Data Sniffing - Monitoring of Machine Learning for Online Adaptive Systems *

Yan Liu yanliu@csee.wvu.edu
Tim Menzies tim@menzies.com
Bojan Cukic cukic@csee.wvu.edu

Lane Department of Computer Science & Electrical Engineering
West Virginia University
Morgantown, WV 26505, U. S. A.

Abstract

Adaptive systems are systems whose function evolves while adapting to current environmental conditions. Due to the real-time adaptation, newly learned data have a significant impact on system behavior. When online adaptation is included in system control, anomalies could cause abrupt loss of system functionality and possibly result in a failure.

In this paper we present a framework for reasoning about the online adaptation problem. We describe a machine learning tool that sniffs data and detects anomalies before they are passed to the adaptive components for learning. Anomaly detection is based on distance computation. An algorithm for framework evaluation as well as sample implementation and empirical results are discussed. The method we propose is simple and reasonably effective, thus it can be easily adopted for testing.

1 Introduction

Adaptive systems can be applied to domains where autonomy is significant or environmental conditions tend to be unpredictable. Usually, the aim of an adaptive system is to perform appropriately under both foreseen and unforeseen circumstances through adaptation. If the adaptation occurs after deployment of the system, the system is called online adaptive system. In recent years, online adaptive systems have been proposed and implemented in flight control with the expectation to react promptly to unforeseen flight conditions and subsystem failures.

As an example of online adaptive systems, Figure 1 illustrates a simple development paradigm. Before Wednesday, the system is built and validated on the training data sets.

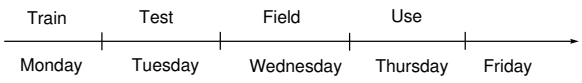

Figure 1. An Online Adaptive System Cycle

After fielding, there is unseen data as well as seen data entering the system and making it learn and change. Thus, the system will react distinctively with respect to the specific data.

With the growing usage of such systems, validation techniques have been developed to assure system stability and reliability. Most of them are static techniques focusing on the system performance before fielding. Effective methods for validating the running system dynamically are rare. As one promising dynamic strategy, novelty detection estimates the reliability of the outputs by using the probability density estimates with respect to the data items already seen by the system [1]. However, this technique usually takes a relatively large amount of computing effort and brings burden onto the system performance, which makes it inapplicable for validating online adaptive systems in a real time manner without impelling excessive computational effort on the running system.

In this paper, we propose a dynamic method based on distance measurement to validate the system adaptation. By sniffing the incoming data in real time not only before but also after it enters the system, we are allowed to prevent the anomalies from system adaptation and discard surprising results that may cause unreliable system performance.

The paper is organized as follows. We first overview recent related work and on-going research effort on validating online adaptive systems. In section 3, we describe a framework comprising of two agents that assures the system performance. Section 4 presents the data sniffing strategy for assessing an online adaptive system. We describe the distance measuring techniques and propose an algorithm for

*This work was supported in part by NASA through cooperative agreement NCC 2-979. The opinions, findings, conclusions and recommendations expressed herein are those of the authors and do not necessarily reflect the views of the sponsor.

testing our approach. Section 5 presents empirical results we obtained from implementing our algorithm in two different domains. In Section 6, conclusions are made and future work is described.

2 Related Work

Most proposed techniques for validating online adaptive systems are based on empirical evaluation through simulation and/or experimental testings. In recent years several experiments evaluated adaptive computational paradigms (neural networks, AI planners) for providing fault tolerance capabilities in control systems [3, 4]. In an on-going effort, a group of researchers at NASA Ames Research Center are defining life cycle validation and verification methods applicable to systems which are integrated with adaptive software components [5]. In some cases, adaptive components are modified to provide support for test based validation of results. An example is the architecture of validity net proposed by Leonard et. al. [1]. Researchers like J.A Leonard have conducted experiments on radial basis functions neural networks by extending the network structure with additional output nodes to calculate confidence intervals for the outputs. Experimental success in research suggests its significant potential for future use.

Analytical methods can provide assurance for some system models with respect to the defined properties. In recent research, Mili et. al. [6] proposed an abstract computational model for online adaptive systems. This model captures the functional properties of an online adaptive system by abstracting away random factors in the function of the system and focus exclusively on details that are relevant to the learning algorithm and the learning data. While this is a generic model that establishes functional properties of adaptive systems using refinement-based reasoning, it is impractical for real time validation.

3 Framework

There are two major phases an online adaptive system assumes while in use. An incoming example is unclassified before it enters the adaptive component, which we define as the pre-adaptation phase. After possible system adaptation, the example is classified and is going to be used as input for the next component, for example, a controller. We call this phase post-adaptation. In order to validate the inputs before they enter the adaptive system as well as the outputs generated by the system, our approach employs the same mechanism to detect anomalies in both phases. Primarily, when peculiar data is presented in the pre-adaptation phase, the system is allowed to either accept the data with some caution for adaptation, or block the classified data from further use.

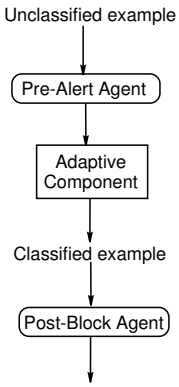

Figure 2. An Adaptive System Integrating Pre-alert Agent and Post-block Agent

According to the pre-adaptation and post-adaptation phases, we define two agents, referred to as *pre-alert agent* and *post-block agent*, respectively. Figure 2 illustrates how an adaptive system comprising of a pre-alert agent and a post-block agent works. The pre-alert agent determines whether a new instance without an assigned class is causing unexpected adaptations for the system and provides quantitative caution if necessary. After the system generates class values for such input data, the class values are examined by the post-block agent to decide their reliability. When the classified example is falling far outside the training domain, the post-block agent will prevent it from entering the next stage, i.e. it will disallow its further use.

In Figure 3, we list the policies for enabling (or disabling) *pre-alert* and *post-block* for different system conditions. More specifically, the policy of enabling *pre-alert* can be explained as allowing the data into the system with particular caution. The *post-block* is enabled when the system conditions necessitate stopping actions based on a poor classification. These actions take two forms:

- the actions taken by the agent using the learner. For example, changing aircraft control surfaces based on the learner's recommendations.
- the updates of the learner's internal theory.

Usually when there is no adaptation, no disaster is imminent and we are not worried about out-of-scope inputs, there is no need to apply either policy. When we are not confident in the suitability of the inputs we can enable pre-alert for particular caution. If instant disaster is possible, one still has the option to ignore classified data. For example, the learner is an optional assistant while the human operator may or may not be active. One particular condition is when we are confident in the input since it is within the scope while abnormal adaptation is caused by some other

Conditions			*Policies*	
Adaptation	*out-of-scope*	*Instant disaster*	*Post-block*	*Pre-alert*
no	*no*	*no*	*disabled*	*disabled*
yes	*yes*	*no*	*disabled*	*enabled*
yes	*no*	*possible*	*enabled*	*disabled*
yes	*yes*	*yes*	*enabled*	*enabled*

Figure 3. Policy Choices on Data Sniffing

factors and instant disaster seems possible (see Figure 3). Under this condition, we may want to enable the post-block policy to discard the unusual output.

4 Data Sniffing Tool

Considering the framework described above, effective tools need to be developed to allow pre-alert agent and post-block agent to determine appropriate actions. Before we present our data sniffing tool, we first introduce our distance hypothesis as the basis of our approach.

Basically, the distance hypothesis is defined as follows.

- Examples seen during use (see Figure 1) that are "close" to examples seen during training and testing do not cause concerns.
- Examples that are "far away" from examples seen during training and testing are causes for concern.

This intuitively simple definition is ambiguous on several important details. Our definitions, shown later, operationalize the following points.

- "close", "far away" - We explore distance metrics from the new example to old examples. The same distance metrics is used for distance between two old examples. We define different distance metrics for unclassified examples and classified examples. For example, by using a vector of bits, when mapping off the bits representing the class attributes, we set the distance metrics from "with class value(s)" to "without class value(s)". Note that an example with class value(s) can only be examined by the post-block agent after the learner runs.
- "causes concern" - Our data sniffer is useless unless it is used. When our sniffer raises a concern, we need to take some kind of appropriate action. As shown in Figure 3, we defined four "sniffer policies" correspondingly to four classes of conditions .

4.1 Distance Measure

Now the problem is - "how shall we operationalize the distance hypothesis?" While investigating in related areas, we found that several clustering algorithms can be used to discover intentional structures in data sets [2]. The classic k-means algorithm forms clusters in numeric domains, thus partitioning instances into disjoint clusters. As a well-understood and relatively successful technique it uses a straightforward Euclidean distance to compute the distance between points. However, one drawback of k-means clustering technique is that we have to determine the number k before we run the algorithm. As a local learner, an online adaptive system can adapt itself rapidly to current feature domain. It is unnecessary to divide the domain into arbitrary k clusters.

This paper explores an alternative algorithm that adopts the distance measurement technique of k-means clustering algorithm. We construct a distance matrix D. D is a $n \times n$ square matrix represents the Euclidean distances between any two examples in the training set where n equals the number of examples in the training sets. Also, we define that $D_{ij} = D_{ji}, 1 \leq i, j \leq n$ and $D_{ii} = 0, 1 \leq i \leq n$. Considering the new data point to be determined outside or inside the covered space, we mingle it with the original data set and recalculate the distance matrix. Thus we can compare these two distance distributions by statistical means. Note that the online adaptation usually deals with small domains consisting of a few dimensions, we choose the t-test as our comparing tool.

4.2 A Testing Algorithm

Having provided the underlying philosophy as detecting "surprising data", we developed an algorithm for testing our approach. Due to characteristics specific to this approach, our algorithm is designed for low dimensional spaces. and thus does not adopt the definition of ad-hoc distance or similar metrics. Before we describe our algorithm, it is necessary to explain the definitions and notations that will be used in this section.

- An example (or datum) x is a single data item used by the algorithm. It typically consists of a vector of values of N attributes: (A_1, A_2, . . . A_N).
- N is the dimensionality of an example in a certain data set. The number of data attributes is noted as N_d, and the number of the class attributes is noted as N_c. Here, $N = N_d + N_c$.
- C_d refers to the number of examples contained in the data set. C_m is the number of new examples. Note that in real applications $C_m = 1$, we here set $1 \leq C_m \leq C_d$ for experimental purpose.

We compute the distance by the following rule based on standard Euclidean distance.

x_i and x_j are two examples with attribute values ($A_{(i,1)}$, $A_{(i,2)}$,..., $A_{(i,N)}$) for x_i and attribute values ($A_{(j,1)}$,

$A_{(j,2)}$,..., $A_{(j,N)}$) for x_j. The distance between x_i and x_j, noted as $D_{(i,j)}$ is computed as:

$$D_{(i,j)} = D_{(j,i)} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{N}(A_{(i,k)} - A_{(j,k)})^2}.$$

We first take a training set S from a given application domain with moderate values of N and C_d. By randomly choosing one example from S, we mutate some attribute values of this example to obtain a likely faulty instance. Specifically, as for the pre-alert agent, we mutate the values of these data attributes for a predefined class to obtain a testing mutant. Since the post-block agent only works after the class values have been generated, possible mutations can be executed not only on data attributes but also on class attributes. We compare these mutants with all old examples instead of a certain class. We can use the same algorithm shown in Figure 4 to implement both tests. The inputs and outputs of the algorithm are follows:

Input :

Data set $S = x_1, x_2, ..., x_{C_d}$.

C_m, the number of mutated examples.

$vstep$, the standard deviation step utilized for the mutation. For instance, when $vstep = 2$, $A_1 = 3$ and the standard deviation of this attribute is 0.5, we modify A_1 to either 4 or 2 with the assumption that the distribution of its values is Gaussian.

Output:

The rejection percentage over C_m new examples based on statistical t-tests.

The algorithm starts with initializing the distance matrices D_1, D_2 and D_m to zeros. In the following loop, the procedure *ComputeDistance($D_1(i,j)$)* computes the distance from example i to example j. Then the algorithm enters the major loop. It iterates C_m times to do the testing. First, it draws an example randomly from data set S through procedure *RandomSample(S, x)* and then modifies it by *MutateExample(x, $vstep$, j)*. After that, procedure *ComputeNewDistance(D_m, x, S)* simply computes the matrix D_m which consists of distance values from this mutated example x to all other examples in S. In procedure *CombineDistance(D_2, D_1, D_m)* we put these two matrix D_1 and D_m together into one matrix D_2 for comparing them in the next procedure *ttest2(D_1,D_2,α)* to perform a t-test.

The t-test examines these two distance matrices to test the null hypothesis, i.e. the average distance for all values in D_1 equals to the average distance of all values in D_2. If

1. $D_1 \leftarrow D_2 \leftarrow D_m \leftarrow 0$;
$count \leftarrow 0$.
2. for $i \leftarrow 1$ **to** C_d
  for $j \leftarrow i$ **to** C_d
    ComputeDistance($D_1(i,j)$);
  end
end
3. for $j \leftarrow 1$ **to** N
  for $k \leftarrow 1$ **to** C_m
    RandomSample(S, x);
    MutateExample(x, $vstep$, j);
    ComputeNewDistance(D_m, x, S);
    CombineDistance(D_2, D_1, D_m);
    $count \leftarrow count +$ **ttest2**(D_1,D_2,α);
  end
end
4. return $\frac{count}{C_m}$.

Figure 4. Algorithm

the null hypothesis is passed, then it implies that the new example x does not change the topological property of the original domain. Recalling that there is no value of class attributes available for the pre-alert agent, we modify the algorithm slightly by changing the mutation number from N to N_d which excludes the class attributes. When computing the distance we set the N to N_d as well. In this case, we filter the data by the specific values of class attributes into a subset for running the algorithm.

5 Empirical Results

For comparison purposes, we offer two sets of results - one where data sniffing works very well and the other where it fails. Our discussion section offers some speculation on why the failure occurred.

5.1 Experiments

The first data set is the circuit data provided by Hu and Menzies [7]. The data set has 192 examples. Each example consists of eighteen data attributes and one class attribute. All nineteen attributes have discrete values. The values of class attribute are enumerated as $4, 5, 6, 7$. In particular, the data attributes cause some constraints on our experiments that the only deviation step we can set for the algorithm is one, i.e. $vstep = 1$.

In Figure 5 and 6, x axis represents the number of mutated attributes, in the algorithm referred to as k. y axis

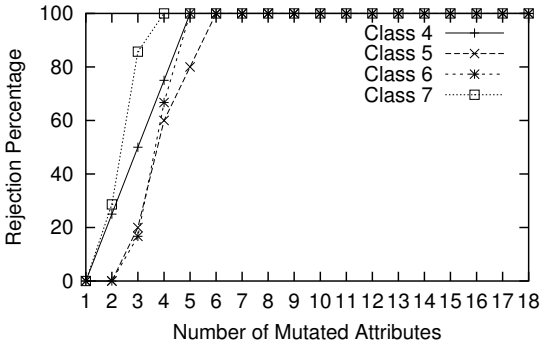

Figure 5. Pre-alert Testing for Circuit Data, i.e. assessing distance without class when $vstep = 1$ at confidence level 0.05 ($\alpha = 0.05$).

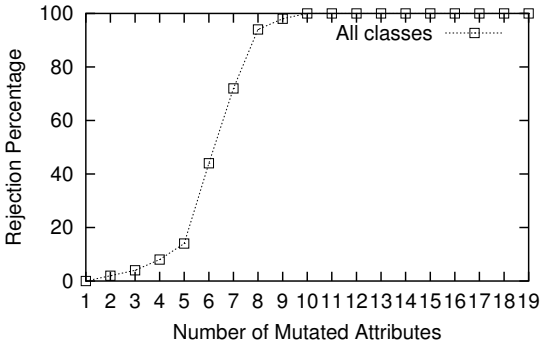

Figure 6. Post-block Testing for Circuit Data, i.e. assessing distance with class when $vstep = 1$ at confidence level 0.05 ($\alpha = 0.05$).

represents the rejection percentage obtained from running the algorithm for each class. Each point shows the rejection percentage for that specific class when certain attribute values were mutated. One trend seen in Figure 5 is that for all class values, the rejection percentage increases with the increase of the number of mutated attributes. After the number of mutated attributes reaches 6, all four runs returned 100% rejection. The same trend shows in Figure 6. Although it is weaker than the one seen in Figure 5, at mutation number 10, the algorithm fully rejects all mutated examples. The possible explanation for this phenomenon is that the class value is the essential representative for the complete example. Without comparing to a particular class value, the mutations cannot bring significant difference into the example.

We conducted another experiment for the scale balance data provided by Hume [8] from the machine learning database built by University of California, Irvine. Each example of this data set is classified as having the balance scale tip to the right (class 'R'), tip to the left (class 'L'), or be balanced (class 'B'). There are four data attributes, which are the left weight, the left distance, the right weight, and the right distance. The class value is determined by the greater of (left-distance $\times$ left-weight) and (right-distance $\times$ right-weight). If they are equal, it is balanced. The data file consists of 625 examples covering all possible weight and distance values. We run the algorithm on class 'L' examples for pre-alert testings and then the whole data set for post-block testings.

With experiments for pre-alert testing, Figure 7 evidently shows growth of rejection percentage with the increase of deviation step from 1 to 4. The increasing trend of rejection percentage according to the number of mutated attributes also shows in this run. However, the rejection percentage is never greater than 50% and the results are weaker than those in the first experiment.

The weakest results are illustrated by Figure 8 from the experiments of post-block testings. The algorithm was not able to identify the anomalies effectively until the deviation step was set to 5. The possible reason for this occurrence is that the characteristics of the data is so strong that mutating a single or further multiple attributes might still keep the example being correct, which implies that it cannot be driven very far from the original example by simple mutations.

5.2 Discussion

We presented examples where data sniffing worked and where it failed. While a good demonstration was obtained from the experiment with the circuit data, the other set of experiments with scale balance data actually showed us the drawback of our testing algorithm as well as the existence of other factors that might affect the performance of our approach. By comparing these two domains we try to characterize domains where data sniffing may fail.

The intrinsic classification rule of scale balance data makes a single mutation not sufficient for generating faulty example. For instance, the first example shown in Figure 9 has a class value 'L'. Modifying single value of *Left weight* from 2 to 1 or to 3 with $vstep = 1$ does not change the class value. Furthermore, the probability of obtaining a faulty example is fairly low, even $vstep$ and C_m increase. On the contrary, examples in the circuit data file cannot be related to each other by a single mutation. Instead, modifications of single attribute in circuit data may easily cause a faulty example. Since our testing algorithm was not able to detect the relational examples before examining them, it considered them as safe examples seen in the training data set. Based on the above experience we hypothesize that

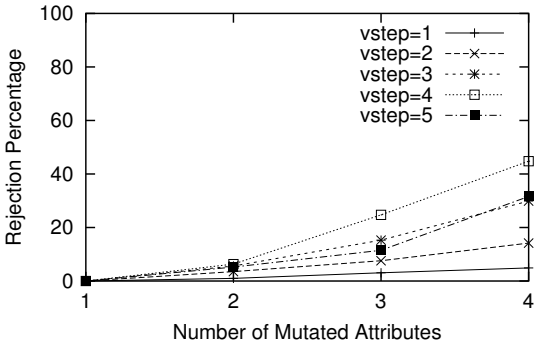

Figure 7. Pre-alert Testing for Scale Balance Data, i.e. assessing distance without class with different deviation steps when $\alpha = 0.05$.

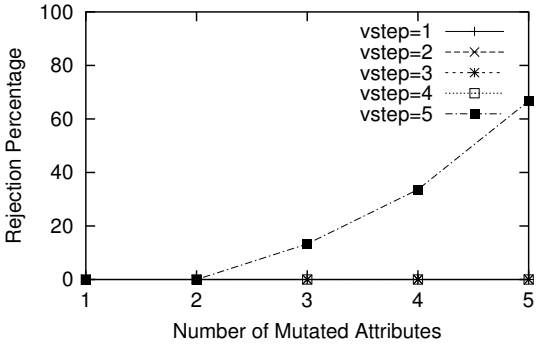

Figure 8. Post-block Testing for Scale Balance Data i.e. assessing distance with class with different deviation steps when $\alpha = 0.05$.

Euclidean-based data sniffing will fail for highly relational data.

6 Conclusion

This paper proposes a novel tool that uses simple data sniffing strategy for validating online adaptive systems. We offer a verification framework comprising of two agents that check the validity of inputs and classifications. Empirical results were attained from running our testing algorithm on different data sets. We can conclude that the the method works reasonably well for data that is not highly relational. Since our approach is very straightforward, it can be easily applied for testing certain domains by simply running the mutator.

In the future, we expect to refine our algorithm to distinguish faulty and correct examples before executing the rejection, causing a more accurate rejection percentage computation. Therefore, available machine learning tools detecting anomalies in certain data domains can be potentially employed. Future work will focus on developing a refined distance computation method which does not make any (implicit or explicit) hypothesis concerning the topology of the domain. Furthermore, sophisticated measurement techniques, which can capture the distance in terms of feature space will be investigated.

Left weight	*Left distance*	*Right weight*	*Right distance*	*Class*
2	*3*	*1*	*2*	*L*
1	*3*	*1*	*2*	*L*
3	*3*	*1*	*2*	*L*
4	*3*	*3*	*3*	*L*

Figure 9. Examples in Scale Balance Data

References

[1] J.A. Leonard, M.A. Kramer, and L.H. Ungar. Using radial bais functions to approximate a function and its error bounds. *IEEE Transactions on Neural Networks*, 3(4):624-627, July 1992.

[2] I. H. Witten, E. Frank. *Data Mining :Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann Publishers, 210-226, 2000.

[3] M. Napolitano, G. Molinaro, M. Innocenti, and D. Martinelli. A complete hardware package for a fault tolerant flight control system using online learning neural networks. *IEEE Control Systems Technology*, January, 1998.

[4] M. Napolitano, C.D. Neppach and V. Casdorph. A neural network-based scheme for sensor failure detection, identification and accomodation. *AIAA Journal of Control and Dynamics*, 18(6):1280-1286, 1995.

[5] M.A. Boyd, J, Schumann, G. Brat, D. Giannakopoulou, B. Cukic and A. Mili. Validation and verification process guide for software and neural nets. Technical report, NASA Ames Research Center, September 2001.

[6] A. Mili, B. Cukic, Y. Liu and R.B. Ayed. Towards the verification and validation of online learning adaptive systems. *Annals of Software Engineering*, 2002 (accpeted for publication).

[7] T. Menzies and Y. Hu, Constraining discussions in requirements engineering, *First International Workshop on Model-based Requirements Engineering*, 2001. `http://tim.menzies.com/pdf/01lesstalk.pdf`

[8] T. Hume, Three aspects of cognitive development, *Cognitive Psychology*, 8, 481-520, 1994.