Software and System Reliability Engineering for Autonomous Systems Incorporating Machine Learning

Publisher: IEEE

Aiden Gula ; Christian Ellis ; Saikath Bhattacharya ; Lance Fiondella **All Authors**

Abstract:
Artificial intelligence and machine learning have attracted significant interest as enablers of autonomous systems. However, these techniques are susceptible to a variety... **View more**

Metadata

Abstract:
Artificial intelligence and machine learning have attracted significant interest as enablers of autonomous systems. However, these techniques are susceptible to a variety of failures as well as adversarial attacks, suggesting the need for formal reliability and resilience engineering methods. Tempered by the knowledge that machine learning is not a panacea and that private industry, infrastructure management, and defense systems are regularly subject to external attack, it is essential to assess the possible failures and corresponding consequences that these technologies may inadvertently introduce. This paper seeks to bridge the gap between traditional and emerging methods to support the engineering of autonomous systems incorporating machine learning. Toward this end we seek to synthesize methods from established fields such as system and reliability engineering as well as software testing with recent trends in the design and test of machine learning algorithms. The proposed approach should provide organizations with additional structure to comprehend and allocate their risk mitigation efforts in order to address issues that will inevitably arise from these less well understood technologies.

Published in: 2020 Annual Reliability and Maintainability Symposium (RAMS)

Date of Conference: 27-30 January 2020

DOI: 10.1109/RAMS48030.2020.9153595

Date Added to IEEE *Xplore*: 31 July 2020

Publisher: IEEE

ISBN Information:

Conference Location: Palm Springs, CA, USA

ISSN Information:

Contents

Summary & Conclusions

Artificial intelligence and machine learning have attracted significant interest as enablers of autonomous systems. However, these techniques are susceptible to a variety of failures as well as adversarial attacks, suggesting the need for formal reliability and resilience engineering methods. Tempered by the knowledge that machine learning is not a panacea and that private industry, infrastructure management, and defense systems are regularly subject to external attack, it is essential to assess the possible failures and corresponding consequences that these technologies may inadvertently introduce. This paper seeks to bridge the gap between traditional and emerging methods to support the engineering of autonomous systems incorporating machine learning. Toward this end we seek to synthesize methods from established fields such as system and reliability engineering as well as software testing with recent trends in the design and test of machine learning algorithms. The proposed approach should provide organizations with additional structure to comprehend and allocate their risk mitigation efforts in order to address issues that will inevitably arise from these less well understood technologies.

SECTION 1
Introduction

In recent years, artificial intelligence (AI) and machine learning (ML) have enjoyed a resurgence thanks to the confluence of big data and high-performance computing. How to properly align these technologies with public purpose [1] is also experiencing intense discussion among policy makers and engineering leadership. From a technical perspective, test and evaluation of unmanned autonomous systems [2] continues to lack formal quantitative definitions of notions such as adaptability, emergent behavior, and human trust. Challenges faced by the policy and technical communities are indicative of the same underlying challenge, namely the need to articulate and regularly refine repeatable and structured methods for the design and assessment of systems incorporating AI and ML that demonstrate properties, including reliability and safety. In the absence of such methods, it will be difficult or impossible to ensure these systems will be accepted as trusted partners by humans and effectively interoperate with them. Clear guidance to incorporate AI and ML technologies into a system are needed to mitigate undesirable difficulties, delays, and potential project failure.

Past research on autonomous systems includes the work of Huang et al. [3] who developed the ALFUS (Autonomy Levels for Unmanned Systems) Framework, which identifies common definitions and terminology to characterize autonomy and facilitate communication in the unmanned systems (UMS) community as well as to specify quantitative metrics to assess autonomous system capabilities. Creation of the ALFUS was informed by ASTM, AIAA, ISO, and IEEE standards, but primarily influenced by and influencing ASTM standards for robotics in physical and simulated environments, including robotic search and rescue [4], unmanned air systems (UAS) [5] and unmanned undersea Vehicles (UUV) [6]. Informed by Sheridan's scale of degrees of automation [7] and Parasuraman's model for types and levels of human interaction with automation [8], NASA scientists working on the SMART (Spacecraft Mission Assessment and Replanning Tool) developed the Level of Autonomy (LOA) Assessment Tool [9], a design time methodology, to determine the appropriate level of autonomy for each function of the SMART flight management system, instead of assuming that the ultimate goal of all systems should be complete autonomy. This approach creates the potential to assess tradeoffs between system cost, design effort and corresponding timeline, safety, operational efficiency, operator trust, and lifecycle cost. Recognizing that capabilities enabled by autonomous systems pose new challenges, a 2015 workshop on the test and evaluation of autonomous systems [10] identified eight shortcomings to existing practices.

To incorporate AI and machine learning, new system engineering methods [11, 12] will be needed, especially

for systems of systems. Moreover, AI and machine learning methods promise to transform the system engineering process itself. However, there is a pressing need to understand how ML capabilities can be incorporated into existing processes. To address this need, this paper explicitly documents how machine learning methods map to traditional concepts from reliability engineering such as (i) reliability growth modeling and reliability engineering, (ii) fault tolerance, (iii) software testing, and (iv) failure modes and effects criticality analysis (FMECA). In documenting these connections in a simple and intuitive manner, we seek to: (i) provide the Test and Evaluation Community with a familiar framework in which to assess autonomous systems and (ii) facilitate effective communication between diverse stakeholders such as system engineers, advanced algorithm designers, and testers as well as leadership. The remainder of the paper is organized as follows: Section 2 places reliability engineering of machine learning in the context of system and software reliability engineering. Section 3 identifies relationships between machine learning and reliability growth modeling and reliability engineering. Section 4 discusses software testing and failure modes, effects and criticality analysis of machine learning. Section 5 offers conclusions and directions for future research.

SECTION 2
Relationship of Machine Learning To System and Software Reliability Engineering

To approach machine learning from a reliability engineering perspective, it is important to recognize machine learning as an enabler of autonomy that commonly resides in software. While hardware-based machine learning is also an established field of research that can achieve substantial speedup and other desirable attributes, hardware implementation typically occurs after the utility of functionality is demonstrated in software. It is therefore important to recognize reliability engineering of machine learning as a predominantly software reliability engineering problem to address within the context of the system engineering process. Moreover, it is important to distinguish between traditional and machine learning-based software components, including the fact that machine learning accuracy depends on training data representative of the operational environment.

Abstractly, machine learning components perform the task of perception and their outputs inform decision making, which can range from simple deterministic cases to advanced rule-based artificial intelligence, after which the system executes the actions. Thus, most systems incorporating machine learning will likely also include traditional software components, necessitating test of traditional and machine learning components as well as their interactions. Architecture-based software reliability [13] characterizes the reliability of an application in terms of its components and the inter-component transition probabilities, which are influenced by inputs from sensors, users, and configuration files. Software reliability growth models [14] can be applied to a complete software application or major components in order to produce reliability estimates and predict time between failures.

Given an estimate of software reliability, one may combine software and hardware reliability estimates to produce a system reliability estimate with reliability modeling paradigms such as reliability block diagrams or fault trees [15]. The realism of traditional system reliability models considering software, architecture-based software reliability models, and software reliability growth models can be improved and additional enhancements will be needed to address machine learning.

SECTION 3
Reliability Growth Modeling and Reliability Engineering of Machine Learning

This section identifies relationships between machine learning methods and traditional reliability growth modeling and engineering concepts. Growth modeling is discussed before reliability engineering to parallel the order in which these concepts are normally presented in machine learning. While machine learning terminology was developed to facilitate communication among those studying it, these terms have also significantly contributed to confusion within the broader engineering community. Therefore, classical statistical terms are also given here to further simplify the exposition. Machine learning is computational statistics. Training a machine learning model with data is synonymous with statistical model fitting. A trained

model is subsequently used to make predictions on unobserved data and is therefore a form of statistical model prediction. The algorithmic training process attempts to identify parameter values that enable accurate prediction and is therefore a form of parameter estimation. Machine learning models often employ familiar objective functions such as regression and maximum likelihood as well as corresponding optimization algorithms.

3.1 Reliability Growth Modeling

Traditional reliability growth modeling characterizes how the reliability of a system increases during testing. These models assume that, as failures are experienced, their underlying sources are identified and removed, leading to reliability growth.

Figure 1 shows the machine learning equivalent of a reliability growth curve.

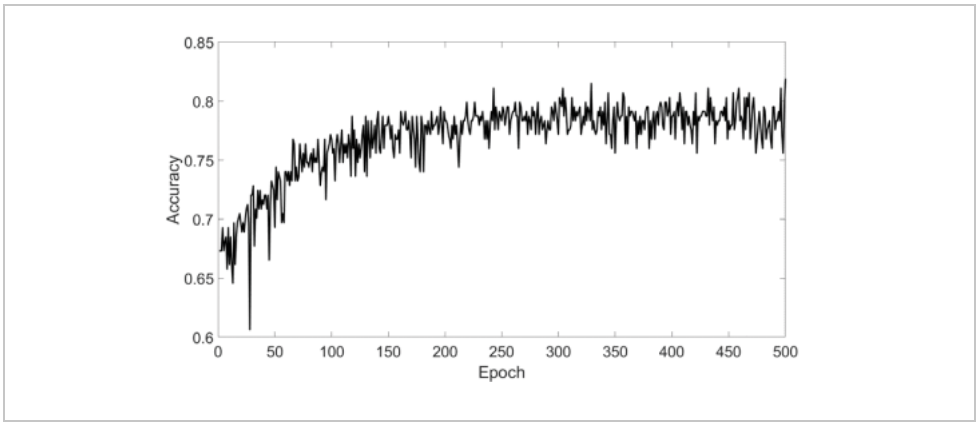

Figure 1
Epoch vs accuracy

Since the accuracy of a machine learning algorithm tends to improve with more training data, time and reliability are replaced with training iterations (epochs) and accuracy on the x- and y-axes respectively. Here, accuracy is defined as the fraction of correct predictions divided by the total number of predictions. Hence, accuracy and reliability tend to be used synonymously. This is appropriate for discrete machine learning problems such as classification, which seeks to place observations (input) into one of two or more categories (output). Binary classification is the special case where there are only two categories. The method underlying Figure 1 is also applicable to classifiers with more than two categories.

The recent empirical success of machine learning methods has also limited attention to the theoretical foundations of machine learning. However, rigorous mathematical statistics underlying machine learning has been studied for nearly fifty years [16]. The Probably Approximately Correct (PAC) [17] learning framework defines the class of learnable concepts in terms of the sample size required to obtain an approximate model. Since machine learning is computational statistics, it also explicitly considers the corresponding time and space resource requirements. A simplified definition in the language of reliability engineering is that a problem is PAC-learnable, if there is an algorithm such that for $\varepsilon > 0$ and $\delta > 0$

$$\Pr\{R(m) > 1 - \varepsilon\} \geq 1 - \delta \tag{1}$$

View Source

reliability $R(\cdot)$ of a fitted model attained by a sample size m greater than a polynomial function of $1/\varepsilon$, $1/\delta$, the cost of representing input elements, and the size of the concept to be learned is greater than 1-ε with confidence or probability $1-\delta$. Furthermore, if the algorithm runs in polynomial time as a function of the factors that determine the sample size requirement, it is said to be efficiently PAC-learnable. This result is distribution free in the sense that it does not matter what the underlying distribution from which the sample data was taken. This $\delta - \varepsilon$ relation can directly inform feasibility, especially when the cost data is nontrivial, although clever methods to generate and annotate data within modeling and simulation as well as other environments have emerged.

Additional considerations [18] include the notion of consistency and finiteness of possible model fits, referred to as the hypothesis set. Consistency corresponds to the cases where a model fit is error free, so that reliability is 1.0. An example of a finite hypothesis set is a binary neural network, which possess binary weight parameters and activation functions, whereas continuous weight parameters render the cardinality of the hypothesis set infinite.

3.2 Reliability Engineering

Machine learning researchers and industry practitioners do not discuss the underlying problem of overfitting machine learning models. Proper methods to validate the predictive capability of statistical models quantitatively assess performance on data not used to fit the model.

Model selection attempts to reduce error, which is decomposed into estimation and approximation error. Estimation error is a function of the model fit, whereas approximation error cannot be estimated, but describes how well the model fit approximates the Bayes error or average noise. Empirical risk minimization seeks to minimize the error on a training sample. Ultimately, a tradeoff between estimation and approximation error is required, which is related to the classical dilemma of model complexity and in and out of sample goodness of fit posed by training and validation data. Moreover, cross-validation methods impose tradeoffs between variance and bias, but reducing bias can also increase computational requirements.

Figure 2 shows the loss in predictions made by the classifier shown in Figure 1 as well as on testing data.

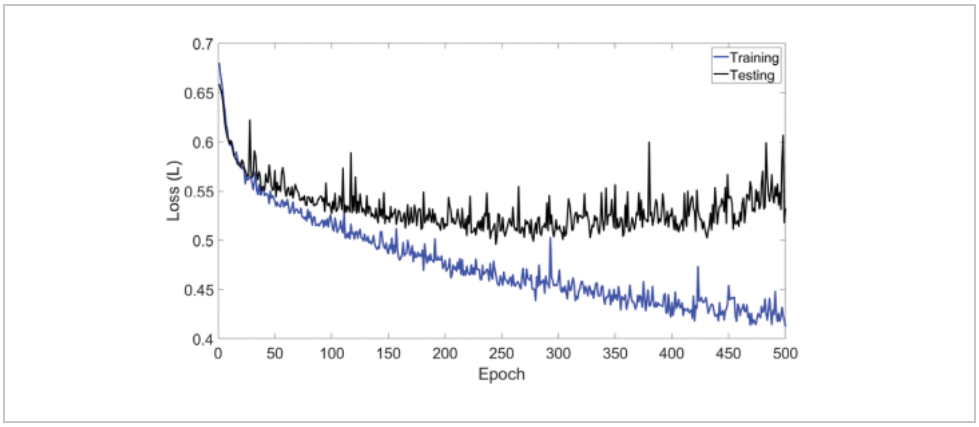

Figure 2
Impact of model fitting on loss in training and testing

Figure 2 indicates that the loss decreases monotonically on the training data, but that loss decreases and then increases on the testing data, indicating that model overfitting has occurred because the algorithm performs well on the training data, but worse on the testing data.

Regularization [19] is a method to avoid overfitting by penalizing a model according to its complexity. A general form of the regularization of the hypothesis h is

$$R(h) = L(h) + \lambda * C(h) \tag{2}$$

View Source

where L is the empirical loss and $\lambda>0$ the penalty applied to the complexity function $C(h)>0$. In addition to promoting simplicity, regularization enables dimensionality reduction.

In some cases, the amount of data is too small to reserve a subset for validation. K-fold cross validation uses the data for both training and testing by dividing a dataset of size m into n subsets of equal size. The learning algorithm is then trained on (n-1) subsets and validated with the remaining subset. K-fold cross validation is applied iteratively to improve a model's predictive accuracy and is also employed in conjunction with regularization.

In machine learning, fault tolerance is commonly referred to as ensemble learning [20]. This includes both

unweighted and a variety of weighted [21] majority voting techniques. Bootstrapping [22] is a popular technique to avoid overfitting in the context of ensemble classifiers.

SECTION 4
Software and System Testing Considering Machine Learning

This section compares traditional software testing methods with methods for machine learning as well as failure modes, effects and criticality analysis for machine learning.

4.1 Traditional software testing

A mature software testing process begins early in the system development lifecycle. Model driven test design [23] decomposes testing into a series of small tasks to simplify test generation. Test development can be divided into four tasks: design, automation, execution, and evaluation. Test automation controls test execution, compares desired and actual outcomes, and other reporting functions, while measures of code coverage such as input space partitioning and graph coverage quantitatively approximate the thoroughness of testing.

Complete coverage is often infeasible, so the input space is broken down into subsets with the goal of maximizing the number of discovered defects per subset. To maximize coverage, the number of test requirements must be satisfied by the tests completed. Often an organization will implement a test automation framework that runs existing sets and new tests for the changes that were made. This assures no new faults are introduced as new features are added.

To illustrate the tradeoff between model complexity and accuracy, Figure 3 provides an input-domain view of testing performed on a binary classifier.

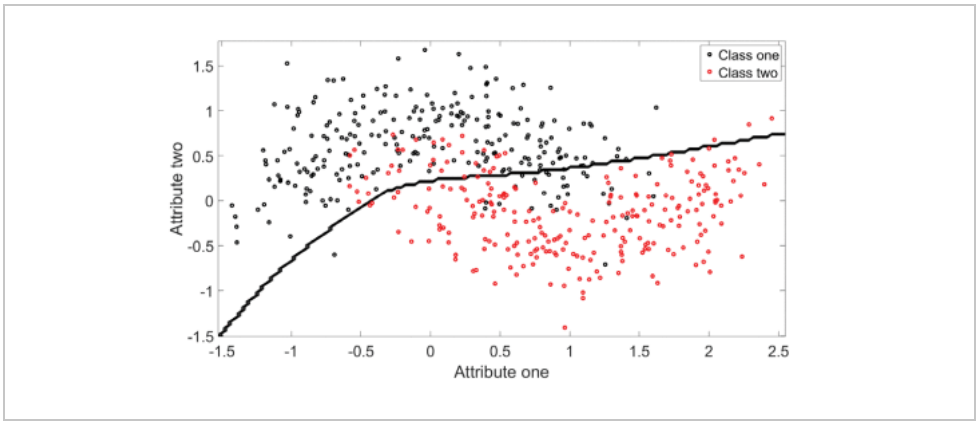

Figure 3
Input domain view of testing machine learning algorithm

In Figure 3, black dots above the black line and red dots below the line represent correct classifications. The two input attributes are continuous valued, making the input space infinite Several misclassifications of class one and two are visible. While a more complex model can reduce the classification error on the observed data, it risks complicating the model.

4.2 Failure modes, effects and criticality analysis

Failure modes, effects and criticality analysis (FMECA) [24] is a method to specify how a system or subsystem fails, the consequences of those failures as well as their criticality. The FMECA process often includes fault tree analysis [15] and related methods to characterize the logical structure of failure propagation within a system and the corresponding probabilities in order to quantify risk and provide a systematic strategy for mitigation.

Machine learning algorithms enable autonomous systems to make decisions, which are prone to failures that can be considered as part of the FMECA. For the sake of illustration, consider a simplified scenario in which an autonomous vehicle is faced with the binary classification problem of determining if the road ahead within safe stopping distance is free of pedestrians or not. Potential failure modes include stopping for a nonexistent pedestrian, which will inconvenience the passengers and not stopping for a pedestrian, which will endanger the pedestrian. Clearly, the latter of these two failure modes should be assigned a greater criticality.

Table 1 shows the confusion matrix associated with the pedestrian identification problem for autonomous vehicles, including the actual and predicted values, categorizing into true (T) and false (F) positives (P) and negatives (N) as well as the criticality of errors associated with off-diagonal elements if acted upon by the autonomous system.

Table 1 Confusion Matrix For Pedestrian Identification

		Actual Values	
		Pedestrian	No Pedestrian
Predicted Values	Pedestrian	*TP*	*FP* (Minor)
	No Pedestrian	*FN* (Catastrophic)	*TN*

Defining a cost matrix *C* based on Table 1, a false negative (c_{21}) is significantly more costly than a false positive (c_{12}). Thus, the cost of misclassification need not be symmetric and traditional methods from binary systems reliability theory are inadequate. Specifically, accuracy $\left(A = \frac{TP+TN}{TP+TN+FP+FN}\right)$ is most closely related to the concept of reliability, whereas sensitivity or true positive rate $\left(TPR = \frac{TP}{TP+FN}\right)$ provides a quantitative measure to maximize in order to avoid this critical failure mode and specificity or true negative rate $\left(TNR = \frac{TN}{TN+FP}\right)$ offers a complementary but conflicting measure to maximize that ensures the autonomous vehicle can provide useful service to the passenger.

To address these competing concerns, testing must therefore quantify the probability of entries in the confusion matrix to express cost as

$$C(Pedestrian) = c_{21} \times \Pr\{FN\} + c_{12} \times \Pr\{FP\}$$

View Source

where Pr{*FN*} and Pr{*FP*} denote the probability of false negatives and positives respectively. This cost expression is at least partially subjective, and the designer must choose appropriate values. For example, the cost of a false negative (c_{21}) may be the average cost of an insurance claim plus reputation cost, whereas the cost of a false positive (c_{12}) could be based on other factors such as customer frustration that leads to switching to the manufacturer's competition. It is also the case that safety critical failures such as false negatives are subject to regulatory oversight that require the manufacturer achieve a probability of failure no worse than a typical human driver. An additional example of this kind of cost modeling for machine learning in the context of reliability engineering includes the work of Nagaraju et al. [25] who quantify the impact of correlation between reliability and prognostics and health management on availability, safety, and cost.

The confusion matrix can be generalized to the *n*dimensional case with cost matrix $C_{n\times n}$ to characterize the reward or loss/penalty/cost associated with correct or incorrect classification. Moreover, this generalization may be interpreted at various levels of abstraction. At a lower-level of abstraction one may wish to consider the categories of objects a deep learning (DL)-based neural network for computer vision is trained to recognize in an image of a scene, whereas a higher-level of abstraction can be applied to the quality of decisions made by an autonomous systems.

A generalized cost equation to score tests [26] is

$$F(C) = \sum_{j=0}^{n} \sum_{i=0}^{n} c_{ij} p_{ij} \tag{3}$$

View Source

where class (category) zero corresponds to 'nothing'. In the context of the DL-based neural network, this enables consideration of the consequences of seeing something when nothing is there as well as seeing nothing when something is there. In the latter context of an autonomous system's decisions, this corresponds to doing nothing when something should have been done and doing something when nothing should have been done. c_{ij} in Equation (3) denotes the cost of classifying an object of class i in class j. For example, while unfortunate, not seeing a small animal in its path may be assigned a lower cost than missing a child. Moreover, the default for correct classification can be set to c_{ii}=0, but this need not follow in all cases. p_{ij} is the probability of classifying an object of class i in class j, which weights the off-diagonal costs (rewards or penalties depending on their sign).

While expressions such as (3) can be used to quantify criticality (cost) and risk (probability) by testing and certification authorities, machine learning researchers have also proposed methods to train a classifier in light of cost. This method is known as cost sensitive learning [26], which attempts to minimize a loss function composed of a weighted sum of sensitivity and specificity. Data and algorithmic methods have been proposed to address this problem. Class rebalancing [27] is a data-based method which changes the proportion of training examples in the set by under sampling the more prevalent data and oversampling the underrepresented data, although under sampling can lose useful information, while oversampling can bias the classification thresholds. Algorithmic methods [28] modify the learning process to improve the sensitivity of the classifier in order to mitigate the consequences of catastrophic misclassifications.

SECTION 5
Conclusions and Future Work

This paper sought to bridge the gap between traditional system reliability engineering and machine learning methods to support the engineering of autonomous systems incorporating machine learning. Toward this end we explicitly identified similarities and dissimilarities between these fields, including reliability growth modeling, reliability engineering, fault tolerance, software testing, and failure modes, and effects criticality analysis. The discussion is intended to provide individuals familiar with system reliability engineering insight into machine learning design and test, so that they can communicate more comfortably with machine learning engineers who may be unfamiliar with the broader system context within which their algorithms must operate. Only through mutual understanding will system engineering and risk mitigation efforts succeed.

Future research will further elaborate the connections between reliability engineering and machine learning methods.

The relationship between adversarial machine learning and failure modes, effects and criticality analysis is an emerging area, where more thorough understanding will assist in the engineering of resilient autonomous systems. Application of techniques from machine learning to support reliability engineering will also be explored.

Authors