Neuro-Ensemble for Time Series Data Classification

Conference Paper · October 2018

DOI: 10.1109/DSAA.2018.00015

3 authors:

Soukaina Filali Boubrahimi
Utah State University
34 PUBLICATIONS 149 CITATIONS
SEE PROFILE

Ruizhe Ma
University of Massachusetts Lowell
28 PUBLICATIONS 135 CITATIONS
SEE PROFILE

Rafal A. Angryk
Georgia State University
217 PUBLICATIONS 1,501 CITATIONS
SEE PROFILE

Neuro-Ensemble for Time Series Data Classification

Soukaïna Filali Boubrahimi , Ruizhe Ma , Rafal Angryk
Department of Computer Science, Georgia State University, Atlanta, GA 30302
Email: {sfilaliboubrahimi1, rma1, rangryk}@cs.gsu.edu

***Abstract*—Combining a set of classification algorithms is a powerful technique in improving the accuracy of individual classifiers. There are two main paradigms in combining classifiers: classifier selection, where each classifier is considered as an expert in some local area of the feature space, and classifier fusion, where all classifiers are trained over the entire feature space and they are considered as competitive and complementary to each other. In this paper, we propose a new ensemble technique, *Neuro-Ensemble*, that follows the classifier fusion paradigm applied on time series data. The Neuro-Ensemble exploits the idea that different classifiers participating in the ensemble have varying degrees of expertise on learning different class labels and it optimizes the ensemble using a shallow Multi-Layer Perceptron (MLP) based meta-learner to capture the expertise of individual classifiers. Every neuron in the MLP represents a classifier that contributes with a vote and performs activation and state computations. This work is the first attempt to train a neural network for learning the expertise of each classifier in an ensemble and optimize the entire classification schema based on class-level expertise weights. We validated our Neuro-Ensemble on 43 real-world time series datasets from the UCR repository. Our experimental results show the effectiveness and efficiency of our approach in comparison with individual baseline learners and ensemble techniques.**

***Index Terms*—Heterogeneous Classifier Fusion, Optimization, Multi-Layer Perceptron, Meta-learning, Time Series Classification**

I. Introduction

Combining the outputs from different supervised learning models, referred to as ensembling, is a common practice in several data mining communities. The main motivation behind ensemble learning models is to improve the performance in terms of classification accuracy. According to the No Free Lunch (NFL) theorem, it is generally accepted that there is no one model that dominates over all types of learning problems [1]. In addition, different classifiers may be useful in representing different aspects of the problem. In other terms, while some classifiers might be powerful in classifying a given class c, other classifiers might be less accurate in classifying c but more useful when classifying other classes. The aim of the ensemble is to leverage the strength of the base learners to optimize the final classifier accuracy. Another motivation for using ensemble methods is for scaling algorithms when applied to very large databases without compromising accuracy. One of the solutions to this problem is to horizontally partition the database into manageable sections and assign every data partition to a learner. The final ensemble represents the combination of all the base learners with a given fusion strategy.

Ensembles can be inspected under two categories based on the types of base learners used: (1) *Homogeneous* ensembles, and (2) *Heterogeneous* ensembles. When a combination of base models of the same learning algorithm is used, the ensemble is called *Homogeneous*. The difference between base learners participating in a homogeneous ensemble stems from the instances in which they are trained as well as the induced randomness in the learning algorithm. On the other hand, when the ensemble is formed from running different learning algorithms, the ensemble is called *Heterogeneous*. In this paper, we propose a new heterogeneous ensemble based on a variety of base learners.

When creating ensembles using classifier fusion paradigm, classifiers are trained using the entire feature space and their individual success in correctly classifying the instances are reflected as weights in the ensembles. The fusion step usually involves tracking the error of the models in the training and validation phase and using them to weight the classifiers predictions. Simple fusion strategies include weighted voting, weighted voting per class and majority voting. Stacking [2] is another fusion methodology that involves training a meta-learner on the class predictions or class probabilities of the base classifiers. The stacking method learns the inter-dependencies that exist between base classifiers and inherently takes into consideration their diversity [3]. Our work stems from the idea that different classifiers have different expertise on the input space [3]; therefore, weight optimization should be performed on a class level basis prior to classifier fusion.

Problem Formalism: Assume that a feature space represented by a mutually exclusive set of classes $S = \{C_1, C_2, \ldots, C_M\}$, for all $i \in \{1, \ldots, M\}$. A classifier, also called **expert**, predicts the class of a data sample $x \in S$ as $e(x) = j$ such that $j \in \{1, \ldots, M\}$. An ensemble E, is the **fusion** of K different base experts e_k, $k \in \{1, \ldots, K\}$, each of which assigns class probabilities to a data sample as $e_k(x)$.

In this paper, we propose a new stacking strategy that uses batch gradient descent optimization algorithm on a shallow neural network to learn the optimal weights of the classifiers. We embed additional classification logic within every neuron in the neural network that represent a participating classifier in the ensemble. Consequently, the neural network is used solely to learn the weights of the base classifiers on a class level basis. In other words, the aim of the network is to learn which classifiers can more accurately predict a particular class label and therefore assign higher weights to them when compared to their counterparts. Neuro-Ensemble is generic in nature and can be used with a number of, heterogeneous or homogeneous, classifiers on different types of data models. In this work we only focused on time series data model [4] [5] [6] that we mine using kNN classifier coupled with 11 different elastic measures. We evaluated our method on a set of 43 real-world datasets from the University of California at Riverside (UCR) Time Series Classification Archive [7]. We found that the Neuro-Ensemble is competitive with the state-of-the-art ensemble method and other baseline methods.

The rest of the paper is organized as follows: in Section II we review the related works; the proposed Neuro-Ensemble method is introduced in Section III. Section IV defines the base classifiers used in the ensemble and the other baseline ensemble method used to compare our meta-learner, followed by the experimental design in Section V and discussion of the results of our experiments presented in Section VI. Finally, Section VII concludes the paper and shows potential directions for future works.

II. Related Works

In this section, we outline some state-of-the-art ensemble methodologies that are commonly adopted from the classifier selection and classifier fusion paradigms.

A. Classifier Selection

Classifier selection strategy evaluates each of the classification algorithms on the cross-validation set and selects the best performing ones for the final evaluation on the test dataset. A simple but effective method in the classifier selection paradigm is the Evaluation and Selection (ES) method. ES evaluates all the classifiers based on the validation set and selects the most accurate one to be used in the test set. Džeroski indicated that ES should be considered as a baseline when evaluating a new ensemble methodology [8]. The downside of ES is that sometimes it fails to select the best classifier given that the most accurate classifier on the validation set is not necessarily the best classifier on the test set.

Another method for selecting base classifiers consists of distinguishing learning algorithms that have shown to be accurate in the same domain of the problem and selecting them. The former strategy relies on building performance records of simple and efficient learning algorithms on different set of domains and using the learners that perform the best in the domain of the problem. A number of approaches have been investigated to build the domain-specific performance record, some of which use landmarking [9], model-based data characterization [10], histograms [11], statistical and information-theoric measures [12]. Dynamic Classifier Selection is another selection method that uses different classifier for different instance in the test space. Classifier selection is based on the classifiers' performance on nearest training instance neighbors [13].

B. Classifier Fusion

Classifier fusion strategies allow all the classifiers to participate in the final prediction (even the less accurate ones). Voting is one of the widely used strategies when combining classifiers that are either homogeneous, such as Random Forest [14], or heterogeneous. In voting, every classifier predicts the class of the test instance (or the class probabilities), and the class that received the most vote from all the classifiers constitute the ensemble prediction. The former type of voting is called majority voting. Weighted voting is another classifier fusion methodology that discriminates between the models based on their performance. Weighted voting assigns a weight to each classifier that is proportional to their respective skill scores on the validation set (e.g., accuracy rate in validation). In the context of time series data ensemble, Denger and others extracts interval-based features (variance, slope, and mean) from time series subsequences and feed them into a random forest [15]. Buza and others proposed General Model-Selection Framework on Networks (GRAMOFON) which can run under 4 different heuristics: *Basic*, *BasicFast*, *RegOptMST* and *NetworkMST*. The first one performs an exhaustive search over classifier combinations while the last three heuristics all use minimal spanning trees that ensures the evaluation of the performance of all participating classifiers [16]. One of the recent time series ensemble technique based on kNN classifier is the Elastic Ensemble (EE). the former is more transparent than the previous methods in the sense that base learners are assessed independently on the cross-validation set and voting is then applied based on their accuracies. A number of voting heuristics have been coupled with the EE which are: *Equal*, *Best*, *Proportional*, and *Significant*. The *Equal* voting scheme does not discriminate between base learners and assigns them the same weights, *Best* scheme is similar to ES method in the sense that it consider only the base learner with the highest accuracy and sets the other classifiers weights to zero. *Proportional* assigns base classifiers weights based on their cross-validation accuracy that are normalized based on the number of transformations [17]. Finally, *Significant* voting is a hybrid between *Best* and *Proportional* as it includes the classifier with the highest validation accuracy and keeps including new classifiers only when they are significantly better than the classifier with the highest cross-validation accuracy [17]. EE ensemble uses 11 distance measures coupled with 1-NN model. In [17] it has been shown experimentally

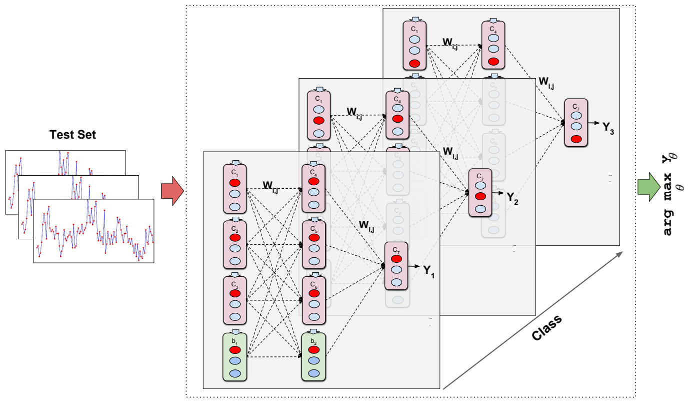
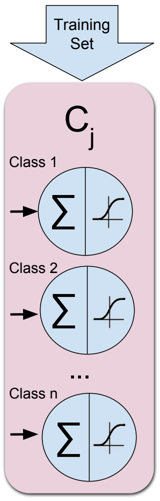
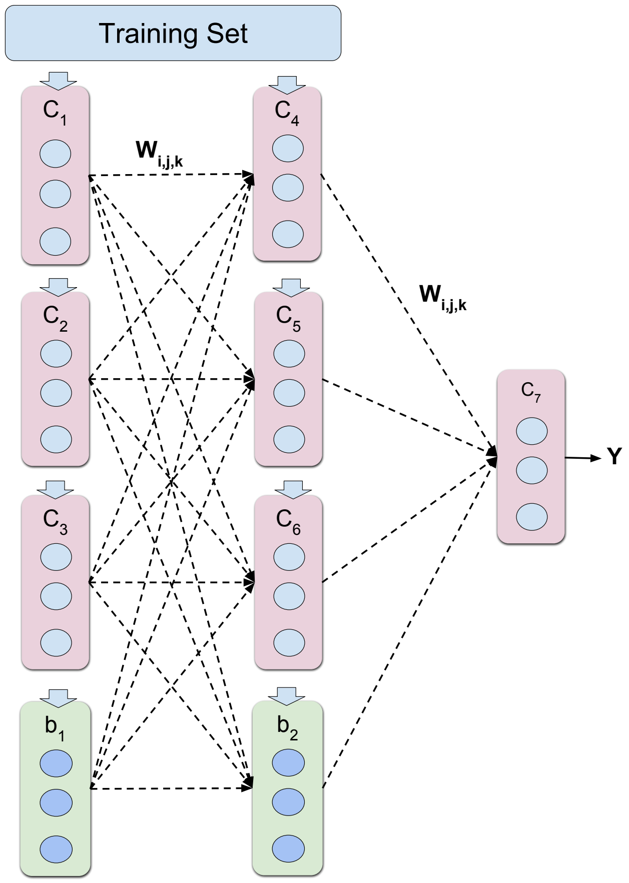


Fig. 1. Neuro-Ensemble Flowchart



(a)



(b)

Fig. 2. (a) Super-neuron representing classifier j closeup with N neurons corresponding to the number of classes in the multiclass classification problem. Every neuron contains an aggregation and activation functions. (b) Neuro-Ensemble overall architecture with one hidden layer and three super-neurons in the input and hidden layers.

that EE coupled with the *Proportional* voting scheme achieved superior results over other heuristics.

Meta-learning is another common strategy that involves two levels of learning [18]. Level-1 (also referred to as level-0) learning consists of training classifiers on the raw (or transformed) input space. The predicted classes (or class probabilities) output of level-1 learning constitutes the new input feature space for the level-2 learning. Our method pertains to the meta-learning category.

III. Neural Network Meta-learning

The goal of the Neuro-Ensemble meta-learner is to model the expertise of each base learner and use it to optimize the final ensemble model accuracy. Our hypothesis is that some learners are more expert in predicting a subset of classes over other base learners. We approach the problem of ensembling from an optimization perspective. Given a set of base learners, the goal is to optimize the cost function and update the weights of each learner based on the class they are the most expert on. Our method is based on a feedforward Multi-Layer Perceptron (MLP) that we specifically adjust to be trained on the class probabilities meta-features generated from the level-1 training of the base classifiers. We use a shallow MLP of 3 layers that we train with backpropagation in conjunction with batch gradient descent optimization algorithm. We define a new **super-neuron** entity that encapsulates a number of neurons equal to the number of classes in the multi-class classification problem. In addition to encapsulating a number of neurons and performing the activation computations, a super-neuron is embedded with classification logic from a given participating classifier in the ensemble. In other words, the super-neuron in Figure 2-a has the ability to classify (using classifier C_j) the samples itself and aggregate its vote vector with vote vectors (class probabilities) it received from previous layers. The encapsulated neurons within each super-neuron propagate their votes only to the next layer encapsulated neurons that have the same indices. In other terms, the vector element representing probability of class j will only be aggregated with the vector element representing the probability of the class j in the next layer. All the super-neurons in the network receive the same training data and make their own predictions on it. The input and output of a super-neuron are class probabilities. An illustration of the super-neuron is shown in Figure 2-a where every inner neuron represents a class.

Conceptually, our proposed approach can be thought of as multiple layers of MLPs trained in parallel, where each MLP layer is specialized in learning the expertise of classifiers in a given class/label. The number of MLP layers corresponds to the number of classes involved in the classification problem. The final ensemble vote constitute the representative class of the MLP that received the highest probability. Figure 1 illustrates the conceptual idea of the Neuro-Ensemble method. It can be noted that at every MLP layer, only 1 class is used for weights optimization at a time. The active neurons within the super-neurons at the level of each layer are shown in red in Figure 1.

Traditionally an artificial neuron entity receives a set of inputs $x_i \ldots x_n$ weighted with their respective weights $w_i \ldots w_n$. The processing involves summing the weighted input and feeding it to a non-linear activation function. In analogy with a neuron, the super-neuron entity receives a set of input vectors $X_i \ldots X_n$ and their associated weight vectors $W_i \ldots W_n$. The dimension of the vector is equal to the number of classes in the classification problem. The input weighting is done by performing a dot product of the input and weight vectors. At this stage, the super-neuron also participates by its own vote v that is added to the weighted sum vector. A component of the resulting vector is defined in Equation 1. After summing the weighted input vectors, the activation function is applied on the new vector component-wise as shown in Equation 2. We used a sigmoid function as the activation function (Equation 3).

$$\begin{gathered} n_{jk}^{(\ell)} = \textstyle\sum_{i=1}^{N_{\ell-1}} a_{ik}^{(\ell-1)} w_{ijk}^{(\ell)} + v_{jk}^{(\ell)} + b_{jk}^{(\ell)}, \\ j = 1, 2, \ldots, N_\ell. \qquad k = 1, 2, \ldots, N_c. \end{gathered} \tag{1}$$

$$a_{jk}^{(\ell)} = f^{(\ell)}(n_{jk}^{(\ell)}) = f^{(\ell)}(\sum_{i=1}^{N_{\ell-1}} a_{ik}^{(\ell-1)} w_{ijk}^{(\ell)} + v_{jk}^{(\ell)} + b_{jk}^{(\ell)}). \tag{2}$$

$$f^{(\ell)}(n_{jk}^{(\ell)}) = \frac{1}{(1 + e^{(-n_{jk}^{(\ell)})})} \tag{3}$$

The class probability vectors are propagated forward through the network using Equations 1-3 to obtain the corresponding network output vector, $\mathbf{Y}(e)$. The weights and biases are then adjusted using the batch gradient descent algorithm to minimize the squared error of the training vector in Equation 4.

$$E = \|\mathbf{Y}(e) - \mathbf{T}(e)\|^2 \tag{4}$$

where $\mathbf{T}(e)^{(q)}$ is the corresponding target vector for the training set vector $\mathbf{s}^{(q)}$ at epoch e. E is a function of all the weights and biases vectors of the network. The updating rules based on the batch gradient descent algorithm are then applied on the weight and bias vectors as shown in Equation 5.

$$\begin{aligned} w_{ijk}^{(\ell)}(e+1) &= w_{ijk}^{(\ell)}(e) - \alpha \frac{\partial E}{\partial w_{ijk}^{(\ell)}(e)} \\ b_{jk}^{(\ell)}(e+1) &= b_{jk}^{(\ell)}(e) - \alpha \frac{\partial E}{\partial b_{jk}^{(\ell)}(e)}, \end{aligned} \tag{5}$$

where $\alpha (> 0)$ is the learning rate. The next step involves computing the partial derivatives in Equation 5. E depends explicitly on the network output $\mathbf{Y}(t)$ (the activations of the last layer, $\mathbf{a}^{(l)}$), which also depends on the net input into the $l-$th layer, $\mathbf{n}^{(l)}$. In turn $\mathbf{n}^{(l)}$ is generated by the activations of the preceding layer and the weights and biases of layer l as shown in Equation 6.

$$\begin{aligned} E &= \|\mathbf{Y} - \mathbf{T}(e)\|^2 = \|\mathbf{a}_{jk}^{(l)} - \mathbf{T}(e)\|^2 \\ &= \|f^{(l)}(\mathbf{n}_{jk}^{(l)}) - \mathbf{T}(e)\|^2 \\ &= \left\| f^{(l)} \left(\sum_{i=1}^{N_{l-1}} a_{ik}^{(\ell-1)} w_{ijk}^{(\ell)} + v_{jk}^{(\ell)} + b_{jk}^{(\ell)} \right) - \mathbf{T}(e) \right\|^2 \end{aligned} \tag{6}$$

The partial derivatives of E with respect to the elements of $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are then derived using the chain rule for differentiation. For a general layer, ℓ, the partial derivatives are defined in Equation 7.

$$\begin{aligned} \frac{\partial E}{\partial w_{ijk}^{(\ell)}} &= \sum_{n=1}^{N_\ell} \frac{\partial E}{\partial n_n^{(\ell)}} \frac{\partial n_n^{(\ell)}}{\partial w_{ijk}^{(\ell)}} \\ \frac{\partial E}{\partial b_{jk}^{(\ell)}} &= \sum_{n=1}^{N_\ell} \frac{\partial E}{\partial n_n^{(\ell)}} \frac{\partial n_n^{(\ell)}}{\partial b_{jk}^{(\ell)}} \end{aligned} \tag{7}$$

for $j = 1, 2, \ldots, N_\ell$ and $k = 1, 2, \ldots, N_c$.

An example of a super-neuron based MLP architecture on a 3 class classification problem is shown in Figure 2-b. In this paper, we are particularly interested in heterogeneous classifiers fusion. Therefore, every super-neuron embed a different learning algorithm. We used the **best-first** heuristic to order the classifiers in the network. The best-first heuristic places the most accurate base classifiers on the level-1 validation in the hidden layer and the less performant classifiers are placed in the former layer (input layer). The full adjusted batch gradient descent algorithm that we used is shown in Algorithm 1.

The next section introduces the base classifiers used by the Neuro-Ensemble that are the same ones used by the Elastic Ensemble [17]. The base learners are all variations of 1-NN classifier coupled with different distance measures and metrics.

IV. Nearest Neighbor Participating Base Classifiers

A number of studies have focused on the use of k nearest neighbors classifier coupled with a distance measures for the time series classification problem. kNN is inherently based on the contiguity hypothesis which states that a test data should have the same label of the training examples in the surrounding radius. In addition to the training data, kNN requires the a priori knowledge of the k parameter and a similarity function that measures the proximity between the time series. Empirically, k=1 has been reported to achieve the best accuracies when coupled with DTW elastic distance [19]. The similarity function is a distance measure that can either be lock-step or elastic. Lock-step measures are used

Algorithm 1 Adjusted Batch Gradient Descent Algorithm

```
Input: Dataset $\mathcal{D}$, learning rate $\alpha$, number of epochs $ep$
Output: The trained $network$
1: $iteration \leftarrow 0$
2: while $iteration < ep$ do
3:     ▷ Propagate the input (Forward Pass)
4:     for each vector component $c$ do
5:         for each input layer unit $i$ do
6:             for each hidden layer unit $j$ do
7:                 $I_{jc} = \sum_i W_{ijc} v_{ic} + v_{jc} + \theta_{jc}$
8:                 $O_{jc} = \frac{1}{1+e^{-I_{jc}}}$
9:             end for
10:             ▷ Backpropagate the errors (Backward Pass)
11:             for each hidden layer unit $j$ do
12:                 $Err_{ic} = O_{jc}(1 - O_{jc}) \sum_k Err_{kc} W_{jkc}$
13:             end for
14:             for each weight $w_{ij}$ in the network do
15:                 $\delta w_{ijc} = \alpha Err_{jc} O_{jc}$
16:                 $w_{ijc} = w_{ijc} + \delta w_{ijc}$
17:             end for
18:             for each bias $\theta$ in the network do
19:                 $\delta\theta_{jc} = \alpha Err_{jc}$
20:                 $\theta_{jc} = \theta_{jc} + \delta\theta_{jc}$
21:             end for
22:         end for
23:     end for
24:     $iteration \leftarrow iteration + 1$
25: end while
26: return $network$
```

solely when the series have the same lengths as they perform a pairwise distance calculation of the time series data points using an L-norm distance, a generalization of other distances (e.g., Manhattan, Euclidean, ...). If the time series do not have equal lengths, re-sampling or interpolation can be used to make their lengths equal. On the other hand, elastic measures are applied also for time series that are not equi-length. In the next subsection we introduce all the measures that are coupled wih 1-NN and used as a base classifier for the EE and our NE ensembles.

A. *Lock-step Measures*

L_n norm or lock-step measures are among the efficient methods for estimating the similarity between time series and are favored due to their simplicity and applicability in indexing mechanisms. This family of measures do not compensate for the translation between the patterns in the time series which makes is relatively cheaper than other measures. L_n norm measures require that the input time series have equal lengths since they assess the time series similarity based on their pairwise distances. If the time series do not have equal lengths, re-sampling or interpolation can be used to make them equal. The general L_n norm distance formula is defined in Equation 8.

$$d_{L_n}(x, y) = \left(\sum_{n=1}^{M} (x_i - y_i)^n \right)^{\frac{1}{n}}, n \in \mathbb{N}^+, \tag{8}$$

where x and y are the input time series and M is the length of the time series. Equation 8 is also known as the L-norm distance which is a generalization of other distances (e.g., Manhattan, Euclidean, ...). When Equation 8 is used with n=1, Manhattan distance is obtained. If n=2, the distance is Euclidean. When n = inf the Equation 8 represents the Tchebychev distance. In this work we consider Euclidean distance from the lock-step measures category.

B. *Elastic Measures*

Elastic measures are invariant to the non-linear variations that happens in the time dimension. kNN classifier has shown to achieve superior results when used with elastic measures. This is mainly due to the fact that time series data are high dimensional in nature and the chronological order of their values is important. Therefore, the proper alignment of sequences using a flexible measure is crucial for the classification task.

1) Dynamic Time Warping (DTW): Dynamic time warping (DTW) is a standard elastic measure for assessing the dissimilarity between two time series [20]. Due to its effectiveness in finding an optimal match between two sequences, DTW has been used in many different domains such as shape interpolation [21] [22] [23], time series matching for incomplete medical data [24] and clustering [25] [26] [27]. DTW works by warping the time series in the time domain in such a way that the final warping cost is minimal. The canonical form of DTW is shown in Equation 9. M and N represent the lengths of the input time series x and y. Initially the D matrix is initialized to $D_{0,0} = 0$ and $D_{i,j}$ to $D_{i,j} = \inf$.

$$\begin{aligned} D_{i,j} = f(x_i, y_j) + min\{D_{i,j-1}, D_{i-1,j}, D_{i-1,j-1}\} \\ s.t: i \in (1, M), j \in (1, N) \end{aligned} \tag{9}$$

2) Derivative Dynamic Time Warping (DDTW): Another variation of the original DTW was proposed by Keogh and Pazzani [28] that transforms the input series into their estimated derivative (first order differences). The estimate represents the average of the line slopes passing through the given point and its left neighbor and the slope of the line passing through the given point and its right neighbor. The squared difference is then applied to compare the derivative of the sequences. The derivative estimation has shown to be more robust to noise and outliers and is defined in Equation 10.

$$D_x[q] = \frac{(q_t - q_{t-1}) + (q_{t+1} - q_{t-1})/2}{2} \tag{10}$$

3) Weighted Dynamic Time Warping (WDTW): Weighted Dynamic Time Warping was proposed by Jeong et al. to suppress the underline assumption of DTW and DDTW which supposes that all the points in the time series have the same weights; therefore, it is possible for points to be matched with neighboring points of the other series that are far in proximity

Algorithm 2 Longest Common Subsequence Measure

Input: First sequence s_1, Second sequence s_2

Output: LCSS distance $dist$

```
1: $M_{mxm} \leftarrow \text{INITIALIZEMATRIX}(0)$
2: for $i \leftarrow$ m to 0 do
3:     for $j \leftarrow$ m to 0 do
4:         $M_{i,j} = M_{i+1,j+1}$
5:         if $s_{1_i} = s_{2_j}$ then
6:             $M_{i,j} \leftarrow M_{i,j} + 1$
7:         else
8:             if $M_{i,j+1} > M_{i,j}$ then
9:                 $M_{i,j} \leftarrow M_{i,j+1}$
10:            end if
11:            if $M_{i+1,j} > M_{i,j}$ then
12:                $M_{i,j} \leftarrow M_{i+1,j}$
13:            end if
14:        end if
15:    end for
16: end for
17: return $M_{1,1}$
```

[29]. To mitigate this problem, WDTW introduces a positive weight bias to points that are close to the point to be matched. The weight function that is used is the modified logistic weight function. The weight value of a given point is determined by Equation 11.

$$w_i = \left[\frac{w_{max}}{1 + \exp -g(i - m_c)}\right]; \tag{11}$$

such that w_{max} is the user defined upper bound of the weights, m_c is the midpoint of the time series. m is the length of the time series, and g is constant found empirically and it controls the slope of the of the logistic function. The larger is the g parameter the more is the penalty.

4) Weighted Derivative Dynamic Time Warping (WDDTW): WDDTW is the combination of the aforementioned DDTW and WDTW measures. WDDTW transforms the time series into their first order differences introduced in Equation 10 and then applies WDTW on the sequence derivatives to assess their similarity using Equation 11.

5) Longest Common Subsequence (LCSS): LCSS is another state-of-the-art similarity measure that gained a lot of popularity in the biology field where it is used for comparing DNA sequence (genes). LCSS was later used in time series data mining and was particularly popular for its robustness to noise. LCSS finds the longest subsequence between two sequences and then defines the distance using the length of this subsequence. LCSS uses two parameters δ and ϵ that should be optimized using the validation data.δ is defined as a percentage of the original time series and it signifies the sliding windows size used to match points across 2 sequences [30]. ϵ is constant that represents the matching threshold such that $\epsilon \in [0, 1]$. Algorithm 2 shows full details about the LCSS measures.

6) Move Split Merge (MSM): MSM computes the distance between two sequences by transforming one sequence into another one using only three operations: move, split and merge. The move operation allows a simple edit of the value of a given point in the sequence. Split operation duplicates the value of a given point to the next neighboring position in the sequence. The merge operation allows two nodes of the same values to merge in one and therefore deleting one node from the sequence. Inserting a value to the sequence requires a split operation followed by a move to modify the newly duplicated value. MSM is a deterministic distance measure that is starting-point invariant and has been proven to obey the triangular inequality property [31]; therefore, it has the desirable property of being metric. The full description of the MSM metric is shown in Algorithm 3.

Algorithm 3 Move Split Merge

Input: First sequence x_1, Second sequence x_2

Output: MSM distance $dist$

```
1:   ▷ Parameter Initialization
2: $Cost(1,1) \leftarrow |x_{1_1} - x_{2_1}|$
3: for $i \leftarrow$ 2 to m do
4:     $Cost(i,1) = Cost(i-1,1) + C(x_{1_1}, x_{1_{i-1}}, x_{2_1})$
5: end for
6: for $j \leftarrow$ 2 to n do
7:     $Cost(i,1) = Cost(1,j-1) + C(x_{2_j}, x_{1_1}, x_{2_{j-1}})$
8: end for
9:   ▷ Main Loop
10: for $i \leftarrow$ 2 to m do
11:     for $j \leftarrow$ 2 to n do
12:         $Cost(i,j) = min\{Cost(i-1,j-1) + |x_{1_i} - x_{2_1}|$
            $\}$
13:     end for
14: end for
15: return $Costm, n$
```

7) Time Warp Edit Distance (TWE): TWE is another metric that was proposed by Marteau[32]. TWE adapts the popular Edit Distance measure to time series matching by introducing penalty for insertion and deletion of values in the sequences and a *stiffness* parameter that controls the elasticity of the match on the time dimension. More details on the TWE metric can be found in [32].

8) Edit Distance with Real Penalty (ERP): ERP, also known as the marriage of $Lp - norms$ and Edit Distance [33], is another adaptation of Edit Distance for time series data matching. ERP redefines the delete operation of the traditional Edit Distance by differentiating between deletion from sequence x_1 to match sequence x_2 (delete_x_1) and vice versa (delete_x_2). When a match is found, an *L-p norm* distance is applied; otherwise, a penalty is applied when a match is not found.

V. Experimental Setup

In this section, we introduce the UCR datasets we used and explain our experimental settings. We follow up by defining the sampling methodology we adopted.

TABLE I
UCR DATASETS METADATA

ID	Dataset Name	DS Size	TS Length	$\mathcal{L}$
1	Adiac	781	176	37
2	ArrowHead	211	251	3
3	Beef	60	470	5
4	BeetleFly	40	512	2
5	BirdChicken	40	512	2
6	CricketX	780	300	12
7	CricketY	780	300	12
8	CricketZ	780	300	12
9	ECG200	200	96	2
10	FacesUCR	2250	131	14
11	Herring	128	512	2
12	Lightning2	121	637	2
13	Lightning7	143	319	7
14	MedicalImages	1141	99	10
15	OliveOil	60	570	4
16	Phoneme	2110	1024	39
17	Plane	210	144	7
18	SwedishLeaf	1125	128	15
19	ToeSegmentation1	268	277	2
20	WordSynonyms	905	270	25
21	Worms	258	900	5
22	MiddlePhalanxOAG	554	80	3
23	ShapeletSim	200	500	2
24	Computers	500	720	2
25	Earthquakes	461	512	2
26	Fish	350	463	7
27	GunPoint	200	150	2
28	Haptics	463	1092	5
29	Meat	120	448	3
30	OSULeaf	442	427	6
31	RefrigerationDevices	750	720	3
32	Ham	214	431	2
33	LargeKitchenAppliances	750	720	3
34	ProximalPhalanxOAG	605	80	3
35	Car	120	577	4
36	CBF	930	128	3
37	Coffee	56	286	2
38	DistalPhalanxTW	539	80	6
39	FaceFour	112	350	4
40	ScreenType	750	720	3
41	SmallKitchenAppliances	750	720	3
42	SyntheticControl	600	60	6
43	Trace	200	275	4

A. Datasets

The performance of the ensemble methodologies and other baselines are evaluated on 43 datasets from the UCR Machine Learning repository [7]. Table I presents the details of the datasets used in this study (dataset name, dataset size, time series length, and number of classes).

B. Sampling

In order to compare the performance of different ensemble algorithms, we have set some desiderata. The first requirement when evaluating an ensemble methodology is that there should be still room for improvement of every base classifier participating in the ensemble. Theoretically, the Naive Bayes error represents the optimal level of learning of a classifier where no more improvement can be made [1]. To make the

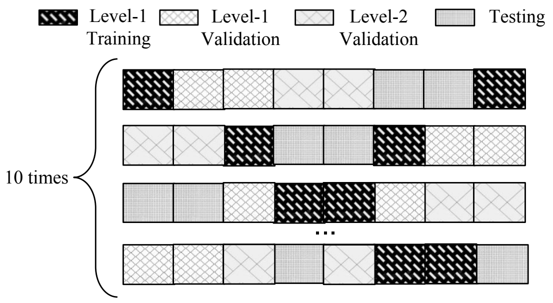


Fig. 3. Sampling methodology (25/25/25/25)

problem challenging, the base models should be trained with few instances before they reach the Bayes optimal error. In order to meet this requirement, we used a stratified 25% of each dataset for training the base classifiers participating in the ensemble.

As mentioned earlier, our proposed Neuro-Ensemble meta-learner involves two levels of learning. Level-1 learning consists of training the base learners and validating them. We reserved another stratified holdout sample of 25% of the dataset for validating the level-1 base learners. Level-2 learning involves using the meta-features generated from the level-1 learners and feeding them to the Neuro-Ensemble. In this paper, the meta-features represent the class predictions on the level-1 training and level-1 validation datasets. In other terms, 50% of the dataset is used for level-2 training. To validate the level-2 meta-learner, we used another stratified holdout sample of 25% of the datasets as a level-2 validation set. Training on a large dataset can reduce the number of instances used for testing, resulting in high variability in the classifier performance [34]. Therefore, we reserved the last quarter of the dataset as the never-seen testing set of the ensemble. The stratified 25/25/25/25 split is repeated 10 times.

VI. EXPERIMENTAL RESULT

In this section, we will discuss three criteria of the experiments we conducted to show the efficiency of our ensemble from different perspectives. We first discuss the accuracy performance of the Neuro-Ensemble in comparison with the participating base classifiers, we then explore the relationship between time series lengths and the ensemble methodology efficiency. Finally, we compare the overall accuracy and execution times of the Neuro-Ensemble and the Elastic Ensemble method coupled with the *Proportional* voting heuristic.

The scatter plot and boxplot in Figure 4 shows the average test accuracy of the Neuro-Ensemble and the 11 participating classifiers on the 10-folds validation sets of each of the 43 datasets respectively. The figure also shows the minimum and maximum accuracy of the base classifiers. It can be observed that the Neuro-Ensemble is, in most of the cases, superior to its underlying base classifiers which makes it better than the Evaluation and Selection (ES) method. The former selects the most accurate classifier based on their performance on the validation set and use it for the testing. Figure 4 shows the

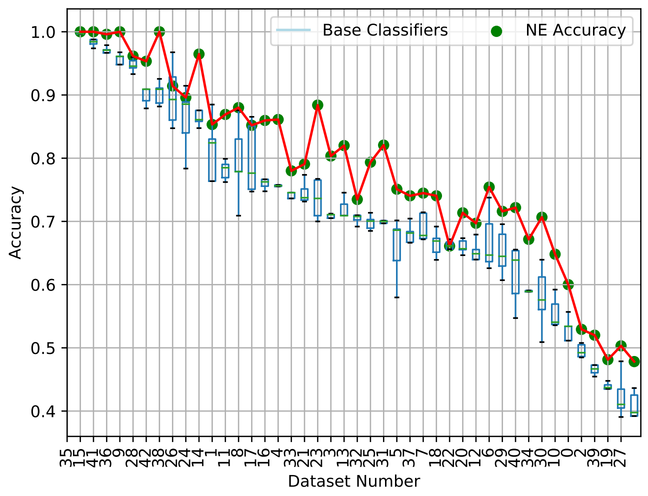


Fig. 4. Average accuracy of the Neuro-Ensemble (NE) method and boxplot accuracies of all the 11 participating base classifiers on the 10-fold validation set.

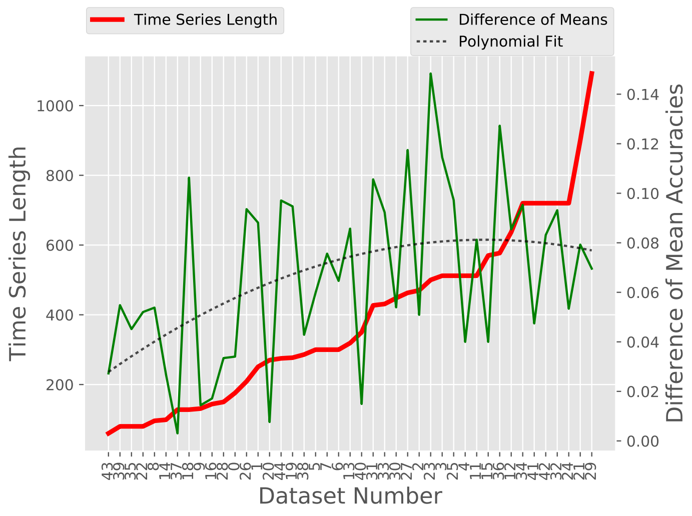


Fig. 5. Average Accuracy Difference of Neuro-Ensemble (NE) method and the mean accuracies of the 11 participating base classifiers

performance distribution of all the base classifiers on the test data, which means that even with the optimal choice of the best classifier on the validation set, Neuro-Ensemble is almost always superior than Evaluation and Selection.

In order to make more sense of the improvement difference of NE compared to ES, we evaluate the average performance difference of NE and the base classifiers mean accuracies. Figure 5 shows the ordered list of datasets based on their corresponding time series lengths that is shown in the (red) line plot. The second (green) line plot shows the difference of accuracies between NE and the mean accuracies of the base classifiers. The dotted line plot represents a polynomial fit of the difference of means. The first observation that can be made is that there is upward trend on the polynomial fit that stops at

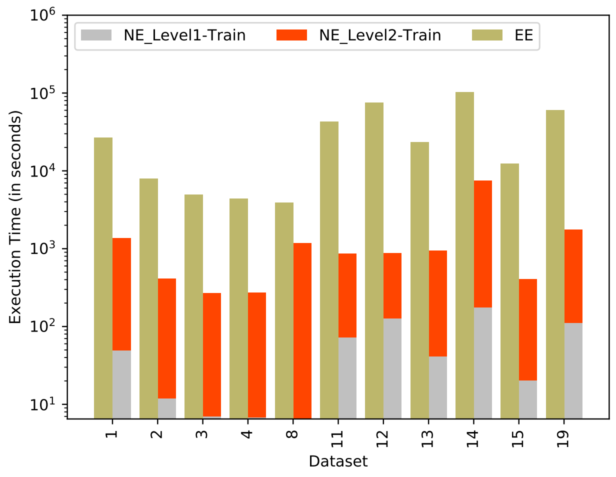


Fig. 6. Training times of NE and EE

around time series length (≈ 600). This suggests that the use of NE is useful for time series datasets whose sequences lengths fall in the range $[40, 600]$. When the time series length is high, the ensemble performance gain is comparatively limited. We explain this phenomena by the types of base classifiers used in this study. Empirically, it has been shown that shape-based time series classifiers that rely on the use of distance measures are not particularly efficient for long time series [35]. As a result, the base classifiers are reaching their optimal Naive Bayes error quickly which does not leave any room for improvement for the ensemble. This hypothesis needs further validation by using other base classifiers in NE that are better suited for longer time series data such as interval based and shapelet-based classifiers [36].

In the next round of experiments, we compared our method to the state-of-the-art Elastic Ensemble coupled with *Proportional* voting heuristic with a focus on a subset of datasets where the Neuro-Ensemble has shown to have a significant difference over the use of the best base classifiers. The datasets of focus fall in the upward trend of the polynomial fit line shown in Figure 4. We compared our method to the Elastic Ensemble based on their respective training times and test accuracies. Figure 8 shows the bar plots representing the training time of the Elastic Ensemble and the stacked bar plots showing the level-1 and level-2 training times of our Neuro-Ensemble. In all of the cases, the total training time of the Neuro-Ensemble is significantly less than the training time of the Elastic Ensemble. NE is at least *3x* faster than EE and up to 86*x* faster (the plot is in log-scale). Another observation that can be made is that the level-2 training time is proportional to the number of classes of the datasets shown in Table I. As explained earlier, our proposed ensemble can be conceptually thought of a multi-layer perceptron of $\mathcal{L}$ layers as illustrated in Figure 1. The more classes the problem has, the more layers exists which necessitate more training. It is also important to note that level-2 training time of NE is always

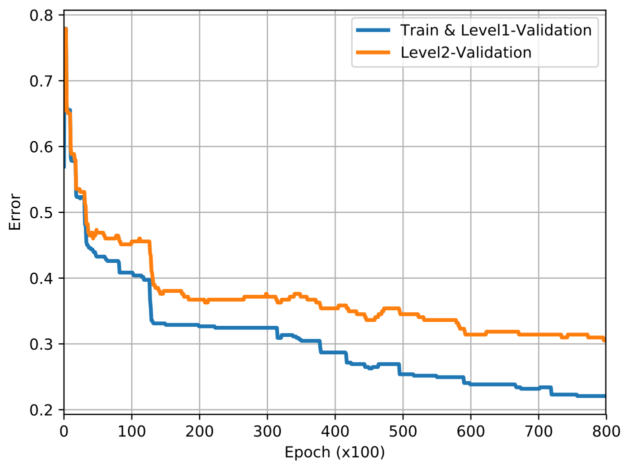


Fig. 7. Learning Curve of the Neuro-Ensemble on the Beetfly dataset

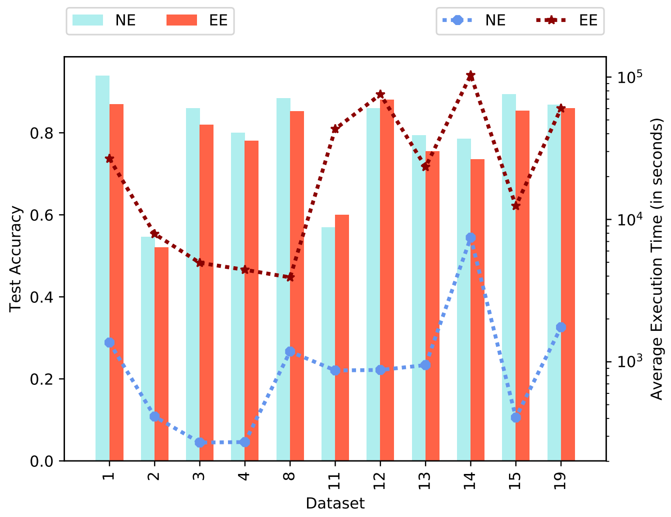


Fig. 8. Test accuracy and average execution time of NE and EE

greater that the level-1 training of the base classifiers. This finding is expected since we used $80,000$ epochs to train NE to make sure that we do not prematurely stop the training. An example of learning curves for the *Beetlefly* dataset is shown in Figure 7. The curve shows that after $\approx 70,000$ epochs, the validation error stagnates which indicates the sweet spot where training should be ended to avoid overfitting the model.

Figure 8 shows the mean accuracy of NE and EE methods with their corresponding training times. NE is generally more accurate than EE and takes significantly less time to train (up to $86x$ faster). The highest accuracy improvement were noticed in dataset 1 and dataset 8. After mapping those datasets to Table 1, we noted that they have a relatively high number of labels. This suggests that NE works particularly better for the case of classification problems with a high number of classes.

VII. Conclusion

In this paper, we presented a new heterogeneous ensemble based on a modified neural network meta-learner applied on time series data. We addressed the problem of classifiers fusion from an optimization perspective. We used an adjusted Multi-Layer Perceptron learner based on super-neurons that models the class probabilities. A super-neuron entity has embedded classification capability in addition to the activation and state computations. The adjusted network is trained on class probabilities emanating from every super-neuron. Consequently, we are able to optimize the cost function and update the weights and biases per class/label basis. Our proposed approach learns the expertise of every super-neuron in the network and assigns higher weights for the classes it is most expert on and vice versa. This scheme ensures that base learners will participate in the ensemble prediction only for the classes they are most expert on and marginally contribute to others. We tested our method on 43 datasets from the UCR repository by comparing its relative performance to the 11 base classifiers and state-of-the-art Elastic Ensemble. We found that our approach improves the simple Evaluation and Selection method when the time series length is not very long and the datasets contains many classes.

Neuro-Ensemble can be coupled with different heuristics when ordering the heterogeneous base learners across the network. In this work, we explored the best-first heuristic. As a future direction of this work, we would like to explore other heuristics for classifiers ordering in the network. In addition, we want to explore other architectures that are more complex than a simple shallow network.

VIII. Acknowledgment

We would like to thank Anthony Bagnall for making their Elastic Ensemble as well as all the distance measures source code available. This project has been supported in part by funding from the Division of Advanced Cyber infrastructure within the Directorate for Computer and Information Science and Engineering, the Division of Astronomical Sciences within the Directorate for Mathematical and Physical Sciences, and the Division of Atmospheric and Geospace Sciences within the Directorate for Geosciences, under NSF award #1443061. It was also supported in part by funding from the Heliophysics Living With a Star Science Program, under NASA award #NNX15AF39G.

References

[1] D. H. Wolpert, W. G. Macready *et al.*, "No free lunch theorems for search," Technical Report SFI-TR-95-02-010, Santa Fe Institute, Tech. Rep., 1995.

[2] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[3] G. Brown and L. I. Kuncheva, "good and bad diversity in majority vote ensembles," in *International Workshop on Multiple Classifier Systems*. Springer, 2010, pp. 124–133.

[4] S. M. Hamdi, D. Kempton, R. Ma, S. F. Boubrahimi, and R. A. Angryk, "A time series classification-based approach for solar flare prediction," in *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2543–2551.

[5] R. Ma, S. F. Boubrahimi, S. M. Hamdi, and R. A. Angryk, "Solar flare prediction using multivariate time series decision trees," in *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2569–2578.

[6] S. F. Boubrahimi, B. Aydin, P. Martens, and R. Angryk, "On the prediction of¿ 100 MeV solar energetic particle events using goes satellite data," in *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2533–2542.

[7] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, "The ucr time series classification archive," http://www.cs.ucr.edu/~eamonn/time_series_data, 2015.

[8] S. Džeroski and B. Ženko, "Is combining classifiers with stacking better than selecting the best one?" *Machine learning*, vol. 54, no. 3, pp. 255–273, 2004.

[9] B. Pfahringer, H. Bensusan, and C. G. Giraud-Carrier, "Meta-learning by landmarking various learning algorithms." in *ICML*, 2000, pp. 743–750.

[10] H. Bensusan, C. G. Giraud-Carrier, and C. J. Kennedy, "A higher-order approach to meta-learning." *ILP Work-in-progress reports*, vol. 35, 2000.

[11] A. Kalousis and T. Theoharis, "Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection," *Intelligent Data Analysis*, vol. 3, no. 5, pp. 319–337, 1999.

[12] P. B. Brazdil, C. Soares, and J. P. Da Costa, "Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results," *Machine Learning*, vol. 50, no. 3, pp. 251–277, 2003.

[13] K. Woods, W. P. Kegelmeyer, and K. Bowyer, "Combination of multiple classifiers using local accuracy estimates," *IEEE transactions on pattern analysis and machine intelligence*, vol. 19, no. 4, pp. 405–410, 1997.

[14] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[15] 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.

[16] D.-I. K. A. Buza, "Fusion methods for time-series classification," *UPDATE*, vol. 2, p. 27, 2011.

[17] J. Lines and A. Bagnall, "Ensembles of elastic distance measures for time series classification," in *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 2014, pp. 524–532.

[18] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta, *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[19] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana, "Fast time series classification using numerosity reduction," in *23rd ICML*. ACM, 2006, pp. 1033–1040.

[20] E. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," *KIS*, vol. 7, no. 3, pp. 358–386, 2005.

[21] S. F. Boubrahimi, B. Aydin, M. A. Schuh, D. Kempton, R. A. Angryk, and R. Ma, "Spatiotemporal interpolation methods for solar event trajectories," *The Astrophysical Journal Supplement Series (APJs)*, vol. 236, no. 1, p. 23, 2018.

[22] S. F. Boubrahimi, B. Aydin, D. Kempton, and R. Angryk, "Spatio-temporal interpolation methods for solar events metadata," in *IEEE Big Data 2016*. IEEE, 2016, pp. 3149–3157.

[23] S. F. Boubrahimi, B. Aydin, D. Kempton, S. S. Mahajan, and R. Angryk, "Filling the gaps in solar big data: Interpolation of solar filament event instances," in *BDCloud 2016*. IEEE, 2016, pp. 97–104.

[24] P. Tormene, T. Giorgino, S. Quaglini, and M. Stefanelli, "Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation," *Artificial intelligence in medicine*, vol. 45, no. 1, pp. 11–34, 2009.

[25] R. Ma, S. F. Boubrahimi, and R. Angryk, "Time series distance density cluster with statistical preprocessing," in *DaWaK 2018, 20th International Conference on Big Data Analytics and Knowledge Discovery*.

[26] R. Ma, R. A. Angryk, P. Riley, and S. F. Boubrahimi, "Coronal mass ejection data clustering and visualization of decision trees," *The Astrophysical Journal Supplement Series*, vol. 236, no. 1, p. 14, 2018.

[27] S. F. Boubrahimi, R. Ma, B. Aydin, S. M. Hamdi, and R. Angryk, "Scalable knn search approximation for time series data," in *ICPR 2018, International Conference on Pattern Recognition 2018.*

[28] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping," in *Proceedings of the 2001 SIAM International Conference on Data Mining*. SIAM, 2001, pp. 1–11.

[29] 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.

[30] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE, 2002, pp. 673–684.

[31] A. Stefan, V. Athitsos, and G. Das, "The move-split-merge metric for time series," *IEEE transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1425–1438, 2013.

[32] P.-F. Marteau, "Time warp edit distance with stiffness adjustment for time series matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 306–318, 2009.

[33] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, pp. 792–803.

[34] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine learning*, vol. 36, no. 1-2, pp. 105–139, 1999.

[35] 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.

[36] J. J. Rodríguez, C. J. Alonso, and J. A. Maestro, "Support vector machines of interval-based features for time series classification," *Knowledge-Based Systems*, vol. 18, no. 4-5, pp. 171–178, 2005.