Support Vector Analysis for Computational Risk Assessment, Decision Making, and Vulnerability Discovery in Complex Systems

Andrei V. Gribok[a], Curtis L. Smith[a]
[a] Idaho National Laboratory, Idaho Falls, USA, Andrei.gribok@inl.gov, Curtis.Smith@inl.gov

Abstract: This paper aims to present a fundamentally new way of exploring supervised binary classification by using support vector machines (SVMs) to intelligently guide the sampling process through very high-dimensional parameter spaces in order to analyse logical flaws in the complex design of a system. Currently, the machine learning (ML) field is dominated by pattern recognition, data representation, and forecasting, whereas research into ML techniques for discovering logical fallacies is lacking. The proposed methodology utilizes ML to develop a computational method for tackling the problem of analyzing logical constructs such as fault trees represented as Boolean expressions.
The primary goal of the proposed methodology is to use intelligently guided space sampling methods to drastically reduce the number of system configurations needing analysis. This methodology will enable researchers to auto-detect potential vulnerabilities in system designs, devices, and networks in order to reveal previously unseen issues, minimize human error, and reduce costs by allowing analysts to focus on critical areas via intelligent and efficient sampling of the system's parameter space.

1. INTRODUCTION

A primary limitation of modern probabilistic risk assessment (PRA) techniques is that development of risk scenarios and system vulnerabilities is performed manually by analysts and critically depends on their qualifications, the system information available to them, and their ability to understand and "discover" system vulnerabilities (and to properly describe these vulnerabilities via Boolean logic). In other words, modern PRA is a method for documenting analysts' discoveries, not for suggesting new, previously unknown risks. A method is needed for auto-detecting potential vulnerabilities in system designs in order to reveal previously unseen issues, minimize human error, and reduce human costs by allowing analysts to focus on critical areas by intelligently and efficiently sampling the system's parameter space. In addition, the method must not be reliant on waiting for training data, since "real data" in this context would represent system/subsystem failures.

This paper aims to present a fundamentally new way of exploring supervised binary classification, based on using support vector machines (SVMs) to intelligently guide the sampling process through very high-dimensional parameter spaces in order to analyze logical flaws in the complex design of a system. Currently, the machine learning (ML) field is dominated by pattern recognition, data representation, and forecasting, whereas research into ML techniques for discovering logical fallacies is lacking. The proposed methodology utilizes ML to develop a computational method for tackling the problem of analyzing logical constructs such as fault trees represented as Boolean expressions.

The primary outcome from this project will be a new, broadly applicable methodology that uses intelligently guided space-sampling methods to drastically reduce the number of system configurations needing analyzed. The methodology will enable researchers to auto-detect potential vulnerabilities in system designs, devices, and networks in order to uncover previously unseen issues, minimize human error, and reduce costs by allowing analysts to focus on critical areas via intelligent and efficient sampling of the system's parameter space. The second outcome will be a demonstration of ML benefits, as derived from a case study involving the application of a computational risk assessment to PRA.

2. METHODS

2.1. PRA and Support Vector Machines

PRA is a mature and influential technology that relies on two core methodologies: fault tree analysis and statistical parameter estimation. The fault tree analysis employs Boolean logic to combine various paths to failure. Once such paths are exhaustively enumerated, elementary probability rules are applied to aggregate component probabilities of failure into a total probability of failure for a system. This approach allows for delineating system responses to different initiating events, calculating the system's probability of failure under different scenarios, and determining system vulnerabilities. But despite its impressive past successes, PRA is plagued by inherent fundamental limitations [1],[2],[6].

One primary limitation of PRA is that development of risk scenarios and system vulnerabilities is performed manually by analysts and critically depends on their qualifications, the system information available to them, and their ability to understand and "discover" system vulnerabilities (and to properly describe these vulnerabilities via Boolean logic). In other words, modern PRA is a method for documenting analysts' discoveries, not for suggesting new, previously unknown risks [1],[10],[11]. Currently, PRA methodologies are both a science and an art.

A natural extension of this limitation is the difficulty of applying PRA to highly complex and dynamic systems, due to the number of permutations analysts must consider when evaluating risk scenarios. For large multi-component systems, this number is astronomical, beyond the capacity of humans to efficiently handle. In theory, it would also be possible to determine system behavior and vulnerabilities by collecting failure data; however, most modern complex systems are reliable enough to make actual system failure data points sparse. Our approach bypasses the need for failure data by producing systems and associated system insights via an automated approach that is then evaluated using advanced computational methods.

While modern PRA can analyze sensitivity of top event to basic initiating events through importance measures, no methodology exists for enabling the discovery and analysis of logical flaws, vulnerabilities, and weaknesses in fault trees and the complex design of a system. A novel method is needed for auto-detecting possible vulnerabilities in system designs in order to reveal previously unseen issues, minimize human error, and reduce human costs by allowing analysts to focus on critical areas.

This paper aims to demonstrate a fundamentally new way of exploring supervised binary-classification SVMs [7],[4] to intelligently guide the sampling process through very high-dimensional parameter spaces in order to analyze logical flaws in the complex design for a system. Currently, the ML field is dominated by pattern recognition, data representation, and forecasting [3],[5],[8],[9], whereas research into ML techniques aimed at discovering logical fallacies is lacking. The reason for selecting fault trees as a test bed for the proposed methodology is that they represent logical constructs in their purest, most concise, and most rigorous form.

The proposed approach is built on a regularized kernel-based technique called SVM [4],[5], which corresponds to minimization of the following functional:

$$E(f) = \frac{C}{N}\sum_{i=1}^{N}\left|Y_i - f(x_i)\right|_{\varepsilon} + \frac{1}{2}\left\|f\right\|^2 \tag{1}$$

where $$\left|Y_i - f(x_i)\right|_{\varepsilon} = \begin{cases} 0 \;\; if \;\; \left|Y_i - f(x_i)\right| < \varepsilon \\ \left|Y_i - f(x_i)\right| \quad otherwise \end{cases}$$

is Vapnik's ε-insensitive loss function, C is the regularization parameter, Y is vector of measurements, X is a vector of inputs, f is the separating hyperplane, and N is the number of training patterns. The loss

function assigns zero loss to any errors less than ε, thus safeguarding against overfitting. In other words, this function fits not a crisp value but a tube with radius ε. This is similar to a fuzzy description of the function. The other important thing about this loss function is that it minimizes the least modulus but not the least squares. Strictly speaking, it is known that least squares are optimal for Gaussian noise models and that the least squared loss is sensitive to outliers, since it tends to penalize more large deviations. However, if the noise is not Gaussian, then, as Huber showed, under assumptions of symmetry and convexity of noise probability density function [4],[5], the best approximation is provided by the least modulus, not the least squares. Thus, if one possesses only general information about noise density (as is usually the case), it is better to minimize the least modulus. Parameter ε also plays a very important role in providing a sparse representation of the data, which is important for data reduction. Note that Eq. (1) is very similar to the ridge functional with $\lambda = 1/2C$.

In [5],[4], it is shown that, under very general conditions, the minimizer of Eq. (1) can be written as:

$$f(x) = \sum_{i=1}^{N} c_i * K(x_i, x) \tag{2}$$

where c_i is the solutions of a quadratic programming (QP) problem. $K(x_i,x)$ is the so-called kernel function, which defines the generalized inner product. Several choices of kernel K are available, such as Gaussian, sigmoid, polynomial, and splines. What is interesting is that, for different types of kernels, SVM corresponds to different approximation techniques; for example, the Gaussian kernel corresponds to RBF networks, sigmoid to multilayer perceptron with one hidden layer, and polynomial to polynomial approximation. In this manner, SVM comprises different learning techniques "under one roof." Kernel K should be chosen prior to applying the SVM. Only coefficients c_i are defined by the data, and are obtained by maximizing the following quadratic form:

$$min(\,E(c)) = \frac{1}{2}\sum\nolimits_{i,j=1}^{N} c_i c_j K(x_i, x_j) - \sum\nolimits_{i=1}^{N} c_i Y_i + \varepsilon \sum\nolimits_{i=1}^{N} |c_i|$$

$$subject\ to\ the\ constraints$$

$$\sum_{i=1}^{N} c_i = 0, -\frac{C}{N} \le c_i \le \frac{C}{N}, \qquad i = 1 \ldots N \tag{3}$$

The fundamental thing about Eq. (3) is that, as a QP problem, it possesses *a unique* solution. In contrast to neural network training, in which a complex nonlinear optimization with many local minima is performed, SVM conducts quadratic optimization using a *single global minimum*, meanwhile providing capabilities for learning any nonlinear relations and emulating neural networks. Due to the nature of this QP problem, only a certain number of coefficients c_i will be nonzero, and the data points associated with them are called support vectors (SVs), thus the name SVM. Parameters C and ε are regularization parameters that control the flexibility or complexity of SVMs. These parameters should be selected by user through resampling or other standard techniques. However, it should be emphasized that, in contrast to classical regularization techniques [6],[7], a clear theoretical understanding of C and ε remains elusive, and is the subject of both theorical and experimental efforts. In this paper, the SVM parameters were selected via cross-validation.

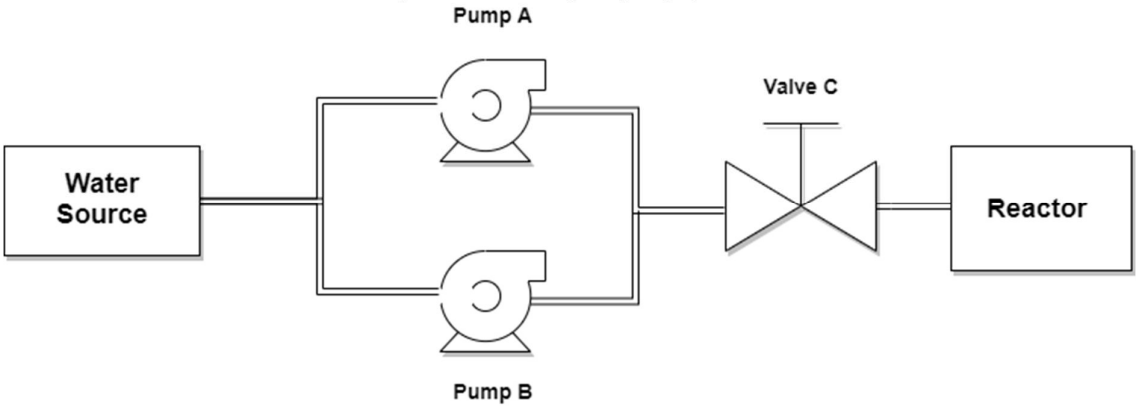
2.2. Analysis of Systems with Existing Fault Trees

For existing systems with available fault trees, the proposed novel methodology first stochastically generates large volumes of training data by "rewiring" the fault trees of the target system. Rewiring includes randomly changing the gate logic and the occurrence of fundamental events (i.e., basic or initiating events). Rewiring existing target datasets skews the training data toward existing systems but

provides the needed variation by generating millions of training examples. The combination of fault tree logic and Boolean variables representing initiating events can be called a configuration.

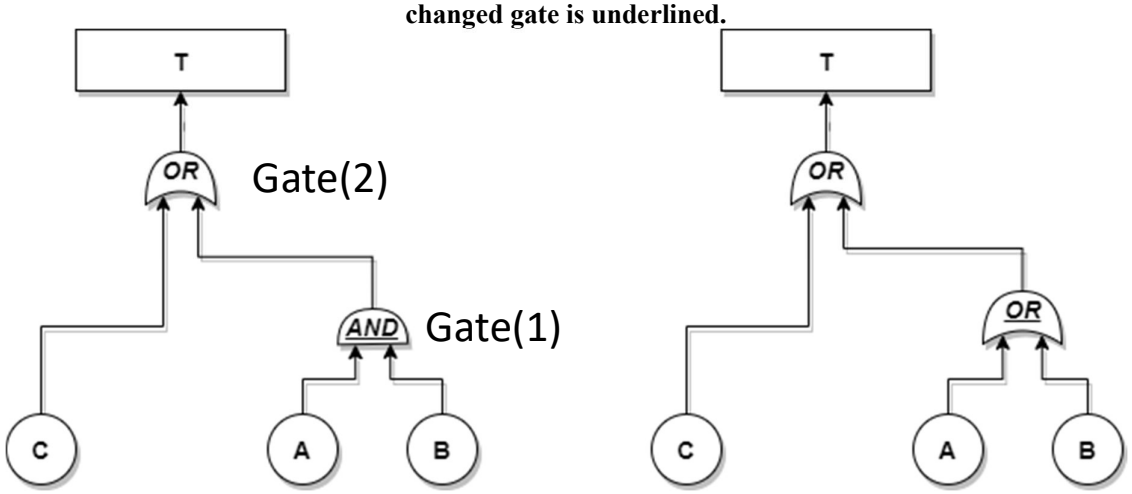
A case in point is the example given in Figure 1, which shows a water pumping system [[2]] that can be tested for its reliability. In this case, the top undesirable event is T = "no water flow to the reactor." The primary initiating events that can cause the top event to occur are: A = "pump A fails to run"; B = "pump B fails to run"; and C = "valve C fails to open." The minimal cut sets' expression for the top event in this example is T = C∨A∧B; that is, no water flow to the reactor will occur *if either* valve C fails to open *or* both pumps A *and* B fail.

Figure 1. Water pumping system.



Fault tree analysis is an analytical technique for determining ways in which the top undesirable event might occur. Figure 2 shows the original fault tree for the system, along with a rewired fault tree. For the rewired tree, the AND gate has been changed to an OR gate.

Figure 2: Original (left) and rewired (right) fault trees for the water pumping system. The changed gate is underlined.



The rewired fault tree is of interest because it represents a situation in which a common cause failure in the two pumps was overlooked when developing the original fault tree. For a common cause failure in the two pumps, the failure of one effectively implies failure of the other, thus changing the AND gate to an OR gate. The truth tables for the two fault trees are shown in Tables 1 and 2, where T is Boolean "1" (meaning failure has occurred) and F is Boolean "0" (meaning the component is operational).

Table 1: Truth Table for the Original Fault Tree

A	T	T	T	**T**	F	F	F	F
B	T	T	F	**F**	T	T	F	F
C	T	F	T	**F**	T	F	T	F
T=C∨A∧B	T	T	T	**F**	T	F	T	F

Table 2: Truth Table for the Rewired Fault Tree

A	T	T	T	**T**	F	F	F	F
B	T	T	F	**F**	T	T	F	F
C	T	F	T	**F**	T	F	T	F
T=C∨A∨B	T	T	T	**T**	T	T	T	F

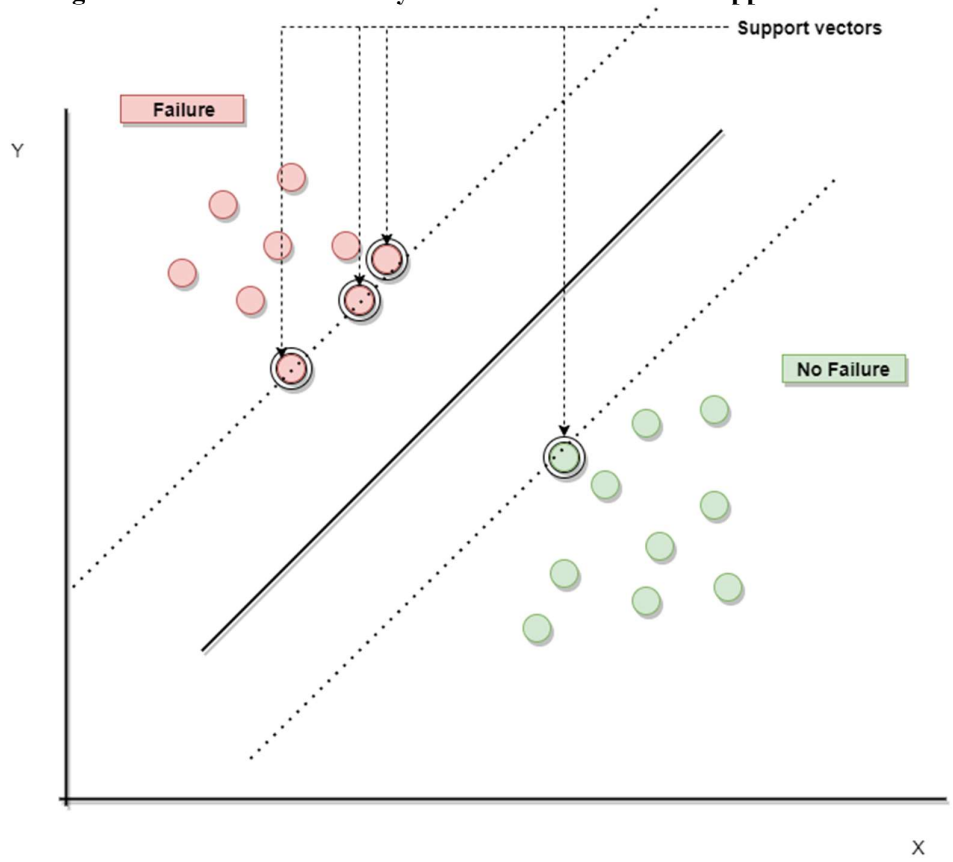
Note that, under the same combination of initiating events, the two fault trees produced opposite results in terms of system failure, as shown in the columns marked in red. For the original fault tree, failure of pump A does not cause the top event to occur, but for the rewired tree, the system fails. Since the two fault trees produce opposite results regarding the occurrence of the top event, the SVM categorizes them into two opposite classes.

SVM is a pattern recognition technique that requires representation of the input data as numerical, as it operates on Euclidean distance. The fault trees are represented as binary vectors, with logical OR represented as 1 and logical AND represented as 0.

For an existing fault tree, the machine-generated data vectors train the supervised binary-classification SVM with two output classes: occurrence and non-occurrence of the top event. During training, along with classifying each input data vector, SVM algorithms also find SVs in the training data, as shown in Figure 3. By the very nature of the training algorithm, SVM focuses only on those points that are most difficult to tell apart. Because in our case the points are realizations of fault trees, SVM will discover the most similar fault trees from both classes, thus also pointing to the most "vulnerable" configurations.

Figure 3 illustrates the process of binary classification using SVM in 2-D space. It is for illustrative purposes only, as the higher dimensional spaces to be used in this project cannot be visualized. Even for the water pumping system in Figure 1, the input space will be 5-D, as there are three Boolean variables (representing basic events) and two logical gates. In the proposed novel methodology, the training data will represent insights from fault trees, so the algorithm will find support fault trees. The SVs are the input data most important for separating the two classes—failure vs. non-failure—and most notably, they represent only a very small portion of the input data. Since in this case the input data are characterizations of the fault trees, the SVM training will produce system configurations that represent the "borderline" between failure and non-failure scenarios. These support trees can be further scrutinized for insights into the system's logical vulnerabilities and risks.

Figure 3: Illustration of binary-classification SVM with support vectors.



3. RESULTS AND DISCUSSION

In this paper, we first demonstrated the proposed approach on the two-pump system shown in Figure 1. This system has only two logical gates and thus only four combinations representing the different logical schemes. The truth tables for those combinations are shown in Tables 3–6, where T is Boolean "1" (meaning failure has occurred) and F is Boolean "0" (meaning the component is operational). The truth densities for every Boolean function were also calculated, representing the ratio of the function's true values to the total number of outputs for that function. If the true value of the output represents a failure, the truth density gives an estimate of the reliability of the system represented by that specific function.

As seen from the tables, the least reliable configuration for the two-pump system is that of Table 4. That configuration has both the logical operators set to OR and has a truth density of 87.5%. The most reliable function is that of Table 6, with both the logical operators being set to AND. In general, systems with a larger number of AND operators tend to be more reliable than those with a larger number of OR operators.

Table 3: Truth Table for T = C OR (A AND B); Truth Density = 5/8 = 62.5%

A	T	T	T	T	F	F	F	F
B	T	T	F	F	T	T	F	F
C	T	F	T	F	T	F	T	F
T=C∨A∧B	**T**	**T**	**T**	F	**T**	**F**	**T**	**F**

Table 4: Truth Table for T = C OR (A OR B); Truth Density = 7/8 = 87.5%.

A	T	T	T	**T**	F	F	F	F
B	T	T	F	**F**	T	T	F	F
C	T	F	T	**F**	T	F	T	F
T=C∨A∨B	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**F**

Table 5: Truth Table for T = C AND (A OR B); Truth Density = 5/8 = 37.5%

A	T	T	T	**T**	F	F	F	F
B	T	T	F	**F**	T	T	F	F
C	T	F	T	**F**	T	F	T	F
T=C∧A∨B	**T**	**T**	**T**	**F**	**F**	**F**	**F**	**F**

Table 6: Truth Table for T = C AND (A AND B); Truth Density = 1/8 = 12.5%

A	T	T	T	**T**	F	F	F	F
B	T	T	F	**F**	T	T	F	F
C	T	F	T	**F**	T	F	T	F
T=C∧A∧B	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**T**

Table 3 represents the truth table for the original two-pump system, with a truth density of 62.5%. Changing the AND operator to OR, we obtained the Boolean function represented in Table 4. We see that changing the gate made the system less reliable, as its truth density was 87.5%. This heightened truth density was due to the two columns marked in red. While the values of the arguments of the Boolean functions are identical, changing the gate added two additional true values to the Boolean function's output, thus increasing the truth density. Table 5 shows the truth table for the function with two operators switched, changing the truth density to 37.5% and making the system more reliable. On the other hand, in Table 6 we see that setting both logical operators to AND produces a highly reliable system with a truth density of 12.5%. The red columns in Table 6 indicate the functions' values that were changed from the original configuration. Table 7 shows all possible configurations for the two-pump system. The four different combinations of logical operations are color-coded. For example, the red columns correspond to the original Boolean function T = C OR (A AND B). Column T corresponds to the top event and is used as response variable for SVM training. A SVM with an RBF kernel was trained using the data from all 32 configurations. The hyperparameters of the SVM were optimized via cross-validation. Since cross-validation splits data randomly, there is variation in the hyperparameter values and hence the number and values of SVs produced by the SVM. The SVM was trained 100 times—each time using a different cross-validation split—and after each training, the SVs were recorded. Each time an input vector was selected as a SV, it was counted. A histogram was built showing how many times a given input configuration was selected as an SV. This histogram for the two-pump system is shown in Figure 4. The y-axis in Figure 4 represents the counts for each configuration to be selected as a SV, out of the 100 training sessions. For example, configuration 1 was never selected as a SV, while configuration 2 was selected in each of the 100 training sessions. As seen from the histogram, some configurations are consistently selected as SVs, while others are never selected. Intuitively, the ones that are never selected are deep into failure or non-failure categories, while frequently selected are borderline.

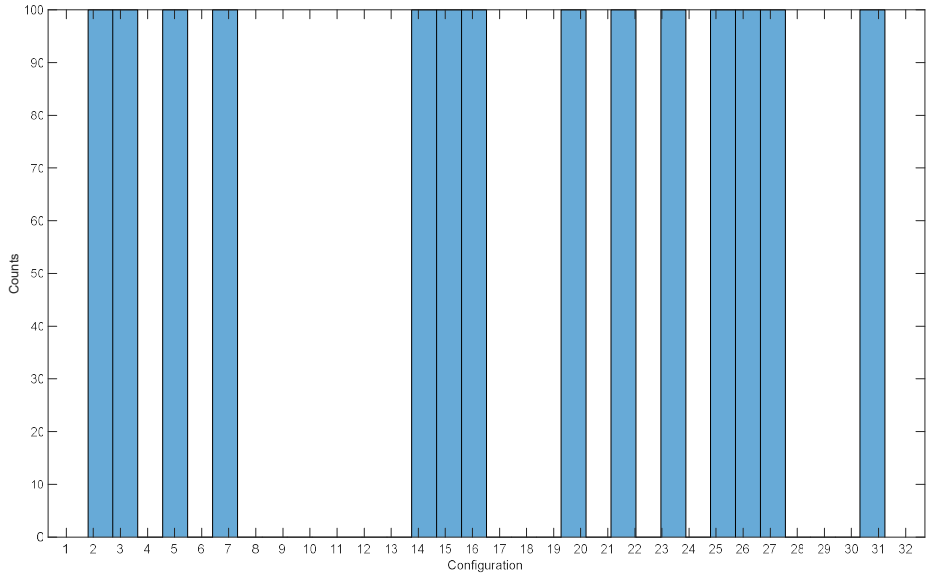
Notice that the blue-coded combination of gates T = C AND (A AND B), with both logical operators set to AND, has the fewest number of configurations selected as SVs. This combination of gates has the lowest truth density and is deeply into the safe category. The same is true for the green-coded combination T = C OR (A OR B), which features both operators set to OR, has the highest truth density, and is deeply into unsafe category. The other two gate combinations (with truth densities of 62.5 and 37.5%) have the largest number of configurations consistently selected as SVs—namely, 4 out of 8.

Table 7: All possible configurations for the two-pump system. The logical gates are coded OR = 1, and AND = 0.

Configuration	C	Gate(2)	A	Gate(1)	B	T
1	0	1	0	0	0	0
2	1	1	0	0	0	1
3	0	1	1	0	0	0
4	1	1	1	0	0	1
5	0	1	0	0	1	0
6	1	1	0	0	1	1
7	0	1	1	0	1	1
8	1	1	1	0	1	1
9	0	0	0	0	0	0
10	0	0	1	0	0	0
11	1	0	0	0	0	0
12	0	0	1	0	1	0
13	0	0	0	0	1	0
14	1	0	1	0	1	1
15	1	0	1	0	0	0
16	1	0	0	0	1	0
17	1	1	1	1	1	1
18	1	1	0	1	1	1
19	1	1	0	1	0	1
20	0	1	0	1	0	0
21	1	1	1	1	0	1
22	0	1	0	1	1	1
23	0	1	1	1	0	1
24	0	1	1	1	1	1
25	1	**0**	1	**1**	0	1
26	1	**0**	0	**1**	1	1
27	0	**0**	1	**1**	1	0
28	1	**0**	1	**1**	1	1
29	0	**0**	0	**1**	0	0
30	0	**0**	0	**1**	1	0
31	1	**0**	0	**1**	0	0
32	0	**0**	1	**1**	0	0

This demonstrates that, of the four different combinations of gates, these two combinations may require further analysis and scrutiny, as they produce the largest number of borderline configurations. Also note that specific configurations that are deeply into the safe or unsafe categories are never selected as SVs, indicating them to be more or less irrelevant for classification. For example, configuration 9, in which all the components are operational and both logical operators are set to OR, is never selected as an SV, as it is a very safe configuration. Similarly, configuration 17, in which all the components failed and both logical operators are set to AND, is never selected as a SV, since it is deeply in the unsafe category. Under this approach, analysts will look for those combinations of logical gates and specific configurations that appear most often as SVs, as they represent the most vulnerable situations and require in-depth analysis.

Figure 4: Support vector configuration histogram for the two-pump system.



An attractive feature of SVM is that the number of SVs is significantly smaller than the total number of input patterns (often only a fraction of a percent), as demonstrated in Table 8, which represents a SVM analysis of the Boolean expression A&B|C|D&E|F&G|J&K. The total number of configurations generated by this expression, including all possible rewirings and basic event permutations, is 131,072.

The first column in Table 8 relates how many input vectors out of the total 131,072 were used to train the SVM. The other columns represent the percentage/number of SVs with respect to the total number of inputs, along with the type of kernel function used, the training loss (error) achieved, and the run time required to train the SVM on a laptop computer. The most important column, marked in bold, is the percentage of all input vectors that are SVs. As we can see, the number of SVs is a fraction of a percent of the total amount of possible configurations. Having identified those support configurations, we can focus our analysis on a very small sample, thus facilitating the overall safety analysis and design. Table 8 also demonstrates that the percentage of SVs is very stable and does not depend on the number of training vectors or SVM parameters (e.g., kernel type). Analysis of the SVs showed that SVM consistently selects the same configurations, as the support configuration and class-membership of those configurations normally depend on a single gate.

Table 8: SVM Analysis of the Boolean Expression A&B|C|D&E|F&G|J&K

Number of inputs used for training	Percent of all input vectors	Number of SVs	Percent of used training inputs	**Percent of all input vectors**	Loss	Time (sec)	Kernel function
512	0.3906	110	21.4844	**0.0839**	0.2056	23.5	polynomial
516	0.3937	89	17.2481	**0.0679**	0.3968	24.3	polynomial
548	0.4181	120	21.8978	**0.0916**	0.2376	87.6	polynomial
612	0.4669	110	17.9739	**0.0839**	0.1428	71.1	Gaussian
708	0.5402	141	19.9153	**0.1076**	0.1308	63.3	polynomial
836	0.6378	339	40.5502	**0.2586**	0.1234	49.6	Gaussian
996	0.7599	197	19.7791	**0.1503**	0.1013	56.6	polynomial
1188	0.9064	291	24.4949	**0.2220**	0.0910	75.2	Gaussian
1412	1.0773	282	19.9717	**0.2151**	0.0866	52.3	polynomial
1668	1.2726	413	24.7602	**0.3151**	0.0673	56.6	Gaussian
1956	1.4923	359	18.3538	**0.2739**	0.0671	83.3	polynomial

Number of inputs used for training	Percent of all input vectors	Number of SVs	Percent of used training inputs	**Percent of all input vectors**	Loss	Time (sec)	Kernel function
2276	1.7365	423	18.5852	**0.3227**	0.0445	247.9	polynomial
2628	2.0050	460	17.5038	**0.3510**	0.0459	98.7	polynomial
3012	2.2980	462	15.3386	**0.3525**	0.0374	94.9	polynomial
3428	2.6154	538	15.6943	**0.4105**	0.0398	2011.5	polynomial
3876	2.9572	384	9.9071	**0.2930**	0.0344	84.6	polynomial
4356	3.3234	642	14.7383	**0.4898**	0.0194	104.9	polynomial
4868	3.7140	648	13.3114	**0.4944**	0.0220	126.6	polynomial
5696	4.3457	713	12.5176	**0.5440**	0.0139	163.0	polynomial
6288	4.7974	769	12.2296	**0.5867**	0.0101	697.7	polynomial
6912	5.2734	830	12.0081	**0.6332**	0.0097	194.9	polynomial
7568	5.7739	834	11.0201	**0.6363**	0.0093	1020.9	polynomial
8256	6.2988	826	10.0048	**0.6302**	0.0058	212.3	polynomial
8976	6.8481	1059	11.7981	**0.8080**	0.0101	192.8	Gaussian
9728	7.4219	965	9.9198	**0.7362**	0.0056	1112.2	polynomial
10512	8.0200	1017	9.6747	**0.7759**	0.0028	1163.8	polynomial
11328	8.6426	979	8.6423	**0.7469**	0.0027	414.3	polynomial
12176	9.2896	980	8.0486	**0.7477**	0.0043	1130.7	polynomial
13056	9.9609	1019	7.8048	**0.7774**	0.0040	1071.0	polynomial
13968	10.6567	1020	7.3024	**0.7782**	0.0021	998.9	polynomial

4. CONCLUSIONS

SVM can be a valuable tool for analyzing vulnerabilities in complex systems. The presented results demonstrate it to be capable of finding a small number of SVs in binary patterns which represent fault trees of certain engineering systems. The detected SVs are consistent and do not depend on the SVM parameters. For relatively large Boolean expressions, the number of SVs is only a fraction of the number of training patterns. Analysts can further scrutinize the SVs to improve system reliability and resiliency.

While the application of SVM to regression and classification problems is a very well-researched topic, the utility of SVs themselves is less understood, constituting a research gap in ML. The scientific goal of the proposed project is to develop a novel SV-based methodology for intelligently sampling high-dimensional spaces and discovering logical flaws in the experiment design, traffic patterns, and reliability constraints of systems, using PRA as a proving ground. An innovative aspect of this research is that, if proven viable, it will reveal logical flaws that indicate how a complex system may be improved (i.e., moved toward the success state) or otherwise potentially impacted (i.e., moved toward the failure state).

Auto detection of logic paths and possible vulnerabilities in system designs will improve risk quantification by revealing previously unseen issues, minimizing human error in regard to the completeness of the risk assessment, and enabling subject matter experts to focus on critical areas, thanks to intelligent and efficient sampling of the system's parameter space. The future work will include analysis of more complicated fault trees and in-depth analysis of selected support configurations.

Acknowledgements

This research was supported by Idaho National Laboratory Directed Research and Development Fund under contract no. #20A1054-008FP.

References

[1] A. Mosleh. *"PRA: A perspective on strength, current limitations, and possible improvements,"* Nuclear Engineering and Technology, volume 46, number 1, pp. 1–10, (2014). doi:10.5516/NET.03.2014.700.

[2] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. "*Fault Tree Handbook,"* NUREG-0492, U.S. Nuclear Regulatory Commission, January 1981.

[3] SECY-17-0112, "*Plans for Increasing Staff Capabilities to Use Risk Information in Decision-Making Activities,"* November 13, 2017.

[4] V. N. Vapnik. *"Statistical Learning Theory,"* John Wiley & Sons, New York, NY, USA, (1998).

[5] V. N. Vapnik. *"The Nature of Statistical Learning Theory,"* Springer-Verlag, New York, NY, USA, (1995).

[6] C. M. Bishop. *"Neural Networks for Pattern Recognition,"* Oxford University Press, Oxford, UK, (1995).

[7] S. Haykin. *"Neural Networks: A Comprehensive Foundation,"* Prentice Hall, Upper Saddle River, NJ, USA. (1999).

[8] Y. LeCun, Y. Bengio, and G. Hinton. *"Deep learning,"* Nature, volume 521, pp. 436–444, (2015). doi:10.1038/nature14539.

[9] R. O. Duda, P. E. Hart, and D. G. Stork. *"Pattern Classification,"* John Wiley & Sons, New York, NY, USA. (2001).

[10] S. B. Akers. *"Binary decision diagrams,"* IEEE Transactions on Computers, volume C-27, number 6, pp. 509–516. (1978). doi:10.1109/TC.1978.1675141.

[11] D. Harris, and S. Harris. *"Digital design and computer architecture (2nd ed.),"* San Francisco, Calif.: Morgan Kaufmann. p. 129, (2012).