsewhere.

Then, the optimal nearest-centroid classification rule is given by the function $f_m : \mathbb{R}^K \to \{1, 2\}$ defined as

$$f_m(\boldsymbol{x}) = \begin{cases} 1 & \text{for } ||\boldsymbol{x} - \bar{\boldsymbol{\theta}}_2^m||_2^2 - ||\boldsymbol{x} - \bar{\boldsymbol{\theta}}_1^m||_2^2 \geq 0 \\ 2 & \text{for } ||\boldsymbol{x} - \bar{\boldsymbol{\theta}}_2^m||_2^2 - ||\boldsymbol{x} - \bar{\boldsymbol{\theta}}_1^m||_2^2 < 0. \end{cases} \tag{4}$$

We highlight that, in the above equations, the centroids elements associated to the set of indices $\mathcal{E}$ play no role for the sake of classification; thus, the computational effort required to compute (4) is reduced with respect to the one associated to (3). This also enables the possibility to avoid the computation of the corresponding features during the classification task, thus reducing its time requirements.

The most appropriate level of sparsity for $\bar{\boldsymbol{\theta}}_1^m$ and $\bar{\boldsymbol{\theta}}_2^m$ (which we will denote in the following with $k$) can be found by cross-validation. We highlight that the computational cost required for training the above sparse centroid classifier is $O(Kn) + O(K \log m)$, being $n = n_1 + n_2$; we refer the reader to [1] for a deeper analysis of the theoretical aspects of this method.

## III. Sparse centroid tree

In this section, we present the proposed method for extending the binary sparse centroid classifier to the multi-class case. As the assignation rule described by (4) is intrinsically binary, we propose a multi-class classification strategy based of sequential binary partitions of the set of classes. This strategy can be easily implemented by building a binary tree where each node applies the sparse centroid classification rule (4) in order to discern between two sets of potential classes. By following the classification path from the root node to one of the leaves, the appropriate sequence of binary partitions will output the class label. The proposed binary tree is thus able to handle simultaneously both the stages of feature selection and classification.

The binary space partitions of the tree classifier are defined in an iterative way. We explain next the procedure for performing a binary partition over a generic classes set $\mathcal{C}_R \subseteq \mathcal{C}$. Building the proposed decision tree classifier will require to apply such operation in an iterative way.

Let $\mathcal{P}(\mathcal{C}_R)$ be the set of all the possible binary partitions of the set $\mathcal{C}_R$. We denote one of its elements as $P = \{P_1, P_2\}$, where $P_1 \cup P_2 = \mathcal{C}_R$ and $P_1 \cap P_2 = \emptyset$. The two sets of classes $P_1$ and $P_2$ represent the current candidate binary partitions of $\mathcal{C}_R$. We can define a binary representation of the data based on this binary partition. let $X$ be a place-holder for either $T$ or $V$; then for each data sample $d = (\boldsymbol{x}, c) \in \mathcal{D}_X$ we build the binary dataset $\mathcal{D}_{X|P}$ associated with the partition $P$ according to the following rule:

$$\begin{aligned} (\boldsymbol{x}, 1) \in \mathcal{D}_{X|P} \quad &\text{if} \quad c \in P_1; \\ (\boldsymbol{x}, 2) \in \mathcal{D}_{X|P} \quad &\text{if} \quad c \in P_2; \end{aligned} \tag{5}$$

if none of the above condition is satisfied, then the observation $\boldsymbol{x}$ is not included in $\mathcal{D}_{X|P}$. After having defined the binary representations for the training and validation datasets, we can apply the sparse nearest-centroid classifier for evaluating the quality of the candidate classes partition. Given a maximum allowed degree of cardinality $\bar{m} < K$, the accuracy of the assignation rule in (4) is evaluated on the validation samples for different levels of cardinality $m = 1, ..., \bar{m}$. The performance index of the split $P = \{P_1, P_2\}$ is chosen to be the highest reachable level of accuracy, attained for a given sparsity degree $k \leq \bar{m}$. By repeating the process for all the elements of $\mathcal{P}$, we choose the partition of the class space $\mathcal{C}_R$ that shows the highest performance index. The chosen partition

is denoted as $P^* = \{P_1^*, P_2^*\}$. The steps required to partition the class space are reported in Algorithm 1. A standard cross-validation scheme can be applied within Algorithm 1 in order to guarantee a meaningful evaluation of the quality of each possible split of the classes set.

**Algorithm 1** Nearest-centroid binary space partitioning

**Input:** Training dataset $\mathcal{D}_T$, validation dataset $\mathcal{D}_V$, set of classes to be partitioned $\mathcal{C}_R$, set of all the classes $\mathcal{C}$, maximum feature vector cardinality $\bar{m}$.

**Output:** Best class space partitioning, $P^* = \{P_1^*, P_2^*\}$, cardinality level $k$.

**for** each element $P = \{P_1, P_2\} \in \mathcal{P}(\mathcal{C}_R)$ **do**
  Build the set $\mathcal{D}_{T|P}$ by applying the class partition (5) to $\mathcal{D}_T$
  Build the set $\mathcal{D}_{V|P}$ by applying the class partition (5) to $\mathcal{D}_V$
  compute the centroids $\bar{\boldsymbol{x}}_1$, $\bar{\boldsymbol{x}}_2$ on $\mathcal{D}_{T|P}$ as in (3)
  **for** $m = 1, 2, ..., \bar{m}$ **do**
    Apply (4) with cardinality degree $m$ to the samples that belong to $\mathcal{D}_{V|P}$. Define $a_m$ as the number of properly classified samples.
    Define the performance index $a(P, m) = \frac{a_m}{|\mathcal{D}_{V|P}|}$
  **end for**
  $k^P = \text{argmax}_m \; a(P, m)$
**end for**
$P^* = \text{argmax}_{P \in \mathcal{P}} \; a(P, k^P)$.
$k^* = k^P$.

**Return:** $P^*$ and $k^*$.

**Algorithm 2** Training of Nearest-centroid binary tree

**Input:** Set of classes $\mathcal{C}$, Dataset partition $\mathcal{D}_T$, $\mathcal{D}_V$, maximum cardinality degree $\bar{m}$.

**Output:** Decision tree classifier.

Set $\mathcal{C}_1 \leftarrow \mathcal{C}$ as input class for the root node $N_1$.
Initialize the set of tree leaves $\mathcal{L} = \{N_1\}$.
**while** $\dim(\mathcal{L}) < \dim(\mathcal{C})$ **do**
  **for** each $N_l \in \mathcal{L}$ **do**
    **if** $\dim(\mathcal{C}_l) > 1$ **then**
      Set $\mathcal{C}_R \leftarrow \mathcal{C}_l$. Apply Algorithm 1
      Assign the decision rule $f_k$ to $N_l$.
      Define the two nodes $N_{l+1}$, $N_{l+2}$ as children of $N_l$.
      Set $\mathcal{C}_{l+1} \leftarrow P_1^*$ and $\mathcal{C}_{l+2} \leftarrow P_2^*$.
      Remove $N_l$ from $\mathcal{L}$.
      Add $N_{l+1}$ and $N_{l+2}$ to $\mathcal{L}$.
    **else**
      Do nothing
    **end if**
  **end for**
**end while**

**Return:** Decision tree classifier

After having defined the above partitioning strategy, the construction of the decision tree classifier is straightforward. The root node performs the first binary partition of the set $\mathcal{C}$. By applying Algorithm 1, we compute the optimal set partition $P^* = \{P_1^*, P_2^*\}$, which will be assigned as input classes for the two children nodes. The same procedure is applied iteratively on each of the children nodes until the number of tree leaves equals the number of classes in $\mathcal{C}$. Algorithm 2 summarizes the procedure for the generation of the decision tree classifier, where we denote with $\mathcal{C}_l$ the set of input classes of the $l$-th node of the tree.

Once the classifier tree is defined, the classification of an incoming sample can be performed by following, from the root to one of the leaves, the path imposed by the assignation rules associated to each node. By following this path, we compute only the features required by the nodes of such path. We remark that the overall number of features required to classify a given sample is the sum of all the features required by the assignation rules that are computed along the classification path, from the root node to one of the leaves, which is different for each output class. As shown by the experiments of section IV, the decision tree structure implicitly carries out a hierarchical partitioning which also gives information about the linear separability of the classes.

We conclude this section with an analysis of the computational complexity of the proposed algorithm. Since the complexity of both the training and the classification stages relies on the chosen class space partitioning and the optimal sparsity degree associated to the sparse centroid classifiers at each node, we will state the computational complexity considering the worst-case scenario.

**Proposition 1** (Computational Complexity)**.** *The worst-case computational complexity required by the Nearest-centroid binary tree for classifying an upcoming datum is $O(C\bar{m})$. The complexity of its training stage is $O(2^C Kn) + O(2^C K \log \bar{m})$.*

To prove the above proposition for the testing stage, consider that a binary tree with $C$ leaves contains $C - 1$ internal nodes. Since the classification complexity of the single sparse centroid classifier is $O(m)$, the worst case classification complexity for the binary tree is $O(C\bar{m})$, where $\bar{m}$ is the maximum cardinality degree set by the user. This means that the classification time is linear with respect to the number of classes and the feature cardinality.

To assess the computational complexity of the training stage, we recall that the number of possible binary partitions of a set with $C$ elements is $2^{C-1} - 1$. Thus, the training complexity is $O(2^C Kn) + O(2^C K \log \bar{m})$.

## IV. Experimental Results

In this section, we provide empirical evidence of the effectiveness of the proposed method. The experimental validation is performed in two stages. We first compare the performance of our algorithm with state-of-the-art time series classifiers. This first stage is performed using various benchmark time series datasets. Then, in the second part of the experimental validation, we show an application of the proposed method in a resource-constrained scenario. In particular, we focus on the specific problem of Human Activity Recognition, showing that the proposed method can be successfully employed for applications with strong resource constraints.

The numerical experiments are carried out in Matlab (whenever not differently specified), making use of the same machine, namely a laptop equipped with 2GHz CPU and 8 GB of RAM. The code and all the datasets are available at https://github.com/tomBradde/Fast-Time-Series-Classification-via-Sparse-Centroids.

### A. Performance over Benchmark Datasets

The purpose of this experimental section is to show how the proposed method can be effectively applied to generic time series classification problems in order to guarantee major speed up in terms of computation time. This speed up comes together with accuracy scores that are compatible with those of state-of-the-art time series classifiers. The proposed sparse centroid tree classifier is evaluated on a set of benchmark time series classification datasets. All the datasets are extracted from the *UCI Machine Learning Repository* [21], which is considered as a standard benchmark database for time series classification problems, see, e.g., [6], [7]. We refer the reader to [21] for further information about the processes used to generate the time series and the references to their original sources. In Table I we report some information about the datasets considered in the experimental evaluation, namely the dimension of test/training set, the number of classes, and the length of the time series observations involved in the problem. For all these datasets, we used the same train/test split proposed in the UCI repository.

As the datasets are provided in the form of raw time series, we map the observation into an initial set of features; this strategy is commonly employed in time series classification, see, e.g., [22], [23]. The features we exploit have been selected according to the literature and include statistical properties [24], spectral representations [25], self-similarities measures [26], and other characteristics related to the time series.

For each dataset, we trained 10 different sparse centroid classifiers, applying a standard cross-validation scheme within Algorithm 1; each classifier has been trained by making use of a different random sample partitioning for performing the cross-validation. The resulting classifiers have been evaluated on the test samples and from the resulting performances we derived the mean value of the accuracy of the classifiers and the associated variance.

In order to validate the effectiveness of the proposed feature selection strategy, we included in our comparison also a decision tree classifier built according to Algorithm 2, making use of the the assignation rule (3) instead of (4). This decision tree classifier makes use of all the available features, without performing any feature selection.

As for terms of comparison, we selected some of the state-of-the-art algorithms analyzed in [7], namely Derivative DTW [11], 1-Nearest Neighbor with SAX [13], Proximity Forest [27], BOSS [28], cBOSS [29], VSM [15], Fast Shapelets [16], and Time Series Forest [30]. In addition, we also considered some deep learning methods, namely Fully Convolutional Neural Networks (FCN), Residual Networks (ResNet) [31], and Time Convolutional Neural Networks (Time-CNN) [32]; to test these classifiers, we used the public code provided in the repository [33]. The numerical experiments for the deep learning methods are carried out in Tensorflow 2.0 making use of a machine equipped with a Nvidia TITAN RTX GPU.

Table II reports the test accuracy results and the cumulative time required by each classifier to perform the training and testing stages. Some values in the table are missing because the classifier training and testing procedures required more than 10 minutes to be completed. For each classifier, we report the average accuracy reached over the completed experiments (i.e. those that required less than 10 minutes to be completed).

The results show that the proposed classifier offers training and classification times that are order of magnitudes better (lower) than the one required by state-of-the-art methods, all the while achieving competitive accuracy levels. Additionally, we notice that the performance of the centroid tree classifier built without performing feature selection is worse than that of the proposed approach, both in terms of accuracy and of time requirements; this confirms that the proposed sparsification strategy may improve *both* time efficiency and accuracy.

### B. Application to Human Activity Recognition

In this section, we experimentally show the applicability of the sparse centroid tree to the specific problem of Human Activity Recognition (HAR) based on sampled sensors data. Since this task is usually required to be performed by portable instrumentation with low computational capacity, the potential solutions must take into account the strong technological constraints imposed by the hardware [4], [34].

Furthermore, due to the increasing number of sensors embedded in devices of common use, the problem is more properly tackled by taking into account multivariate time series data. When multivariate time series are considered, the ensuing amount of data readily becomes difficult to handle, and often dimensionality reduction is desirable. These characteristics motivated us to test the proposed method in such application scenario. We used the public dataset provided by Anguita *et al.* in [35]. The dataset considers six different activities, namely *walking*, *walking upstairs walking downstairs*, *sitting*, *standing*, and *laying*. Each sample is represented by a feature vector of dimension 561. The observations were split in a training set of dimension 9309 and a test set of dimension 1030.

The sparse centroid tree is trained in $32$ s and the mean accuracy over the test samples is $87\%$. The per-class classification performance is shown in Table III, where we indicate also the number of required features. We highlight that the classifier achieves $100\%$ accuracy for the *laying* activity, requiring only $1$ feature. The overall number of features that are required by the sparse centroid tree is $45$, while the mean classification time over all the test samples is only $38$ $\mu$s.

We also provide a performance comparison between the proposed method and the most commonly employed classifiers for human activity recognition, namely Naive Bayes [36], K-nearest neighbour and Random Kitchen Sinks SVM [37]. Since the proposed sparse centroid tree performs simultaneous

TABLE I
BENCHMARK DATASETS CONSIDERED IN THE EXPERIMENTAL EVALUATION.

| | Trace | Synthetic control | UMD | TwoPatterns | CBF | Meat | Car | ProximalPhalanx AgeGroup | Plane | BME |
|---|---|---|---|---|---|---|---|---|---|---|
| Training Samples | 100 | 300 | 36 | 1000 | 900 | 60 | 60 | 400 | 105 | 30 |
| Test Samples | 100 | 300 | 144 | 4000 | 30 | 60 | 60 | 205 | 105 | 150 |
| Length | 275 | 60 | 150 | 128 | 128 | 448 | 577 | 80 | 144 | 128 |
| Classes | 4 | 6 | 3 | 4 | 3 | 3 | 4 | 3 | 7 | 3 |

TABLE II
COMPARISON OF CLASSIFIERS OVER UCI DATASETS.

| | | Trace | Synthetic Control | UMD | Two Patterns | CBF | Meat | Car | Proximal Phalanx AgeGroup | Plane | BME | Average Accuracy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sparse Centroid tree | Time | 1.1 s | 2.8 s | 0.68 s | 7 s | 5.6 s | 0.7 s | 1.3 s | 1.9 s | 1.2 s | 0.2 s | 91.4% |
| | Accuracy | 98 % | 98% | 87% | 90% | 100% | 94% | 70% | 83% | 97% | 97% | |
| | Variance | 3e-5 | 0 | 1e-3 | 1e-5 | 0 | 1.4e-4 | 6e-4 | 5e-4 | 1e-4 | 0 | |
| Centroid Tree | Time | 2.3 s | 5.6 s | 2 s | 36 s | 8.4 s | 1.6 s | 1.8 s | 5.8 s | 2.7 s | 2 s | 79.9% |
| | Accuracy | 78% | 97% | 70% | 84 % | 94 % | 83% | 53% | 80 % | 90% | 70% | |
| Derivative DTW [11] | Time | 72 s | 31 s | 26 s | - | – | 58 s | 101 s | – | 19 s | 4 s | 93.6% |
| | Accuracy | 100% | 98% | 97% | - | – | 95% | 68% | – | 99% | 98% | |
| Sax 1NN | Time | 74 s | – | 157 s | - | – | 27 s | 33 s | 342 s | 63s | 5 s | 89.9% |
| | Accuracy | 99% | – | 97% | - | – | 95% | 72% | 80% | 99% | 87% | |
| Proximity Forest [27] | Time | 105 s | 35 s | 57 s | - | 281 s | 125 s | 314 s | 66 s | 37 | 11 s | 94.9% |
| | Accuracy | 99% | 99% | 100% | - | 100% | 100% | 73% | 84% | 99% | 100% | |
| BOSS [28] | Time | 40 s | 13 s | 20 s | - | 541 s | 42 s | 67 s | 31 s | 14 s | 2 s | 93.2% |
| | Accuracy | 100% | 96% | 100% | - | 97% | 100% | 80% | 80% | 99% | 87% | |
| cBOSS [29] | Time | 14 s | 13 s | 13 s | 320 s | 142 s | 12 s | 16 s | 27 s | 10 s | 3 s | 92.9% |
| | Accuracy | 100% | 95% | 100% | 99% | 100% | 100% | 76 % | 84 % | 99% | 76% | |
| SAX VSM [15] | Time | 72 s | – | 153 s | - | – | 27 s | 33 s | 343 s | 62 s | 5 s | 91.1% |
| | Accuracy | 99% | – | 97% | - | – | 95% | 71% | 80% | 99% | 87% | |
| Fast Shapelets [16] | Time | 45 s | 24 s | 27 s | 300 s | 111 s | 52 s | 120 s | 10 s | 14 s | 3 s | 87.6% |
| | Accuracy | 100% | 95% | 93% | 91% | 97% | 92% | 66% | 82% | 97% | 63% | |
| TSF [30] | Time | 3 s | 5 s | 3.7 s | 28 s | 11 s | 4 s | 4.2 s | 5 s | 2.9 s | 1.6 s | 94.6% |
| | Accuracy | 100% | 99% | 97% | 98% | 99% | 99% | 73 % | 85 % | 99% | 97% | |
| FCN [31] | Time | 117 s | 251 s | 248 s | - | 504 s | 161 s | 160 s | 219 s | 153 s | 236 s | 91.6% |
| | Accuracy | 100% | 98% | 100% | - | 99% | 72% | 89% | 83% | 100% | 83% | |
| ResNet [31] | Time | 184 s | 184 s | 362 s | - | 311 s | 238 s | 224 s | 173 s | 259 s | 356 s | 96.3% |
| | Accuracy | 100% | 99% | 98% | - | 100% | 98% | 90% | 82% | 100% | 100% | |
| Time-CNN [32] | Time | 79 s | 168 s | 68 s | - | 279 s | 72 s | 73 s | 144 s | 79 s | 73 s | 91.7% |
| | Accuracy | 94% | 98% | 95% | - | 100% | 91% | 77% | 80% | 95% | 95% | |

TABLE III
CONFUSION MATRIX OF THE SPARSE CENTROID TREE OVER THE HUMAN ACTIVITY RECOGNITION DATASET.

| | Walking. 22 feat. | Walking Upstairs 33 feat. | Walking Downstairs 33 feat. | Sitting 15 feat. | Standing 15 feat. | Laying 1 feat. |
|---|---|---|---|---|---|---|
| Walking | 165 | 3 | 4 | 0 | 0 | 0 |
| Walking Upstairs | 13 | 135 | 6 | 0 | 0 | 0 |
| Walking Downstairs | 7 | 9 | 125 | 0 | 0 | 0 |
| Sitting | 0 | 0 | 0 | 132 | 39 | 7 |
| Standing | 2 | 0 | 0 | 43 | 146 | 0 |
| Laying | 0 | 0 | 0 | 0 | 0 | 194 |

feature selection and classification, to ensure a fair comparison the other classifiers considered in the evaluation have been trained and tested after a feature selection stage, which selects a number of features comparable to the one required by the sparse centroid tree. We tested various feature selection methods, namely Generalized Fisher, [38], infinite feature selection [39] and infinite latent feature selection [40]. The performance of these classifiers combined with the different feature selection methods are compared with our proposed sparse centroid tree. Table IV reports the results associated with these experiments. In this table, the term "Automatic Feature Selection" indicates the feature selection performed by the sparse centroid tree. We observe that the classification time of the proposed method is approximately one order of magnitude lower than the one obtained with the nearest neighbor and SVM classifiers. Instead, we observe that the classification time of the naive Bayes classifier is about twice the one of the proposed method, and it shows less or equal accuracy when compared to the Sparse Centroid Tree.

## V. CONCLUSIONS

In this work, we proposed a simple and fast classification scheme based on sparse centroids. This method guarantees high accuracy and exhibits a classification complexity which is linear in both the number of classes and the number of features retained by the sparse classifier. The proposed approach has been applied to the problem of time series classification, and experimentally compared with most of the state-of-the art classifiers: experimental evidence testifies that the method may be a superior option in application scenarios that demand fast processing times and low computational power.

TABLE IV
CLASSIFIER COMPARISON WITH VARIOUS FEATURE SELECTION METHODS.

| | 1-Nearest Neighbour | Random Kitchen Sinks SVM | Naive Bayes | Sparse Centroid Tree |
|---|---|---|---|---|
| Sparse Centroid [1] | 93% | 96% | 87% | – |
| Generalized Fisher [38] | 93% | 95% | 85% | – |
| Infinite FS [39] | 89% | 95% | 83% | – |
| Infinite Latent FS [40] | 93% | 96% | 85% | – |
| Automatic Feature Selection | – | – | – | 87% |
| Mean Classification time on test samples | 290 $\mu$s | 400 $\mu$s | 83 $\mu$s | 38 $\mu$s |
| Training time | – | 240 s | 8 s | 32 s |

## REFERENCES


[1] G. C. Calafiore and G. Fracastoro, "Sparse $\ell_1$ and $\ell_2$ center classifiers," *Transactions in Neural Networks and Learning Systems*, pp. 1–14, 2020.

[2] P. Schäfer and U. Leser, "Fast and accurate time series classification with weasel," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 637–646.

[3] M. Christ, A. W. Kempa-Liehr, and M. Feindt, "Distributed and parallel time series feature extraction for industrial big data applications," *arXiv preprint arXiv:1610.07717*, 2016.

[4] J. Lee and J. Kim, "Energy-efficient real-time human activity recognition on smart mobile devices," *Mobile Information Systems*, vol. 2016, 2016.

[5] F. Attal, S. Mohammed, M. Dedabrishvili, F. Chamroukhi, L. Oukhellou, and Y. Amirat, "Physical human activity recognition using wearable sensors," *Sensors*, vol. 15, no. 12, pp. 31 314–31 338, 2015.

[6] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[7] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[8] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.

[9] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in *KDD workshop*, vol. 10, no. 16. Seattle, WA, USA:, 1994, pp. 359–370.

[10] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu, "Weighted dynamic time warping for time series classification," *Pattern recognition*, vol. 44, no. 9, pp. 2231–2240, 2011.

[11] T. Górecki and M. Łuczak, "Using derivatives in time series classification," *Data Mining and Knowledge Discovery*, vol. 26, no. 2, pp. 310–331, 2013.

[12] G. E. Batista, X. Wang, and E. J. Keogh, "A complexity-invariant distance measure for time series," in *Proceedings of the 2011 SIAM international conference on data mining*. SIAM, 2011, pp. 699–710.

[13] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: a novel symbolic representation of time series," *Data Mining and knowledge discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[14] J. Lin, R. Khade, and Y. Li, "Rotation-invariant similarity in time series using bag-of-patterns representation," *Journal of Intelligent Information Systems*, vol. 39, no. 2, pp. 287–315, 2012.

[15] P. Senin and S. Malinchik, "Sax-vsm: Interpretable time series classification using sax and vector space model," in *2013 IEEE 13th international conference on data mining*. IEEE, 2013, pp. 1175–1180.

[16] T. Rakthanmanon and E. Keogh, "Fast shapelets: A scalable algorithm for discovering time series shapelets," in *proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 2013, pp. 668–676.

[17] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *The Journal of Machine Learning Research*, vol. 5, pp. 1205–1224, 2004.

[18] M. Tan, I. W. Tsang, and L. Wang, "Towards ultrahigh dimensional feature selection for big data," *Journal of Machine Learning Research*, 2014.

[19] X. Li, H. Zhang, R. Zhang, Y. Liu, and F. Nie, "Generalized uncorrelated regression with adaptive graph for unsupervised feature selection," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 5, pp. 1587–1595, 2018.

[20] Q. Wang, J. Wan, F. Nie, B. Liu, C. Yan, and X. Li, "Hierarchical feature selection for random projection," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 5, pp. 1581–1586, 2018.

[21] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[22] T. Altay and M. G. Baydoğan, "A new feature-based time series classification method by using scale-space extrema," *Engineering Science and Technology, an International Journal*, 2021.

[23] B. D. Fulcher and N. S. Jones, "Highly comparative feature-based time-series classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3026–3037, 2014.

[24] A. Nanopoulos, R. Alcock, and Y. Manolopoulos, "Feature-based classification of time-series data," *International Journal of Computer Research*, vol. 10, no. 3, pp. 49–61, 2001.

[25] F. Mörchen, "Time series feature extraction for data mining using dwt and dft," 2003.

[26] X. Wang, K. Smith, and R. Hyndman, "Characteristic-based clustering for time series data," *Data mining and knowledge Discovery*, vol. 13, no. 3, pp. 335–364, 2006.

[27] B. Lucas, A. Shifaz, C. Pelletier, L. O'Neill, N. Zaidi, B. Goethals, F. Petitjean, and G. I. Webb, "Proximity forest: an effective and scalable distance-based classifier for time series," *Data Mining and Knowledge Discovery*, vol. 33, no. 3, pp. 607–635, 2019.

[28] P. Schäfer, "The boss is concerned with time series classification in the presence of noise," *Data Mining and Knowledge Discovery*, vol. 29, no. 6, pp. 1505–1530, 2015.

[29] M. Middlehurst, W. Vickers, and A. Bagnall, "Scalable dictionary classifiers for time series classification," in *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 2019, pp. 11–19.

[30] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Information Sciences*, vol. 239, pp. 142–153, 2013.

[31] 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)*. IEEE, 2017, pp. 1578–1585.

[32] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *Journal of Systems Engineering and Electronics*, vol. 28, no. 1, pp. 162–169, 2017.

[33] "Deep learning for time series classification repository," https://github.com/hfawaz/dl-4-tsc, accessed: 2021-09-24.

[34] M. A. Labrador and O. D. L. Yejas, *Human activity recognition: using wearable sensors and smartphones*. CRC Press, 2013.

[35] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones." in *Esann*, vol. 3, 2013, p. 3.

[36] K. Altun, B. Barshan, and O. Tunçel, "Comparative study on classifying human activities with miniature inertial and magnetic sensors," *Pattern Recognition*, vol. 43, no. 10, pp. 3605–3620, 2010.

[37] A. Rahimi, B. Recht *et al.*, "Random features for large-scale kernel machines." in *NIPS*, vol. 3, no. 4. Citeseer, 2007, p. 5.

[38] Q. Gu, Z. Li, and J. Han, "Generalized fisher score for feature selection," *arXiv preprint arXiv:1202.3725*, 2012.

[39] G. Roffo, S. Melzi, and M. Cristani, "Infinite feature selection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4202–4210.

[40] G. Roffo, S. Melzi, U. Castellani, and A. Vinciarelli, "Infinite latent feature selection: A probabilistic latent graph-based ranking approach," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1398–1406.