Hindawi
Science and Technology of Nuclear Installations
Volume 2017, Article ID 2709109, 14 pages
https://doi.org/10.1155/2017/2709109

Research Article

Postprocessing of Accidental Scenarios by Semi-Supervised Self-Organizing Maps

Francesco Di Maio,[1] Roberta Rossetti,[1] and Enrico Zio[1,2]

[1]*Energy Department, Politecnico di Milano, Via La Masa 34, 20156 Milano, Italy*
[2]*Chair on System Science and Energetic Challenge, Fondation EDF, Ecole Centrale, Supelec, Paris, France*

Correspondence should be addressed to Francesco Di Maio; francesco.dimaio@polimi.it

Received 12 June 2017; Revised 12 October 2017; Accepted 30 October 2017; Published 12 December 2017

Academic Editor: Michael I. Ojovan

Integrated Deterministic and Probabilistic Safety Analysis (IDPSA) of dynamic systems calls for the development of efficient methods for accidental scenarios generation. The necessary consideration of failure events timing and sequencing along the scenarios requires the number of scenarios to be generated to increase with respect to conventional PSA. Consequently, their postprocessing for retrieving safety relevant information regarding the system behavior is challenged because of the large amount of generated scenarios that makes the computational cost for scenario postprocessing enormous and the retrieved information difficult to interpret. In the context of IDPSA, the interpretation consists in the classification of the generated scenarios as safe, failed, Near Misses (NMs), and Prime Implicants (PIs). To address this issue, in this paper we propose the use of an ensemble of Semi-Supervised Self-Organizing Maps (SSSOMs) whose outcomes are combined by a locally weighted aggregation according to two strategies: a locally weighted aggregation and a decision tree based aggregation. In the former, we resort to the Local Fusion (LF) principle for accounting the classification reliability of the different SSSOM classifiers, whereas in the latter we build a classification scheme to select the appropriate classifier (or ensemble of classifiers), for the type of scenario to be classified. The two strategies are applied for the postprocessing of the accidental scenarios of a dynamic U-Tube Steam Generator (UTSG).

1. Introduction

The number of dynamic scenarios considered in an Integrated Deterministic and Probabilistic Safety Analysis (IDPSA) of dynamic systems increases with the number of failure events that can occur and the consideration of their timing and sequencing. This can make the computational cost for scenario postprocessing enormous and the retrieved information difficult to interpret [1–4]. The main goal of postprocessing is the classification of the dynamic scenarios generated as safe, failed, Near Misses (NM), and Prime Implicants (PIs) clusters. Safe scenarios are those that, even if several components failures are included, keep the system working in safe conditions. Failed scenarios, instead, result from a combination of failure events that lead the system into a failed condition. Among failed scenarios, PIs are those scenarios containing events representing the minimal combinations of component failure necessary for system failure [5] (i.e., the dynamic systems equivalent of Minimal Cut Sets (MCSs)). Among safe scenarios, NMs are dangerous sequences of events that lead the system to a quasi-fault state [6].

Many methods have been proposed in literature for the classification task. A first step could be distinguishing failed scenarios from safe scenarios, for example, by a fuzzy-*c*-means (FCM) classifier [6], a Mean-Shift Methodology (MSM) [7], or a decision tree [8]. Methods have been proposed for the identification also of PIs and Near Misses. For example, PIs identification has been performed with a differential evolution-based method [9] or a visual interactive method [10], where the number of components whose behavior is specified in the accident sequence is selected as most important feature for the PIs identification: the accident sequences associated with the lowest literal cost are selected and stored as PIs (most reduced sequences, i.e., with least number of events, that cannot be covered by any other implicant). Regarding the identification of the Near Misses

sequences, an unsupervised clustering problem based on an optimized wrapper algorithm and the K-means clustering algorithm has been proposed (MacQueen, 1967) [4]. A comprehensive method for accidental scenarios classification can be provided by Self-Organizing Maps (SOMs) [11], which have been widely used in various engineering and physical applications, including fault detection and diagnosis in complex systems [12, 13]. SOMs capture nonlinear relationships of high-dimensional data and visualize them on a low-dimensional interface, normally a 2D structure of, so-called, neurons. In this structure, data are assigned to the most similar neuron called Best Matching Unit (BMU) (usually by measuring the smallest Euclidean distance), so that the available data are divided into regions with common characteristics (i.e., data with high similarity to the same BMU are mapped close to each other). Three kinds of SOM exist: the Unsupervised SOM (USOM), the Semi-Supervised SOM (SSSOM), and the Supervised SOM (SSOM). We have shown in [14] a SSSOM performs best in identifying safe, failed, NMs, and PIs groups of scenarios. In particular, assigning the set of discrete variables (i.e., the failure sequences) to a BMU, a SSSOM (implemented with a Manhattan distance as similarity measure) is particularly suitable to properly treat the MVL approximation needed for the representation of the dynamic scenarios (the usual binary variables representation used in Boolean Logic, in which the modeling is limited only to the occurrence or not of certain events [2–4, 6, 9, 10, 15], is not sufficient). In this work, it will be shown that the SSSOM performance in classifying different groups of scenarios depends on the feature of the SSSOM that is used as discriminating characteristics for choosing the BMU (e.g., assigning the data to the cluster with the geometric barycenter more similar to the input data or to the cluster with the maximum (minimum) neuron (i.e., with the maximum (minimum) weights) more similar to the input data). The results confirm that depending on this, some classifiers overperform the stand-alone SSSOM for some classes and vice versa. This suggests adopting an ensemble approach for an improved classification of accidental scenarios.

The main objective of this work is to propose a postprocessing tool for dynamic accidental scenarios, which exploits an ensemble of classifiers. In fact, by doing so, it is possible to leverage the classifiers complementary characteristics and to boost overall classification accuracy (in terms of the multiobjective precision sensitivity and specificity) [16]. In general, strategies for boosting diversity include (i) using different types of classifiers (this is the technique we adopt for our application); (ii) training individual classifiers with different data sets; (iii) using different subsets of features. Various methodologies exist for aggregation of the outcomes of individual classifiers: majority vote [17], Borda count [18], threshold voting [Ho 1994], weighted average [19], fuzzy integral [20], fuzzy templates [21], and Dempster-Shafer theory [22]. Furthermore, methods have been developed to dynamically select a classifier from the set of available ones, based on local information [23]: different classifiers perform best in different regions and this aggregation can lead to improving classification results; in a supervised setting, the individual classifier performance can be calibrated based on historical data with known target values; each individual classifier performance value reflects the degree to which we want each classifier to contribute in the ensemble aggregation: the best performing classifier for a given scenario type should contribute most [16]. On these premises, we propose two alternative strategies based on locally weighted aggregation of SSSOMs outcomes: a locally weighted ensemble and a decision tree based on an ensemble. For both strategies, we resort to the Local Fusion (LF) principle [19] for building the ensemble outcome, based on each classifier local performance, measured by the classification accuracy on scenarios in the neighborhood of (i.e., similar to) the test scenario considered.

In the locally weighted ensemble strategy, we ensemble the classification outcomes of the SSSOMs whose assignments to a BMU are given with respect to the different features characterizing the SSSOM (e.g., the Mean Quantization Error (MQE) based SSSOM, the barycenter based SSSOM, the minimum neuron based SSSOM, the maximum neuron based SSSOM, and the stand-alone SSSOM).

Differently, the decision tree based classification scheme chooses one single classifier (or ensemble of classifiers), based on the local performance of the test scenario. In this way, input scenarios with similar characteristics are treated by the same classifier (or ensemble of classifiers) for which the branch of the tree is most effective.

The feasibility of combining local information for postprocessing IDPSA scenarios for their classification into safe, failed, NMs, and PIs, is demonstrated with respect to a dynamic U-Tube Steam Generator (UTSG) of a NPP [24]. For IDPSA scenarios generation, a dynamic simulation model has been implemented in SIMULINK and a multivalued logic (MVL) scheme [4] has been adopted for describing the different component operational states in the scenarios.

The paper is organized as follows. In Section 2, the UTSG and its SIMULINK model are presented. In Section 3, the SSSOMs are presented and different features are considered as discriminating characteristics for the classification; also the LF process for ensembling is outlined. In Section 4, the locally weighted ensemble of SSSOMs and the decision tree based ensemble of SSSOMs are presented, and the results on the case study considered are reported. In Section 5, some conclusions and final remarks are given.

2. Case Study

We consider the dynamic scenarios of a UTSG used in nuclear power plants for scenarios generation. A SIMULINK model has been used to describe the UTSG response at different power levels P_0 [4]. The component failures considered for UTSG are as follows (Figure 1): the steam valve failure, the safety relief valve failure, the interruption of the communication between the sensor that monitors the water level (governed by the balance between the incoming and exiting feed water) and the Proportional Integrative Derivative (PID) controller, and the PID failure. A mission time (T_{miss}) of 4000 (s) has been considered for allowing complete development also of slow dynamic accident scenarios occurring at early/medium times. The component failures are considered occurring at any continuous time instant, with

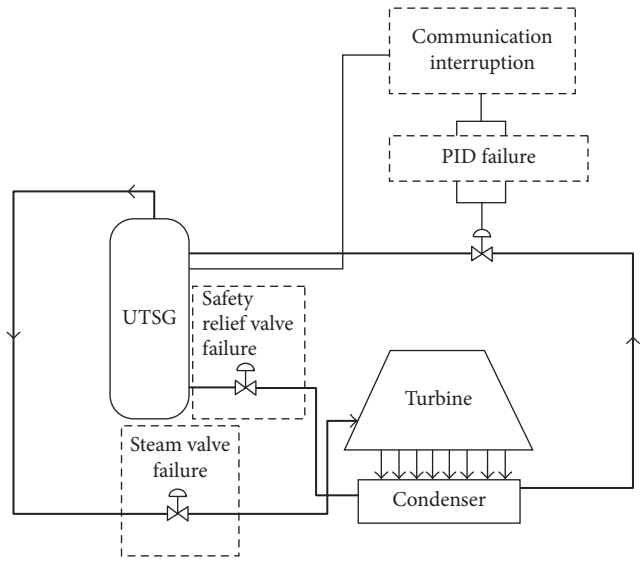

Figure 1: Sketch of the failures that can be injected into the system.

any order in the sequence and magnitude. Assumptions on the failure occurrence process have been made in order to (i) favor the occurrence of multiple failures in the scenarios, (ii) capture the dynamic influence of all factors of interest, and (iii) treat a comprehensive (but still manageable) problem for which scenarios postprocessing is required for a robust risk quantification.

For the tractability of the problem, we resort to a multivalued logic (MVL) computational framework in which the components can fail at discrete times and magnitudes [9]. The discretization consists in the following:

(i) Time: for each component, the mission time (T_{miss}) is divided into four intervals, labeled $t = 1$ for a failure in $[0, 1000]$ (s), $t = 2$ in $[1000, 2000]$ (s), $t = 3$ in $[2000, 3000]$, and $t = 4$ in $[3000, 4000]$. If $t = 0$ the component does not fail in T_{miss}.

(ii) Component failure magnitudes:

(a) the steam valve failure magnitude is indicated as 1, 2, or 3 for failure states corresponding to stuck at 0%, at 50%, and at 150% of the Q_e value that should be provided at power level P_o, respectively; if the steam valve magnitude is indicated as 0, the component does not fail in T_{miss};

(b) the safety relief valve failure magnitude is indicated as 1, 2, 3, and 4, if it is stuck between $[0.5, 12.6]$ (kg/s), $[12.6, 25.27]$ (kg/s), $[25.27, 37.91]$ (kg/s), and $[37.91, 50.5]$ (kg/s), respectively; if the safety relief valve magnitude is indicated as 0, the component does not fail in T_{miss};

(c) the communication between the sensor measuring the water level and the PID controller is labeled with 0 if the communication works, with 1 otherwise;

(d) the PID controller failure magnitude is discretized into 8 equally spaced magnitude intervals, labeled from 1 to 8, representative of failure states corresponding to discrete intervals of output value belonging to $[-18, 18]$% of the Q_e value that should be provided at P_o; if the PID controller magnitude is labeled as 0, the component does not fail in T_{miss}.

Each scenario is represented by a set of values of the multivalued variables of time, magnitude, and order of occurrence, which are resumed into a sequence vector: each sequence is, thus, an MVL vector $\overline{X} = [x_1, x_2, \ldots, x_d]$, of length $d = 12$. For example, $[2, 3, 1, 3, 1, 3, 2, 1, 2, 4, 6, 4]$ corresponds to a scenario where

(i) the steam valve fails stuck at its maximum allowable value (3) at a time (2) in $[1001, 2000]$ (s) and it is the first (1) event occurring along the sequence;

(ii) the safety relief valve fails third (3) in the time interval (3) equal to $[2001, 3000]$ (s), with a magnitude (1) belonging to $[0.5, 12.6]$ (kg/s);

(iii) the communication between the sensor measuring the water level and the PID controller is the second (3) failure event (1) in the sequence, and it occurs in the time interval (2) of $[1001, 2000]$ $[1001, 2000]$ (s);

(iv) the PID controller fails stuck as fourth (4) in the time interval (4) of $[3001, 4000]$ (s), with a magnitude (6) belonging to $[6, 10]$% of the Q_e value that should be provided at P_o.

All possible combinations of multiple component failures, each represented by a MVL vector of time, magnitude, and order of occurrence, lead to a total of $N = 100509$ accidental MVL scenarios to be treated for the quantification of the risk related to the UTSG operation.

3. The Ensemble

The design of a successful ensemble consists of two important parts [25, 26]: (1) the design of the individual classifiers (Section 3.1); (2) the design of the aggregation mechanism (Section 3.2) [27].

3.1. Design of Classifiers. For postprocessing the $N = 100509$ multivalued dynamic scenarios of the UTSG, we resort to a Semi-Supervised Self-Organizing Map (SSSOM) based on the Manhattan distance (shown in Figure 2(c)). This SSSOM has been shown in [14] to be efficient for grouping the scenarios in four distinct regions of the map and retrieving safety relevant information, and it is hereafter shown to be capable of further improvement when trained to classify new data based on different features of this same SSSOM to be used as BMUs and, then, their outcomes are ensembled into the final classification: we shall see that certain classifiers overperform the others for certain classes and vice versa. Specifically, we build $K = 5$ classifiers the stand-alone SSSOM, the MQE based SSSOM, the barycenter based SSSOM, the minimum neuron based SSSOM, and the maximum neuron based

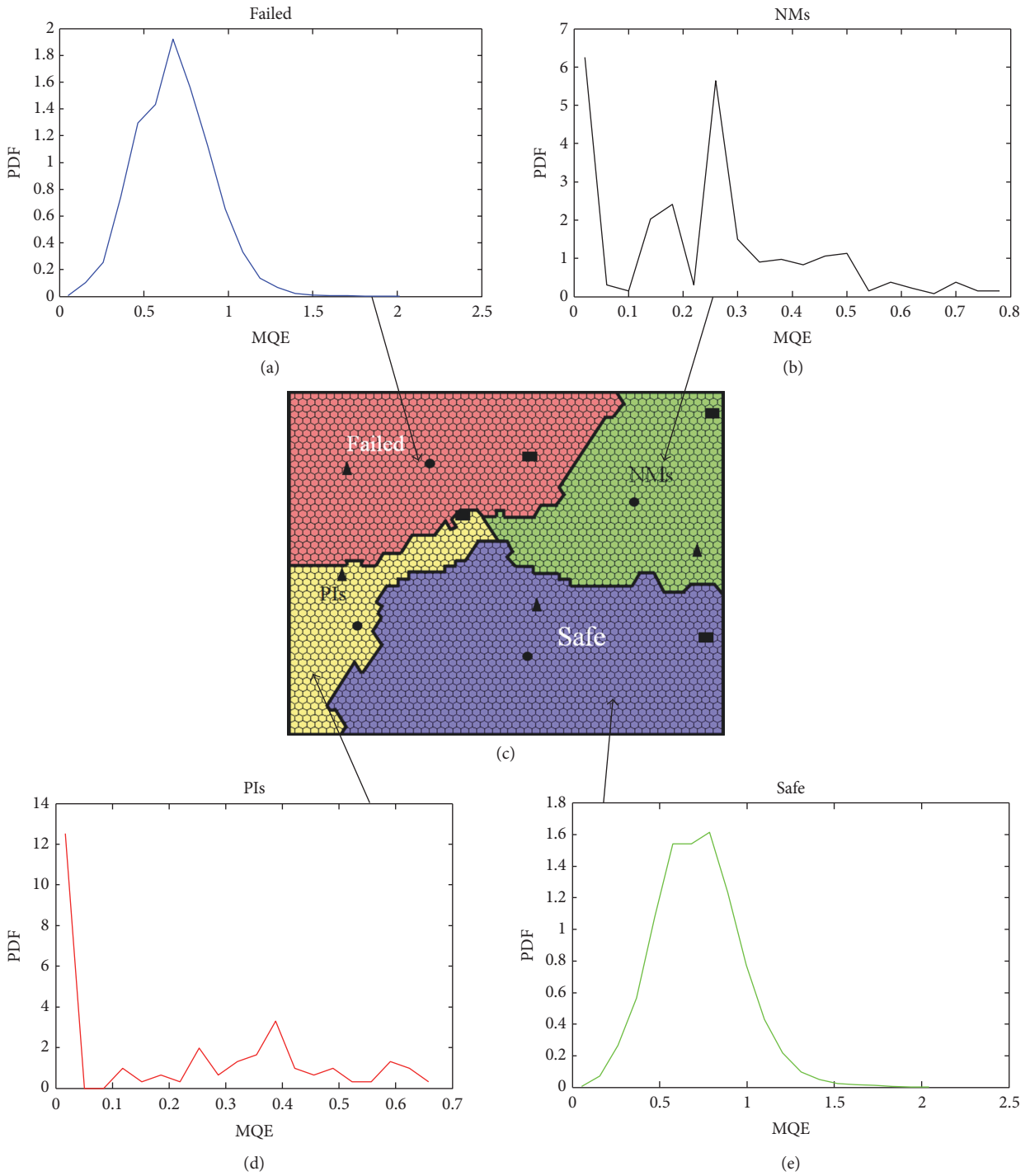

FIGURE 2: The stand-alone SSSOM (c): different shades of color indicate different classes, circles are the geometric barycenters of the classes, triangles are the minimum neurons of the classes, and rectangles are the maximum neurons of the classes. (a) PDF of the MQE for failed scenarios; (b) PDF of the MQE for NMs scenarios; (d) PDF of the MQE for PIs scenarios; (e) PDF of the MQE for safe scenarios.

SSSOM and show how, for different classes, none of these is the best and all would mutually benefit from each other, namely.

3.1.1. The Stand-Alone SSSOM. A SSSOM of $M = 3025$ neurons $C = [c_1, c_2, \ldots, c_M]$, each of which is assigned a weight vector $\overline{w}_m = [w_1, w_2, \ldots, w_d]$, is trained on the $N = 100509$ UTSG dynamic scenarios $\overline{X}$ belonging to a $d = 12$-dimensional space, say $\overline{\overline{X}} = [\overline{X}_1, \overline{X}_2, \ldots, \overline{X}_N]$, where the nth sample is $\overline{X}_n = [x_1, x_2, \ldots, x_d]$. For the sake of completeness, the training algorithm is presented in the Appendix. In particular, this SSSOM is constructed by replacing the Euclidean

distance as similarity measure between the generic scenario in input $\overline{X}_n$ and the weight $\overline{w}_m$ of the M neurons of the map, with the Manhattan distance:

$$d_{\text{Manhattan}}\left(\overline{X}_n, \overline{w}_m\right) = \sum_{k=1}^{d} \left\| \overline{X}_k - \overline{w}_k \right\|, \tag{1}$$

where $\|\cdot\|$ is the absolute value of the difference between the two vectors along the d-dimension [14]. By doing so, the MVL formalism is accommodated within the similarity assessment between data vector and neurons. The map of Figure 2(c) has been built with $t_{\text{tot}} = 15$ training epochs and a $\varepsilon(t = 0) = 0.01$ factor (for the meaning of these parameters, see the Appendix). Different shades of color represent the different $G = 4$ classes $t = [1, 2, 3, 4]$ for safe, NMs, failed, and PIs, respectively.

3.1.2. The MQE Based SSSOM. Let us consider a generic scenario $\overline{X}_n$ and a generic neuron of the map $\overline{w}_i$. As stated in the Appendix (A.5), a commonly used quality measures that can be used to determine the performance of the map is the MQE and it can be defined as in the following equation.

$$\text{MQE} = \frac{1}{N} \sum_{n=1}^{N} \left\| \overline{X}_n - \overline{w}_i \right\|, \tag{2}$$

where $\overline{w}_i$ are the weights associated with the BMU neuron c_i.

Basically, the lower the MQE of the BMU is, the more the scenario features vector is similar to its weight vector and, thus, the more the knowledge is learnt by the SSSOM. Computing the MQE for each input data and grouping them classwise, we can obtain the empirical probability density functions (PDF) referring to the distribution of the MQE for each class (Figures 2(b), 2(a), 2(e), and 2(d)). Equation (3) shows an example of computation of the MQE for a generic class g:

$$\text{MQE}_g = \frac{1}{N_g} \sum_{n=1}^{N_g} \left\| \overline{X}_{n_g} - \overline{w}_i \right\|, \tag{3}$$

where N_g is the number of scenarios belonging to the class g, $\overline{X}_{n_g}$ is a generic scenario belonging to the class g, and $\overline{w}_i$ is the weight vector of the BMU neuron in the map to which $\overline{X}_{n_g}$ is assigned. The classification of a new input to a g class with the MQE based SSSOM proceeds as follows: its MQE_g is calculated as in (2) and, then, it is assigned to the class with the larger PDF value for the calculated MQE. The rationale is that, for a particular value of MQE, the larger the PDF, the more probable the value: if for a class g, the PDF associated with a MQE value is larger than for the other classes, it is more probable that the scenario belongs to that class. For example, if the MQE of an input is equal to 1.5, we assign it to the safe class because the PDF of the safe class (Figure 2(e)) is larger than the other classes, for this value of MQE. In general, we can notice in Figure 2 that NMs and PIs classes have PDF skewed towards low values of MQE, whereas failed and safe classes have larger MQE values.

3.1.3. The Barycenter Based SSSOM. The same SSSOM trained as in the Appendix and shown in Figure 2(c) is exploited as an alternative classifier by using the geometric barycenter of each cluster as a reference for the choice of the BMU (circles in Figure 2). When a new $\overline{X}_n$ is fed to this SSSOM, we select the closest of the four barycenter neurons as the most similar neuron, where similarity is quantified based on the Manhattan distance:

$$d_{\text{Manhattan}}\left(\overline{X}_n, \overline{w}_{g_{\text{bar}}}\right) = \sum_{k=1}^{d} \left\| \overline{X}_k - \overline{w}_{g_{\text{bar}}} \right\|, \tag{4}$$

where $\overline{w}_{g_{\text{bar}}}$ is anyone of the four barycenters of the four classes. The rationale is that the geometric barycenter is most representative of the characteristics of the class.

3.1.4. The Minimum Neuron Based SSSOM. Considering again the SSSOM trained as in the Appendix and shown in Figure 2(c) for each cluster g we locate on the map the neuron with the minimum weight $\overline{w}_{g_{\min}}$ (represented in the map with a triangle in the map of Figure 2) and for the classification we assign the new vector $\overline{X}_n$ to the cluster with the minimum neuron most similar to the considered input, based on the Manhattan distance:

$$d_{\text{Manhattan}}\left(\overline{X}_n, \overline{w}_{g_{\min}}\right) = \sum_{k=1}^{d} \left\| \overline{X}_k - \overline{w}_{g_{\min}} \right\|. \tag{5}$$

The rationale is that if the vector of the features of a scenario is similar to that of neuron with the minimum weight of a specific cluster, it will be assigned to this cluster because it is very different to the neurons with minimum weight vectors of the other classes.

3.1.5. The Maximum Neuron Based SSSOM. The maximum neuron based SSSOM is complementary to the previous one in that it is based on the neuron with the maximum weights for each cluster $\overline{w}_{g_{\max}}$ represented by a rectangle in the map of Figure 2(c).

3.1.6. Classification Performance. The four classifiers of Sections 3.1.2–3.1.5 are compared to the stand-alone SSSOM of [14], on the UTSG scenario postprocessing task. The performances of the classifiers are quantified by the calculation of [28]:

(i) Precision: the larger, the better the capability of the kth classifier to not include samples of other classes in the considered gth class:

$$\text{Pr}_g = \frac{n_{gg}}{n'_g}, \tag{6}$$

where n'_g is the total number of scenarios assigned to the gth class and n_{gg} is the number of scenarios belonging to class g and correctly assigned to class g.

Table 1: MQE based SSSOM performances: precision, sensitivity, and specificity for each class.

MQE based	Safe	Failed	NMs	PIs
Precision	0.674	0.3803	0.0406	0.0083
Sensitivity	0.4699	0.475	0.491	0.6333
Specificity	0.6006	0.5674	0.9616	0.9326

Table 2: Barycenter based SSSOM performances: precision, sensitivity, and specificity for each class.

Barycenter based	Safe	Failed	NMs	PIs
Precision	0.6459	0.4816	0.0097	0.0014
Sensitivity	0.2975	0.3859	0.5934	0.3444
Specificity	0.7134	0.7678	0.7996	0.7827

Table 3: Minimum neuron based SSSOM performances: precision, sensitivity, and specificity for each class.

Minimum neuron based	Safe	Failed	NMs	PIs
Precision	0.7437	0.5418	0.0325	0.0025
Sensitivity	0.7145	0.427	0.0994	0.2667
Specificity	0.5666	0.7981	0.9902	0.9054

(ii) Sensitivity: the larger, the better the capability of the kth classifier to correctly recognize samples belonging to the gth class:

$$\mathrm{Sn}_g = \frac{n_{gg}}{n_g}, \tag{7}$$

where n_g is the total number of scenarios belonging to the gth class.

(iii) Specificity: the larger, the better the capability of each gth class of the kth classifier to reject the samples of all the others:

$$\mathrm{Sp}_g = \frac{\sum_{k=1}^{G}\left(n'_k - n_{gk}\right)}{N - n_g} \quad \text{for } k \neq g, \tag{8}$$

where n'_k is the total number of samples assigned to the kth class:

$$n'_k = \sum_{g=1}^{G} n_{gk}. \tag{9}$$

In Tables 1–5, the performances for the MQE based, the barycenter based, the minimum neuron based, the maximum neuron based, and the stand-alone SSSOMs, for each class, are reported.

For failed and PIs scenarios, the parameters of the stand-alone SSSOM (precision of 0.83 and 0.016, sensitivity of 0.773 and 0.911, and specificity of 0.911 and 0.949, respectively) are larger than for all the other classifiers. For example, for the minimum neuron based SSSOM, the precision for failed scenarios is equal to 0.5418, which is much lower than the precision obtained with the stand-alone SSSOM. It is worth noticing that this is always true for all the parameters values when dealing with failed and PIs scenarios. On the contrary, looking at the NMs and safe scenarios, we see that the other classifiers overcome the stand-alone SSSOM performances. For example, the specificity in classifying safe scenarios is higher for the maximum neuron based SSSOM than for the stand-alone SSSOM (0.955 versus 0.927), and the precision in classifying NMs is higher for the MQE based SSSOM than for the stand-alone SSSOM (0.0406 versus 0.034), and also the specificity in classifying NMs for both the MQE based SSSSOM (0.9616) and the minimum neuron based SSSOM (0.9902) is higher than the stand-alone SSSOM (0.911).

Table 4: Maximum neuron based SSSOM performances: precision, sensitivity, and specificity for each class.

Maximum neuron based	Safe	Failed	NMs	PIs
Precision	0.833	0.3864	0.0066	0.002
Sensitivity	0.1278	0.3514	0.5813	0.6222
Specificity	0.955	0.6881	0.7084	0.7167

Table 5: Stand-alone SSSOM performances: precision, sensitivity, and specificity for each class.

SSSOM [14]	Safe	Failed	NMs	PIs
Precision	0.949	0.83	0.034	0.016
Sensitivity	0.78	0.773	0.957	0.911
Specificity	0.927	0.911	0.911	0.949

In Figure 3, a 3D representation of the performance parameters values of Tables 1–5 is given for each implemented SSSOM and each scenario class: stars indicate the MQE based SSSOM values, circles the barycenter based SSSOM values, squares the minimum neuron based SSSOM values, diamonds the maximum neuron based SSSOM, and crosses the stand-alone SSSOM values.

Figure 3(a) confirms that the stand-alone SSSOM, on average, overperforms the other classifiers, except for safe and NMs classes: for these scenarios in Figures 3(b) and 3(d), respectively, a Pareto front can be identified and highlighted with a solid line for the suboptimal solutions of classifiers that do not dominate all the others with respect to all the three performance objectives. For example, we can see that the precision in classifying the NMs is higher for the MQE based SSSOM (0.0406) than for the stand-alone SSSOM (0.034) and so is the specificity in classifying NMs (0.9616 versus 0.911), but the sensitivity for the same class is higher for the stand-alone SSSOM (0.957 versus 0.491).

These results suggest the possibility of developing a general method for aggregating the multiple classifiers outputs considered as an ensemble, whose aggregation mechanism (as proposed in the following sections) would consider the local performances of the different classifiers in dealing with the different types of scenarios and automatically selecting the classifier to be used.

As a last remark, it is worth clarifying that the proposed ensemble method is designed such that it could be applied to other cases than the UTSG here presented. Therefore, even if, under this circumstances, one might argue it would be more

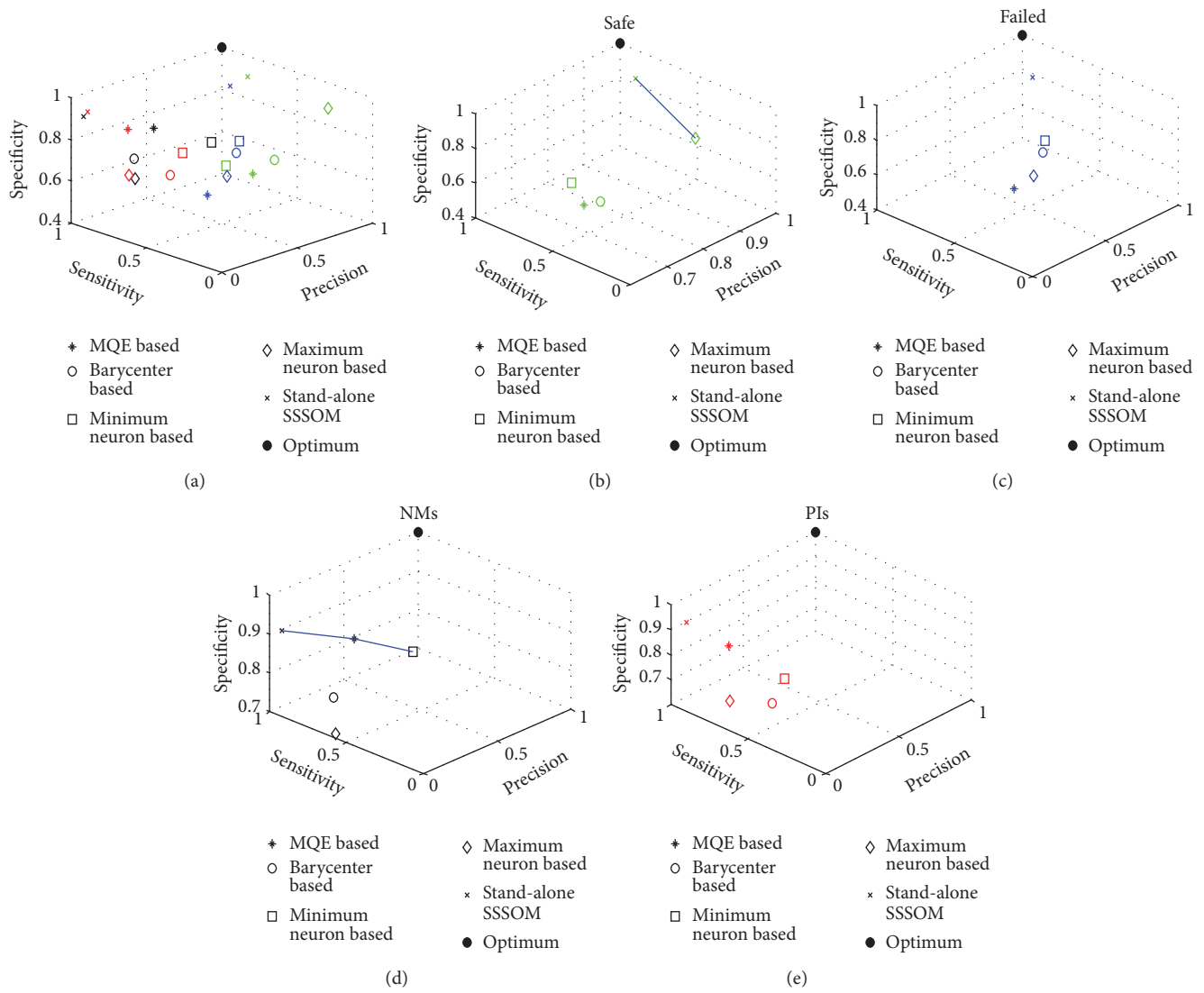

Figure 3: 3D representation of the performance parameters. Red refers to PIs, black to NMs, blue to failed, and green to safe scenarios.

convenient to rely on the stand-alone SSSOM only when dealing with failed and PIs scenarios and on the ensemble for the other classes of scenarios because (i) it is always true that the stand-alone SSSOM overcomes the other classifiers when dealing with failed and PIs scenarios and (ii) it is not when dealing with NMs and safe scenarios, not to limit the generality of the proposed ensemble method, we proceed by devising an automatic aggregation mechanism, rather than case based.

3.2. Design of the Aggregation Mechanism: The Locally Weighted Fusion. Let w_Q^k be the weight that classifier k carries in assigning scenario $\overline{X}_n$ to a class of a dataset of N scenarios to be classified:

$$w_{\overline{X}_n}^k = \frac{1}{\mathrm{me}_{\overline{X}_n}^k}, \tag{10}$$

where the Mean Error (ME) $\mathrm{me}_{\overline{X}_n}^k$ is the error that classifier k makes in classifying the scenario $\overline{X}_n$, defined as

$$\mathrm{me}_{\overline{X}_n}^k = \frac{\sum_{n=1}^{N} e_n^k}{N} \tag{11}$$

and e_n^k is the error that the classifier k commits in classifying the nth scenario whose real class is t_n (with $n = 1, \ldots, N$). In this work, the error e_n^k is computed in two different ways: being $\hat{y}_n^k = 1, \ldots, G$ the class the kth classifier assigns to $\overline{X}_n$; the first way for computing $e_{n_{(1)}}^k$ is given in the following equation:

$$e_{n_{(1)}}^k = \left\{ f(x) = \begin{cases} 0, & \text{if } \hat{y}_n^k = t_n \\ 1, & \text{if } \hat{y}_n^k \neq t_n, \end{cases} \right. \tag{12}$$

where the error is null if the estimated class $\hat{y}_n^k$ is the same as the real class t_n (where $\hat{y}_n^k = 1$ and $t_n = 1$ means that the estimated and real class of the scenario are safe, respectively, $\hat{y}_n^k = t_n = 2$ means failed, $\hat{y}_n^k = t_n = 3$ means NMs, and $\hat{y}_n^k = t_n = 4$ means PIs), whereas in the second way $e_{n_{(2)}}^k$ is calculated as the Manhattan distance between the real and the predicted class by the following equation:

$$e_{n_{(2)}}^k = \left\| \hat{y}_n^k - t_n \right\|. \tag{13}$$

Usually the error is computed by relying on a subset of N, called neighbor set of scenarios to $\overline{X}_n$ and defined as in (10):

$$P\left(\overline{X}_n\right) = \left\{ u_j \mid u_j \in N\left(\overline{X}_n\right) \right\}, \tag{14}$$

where $u_j = \langle x_{1,j}, x_{2,j}, \ldots, x_{d,j} \rangle$ is a set of d-dimensional scenarios, $N(\overline{X}_n)$ is the neighborhood of $\overline{X}_n$ that is in this work defined as a set of $N_{\overline{X}_n} = 100$ scenarios (i.e., a subset of N scenarios) whose Manhattan distance for the instance to be classified is lower than 10 (i.e., being $d = 12$, a threshold value equal to 10 means $\overline{X}_n$ and its neighbors have not to differ too much), and, thus, $j = 1, \ldots, 100$.

$$\text{dist}_{\text{Manhattan}}\left(\overline{X}_n, u_j\right) = \sum_{l=1}^{d} \left\| \overline{X}_{nl} - u_l \right\|. \tag{15}$$

In this way, the kth classifier performance is expected to be similar to the one that would be obtained with a new (unknown) scenario. A weight $w_{\overline{X}_n}^k$ can, thus, be associated with each of the individual k classifiers of the ensemble depending on its performance, as it will be shown in the next section.

4. The Proposed Ensemble Strategies

In the following, we describe the details of the implemented ensemble strategies, namely, the locally weighted ensemble of SSSOMs and the decision tree based on an ensemble of SSSOMs. These approaches rely on the five classifiers introduced in Sections 3.1.1–3.1.5 (the stand-alone SSSOM, the MQE based SSSOM, the barycenter based SSSOM, the minimum neuron based SSSOM, and the maximum neuron based SSSOM, respectively), whose classification outcomes are combined into two different ways, as we shall see in what follows.

4.1. Locally Weighted Ensemble of SSSOMs. For the locally weighted ensemble of SSSOMs, we directly apply the algorithm of the neighborhood based approach, as described in Section 3.2 to the $N = 100509$ dynamic scenarios. For each scenario to be classified we retrieve the 100 neighbors based on the considerations made before: relying on the neighborhood of each scenario we compute the classification errors (with both (12) and (13)) and, through the errors, also the weights associated. The classification outcomes of the five different trained SSSOMs are ensembled and the assignment to a class is given accounting for the different performances of these classifiers when assigning the weight (the larger the number of neighbors of the input scenario correctly classified, the lower the error, and the larger the weight and the reliability for the kth classifier). For the computation of the weight associated with each classifier k for each scenario, thus, we resort (10) and (11) where, for the nth generic scenario and the kth classifier, we calculate

$$w_{\overline{X}_n}^k = \frac{1}{\text{me}_{\overline{X}_n}^k}, \tag{16}$$

where $\overline{X}_n$ is one of the $N = 100509$ dynamic scenarios, $w_{\overline{X}_n}^k$ is the weight associated with this scenario, and $\text{me}_{\overline{X}_n}^k$ is the ME associated with this scenario and computed as in (13):

$$\text{me}_{\overline{X}_n}^k = \frac{\sum_{j=1}^{N_{\overline{X}_n}} e_j^k}{N_{\overline{X}_n}}, \tag{17}$$

where $N_{\overline{X}_n} = 100$ and e_j^k is the classification error. Once the weights are computed for all the $K = 5$ classifier, the input data $\overline{X}_n$ is assigned to the class with the larger weight $k = \arg((\max_k(w_{\overline{X}_n}^k)))$, because this is the most reliable classifier for the nth vector.

4.1.1. Training of the Locally Weighted Ensemble of SSSOMs. Table 6 shows the classification results for all scenarios and for those of NMs and PIs classes. The rows of the table report the results obtained when (12) and (13) are used for computing the classification error. We can see that, in both cases, the total number of correctly assigned scenarios (irrespectively of being safe, failed, NMs, and PIs) exploiting the locally weighted ensemble of SSSOMs increases with respect to the stand-alone SSSOM (whose results are reported in Table 7): this latter, in fact, scores a total amount of 78288 rightly assigned scenarios [14], while with the locally weighted ensemble of SSSOMs, we achieve 84141 overall correct assignments when the error is given by (12) and 81512 overall correct assignments when we resort to the Manhattan distance of (13) for computing the error.

If we, instead, focus on NMs and PIs classes, both ensembles are penalized with respect to NMs and PIs classification

Table 6: Locally weighted ensemble classification results: method using (12) and (13).

Approach	Correctly assigned		
	Total	NMs	PIs
Locally weighted ensemble by using (12)	84141	104	66
Locally weighted ensemble by using (13)	81512	308	77

Table 7: Stand-alone SSSOM classification results.

Approach	Correctly assigned		
	Total	NMs	PIs
Stand-alone SSSOM	78288	318	82

Table 8: Locally weighted ensemble percentage classification result using (12) and (13).

Approach	Correctly assigned		
	Total	NMs	PIs
Locally weighted ensemble by using (12)	83.71%	31.33%	73.33%
Locally weighted ensemble by using (13)	81.1%	92.77%	85.56%

Table 9: Locally weighted ensemble performance parameters using (12).

Ensemble (1)	Safe	Failed	NMs	PIs
Precision	0.9443	0.8162	0.0338	0.0186
Sensitivity	0.8453	0.8277	0.3133	0.7333
Specificity	0.9124	0.8958	0.9704	0.9653

Table 10: Locally weighted ensemble performance parameters using (13).

Ensemble (2)	Safe	Failed	NMs	PIs
Precision	0.9488	0.8359	0.0437	0.0195
Sensitivity	0.8266	0.782	0.9277	0.8556
Specificity	0.9216	0.9142	0.9327	0.9615

(second and third column): the stand-alone SSSOM correctly assigns 318 out of 332 NMs and 82 out of 90 PIs (as reported in Table 7). It is worth pointing out that, even if the ensembles do not correctly classify all NMs and PIs scenarios, we can consider these results satisfactory for the operational risk quantification which the classification is aiming at contributing to (i.e., the consequences of the scenario occurring and to its probability of occurrence): as already said, PIs normally are made of many component failures but because of this also have low probability of occurrence and, thus, the risk that is not accounted for due to the misclassification of PI is very low, whereas for the NMs those scenarios that are not correctly classified are classified as either safe (with no extra risk quantification being both safe and NMs leading to safe states) or failed scenarios (with a conservative overestimation of the system operational risk).

Table 8 reports the same results in terms of percentage of correct assignment.

Looking at the two locally weighted ensembles, we can say that the one based on the Manhattan distance is more effective in the assignment of NMs and PIs than the other: we see in fact that the percentage of correctly assigned NMs is 31.33% when (12) is used and 92.77% when (13) is used, whereas for PIs the percentage increases from 73.33% to 85.56% when the Manhattan distance is used. Even if (12) is used, the percentage of correct assignment is larger than when the Manhattan distance is used (83.71% versus 81.1%), since NMs and PIs are the most safety relevant classes and, thus, are those we have to guarantee to be better classified during the postprocessing of dynamic scenarios.

Furthermore, Tables 9 and 10 list the precision, sensitivity, and specificity values for the two ensembles for all the four classes. The best performances are obtained by using (13): the precision is larger using (13) than (12) for all the four classes and what we gain in terms of sensitivity in classifying NMs and PIs scenarios and specificity in classifying safe and failed scenarios justifies a negligible loss in terms of sensitivity in classifying safe and failed scenarios and specificity in classifying NMs and PIs scenarios. In fact,

(i) the specificity for NMs and PIs decreases (from 0.9704 to 0.9327 and from 0.9653 to 0.9615, respectively);

(ii) the sensitivity for safe and failed scenarios decreases (from 0.8453 to 0.8266 and from 0.8277 to 0.782, respectively).

Using (13), we gain a consistent benefit; namely,

(i) the sensitivity for NMs and PIs increases (from 0.3133 to 0.9277 and from 0.7333 to 0.8556, respectively);

(ii) the specificity for safe and failed scenarios increases (from 0.9124 to 0.9216 and from 0.8958 to 0.9142, respectively).

In conclusion, it is possible to assert that the approach based on (13) leads to superior results of classification.

Table 11: Ensemble classification results.

Approach	Correctly assigned		
	Total	NMs	PIs
Locally weighted ensemble by using (12)	1673	2	9
Locally weighted ensemble by using (13)	1599	8	11

4.1.2. Test of the Locally Weighted Ensemble of SSSOMs. We test the locally weighted ensemble of SSSOMs approach with a set of scenarios in which the time is not discretized anymore, but it is continuous. A new set of input data $\overline{\overline{X}}_{\text{test}}$ of 2000 scenarios have been generated, in which components can fail randomly between 0 and the mission time of 4000 (s). Then, the trained classifiers are used to classify $\overline{\overline{X}}_{\text{test}}$. In Table 11, the results of the test conducted on the locally weighted ensemble of SSSOMs are reported.

Within the set of the 2000 input data, there are 8 NMs and 11 PIs. We can see in Table 11 that the test classification results confirm the considerations made for the training. The ensemble based on the Manhattan distance is more efficient in the assignment of NMs and PIs than the other, even if the total correct assignment is larger for (12) than for (13): using (9) all the NMs and PIs scenarios are correctly classified, whereas only 2 NMs and 9 PIs are assigned to the right class, if (12) is used.

4.2. Decision Tree Based on an Ensemble of SSSOMs. The decision tree based classification scheme chooses one classifier (or ensemble of classifiers), depending on its local performance of the test scenario considered (see Figure 4). In this way, a test scenario with similar characteristics is treated

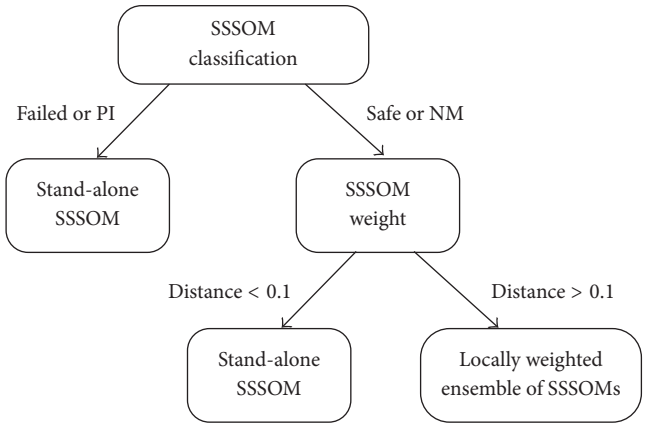

Figure 4: Decision tree based classifier.

with the same classifier (or ensemble of classifiers) for which the branch is most effective. A decision tree [29] is defined as a classification procedure that recursively partitions the scenarios into smaller subdivisions on the basis of a set of rules defined at each branch. The tree is composed of a root node (formed from all the input scenarios), a set of internal nodes (splits), and a set of terminal nodes (leaves), in which the scenarios are divided per groups with common characteristics. In this framework, the scenarios are classified by sequentially subdividing them according to the decision framework defined by the tree, and a class label is assigned to each scenario according to the leaf node into which the scenario falls.

In the following, we describe the decision rules used for the construction of the tree structure:

(i) If the stand-alone SSSOM classifies the scenario as failed or PI, then we accept such classification. The rationale is that as shown before in Figure 3 and Tables 1–5, the stand-alone SSSOM is the best classifier of failed and PI scenarios.

If the stand-alone SSSOM classifies the scenario as safe or NM, we consider multiple classifiers, as the three objectives (precision, sensitivity, and specificity) show a Pareto front where two (or three) classifiers can provide equally plausible classification results (see Tables 1–5). In this case, we aggregate the classification outcomes of the classifiers on the Pareto front, weighting them proportionally to the inverse distance between the point in the space of the three objectives for the specific classifier and the optimum, represented by the point $[1, 1, 1]$ in the same space: the lower the distance, the larger the associated weight.

If the stand-alone SSSOM classifies the scenario as safe or NM

(i) the scenario is assigned to the class given by the stand-alone SSSOM with a distance < 0.1;

(ii) if the stand-alone distance is larger than 0.1, we resort to the locally weighted ensemble of SSSOMs (Section 4.1).

Table 12: Decision tree classification result using (12) and (13).

Approach	Correctly assigned		
	Total	NMs	PIs
Decision tree using (12)	82520	104	82
Decision tree using (13)	80192	306	82

It is worth mentioning that the threshold distance is chosen equal to 0.1 (i.e., a reasonably low error), because in such way we would rely on the stand-alone SSSOM (for safe or NMs) only if the assignment can be done with large confidence (otherwise, we resort to the locally weighted ensemble of SSSOM).

4.2.1. Training of Decision Tree Based on an Ensemble of SSSOMs. Table 12 shows the classification results for all scenarios and for those of NMs and PIs classes. As for the locally weighted ensemble of SSSOMs, we focus, in particular, on these two classes, because these are the two more relevant for quantifying the operational risk of the system. Both the decision trees based on (12) and (13) overperform the classification of the stand-alone SSSOM (whose results are reported in Table 7): the approach that uses (12) scores 82520 correctly assigned scenarios, whereas the approach based on (13) scores 80192 correct assignments. As for the locally weighted ensemble, both classifications are penalized with respect to NMs if compared with the stand-alone SSSOM, because the first approach based on (12) correctly classifies 104 NMs, whereas the approach that uses (13) correctly classifies 306 NMs. The number of correctly classified PIs corresponds to the number of the stand-alone SSSOM classified rightly, because in the classification algorithm of the decision tree, the classification of the PIs leans on the stand-alone SSSOM only.

Table 13 reports the classification result of the decision tree in terms of percentages.

Looking at the two decision trees, the one based on the Manhattan distance is more effective in the classification of NMs than the other: we see in fact that the percentage of correctly assigned NMs is 31.33% when (12) is used and 92.17% when (13) is used. The PIs have the same percentage of correctly classified scenarios because they are assigned with the same classifier (the stand-alone SSSOM). Using (12), the percentage of correct assignment is larger than when the Manhattan distance is used (82.1% versus 79.79%), but we know that NMs and PIs are the most safety relevant classes and, thus, are those we have to guarantee to be better classify during the postprocessing of dynamic scenarios.

Tables 14 and 15 list the precision, sensitivity, and specificity values for the two decision trees for all the four classes. The best performances are obtained by using (13): the precision is larger using (13) than (12) for all the four classes and, as for the locally weighted ensembles, what we lose in terms of sensitivity in classifying safe and failed scenarios and specificity in classifying NMs is justified by the growth in terms of sensitivity in classifying NMs and specificity in classifying safe and failed scenarios. In fact,

(i) the specificity for NMs decreases (from 0.9707 to 0.934);

Table 13: Decision tree percentage classification results using (12) and (13).

Approach	Correctly assigned		
	Total	NMs	PIs
Decision tree using (12)	82.1%	31.33%	91.11%
Decision tree using (13)	79.79%	92.17%	91.11%

Table 14: Decision tree performance parameters using (12).

Decision tree (1)	Safe	Failed	NMs	PIs
Precision	0.9473	0.8087	0.0343	0.0158
Sensitivity	0.8216	0.8275	0.3133	0.911
Specificity	0.9197	0.891	0.9707	0.9492

Table 15: Decision tree performance parameters using (13).

Decision tree (2)	Safe	Failed	NMs	PIs
Precision	0.9498	0.8266	0.0442	0.0159
Sensitivity	0.8058	0.7824	0.9217	0.911
Specificity	0.9251	0.9083	0.934	0.9496

Table 16: Decision tree classification results.

Approach	Correctly assigned		
	Total	NMs	PIs
Decision tree using (12)	1544	8	11
Decision tree using (13)	1537	8	11

(ii) the sensitivity for safe and failed scenarios decreases (from 0.8216 to 0.8058 and from 0.8275 to 0.7824, respectively).

Using (13),

(i) sensitivity for the NMs increases (from 0.3133 to 0.9217),

(ii) specificity for the safe and failed scenarios increases (from 0.9197 to 0.9251 and from 0.891 to 0.9083, respectively).

Also in this case we can say that the Manhattan approach for the computation of the classification error (as in (13)) is preferable.

4.2.2. Test of Decision Tree Based on an Ensemble of SSSOMs. We test the decision tree based on an ensemble of SSSOMs approach with the same set of scenarios in which the time is continuous, as in Section 4.1.2. Table 16 shows the classification results.

In this case, the two approaches (using (12) and (13)) lead to the same results of classification of NMs (as regards PIs, it is obvious that the classification leads to the same number of correctly classified scenarios, because only the stand-alone SSSOM is used for the class assignment): this can be due to the small number of scenarios used for the test phase, if compared with the total number of scenarios ($N = 100509$). Similarly to other cases, the total number of correctly classified vectors is greater using (12) that scores 1544 correctly classified scenarios, than using (13) which obtains 1537 right classifications. But since the NMs and the PIs are the two classes for which we need to guarantee a good classification for the quantification of the operational risk of the system and considering the results of the training phase, we can say that the approach that uses (13) is better than the approach based on (12).

Both the locally weighted ensemble of SSSOMs and the decision tree based on an ensemble of SSSOMs show better performances when the classification error is computed with (13), rather than with (12). This is due to the fact that (12) smooths down the large errors of the classifier (e.g., using (12) an error computed for a safe scenario (labeled with 1) misclassified as NMs (labeled with 2) is equal to an error computed for a safe scenario (labeled with 1) assigned to PIs class (labeled with 4)).

In conclusion, we can assert that, between the decision tree based on an ensemble of SSSOMs and the locally weighted ensemble of SSSOMs (both based on (13)), the best compromise choice falls on the former approach: indeed, it can guarantee a very large overall correct assignment rate (80192 out of 100509), a large number of NMs correctly classified (306 out of 332 real NMs), and the largest number of PIs assigned to the right class (82 out of 90 real PIs), whereas the weighted ensemble of SSSOMs correctly classifies only 77 real PIs.

5. Conclusions

The postprocessing of IDPSA accidental scenarios of a dynamic system is a fundamental task for retrieving safety relevant information for the system operation and maintenance. In practice the task can be challenged by the combinatorial explosion of the scenarios generated due to the dynamic dependence of components failure events and the consideration of timing and magnitudes of failure events in the accidental scenarios generation.

In this paper, for UTSG scenario generation, a SIMULINK dynamic simulation model has been used, within a MVL scheme that describes the different component operational states, and has presented two alternatives strategies of scenario classification (i.e., the locally weighed ensemble of SSSOMs and the decision tree based on an ensemble of SSSOMs) with the twofold purpose of (1) leveraging the classifiers complementary characteristics and (2) boosting overall classification accuracy. In general terms, it has been shown (in Section 3.1.6) that ensemble approach can benefit from each independent classifier by capitalizing (within either a locally weighted strategy or a decision tree based strategy) their complementary characteristics. The methodology highlights the need of taking into account different classifiers to recover information that would have been lost if neglected. Examples have been provided with respect to the capability of the ensemble of SSSOM to improve the classification of NMs and PIs, especially, accepting that safe scenarios that do not cause a negative contribution to system operational risk quantification might be misclassified.

Despite this, the overall classification accuracy improvement has been demonstrated (from 78288 correct assignments of the stand-alone SSSOM [14] to more than 80000 correct assignments with the proposed SSSOM ensemble based classification methods).

Appendix

The SSSOM Training

The SSSOM is a powerful visualization tool for high-dimensional data, which are orderly mapped into a low-dimensional structure that usually consists of a 2D regular grid of hexagonal nodes (neurons) that can vary from a few dozen up to several thousand [30]. The SOM concentrates all the information contained in a set of N input samples $\overline{X}$ belonging to a d-dimensional space, say $\overline{\overline{X}} = [\overline{X}_1, \overline{X}_2, \ldots, \overline{X}_N]$, where the nth sample is $\overline{X}_n = [x_1, x_2, \ldots, x_d]$, utilizing a set of M neurons, $C = [c_1, c_2, \ldots, c_M]$ (where $M < N$), each of which is associated with a weight vector $\overline{w}_m = [w_1, w_2, \ldots, w_d]$ (also called "prototypes" or "codebook" vectors) [31, 32]. The weights of each neuron $\overline{w}_m$ are usually randomly initialized between 0 and 1; then, their values are adjusted during a training phase, so as to be able to optimally represent $\overline{\overline{X}}$ and its structure. The unsupervised SOM training mainly consists in three phases: competition, cooperation, and adaptation [33]. Briefly the training phase entails a stimulus (i.e., any of the input samples $\overline{X}_n$ from $\overline{\overline{X}}$) to be presented to the network and the neurons competition so as to identify which is the Best Matching Unit (BMU) that is the most similar to $\overline{X}_n$ in terms of its weight values. Then, a subset of the neighborhood neurons to the BMU are modified by a neighborhood function. Figuratively, the region around the BMU is stretched towards the stimulus. As a result, the neurons on the grid become ordered: neighboring neurons tend to have similar weight vectors.

Competitive Process. The SOM is trained iteratively: for each training step t, one sample vector $\overline{X}_n$ is chosen randomly from the N available input data set $\overline{\overline{X}}$ and the distance between it and all the weight vectors of the SOM is calculated using some distance measure. The neuron c_i whose weight vector $\overline{w}_i$ is closest to the input vector $\overline{X}_n$ is called Best Matching Unit (BMU):

$$c_i = \arg\left\{\min\left\{\left\|\overline{X}_n - \overline{w}_m\right\|\right\}\right\} \quad \text{with } m = 1, \ldots, M, \tag{A.1}$$

where c_i is the BMU and $\|\cdot\|$ is the distance measure (typically Euclidean, but also Binary [Appiah et al., 2009] or, as original in this work, Manhattan).

Cooperation Process. Once c_i is found, its weights vector $\overline{w}_i$ is updated proportionally to the difference between $\overline{w}_i$ and the values of $\overline{X}_n$, accounting also for the characteristics of the neighborhing neurons of the BMU (i.e., BMU and neighbors tightly cooperate to form a specific pattern on the lattice) [11, 34].

Adaptation Process. The tuning function that updates $\overline{w}_i$ is

$$\begin{aligned}\overline{w}_i(t+1) &= \overline{w}_i(t) \\ &\quad + \alpha(t)\left(1 - \frac{d_{mi}}{d_{\max}+1}\right)\left[\overline{X}_n(t) - \overline{w}_i(t)\right],\end{aligned} \tag{A.2}$$

where α is the learning rate, $d_{\max}$ is the size of the neighborhood radius, which decreases during the training phase, and d_{mi} is the topological distance defined as the number of neurons that separates the considered mth neuron and the winning neuron c_i. The learning rate changes during the training phase, as in the following equation [31]:

$$\alpha(t) = \left(\alpha^{\text{start}} - \alpha^{\text{final}}\right)\left(1 - \frac{t}{t_{\text{tot}}}\right) + \alpha^{\text{final}}, \tag{A.3}$$

where t_{tot} is the total number of training epochs and α^{start} and α^{final} are the learning rate at the beginning and end of the training, usually in [0.1, 0.9] and in $[0, \alpha^{\text{start}}]$, respectively [31]. The training is usually performed in two phases: in the first phase, relatively large initial learning rate α^{start} and neighborhood radius $d_{\max}$ are used; in the second phase both learning rate and neighborhood radius are small right from the beginning. This procedure corresponds to first tuning the SOM approximately to the same space as the input data and, then, fine-tuning the map.

An additional input parameter to be set is the number of neurons M composing the map: this is usually set equal to

$$M = 5 \cdot \sqrt{N}. \tag{A.4}$$

Since different parameters and initializations give rise to different maps, it is important to know whether the map has properly adapted itself to the training data [35]. Two commonly used quality measures that can be used to determine the quality of the map and help choosing suitable learning parameters and map sizes are the MQE and the Topographic Error (TE).

MQE is a measure of how good the map can fit the input data, and the best map is expected to yield the smallest average quantization error between the BMU $\overline{w}_i$ and the input data $\overline{X}_n$. MQE is calculated with the following equation:

$$\text{MQE} = \frac{1}{N}\sum_{n=1}^{N}\left\|\overline{X}_n - \overline{w}_i\right\|, \tag{A.5}$$

where N is the number of the input data used to train the map. Practically, the lower the MQE, the better the map.

TE measures how well the topology is preserved by the map. Unlike the MQE, it considers the structure of the map. For each input data, the distance of the BMU and the second BMU (the second weight vector closer to the input data) on the map is considered; if the nodes are not neighbors, then, the topology is not preserved. TE is computed with the following equation:

$$\text{TE} = \frac{1}{N}\sum_{k=1} N \cdot u\left(\overline{X}_k\right), \tag{A.6}$$

where N is the number of input data used to train the map and $u(\overline{X}_k)$ is 1 if the first and second BMU of $\overline{X}_k$ are not direct neighbors of each other, or $u(\overline{X}_k)$ is 0, otherwise.

In supervised training we consider again a set of N input data $\overline{\overline{X}} = [\overline{X}_1, \overline{X}_2, \ldots, \overline{X}_N]$, M neurons $C = [c_1, c_2, \ldots, c_M]$, and weight vectors $\overline{w}_m = [w_1, w_2, \ldots, w_d]$ associated with them. This time we add in input $\overline{Y} = [y_1, y_2, \ldots, y_N]$, called class vector and representing the G classes of each nth input data. The training is still based on the three phases of competition, cooperation, and adaptation but in SSSOM the algorithm differs from the USOM one, since the fused similarity measure is based on weighted combination distances between an object (vector) $\overline{X}_n$ and all units in the X map ($S(\overline{X}, X\text{map})$) and the distances between the corresponding output object y_n and the units in the Y map ($S(\overline{Y}, Y\text{map})$). By $S_{\text{Fused}}(n, m)$ a common winning unit for both maps is determined:

$$S_{\text{Fused}}(n, m) = \varepsilon(t)\, S\left(\overline{X}_n, X\text{map}_m\right) + (1 - \varepsilon(t))\, S\left(\overline{Y}_n, Y\text{map}_m\right). \tag{A.7}$$

The location of the minimum of the above function is the common winning unit c_i and $\varepsilon(t)$ regulates the relative weight between similarities in the X and Y maps, and it is still dependent on the number of training epochs, decreasing linearly during the training.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] P. E. Labeau, C. Smidts, and S. Swaminathan, "Dynamic reliability: towards an integrated platform for probabilistic risk assessment," *Reliability Engineering & System Safety*, vol. 68, no. 3, pp. 219–254, 2000.

[2] T. Aldemir, "A survey of dynamic methodologies for probabilistic safety assessment of nuclear power plants," *Annals of Nuclear Energy*, vol. 52, pp. 113–124, 2013.

[3] E. Zio, "Integrated deterministic and probabilistic safety assessment: concepts, challenges, research directions," *Nuclear Engineering and Design*, vol. 280, pp. 413–419, 2014.

[4] F. Di Maio, M. Vagnoli, and E. Zio, "Risk-based clustering for near misses identification in integrated deterministic and probabilistic safety analysis," *Science and Technology of Nuclear Installations*, vol. 2015, Article ID 693891, 29 pages, 2015.

[5] W. V. Quine, "The problem of simplifying truth functions," *The American Mathematical Monthly*, vol. 59, pp. 521–531, 1952.

[6] E. Zio and F. D. Maio, "Processing dynamic scenarios from a reliability analysis of a nuclear power plant digital instrumentation and control system," *Annals of Nuclear Energy*, vol. 36, no. 9, pp. 1386–1399, 2009.

[7] D. Mandelli, A. Yilmaz, T. Aldemir, K. Metzroth, and R. Denning, "Scenario clustering and dynamic probabilistic risk assessment," *Reliability Engineering & System Safety*, vol. 115, pp. 146–160, 2013.

[8] S. Galushin and P. Kudinov, "An approach to grouping and classification of scenarios in integrated deterministic-probabilistic safety analysis," in *Proceedings of the 12th International Probabilistic Safety Assessment and Management Conference, (PSAM '14)*, Honolulu, Hawaii, USA, June 2014.

[9] F. Di Maio, S. Baronchelli, and E. Zio, "A computational framework for prime implicants identification in non-coherent dynamic systems," *Risk Analysis*, vol. 35, no. 1, pp. 142–156, 2015.

[10] F. Di Maio, S. Baronchelli, and E. Zio, "A visual interactive method for prime implicants identification," *IEEE Transactions on Reliability*, vol. 64, no. 2, pp. 539–549, 2014.

[11] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[12] S. Wu and T. W. S. Chow, "Induction machine fault detection using SOM-based RBF neural networks," *IEEE Transactions on Industrial Electronics*, vol. 51, no. 1, pp. 183–194, 2004.

[13] H. Yu, F. Khan, and V. Garaniya, "Risk-based fault detection using Self-Organizing Map," *Reliability Engineering & System Safety*, vol. 139, pp. 82–96, 2015.

[14] F. Di Maio, R. Rossetti, and E. Zio, "Local fusion of an ensemble of semi supervised self organizing maps for a safety analysis post-processing of accidental scenarios," in *Proceedings of the International Topical Meeting on Probabilistic Safety Assessment and Analysis (PSA '17)*, Pittsburgh, Pa, USA, September 2017.

[15] F. Di Maio, S. Baronchelli, and E. Zio, "Hierarchical differential evolution for minimal cut sets identification: application to nuclear safety systems," *European Journal of Operational Research*, vol. 238, no. 2, pp. 645–652, 2014.

[16] P. P. Bonissone, F. Xue, and R. Subbu, "Fast meta-models for local fusion of multiple predictive models," *Applied Soft Computing*, vol. 11, no. 2, pp. 1529–1539, 2011.

[17] L. Lam and C. Suen, "A theoretical analysis of the application of majority voting to pattern recognition," in *Proceedings of the 12th International Conference on Pattern Recognition and Computer Vision*, pp. 418–420, Jerusalem, Israel.

[18] L. Xu, A. Krzyzak, and C. Y. Suen, "Methods of combining multiple classifiers and their applications to handwriting recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 3, pp. 418–435, 1992.

[19] P. Baraldi, A. Cammi, F. Mangili, and E. E. Zio, "Local fusion of an ensemble of models for the reconstruction of faulty signals," *IEEE Transactions on Nuclear Science*, vol. 57, no. 2, pp. 793–806, 2010.

[20] L. Kuncheva, J. C. Bezdek, and M. A. Sutton, "On combining multiple classifiers by fuzzy templates," in *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS '98)*, pp. 193–197, IEEE, Pensacola Beach, Fla, USA, August 1998.

[21] W.-T. Chen, P. D. Gader, and H. Shi, "Improved dynamic-programming-based handwritten word recognition using optimal order statistics," in *Proceedings of the Statistical and Stochastic Methods in Image Processing II*, pp. 246–256, San Diego, Calif, USA, July 1997.

[22] A. Verikas, A. Lipnickas, K. Malmqvist, M. Bacauskiene, and A. Gelzinis, "Soft combination of neural classifiers: A comparative study," *Pattern Recognition Letters*, vol. 20, no. 4, pp. 429–444, 1999.

[23] K. Woods, W. Philip 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.

[24] J. F. Aubry, G. Babykina, A. Barros et al., "Project APPRODYN: APPROches de la fiabilité DYNamique pour modéliser des systèmes critiques," Tech. Rep., Collaboration CRAN, EDF R&D, INRIACQFD, UTT-ICD, 2012.

[25] L. I. Kuncheva, "Switching between selection and fusion in combining classifiers: An experiment," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 32, no. 2, pp. 146–156, 2002.

[26] K. Tumer and J. Ghosh, "Error correlation and error reduction in ensemble classifiers," *Connection Science*, vol. 8, no. 3-4, pp. 385–404, 1996.

[27] F. Roli, G. Giacinto, and G. Vernazza, "Methods for designing multiple classifier systems," in *Proceedings of the Multi Classifier Systems (MCS '01)*, vol. 2096 of *Lecture Notes in Computer Science*, pp. 78–87, Springer, 2001.

[28] D. Ballabio, V. Consonni, and R. Todeschini, "The Kohonen and CP-ANN toolbox: a collection of MATLAB modules for self organizing maps and counterpropagation artificial neural networks," *Chemometrics and Intelligent Laboratory Systems*, vol. 98, no. 2, pp. 115–122, 2009.

[29] M. A. Friedl and C. E. Brodley, "Decision tree classification of land cover from remotely sensed data," *Remote Sensing of Environment*, vol. 61, no. 3, pp. 399–409, 1997.

[30] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1-3, pp. 1–6, 1998.

[31] D. Ballabio and M. Vasighi, "A MATLAB toolbox for self organizing maps and supervised neural network learning strategies," *Chemometrics and Intelligent Laboratory Systems*, vol. 118, pp. 24–32, 2012.

[32] C. A. Astudillo and B. J. Oommen, "Self-organizing maps whose topologies can be learned with adaptive binary search trees using conditional rotations," *Pattern Recognition*, vol. 47, no. 1, pp. 96–113, 2014.

[33] Z. Fei, S. Ticlin, and H. Tao, "Fault diagnosis of motor bearing using self-organizing maps," in *Proceedings of the 8th International Conference on Electrical Machines nd Systems (ICEMS '05)*, vol. 3, pp. 2411–2414, 2005.

[34] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas, "SOM toolbox for Matlab 5," Report A57, Helsinhi University of Technology, 2000.

[35] S. Gabrielsson and S. Gabrielsson, *The use of Self-Organizing Maps in Recommender Systems; A survey of the Recommender Systems field and a presentation of a state of the art highly interactive visual movie recommender system [Uppsala Master's Thesis in Computer Science]*, Department of Information Technology at the Division of Computer Systems, Computer Science at Uppsala University, 2006.