A Data Mining Framework for Online Dynamic Security Assessment: Decision Trees, Boosting, and Complexity Analysis

Miao He, Junshan Zhang, and Vijay Vittal
School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287
E-mail:{Miao.He, Junshan.Zhang, Vijay.Vittal}@asu.edu

***Abstract*—Online dynamic security assessment provides the real-time situational awareness for assessing the impact of various N-k contingencies, so that appropriate preventive/corrective controls could be armed in a timely fashion. This task is challenging due to the large number of possible contingencies, the massive scale of power systems, and the multi-scale dynamics that occur under varying operating conditions.**

In this study, a data mining framework for online dynamic security assessment using decision trees and a boosting technique is developed, with the following multi-stage processing. 1) In the offline training stage, classifiers consisting of multiple simple decision trees are built based on a given collection of training data, and an iterative algorithm is used to "boost" the accuracy of the classifiers. 2) In the near real-time update stage, the simple decision trees together with their voting weights are updated when new data are available, enabling a smooth tracking of the changes of decision regions. 3) In the online DSA stage, real-time phasor measurements are used to locate the current operating condition into a decision region and obtain timely security decisions. The clustering of contingencies and data preprocessing via dimension reduction of the attributes are also discussed. Numerical testing based on a practical power system demonstrates that the proposed approach works well under a variety of realistic operating conditions.

I. Introduction

Many large-scale fault events in power systems, which led to severe and cascading blackouts, have been attributed to a lack of "situational awareness," that is formally defined as "understanding the current environment and being able to accurately anticipate future problems to enable effective actions" [1]. For instance, the 2005 Houston blackout could have been prevented if the significant phasor angle differences were understood; and indeed, during the 2008 Hurricane Gustav event, effective corrective controls, based on the real-time measurement of system frequency, were implemented [2]. There is clearly an urgent need to enhance the situational awareness by developing low-complexity algorithms for online dynamic security assessment.

Dynamic security assessment (DSA) is an analysis tool that can provide system operators with important information such as voltage, thermal, and transient stability under various probable contingencies. With the real-time or near real-time measurements collected by phasor measurement units (PMUs), *online DSA* can produce accurate decisions for current or impending operating conditions (OCs). Recently, several efforts have been directed towards cost-effective online DSA schemes [3]–[6]. However, it remains a challenging task due to the computational complexity incurred by the large size of the contingency list and the massive scale of power systems. First, the combinatorial possibilities of $N-k$ $(k=1,2,\cdots)$ contingencies makes it intractable to perform detailed analysis (e.g., power flow analysis and time domain simulations) for all contingencies. In practice, contingency screening schemes (see [6] and reference therein) are used to select the *active* contingencies that are likely to cause instability, so that detailed analysis is performed only on those active contingencies. However, the number of active contingencies can still be very large (possibly over thousands for a regional power system [6]). Another challenge for online DSA is the high computational complexity of detailed analysis in processing the high-dimensional measurement data.

To tackle the aforementioned challenges of online DSA, the decision regions for the active contingencies are first intensively studied offline. Then, security decisions are obtained simply by locating the current OC to a specific region. Towards characterizing the decision regions, data mining tools (e.g., decision trees (DTs) [7]) can effectively find the attributes and thresholds that are critical to make accurate decisions, provided that a knowledge base relevant to the current OC is used in offline studies. In practice, a knowledge base can be built offline, through the detailed analysis of the predicted OCs generated from day-ahead load forecast and unit commitments, recent OCs, and anticipated OCs generated from short-term load forecast and dispatch.

A. Related Work

The efficacy of DT based approaches have been verified through several recent studies on practical power systems [3]–[5]. DTs were originally introduced into DSA by [3]. In the three-stage scheme proposed in [3], a DT is first trained offline using a knowledge base, and then, both the DT and the knowledge base are periodically updated, i.e., if the trained DT fails to classify new cases correctly, the new data are incorporated into the knowledge base and a new DT is thus rebuilt from scratch. Then, the updated DT is used to produce prompt decisions in online DSA. In [4], more attributes, including voltage magnitudes, phase angle differences, current flows, and square of voltage magnitudes, are used for DT training and help in making a more accurate

978-1-4577-2159-5/12/$31.00 ©2011 IEEE

prediction. The improvement in accuracy by using multiple optimal DTs and corrective DTs is also discussed. In recent work [5], four important post-contingency security issues, i.e., voltage magnitude violation, thermal limit violation, voltage stability, and transient stability under $N-k$ contingencies, are studied using DTs, where the knowledge base is enriched by new OCs generated by linear interpolation. Furthermore, a group of randomly changed OCs are used to test the robustness of the trained DTs.

B. Summary of Main Results

In this study, a data mining framework for online dynamic security is proposed, as shown in Fig.1, with the basic idea summarized as follows. In the offline training stage, a group of N_{OC} predicted OCs are generated for each period T_1 in the next day, based on load forecast and generation schedules. Then, through offline studies on the predicted OCs for a given contingency list $\mathcal{C}$, a knowledge based consisting $N = N_{OC} \times |\mathcal{C}|$ training data is used to train the classifiers, where $|\mathcal{C}|$ is the number of active contingencies. In the near real-time update stage, new data are incorporated into the classifier to refine the decision regions as needed, e.g., when the day-ahead prediction turns out to be inaccurate and new stressed conditions are expected to occur. These new data are created by using past and anticipated OCs, together with new active contingencies. Through the previous two stages, the decision regions for the OCs of the period T_1 can be accurately characterized by the classifiers. In the online DSA stage, the PMU measurements of the critical attributes are collected for each period T_2, and security decisions are obtained by locating current OC to a decision region. Generally, T_1 is at the scale of hours, and the timescale of T_2 can be on the same order as that of PMU measurements.

The proposed scheme differs from those of existing works [3]–[5] in the following aspects. 1) The classifier is obtained through boosting multiple simple DTs instead of a single fully-grown DT. 2) The simple DTs are gracefully updated by incorporating new cases one at a time, whereas rebuilding DTs is used in [3]–[5]. 3) Training data are assigned with different sets of *data weights* for training the simple DTs. 4) The DTs and the knowledge base are updated only when the new cases are misclassified in [3]–[5]. However, it is observed that when handling noisy training data, the cases correctly classified can also contain valuable information and enhance the accuracy of the classifier. Therefore, all the new cases are incorporated into the classifier and the knowledge base. 5) The active contingencies are partitioned into different clusters, and a classifier is built for each cluster.

The clustering of active contingencies is adopted mainly for two reasons: 1) the complexity of DTs when splitting over the index of the active contingencies is exponential in the number of them, and 2) it is observed that the impact of some contingencies is captured by the same subset of attributes. For each cluster of the active contingencies, a classifier is obtained via boosting simple DTs. Specifically, in the offline training stage, the simple DTs are sequentially trained using training

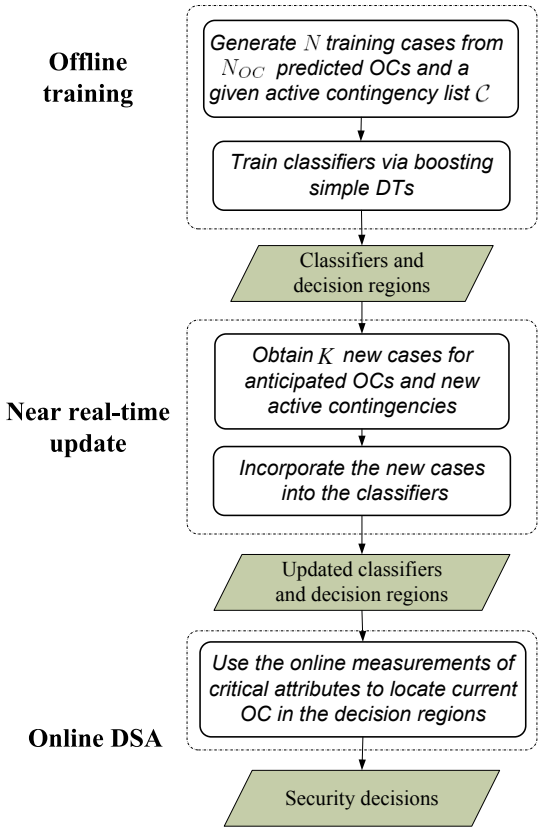

Fig. 1. A data mining framework for online DSA

data with adaptive weights. The misclassified cases by the previous simple DTs are reassigned with higher weights, so that correct decisions can be obtained by subsequent simple DTs for those misclassified cases. For the classifier, by choosing proper voting weights for the simple decision trees, its classification error can be bounded above. In the near real-time update stage, when the classifier obtained so far misclassifies a new case, a higher weight is assigned to the new case to update the subsequent simple DTs, so that the final classifier can smoothly track the change of decision regions. Compared to existing works [3]–[5], the proposed scheme can mitigate the overfitting of the classifiers and produce more accurate decisions. Further, new data corresponding to new OCs or new active contingencies can be incorporated with much lower complexity.

The rest of the paper is organized as follows. A brief introduction to DTs and their application to DSA is presented in Section II, and the possible overfitting of DTs in online DSA is also commented on. In Section III, a classifier via boosting simple DTs is proposed. Specifically, the iterative algorithms for both classifier training and updating are presented, and the computational complexity is also investigated in detail. The clustering of active contingencies and dimension reduction for online DSA is discussed in Section IV. Numerical results are presented in Section V. Finally, Section VI concludes this paper.

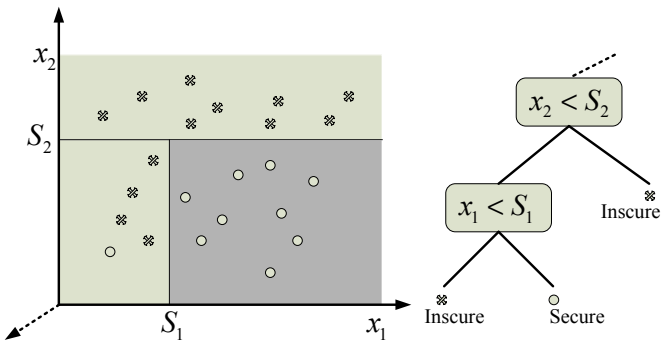

Fig. 2. A subtree of a trained DT: leaf nodes and the corresponding decision regions

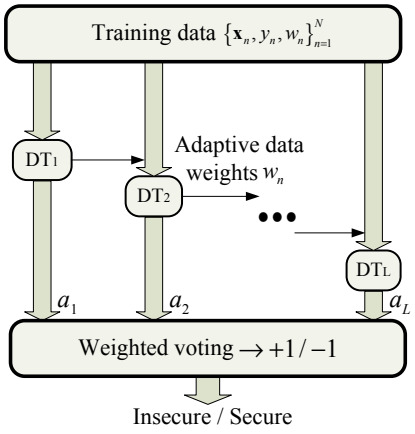

Fig. 3. Classifier via boosting simple DTs

II. Background on Decision Trees

Among various machine learning methods, DTs have good interpretability [7], in the sense that the decision regions generated by DTs can be characterized by several critical attributes. For a DT, only a few tests on the critical attributes are necessary to make a decision. Therefore, DTs can be a good choice for building the classifier of online DSA. In this study, a *Decision tree* is a tree-structured model that maps an observation on the *attributes* $\mathbf{x} = (x_1, x_2, \cdots, x_P)^T$ to a predicted value $\hat{y} \in \{\pm 1\}$ [7]. In a DT, each internal node tests an attribute and decides which child node to drop the observation into, and each leaf node is assigned with a predicted value. Consequently, the path from the root to a leaf node specifies a *decision region* in the attribute space corresponding to that leaf, as illustrated in Fig. 2. Given a collection of training data $\{\mathbf{x}_n, y_n\}_{n=1}^N$, drawn from an unknown distribution $\mathbf{P}$ of the attribute vector $\mathbf{X}$ and the corresponding decision Y, the objective of *decision tree learning* is to find a DT that can fit the training data, and accurately predict the decisions for new cases. Practical state-of-the-art DT learning algorithms are often based on greedy search. For example, in CART [7], the DT fully grows by recursively splitting the given set of training data, and choosing the attributes and tests with the minimum costs, until some predefined stopping criterion (e.g., the minimum node size is 10) is satisfied. Typical metrics for the cost of splitting include classification error, Gini index, etc. [7].

Computational complexity and accuracy are two major metrics when the classifiers of online DSA are designed. For DSA, the attribute vector $\mathbf{X}$ consists of a large number of numerical attributes and one categorical attribute, and the binary decision Y represents the security decision of an OC when a hypothetical active contingency occurs ($Y = +1$ denotes the *insecure* case). Specifically, the numerical attributes include the voltage magnitudes at buses, active/reactive power flows, and phasor angles; and the index of the active contingencies is considered as the categorical attribute. In practical DT learning algorithms [7], when a node spits over an numerical attribute, the N observations of that attribute are first sorted, and then the optimal threshold can be found in one pass over the N observations. If the node splits over a categorical attribute, the number of possible splitting is exponential in the number of different values of the N observations. In this scenario, the clustering of active contingencies can be adopted.

For online DSA, one potential drawback of using a single DT is the proneness to overfit noisy training data, as pointed out in [7]. Since a DT grows by greedy search, for each splitting, it focuses on only a subset of the training data, and thus, may neglect some global features. Fig. 2 illustrates an example of overfitting in the decision region $\{x_2 \leq S_2, x_1 \leq S_1\}$. In particular, for online DSA, this can happen when the cases with wrong decisions (the points in circle) overwhelms a local decision region. Apparently, this results in an unnecessary splitting, and thus a false decision region. In order to mitigate the overfitting of a fully-grown DT and improve the accuracy of the classifier of online DSA, an approach different from existing works [3]–[5] is considered. In the proposed scheme, the classifier for online DSA is obtained via *boosting* simple DTs. Roughly speaking, the term "boosting" [8] refers to the process of training multiple simple DTs sequentially using adaptive data weights, and combining the simple DTs with proper voting weights to boost the accuracy of the classifier. In this study, *simple decision trees* are defined as a collection $\mathcal{H}$ of DTs uniformly with a smaller height J than a single fully-grown DT. Generally, an individual simple DT might have a relatively lower prediction accuracy, but can be less prone to overfitting compared to a fully-grown DT [9]. Further, the classifiers obtained from boosting algorithms are shown to be quite resistant to overfitting [10]. Therefore, boosting simple DTs can produce more accurate classifiers than a single DT.

III. Classifier via Boosting Simple DTs

A primary objective is to find a function $F_L : \mathcal{X} \rightarrow R$ as a weighted voting of L simple DTs, i.e.,

$$F_L(x) = \sum_{l=1}^{L} a_l h_l(x), \tag{1}$$

where $a_l \in \mathcal{R}^+$ is the voting weight[1] of simple DT $h_l \in \mathcal{H}$, $l = 1, 2, \cdots, L$, and the corresponding binary classifier $H : \mathcal{X} \to \{\pm 1\}$, obtained by:

$$H_L(x) = \text{sign}(F_L(\mathbf{x})),$$

so that the classifier H could fit the given data.

A. Offline Training

In order to quantify the performance of the classifier in fitting the training data $\{\mathbf{x}_n, y_n\}_{n=1}^N$, first define the cost function of F_L as follows:

$$\hat{C}_N(F_L) = \frac{1}{N}\sum_{n=1}^{N} \log_2(1 + \mathrm{e}^{-y_n F_L(\mathbf{x_n})}). \tag{2}$$

Then, the offline training problem is formulated as follows:

$$\mathcal{P}_F : \min_{\substack{h_1,\cdots,h_L \in \mathcal{H} \\ a_1,\cdots,a_L \in \mathcal{R}^+}} \hat{C}_N(\sum_{l=1}^{L} a_l h_l). \tag{3}$$

The above cost function is chosen for the following reasons: 1) The cost function lies strictly above the training error, i.e.,

$$\hat{\mathrm{e}}(F_L) \triangleq \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}_{\{y_n \neq H_L(\mathbf{x}_n)\}} \leq \hat{C}_N(F_L).$$

Therefore, the error of classifier can be bounded above by minimizing the cost function $\hat{C}_N(F_L)$. Moreover, different from the indicator functions, the cost function is differentiable w.r.t. F_L, making it possible to solve $\mathcal{P}_F$ in a gradient descent manner. 2) The term $yF_L(\mathbf{x})$ is named as the "margin" of F_L on the case $\{\mathbf{x}, y\}$ [7]. Essentially, the cases with negative margins, i.e., both the miss-detection cases (when $F_L(\mathbf{x}_n) < 0$ but $y_n = +1$), and false alarm cases (when $F_L(\mathbf{x}_n) > 0$ but $y_n = -1$), are penalized. 3) The cost function is shown to be more robust to noise, compared to the exponential cost functions in other boosting algorithms (e.g. Adaboost) [7].

It can be seen from (2) that the cost function is convex and lower-bounded. This fact motivates the usage of a multi-stage optimization strategy, similar to the line search approach [11]. Specifically, initially with F_0 as a zero function, a simple DT $h_l \in \mathcal{H}$ is identified together with a voting weight $a_l \in \mathcal{R}^+$, and added to F_{l-1}, i.e.,

$$F_l = F_{l-1} + a_l h_l, \tag{4}$$

iteratively for $l = 1, 2, \cdots, L$.

1) Simple DTs: $h_l = [- \bigtriangledown \hat{C}_N(F_{l-1})]_{\mathcal{H}}$ is chosen as the descent direction of $\hat{C}_N(F_{l-1})$, where

$$\bigtriangledown \hat{C}_N(F_{l-1})(\mathbf{x}) = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \mathbf{1}_{\{\mathbf{x}=\mathbf{x_n}\}}}{1 + \mathrm{e}^{y_n F_{l-1}(\mathbf{x_n})}}, \tag{5}$$

and $[\cdot]_{\mathcal{H}}$ denotes the projection onto $\mathcal{H}$. Following the approach in [8], h_l is thus the simple DT which maximizes the inner product $\langle h, -\bigtriangledown \hat{C}_N(F_{l-1})\rangle$. Then, the simple DT h_l can be obtained by solving the following problem:

$$\mathcal{P}_{DT}^{(l)} : \min_{h_l \in \mathcal{H}} \frac{1}{N}\sum_{n=1}^{N} w_n^{(l)} \mathbf{1}_{\{y_n \neq h_l(\mathbf{x}_n)\}}, \tag{6}$$

with

$$w_n^{(l)} = \frac{1}{1 + \mathrm{e}^{y_n F_{l-1}(\mathbf{x}_n)}}.$$

For $\mathcal{P}_{DT}^{(l)}$, the objective is exactly to find the simple DT that has the least weighted classification error on the training data. Then, based on (6), the optimal simple DT h_l can be trained through recursive splitting.

2) Voting Weights: The voting weight of simple DT h_l is obtained by solving the following problem:

$$\mathcal{P}_a^{(l)} : a_l = \operatorname*{argmin}_{a \in \mathcal{R}^+} \hat{C}_N(F_{l-1} + a h_l).$$

Define $g_l(a) \triangleq \hat{C}_N(F_{l-1} + a h_l)$. Under the condition that h_l is a descent direction of $\hat{C}_N(F_{l-1})$, it is easy to see $g_l'(0) < 0$. Further, since $g_l''(a) > 0$ holds for $a \in \mathcal{R}^+$, $g_l(a)$ has a unique minimum in R^+.

Proposition 3.1: The classifier via boosting L simple DTs is obtained by solving $\mathcal{P}_{DT}^{(l)}$ in (6) for $l = 1, \cdots, L$, and the data weights and voting weights are given by

$$\begin{cases} w_n^{(l)} = \frac{1}{1+\mathrm{e}^{y_n F_{l-1}(\mathbf{x}_n)}} & n = 1, \cdots, N \\ a_l = \operatorname*{argmin}_{a \in \mathcal{R}^+} g_l(a) & l = 1, \cdots, L \end{cases}. \tag{7}$$

According to (7), the cases with smaller margins $y_n F_{l-1}(\mathbf{x}_n)$ are reassigned with higher weights when used for training the simple DT h_l. Therefore, for the simple DT h_l, it is trained so that correct decisions could be obtained for those cases which are misclassified by previous simple DTs. And for the classifier, by choosing a proper voting weight a_l for h_l, it tries to reduce the overall classification error. In this sense, the classifier generated by the boosting process can fit the training data better as more simple DTs are used. Boosting simple DTs algorithm relates to the multiple optimal DTs algorithm in [4]. Both algorithms aim to enhance the accuracy by using multiple DTs. The major differences are: 1) for boosting, the simple DTs are trained sequentially, in a gradient descent manner, while DTs are usually trained independently in [4]; 2) a weighted voting is adopted in boosting and the voting weights are chosen so that the cost function is minimized, while a majority voting is used in [4]. Therefore, the proposed algorithm can produce much simpler DTs, and meanwhile, guarantee the accuracy of the classifier.

3) Convergence of F_L: According to (2), it is clear that $\hat{C}_N$ is lower-bounded and convex in F_L. Further, as suggested in (5), the gradient $\bigtriangledown \hat{C}_N(F_L)$ is Lipschitz continuous with a Lipschitz constant no greater than $1/N$. Then, based on [8], either the above iterative process obtains a global optimum F_N^* of $\hat{C}_N$ at stage L^* with $\langle \bigtriangledown \hat{C}_N(F_{L^*-1}), h_{L^*}\rangle = 0$, or F_L

[1]Since $-h \in \mathcal{H}$ for any $h \in \mathcal{H}$, we restrict a_l to $\mathcal{R}^+$

converges to F_N^* as $L \to \infty$. Further, as $N \to \infty$, $\hat{C}_N(F_L)$ converges to

$$C(F_L) = \sum_{\{\mathbf{x},y\} \in \mathcal{X} \times \{\pm 1\}} \mathbf{P}_{\mathbf{X}Y}(\mathbf{x}, y) \log_2(1 + \mathrm{e}^{-yF_L(\mathbf{x})}),$$

which achieves the minimum at

$$F^*(\mathbf{x}) = \ln \frac{Pr(Y = +1|\mathbf{X} = \mathbf{x})}{Pr(Y = -1|\mathbf{X} = \mathbf{x})}.$$

It can be seen from (2) $\hat{C}_N$ is continuous for $N \in \mathcal{N}$. Therefore, F^* is a limit point of F_N^*, and H_N^*, i.e., sign(F_N^*), converges to the maximum a posterior probability (MAP) decision rule of $(\mathbf{X}, Y)$.

The process for boosting simple DTs in the offline training stage is summarized in Algorithm 1. Due to the tradeoff between the accuracy of H_L and the number of simple DTs used for boosting, a stopping criterion is employed in Algorithm 1. Another reason for using a stopping criterion, as pointed out in [10], is to avoid overfitting. Based on the line search strategy [11], the stopping criterion is that the Euclidean norm of $[\bigtriangledown \hat{C}_N(F_{l-1})]_{\mathcal{H}}$ is below some level ε_0, i.e., when the gain in the reduction of the cost $\hat{C}_N$ by adding a new simple DT is limited.

Algorithm 1 Train a classifier via boosting simple DTs

1: **Input:** Training data $\{\mathbf{x}_n, y_n\}_{n=1}^N$, $\varepsilon_0 \in (0, 1)$
2: **Initialization:** $F_0 = \mathbf{0}$, $\varepsilon = 1$, $l = 1$.
3: **while** $\varepsilon \geq \varepsilon_0^2$ **do**
4: Compute the data weights according to (III-A1)
5: Find the optimal simple DT h_l by solving (6)
6: $\varepsilon \leftarrow \frac{1}{N} \sum_{n=1}^N w_n^{(l)} y_n h_l(\mathbf{x}_n)$
7: Find the voting weight a_l according to (7)
8: $F_l \leftarrow F_{l-1} + a_l h_l$
9: $l \leftarrow l + 1$
10: **end while**

B. Near Real-time Update

Suppose that L simple DTs are obtained based on the training data, and in the near real-time update stage, K new case are used to update the classifier one at a time. The algorithm for updating the classifier can be developed in a similar way to the offline training. Specifically, for the k-th new case $\{\mathbf{x}_{N+k}, y_{N+k}\}$, $k = 1,2,\cdots,K$, the classifier is updated by incorporating $\{\mathbf{x}_{N+k}, y_{N+k}\}$ with weight $w_{N+k}^{(l)}$ into the simple DT h_l, computing the new voting weight a_l, and then adding it to the classifier as in (4), iteratively for $l = 1, 2, \cdots, L$.

1) Incorporating a New Case into Simple DTs: The $N + k$ data weights at the l-th stage are first computed according to (7). Then, the simple DT h_l is updated using only the new case $\{\mathbf{x}_{N+k}, y_{N+k}\}$ with weight $w_{N+k}^{(l)}$. Incremental tree induction algorithms has been extensively studied (see [12] and the references therein), which can continually modify the DT obtained so far by incorporating new cases without rebuilding a new DT. Simply put, in the algorithm proposed by [12], the DT remains unchanged if a new case is classified correctly. Otherwise, the nodes along the path from the root to the leaf where the new case falls into are iteratively updated, so that optimal test is adopted for each internal node. In this study, the idea of [12] is adopted for updating simple decisions with two modifications. 1) When the leaf node which the misclassified new case falls into has depth J, it stays unchanged. 2) When a new case corresponding to new active contingencies encounters a test on the index of active contingencies, the node and all the nodes below are forced to be updated, since the new case cannot be located in any leaf node in this scenario. When updating the simple DTs, the metric for splitting is the weighted classification error as defined in (6). By using this algorithm, the updated simple DTs can have comparable accuracy to the one rebuilt from scratch, and the average computational cost of updating a well-trained simple DT using a new case is much lower than rebuilding, and is shown to be independent of the number of past cases [12]. The above two features guarantee that the overall computational complexity of online DSA would be greatly reduced, without degrading the accuracy of the classifier, if the classifier and the simple DTs are updated in the above fashion.

2) Voting Weights: After the simple DT h_l is updated, the corresponding voting weight a_l is re-computed as in (7), by using

$$g_l(a) \triangleq \frac{1}{N+k} \sum_{n=1}^{N+k} \log_2(1 + \mathrm{e}^{-y_n(F_{l-1}(\mathbf{x}_n) + a h_l(\mathbf{x}_n))}). \quad (8)$$

The process of updating the classifier is summarized in Algorithm 2. Based on (III-A1), it is clear that the weights of the past $N + k - 1$ cases can change slightly after F_{l-1} is updated. As a result, the updated simple DTs, except h_1, might not be the optimal solutions to (6), because the objective function, i.e., the weighted classification error of the $N + k$ cases, has changed by taking into account the new k cases. Therefore, an extra step is used to test the condition that the updated simple DT h_l is a descent direction of the cost function, i.e., $\langle \bigtriangledown \hat{C}_{N+k}(F_{l-1}), h_l \rangle < 0$. Specifically, if $\sum_{n=1}^{N+k} w_n^{(l)} y_n h_l(\mathbf{x}_n) < 0$, the simple DT is modified simply by sign inversion. Because, under this condition, $-h_l$ is a descent direction of the cost function $\hat{C}_{N+k}(F_L)$. This scenario can happen when new cases with the opposite decisions overwhelm the decision regions of h_l, e.g., when a lot of new cases with correct decisions are added into the region $\{x_2 \leq S_2, x_1 \leq S_1\}$ in Fig. 2.

C. Complexity Analysis

The reduction of the computational complexity is of paramount significance to online DSA. In what follows, the complexity of the algorithms for classifier updating and offline training is discussed for the proposed scheme, with a comparison to those of existing works. For the scheme based on a single DT [5], let J' be the height of the single DT, and K' be the number of updates. For fair comparison, assume that each

Algorithm 2 Update the classifier

Input: A new case $\{\mathbf{x}_{N+k}, y_{N+k}\}$
Initialization: $F_0 = \mathbf{0}$
for $l = 1$ to L **do**
  Compute the data weights according to (III-A1)
  Incorporate $\{\mathbf{x}_{N+k}, y_{N+k}\}$ with weight $w_{N+k}^{(l)}$ into simple DT h_l.
  Compute $\varepsilon = \frac{1}{N+k}\sum_{n=1}^{N+k} w_n^{(l)} y_n h_l(\mathbf{x}_n)$.
  if $\varepsilon < 0$ **then**
    $h_l \leftarrow -h_l$
  end if
  Find the voting weight a_l according to (7)
  $F_l \leftarrow F_{l-1} + a_l h_l$
end for

simple DT is also updated for K' times on average. Further, assume the single DT and simple DTs are well-balanced.

After clustering, the values of the categorical attribute can be very small. It is clear that N can be very large. Therefore, the complexity for a splitting over the categorical attribute can be much lower than that over a numerical attributes. For simplicity, only the $P-1$ numerical attributes are considered in the complexity analysis. Specifically, a splitting over all the $P-1$ attributes with N training data has complexity $\mathcal{O}(PN\log N)$. Further, under the assumption that the DT is well balanced, i.e., the total number of training data at each level of the tree is $\mathcal{O}(N)$, the complexity for building a single DT with height J' is $\mathcal{O}(J'PN\log N)$. Therefore, the complexity of rebuilding DTs with height J' using an incremental knowledge base for K' times is $\mathcal{O}(K'J'PN\log N)$. In the proposed algorithm, the cost of incorporating a new case into the simple DTs is independent of the number of past cases. Whenever an update is made, the cost is incurred by searching for the optimal test of the $P-1$ attributes, for at most J levels.Therefore, the complexity for updating a single DT is $\mathcal{O}(JP)$. Therefore, the overall complexity for updating L simple DTs for K' times is $\mathcal{O}(K'LJP)$. Further, the complexity of evaluating $F_L(\mathbf{x})$ for a case $\mathbf{x}$ is independent of P and L. This is because the classifier uses only the critical attributes, and the function F_L itself can be reduced into a simple function. Based on (7), the complexity of updating the data weights and voting weights are both $\mathcal{O}(KLN)$. Based on the above analysis, the complexity of offline training can be obtained in a similar way. The above analysis is summarized in Table I.

In order to explore the relationship between J and J', it is observed that the capability of a DT to fit the training data depends on the number of the decision regions it generates. For boosting simple DTs, since each simple DT is built by using different sets of data weights, thus the subsets of critical attributes of the simple DTs can be regarded as disjoint. Under the assumption that the simple DTs are well balanced, the total number of the decision regions generated by boosting L simple DTs is $\mathcal{O}(2^{LJ})$. Therefore, $\mathcal{O}(LJ) = \mathcal{O}(J')$ when both classifiers fit the training data. Then, a conclusion can be drawn as follows.

TABLE I
COMPLEXITY OF UPDATE AND TRAINING ALGORITHMS

	A single DT [5]	Boosting simple DTs
Update	$\mathcal{O}(K'J'PN\log N)$	$\mathcal{O}(K'LJP)$
Training	$\mathcal{O}(J'PN\log N)$	$\mathcal{O}(LJPN\log N)$

Proposition 3.2: The proposed scheme reduces the computational complexity in near real-time update stage by a factor of $\mathcal{O}(N\log N)$, compared to the algorithm in [5].

IV. DISCUSSION ON PREPROCESSING

A. Clustering of Active Contingencies

Define $Z_c \triangleq (y_{c,1}, y_{c,2}, \cdots, y_{c,N_{OC}})$ as the decision sequence of an active contingency $c \in \mathcal{C}$ as, where $y_{c,n}$ is the security decision of the n-th OC for contingency c. Intuitively, the Hamming distance between Z_{c_1} and Z_{c_2}, i.e.,

$$d(Z_{c_1}, Z_{c_2}) = \sum_{n=1}^{N_{OC}} \mathbf{1}_{\{Z_{c_1,n} \neq Z_{c_2,n}\}}, \tag{9}$$

quantifies the dissimilarity between the impact of contingencies c_1 and c_2 on the same set of OCs. Particularly, $d(Z_{c_1}, Z_{c_2}) = 0$ implies that contingency c_1 and c_2 have the same impact on all the predicted OCs, and thus can be characterized by the same decision regions. With this insight, a clustering algorithm, which utilizes the Hamming distance of the decision sequences, is proposed based on the k-modes algorithm [13]. In Algorithm 3, K represents the target number of clusters, and D_0 is the size of largest cluster that DTs would handle.

Algorithm 3 Clustering of active contingencies

Input: The number of target clusters K, a threshold D_0, and the decision sequences Z_c $(c = 1, 2, \cdots, |\mathcal{C}|)$
Initialization: Randomly choose K contingencies c'_1, $\cdots$, c'_K, set the corresponding decision sequences as the centroid Z'_k of the clusters $\mathcal{C}_k$ $(k = 1, 2, \cdots, K)$.
for $c = 1$ to $|\mathcal{C}|$ **do**
  Choose $k' = \underset{k}{\operatorname{argmin}}\, d(Z_c, Z'_k)$
  if $|\mathcal{C}_{k'}| \geq D_0$ **then**
    $K \leftarrow K + 1$
    Set Z_c as the centroid of cluster $\mathcal{C}_K$
  else
    $\mathcal{C}_{k'} \leftarrow \mathcal{C}_{k'} \cup \{c\}$
    for $n = 1$ to N_{OC} **do**
      $Z'_{k',n} = \mathbf{1}_{\{\frac{1}{|\mathcal{C}_{k'}|}\sum_{c \in \mathcal{C}_{k'}} Z_{c,n} \geq 1/2\}}$
    end for
  end if
end for
$\mathcal{C}_k \leftarrow \Phi$ for $k = 1, 2, \cdots, K$
Repeat steps 3-15 until the centroids of clusters do not change

When a new active contingency is to be incorporated in the near real-time update stage, the security decisions of the new contingency on all the N_{OC} predicted OCs are first obtained. Then, based on the Hamming distance between the decision sequence of the new active contingency and the centroids of the clusters, the new active contingency is either assigned to an existing cluster or used to create a new cluster. For the latter case, a new classifier is trained for this new active contingency using the N_{OC} cases.

B. Dimension Reduction

Among the numerical attributes, i.e., voltage magnitudes, active/reactive power flows, and phasor angles, there are two kinds of correlations: 1) the attributes measured at the same bus or transmission line are correlated; 2) the attributes measured at adjacent buses and transmission lines are correlated, as described by the Kirchhoffs laws. Therefore, the high dimensional measurement data of the attributes reside in a much lower space, compared to the original attribute space. Based on this observation, principal component analysis (PCA) is utilized to reduce the number of numerical attributes used for classifier training.

Define $\bar{\mathbf{x}}'_n \triangleq \mathbf{x}'_n - \frac{1}{N_{OC}} \sum_{m=1}^{N_{OC}} \mathbf{x}'_m$, and $\mathbf{A} = (\bar{\mathbf{x}}'_1, \bar{\mathbf{x}}'_2, \cdots, \bar{\mathbf{x}}'_{N_{OC}})$, where $\mathbf{x}'_n$ is the measurements of the $P-1$ numerical attributes of the n-th OC, $n = 1, 2, \cdots, N_{OC}$. One challenge in performing PCA is that the number of numerical attributes $P-1$ can be so large that the computational complexity of the eigen-analysis of the matrix $\mathbf{A}\mathbf{A}^T$ is too high. Generally, $N_{OC} \leq P-1$ holds for large-scale power systems. Under this condition, the non-zero eigenvalues and the corresponding eigenvectors of $\mathbf{A}\mathbf{A}^T$ can be obtained from those of $\mathbf{A}^T\mathbf{A}$ [14]. Specifically, let λ_i and u_i $(i = 1, \cdots, N_{OC})$ be the eigenvalues of $\mathbf{A}^T\mathbf{A}$, organized in a descending order, and the corresponding normalized eigenvectors. Then, the M eigenvectors corresponding to the M largest eigenvalues are chosen as the principal components, based on the ratio of the cumulative sum of those M eigenvalues to that of all the eigenvalues. Therefore, for any data $\mathbf{x}$, the corresponding measurements of the M transformed attributes is $\tilde{\mathbf{x}}' = \mathbf{V}^T\mathbf{x}'$, where $\mathbf{V} = (\mathbf{A}u_1, \cdots, \mathbf{A}u_M)$.

V. Numerical Studies

A. A Testing Power System

The system model used in the numerical studies, which consists of over 600 buses, 700 transmission lines, and 100 generators, is a part of the Western Electricity Coordinating Council (WECC) system. In this study, the active contingencies are chosen from a contingency list consisting of 89 $N-1$ and 183 $N-k$ $(k = 2, 3, \cdots)$ contingencies. The OCs used in the numerical studies are based on the realistic data of power flows, bus load and generations, which were recorded every 15 minutes during a representative summer day. The overall load profile is illustrated in Fig. 4. Based on the variation of the aggregated load, each period T_1 is chosen to span 4 hours, and the peak load period 12:00PM-16:00PM is investigated.

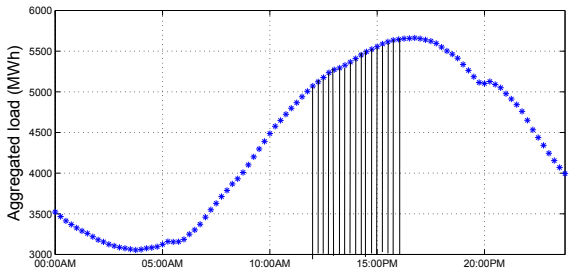

Fig. 4. Aggregated load profile of a day

1) OC Generation: Based on the 17 realistic OCs, more OCs are generated for training and testing purposes. Following the approach [5], 64 OCs are generated by interpolating the bus loads and generator outputs. Then for each of those 81 OCs, 7 new OCs are generated by randomly changing the bus loads within 90% to 110% of their original values. All the generated OCs are tested to make sure they confirm the pre-contingency secure criteria. Among them, 405 are used for training, and the other 162 are used for testing.

2) Active Contingency Selection: To demonstrate the efficacy of the proposed scheme, the thermal stability issue is studied in detail, so that any potential overload could be detected during this peak load period. DSATools™ [15] are used as the offline security assessment software. After power flow analysis [2] of all the 272 contingencies, 86 of them which cause thermal limit violations are chosen as the active contingencies.

B. Performance Evaluation

1) Dimension Reduction: Based on the placement of PMUs in the testing system, 270 quantities which are directly measured by the PMUs are chosen as the original numerical attributes. For PCA, the number of transformed attributes is chosen so that the ratio of the cumulative sum of the corresponding eigenvalues to that of all the eigenvalues is over 99.99%. Then, $M = 24$ transformed attributes are chosen for classifier training, as illustrated in Fig. (5).

2) Clustering, Offline Training and Testing: After clustering, the 86 active contingencies are partitioned into 6 clusters. For each cluster, a classifier is built through boosting simple DTs with a height $J = 3$. The scheme based on a single DT [5] is used as a benchmark for accuracy analysis. For fair comparison, the number of simple DTs are chosen so that LJ is not greater than the height of the single DT before pruning. To evaluate the robustness to noise, 5% of the 405 OCs used for training are randomly assigned with wrong decisions.

The result of testing are summarized in Table. II. For each cluster, boosting simple DTs outperforms the single DT in testing accuracy, which verifies that the classifier via boosting simple DTs is more robust to noisy data. Roughly speaking, the classification error is reduced by half, if boosting simple

[2] In practice, screening schemes can be utilized instead

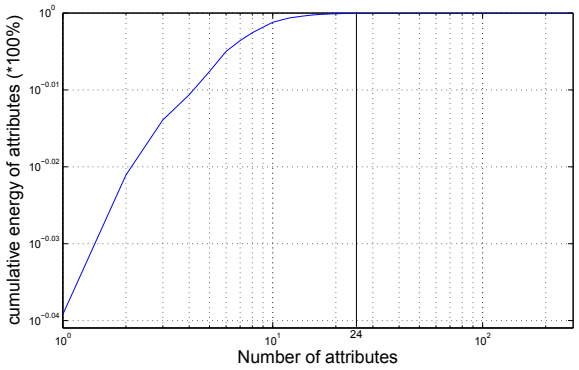

Fig. 5. Dimension reduction of the attributes

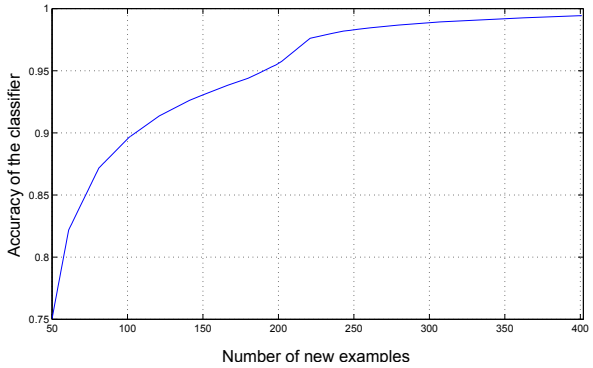

Fig. 6. Testing on the classifier updated by new cases

TABLE II
CLASSIFICATION ERROR ON TESTING DATA

Cluster	No. of contingencies	Boosting simple DTs (%)	A single DT(%)
1	9	1.68	3.54
2	12	3.49	6.75
3	14	3.64	4.61
4	16	2.72	6.17
5	17	4.26	6.42
6	18	4.70	5.94

DTs is utilized instead of a single DT. For $N-k$ contingencies, this improvement in decision accuracy can have a significant impact on the reliability and economics of practical power systems.

3) Incorporating a New Active Contingency: In this study, a classifier is first trained for 8 contingencies of cluster 1. Then, the 402 new cases of the 9th contingency of cluster 1 are used to update the classifier one at a time. For each update, the accuracy of the classifier is evaluated using the 162 testing cases. The classifier achieves an accuracy over 60%, even when the number of new cases is small, as illustrated in Fig. 6, which verifies that the contingencies of the same cluster are correlated and captured by the same subset of numerical attributes. As more cases are used for update, the classifier performs better on the new contingency.

VI. CONCLUSION

In this study, a data-mining framework for online DSA is developed. The proposed scheme focuses on the accuracy and the complexity reduction of the classifier built from boosting simple DTs, and the improvement over existing works is demonstrated through complexity analysis and numerical studies on a practical power system. The proposed scheme is tested for the post-contingency thermal stability issue, and can be used for other security issues, including voltage limit violation, transient stability, etc.

REFERENCES

[1] Pacific Northwest National Laboratory, "What is situational awareness," Jan 2011. http://eioc.pnnl.gov/research/sitawareness.stm.

[2] J. T. Reilly, "Real-time data for system protection and system restoration," in *DIMACS Workshop on Algorithmic Decision Theory for the Smart Grid*, Oct 2010.

[3] K. Sun, S. Likhate, V. Vittal, V. Kolluri, and S. Mandal, "An online dynamic security assessment scheme using phasor measurements and decision trees," *IEEE Transactions on Power Systems*, vol. 22, pp. 1935–1943, Nov 2007.

[4] R. Diao, K. Sun, V. Vittal, R. O'Keefe, M. Richardson, N. Bhatt, D. Stradford, and S. Sarawgi, "Decision tree-based online voltage security assessment using PMU measurements," *IEEE Transactions on Power Systems*, vol. 24, pp. 832–839, May 2009.

[5] R. Diao, V. Vittal, and N. Logic, "Design of a real-time security assessment tool for situational awareness enhancement in modern power systems," *IEEE Transactions on Power Systems*, vol. 25, pp. 957–965, May 2010.

[6] H. D. Chiang, J. Tong, and Y. Tada, "On-line transient stability screening of 14,000-bus models using TEPCO-BCU: Evaluations and methods," in *Power and Energy Society General Meeting, 2010 IEEE*, pp. 1–8, July 2010.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics, Springer-Verlag, 2008.

[8] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean, "Boosting algorithms as gradient descent," in *Neural Information Processing Systems*, pp. 512–518, 1999.

[9] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, pp. 119–139, 1997.

[10] P. Buhlmann and T. Hothorn, "Boosting algorithms: Regularization, prediction and model fitting," *Statistical Science*, vol. 22, pp. 477–505, 2005.

[11] M. J. Box, D. Davies, and W. H. Swann, *Non-Linear optimisation Techniques*. Oliver and Boyd, 1969.

[12] P. E. Utgoff, N. C. Berkman, and J. A. Clouse, "Decision tree induction based on efficient tree restructuring," *Mach. Learn.*, vol. 29, pp. 5–44, Oct 1997.

[13] Z. Huang, "Extensions to the k-means algorithm for clustering large data sets with categorical values," *Data Mining and Knowledge Discovery*, vol. 2, pp. 283–304, 1998.

[14] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *Computer Vision and Pattern Recognition*, pp. 586–591, Jun 1991.

[15] Powertech Labs, "DSATools: Dynamic Security Assessment Software." http://www.dsatools.com.