2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Clustering and Classification of UVM Test Failures Using Machine Learning Techniques

Andy Truong, Daniel Hellström, Harry Duque, Lars Viklund

Axis Communications AB, Lund, Sweden

***Abstract*—When verifying complex hardware designs, verification engineers are often faced with the problem of analyzing failures in large test suites. We describe how machine learning techniques can be applied to automate the clustering and classification of test failures according to root cause. Log files from simulations of UVM test benches were used as input and pre-processed to produce features suitable for machine learning algorithms. The performance of three clustering and eight classification algorithms was evaluated. In addition, the impact of dimensionality reduction on performance and computation time was investigated. Finally, the use of visualization algorithms for clustering was also examined. The results indicate that the most suitable clustering algorithm was DBSCAN, with AMI and ARI scores of 0.593 and 0.545 respectively. The best performing classification algorithm was random forest, with an accuracy of 0.907 and an F_1-score of 0.913.**

Keywords—verification; UVM; root cause; machine learning; clustering; classification

I. INTRODUCTION

When applying coverage-driven constrained random verification, test suites often consist of a large number of test invocations. The same base test is run many times, with different adjustments to random ranges and distributions, and with different random seeds. It is not uncommon that a test suite for a complex IP consists of several thousand test invocations. Thus, a single bug in the design or verification environment frequently results in many test failures. Whenever several tests fail in a run verification engineers typically start by attempting to answer questions such as:

- Do the test failures all have the same root cause or are there multiple issues that need debugging? If there are multiple root causes, which test failures have a common root cause?
- For each root cause, what is the type of the problem? For example, a design bug, a bug in a verification component, or a misconfiguration of the DUT caused by a missing constraint, *etc*.

Answering these questions helps to judge the urgency of the issues and plan the debugging effort. Today, our verification engineers manually inspect the test failures and draw conclusions based on experience and knowledge of the verification environment. This can be time consuming and tedious work. Automating this process can save time and allow verification engineers to quickly focus debugging efforts.

Automation of test failure analysis can be achieved using different approaches. Rule-based approaches involve creating a set of rules that correlate symptoms with root causes. Model-based approaches rely on constructing an approximate model of the system being diagnosed. While these approaches can be effective, they require extensive application-specific knowledge that is often hard to obtain and maintain. An alternative approach is to use machine learning [1]. This approach does not require in-depth knowledge of the system. Instead, machine learning relies on large amounts of data to automatically model relationships between test failure and root cause.

The goal of the work described in this paper is to investigate how machine learning techniques can be applied to automate the task of clustering and classifying test failures. A prototype tool has been developed that applies different machine learning algorithms to tackle these problems, and the produced results have been evaluated. To limit the scope, it was decided to focus on UVM [2] test benches and to use log files from simulations as input data. In principle, the same techniques can be applied to other types of test benches and to other types of data, such as waveforms.

II. Related Work

Machine learning has been applied to different classification and clustering problems in related areas.

Chen et al. investigated how decision trees can be applied to diagnosis of failures in large Internet sites [3]. Their results indicated that the decision tree algorithm was successful to their specific task of failure diagnosis.

Lal and Pahwa investigated the classification of root causes of software failures using different machine learning algorithms [4]. Due to limited access to labeled data, their choice of classification algorithms was restricted to those capable of working with unlabeled data.

In two studies, Chakrabarty et al. investigated the effectiveness of decision trees [5] and support vector machines [6] for board-level functional fault diagnosis. The results indicated that both algorithms can be effective for failure diagnosis.

The application of clustering methods for root cause analysis of software failures was investigated by Podgurski et al. [7]. They concluded that clustering is best used in conjunction with a visualization algorithm since the pure clustering results can be unreliable.

While results from previous investigations were generally positive an extensive comparison of different classification and clustering algorithms, with a larger labeled dataset, is lacking. Furthermore, previous work also relied on pure qualitative analysis, particularly for clustering. Addressing these gaps, could provide us with better results and insights. Finally, the format of the input data and the way it is pre-processed can have a significant impact on the performance of the machine learning algorithm [8]. To the best of our knowledge, the application of machine learning techniques to clustering and classification of test failures in hardware verification has not been investigated.

III. Clustering

The first problem to solve is the division of test failures into groups, or clusters, with a common root cause. Algorithms for performing clustering can be either prototype-based, density-based or graph-based. Prototype-based algorithms form clusters around prototypes which can be centroids (centers of clusters) or a probability distribution function. Density-based algorithms represent a cluster as a dense region of samples that are surrounded by regions with lower density. This is useful when the clusters are uneven in size or when there is noise in the data. Graph-based algorithms represent the data as a graph and the clusters as connected subgraphs [6].

An alternative way of performing cluster analysis is to apply algorithms that visualizes similarities between samples in two or three dimensions. This can be useful if the results of ordinary clustering algorithms are unsatisfactory. Finding the right parameters, such as the number of clusters, is difficult. This often leads to less convincing clusters, and since the obtained results only specify which cluster a sample belongs to it can be difficult to draw conclusions about the quality of the results. Therefore, a visualization algorithm can be used in conjunction with a clustering algorithm to provide a better understanding of how samples relate to each other [6].

A. Algorithms

The following clustering algorithms were evaluated: *k*-mean clustering [10], density-based spatial clustering of applications with noise (DBSCAN) [11], and agglomerative clustering [12]. In addition, visualization of the level of similarity between test failures was investigated, based on t-distributed stochastic neighbor embedding (t-SNE) [13] and multidimensional scaling (MDS) [14].

IV. Classification

The second problem to solve is the classification of test failure root causes. This is typically a supervised learning problem. For training, a dataset where each simulation log file has a corresponding root cause is used. The machine learning algorithms then creates a model that can predict the output for new inputs. In this case an input is a log file from simulation and the corresponding output is a failure root cause.

A. *Failure Causes*

For the work described in this paper a taxonomy of test failure root causes was defined, shown in Figure 1. This taxonomy does not claim to be complete or universally applicable, but rather focuses on the types of failures our verification team found most relevant. Some of the root causes are general, while others are closely related to how our reference models are implemented.

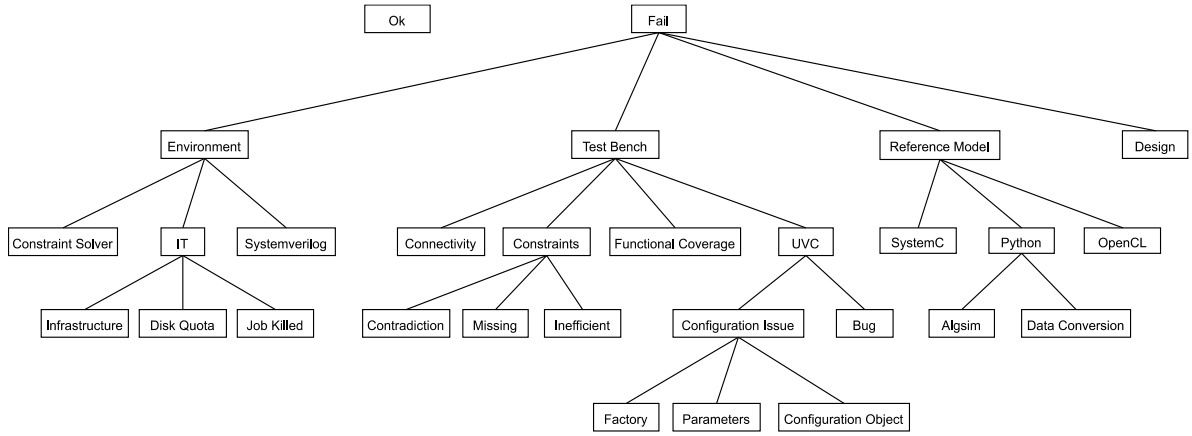

Figure 1. Taxonomy of test failure root causes

B. *Algorithms*

Since there are multiple possible output values a multiclass classifier is required. Some classification algorithms are directly capable of handling multiclass classification. Others are basically binary classifiers but can be generalized to handle problems with multiple classes by using the one-vs-all scheme. This scheme is based on training multiple classifiers of the same type, one for each class, that can determine if a sample belongs to the class or not. The result is then the class that yields the best score [15].

The following classification algorithms were evaluated: logistic regression [16], support vector machines [17] with linear, polynomial and radial kernel functions, naïve Bayes [18], decision tree [19], and the *k*-nearest neighbors algorithm [20], as well as random forests which combines a large number of decision trees [21]. These algorithms were selected because they utilize different strategies that may perform differently on different types of data.

V. Feature Extraction And Selection

The simulation log files used as input consist of semi-structured text. For machine learning algorithms to be able to operate on this kind of data it is necessary to first transform the log file content into numerical feature vectors. What information to use from each type of log message was decided based on domain knowledge. Examples of information gathered from the log files include:

- UVM test name and UVM configuration settings from the command line. In the test benches used for this evaluation, command line arguments are often used to control ranges for random variables, such as image sizes.
- UVM report messages. From these, both general information, e.g. the severity, as well as more specific information, e.g. the category of UVC, is extracted.
- Simulator warning and error messages. This includes errors such as coverage illegal hits and constraint solver failures.

Some of the extracted information is abstracted further. For instance, the simulation time of a message is transformed into a binary value indicating if the event occurred at time zero or later. For most of the information the frequency of occurrence was used as a feature. Counting the frequency of words or phrases is often used in document classification in natural language processing [22].

Our initial feature set consisted of 616 different features. By trimming the amount of information extracted from low severity report messages the number of features was then reduced to 287.

The dimensionality of the feature set a machine learning algorithm is supposed to find solutions in can become very large. One problem with high dimensionality is that the computation time increases. Another problem is that the more dimensions the feature set has, the less likely two samples are to be close to each other. The dimensionality of the feature space can be reduced by performing feature selection or feature extraction. Feature selection methods reduce the dimensionality by removing irrelevant features, whereas feature extraction methods merge existing features [22]. In the evaluation the impact of two feature selection algorithms and one feature extraction algorithm was investigated.

VI. Algorithm Evaluation

The evaluation of the different algorithms was performed using a dataset consisting of log files from 12500 test invocations labeled with their failure root cause. The dataset originated from twenty-nine UVM test benches for different IPs, most of them related to image processing. To create failing test invocations with known root causes, previously encountered typical bugs were manually injected one at a time.

The implementations of algorithms used for the evaluation originated from *scikit-learn* [23]. Pre-processing of log files was implemented using Python regular expressions.

A. Classification

Before the evaluation process, the dataset was divided into a training set containing 10000 samples and a test set containing 2500 samples. The test set was not used until the final evaluation. This reveals how well the model performs on data it has not seen before.

The different classification algorithms were first compared using *k*-fold cross validation [24]. Metrics including accuracy, precision, recall, and F_1-score were used to measure the performance of the classifiers [25]. These metrics were originally defined to handle binary classifications where the predicted result for a sample can be interpreted as either positive or negative. However, they can also be generalized to problems with more than two classes. The most basic metric is *accuracy*, which is the percentage of correct predictions. *Precision* is a measurement of how good the classifier is at not wrongly classifying a negative sample as positive. *Recall* is the complementary metric of precision. It measures how good the classifier is at finding all the sought-after results. The F_1-score, which is the harmonic mean of precision and recall, is often used as a convenient way to compare classifiers [26].

The purpose of the initial comparison was to get an overview of how well the different algorithms performed for the presented problem. Along with the classification algorithms, three different dimensionality reduction algorithms were evaluated to determine their impact on classification performance and computation time. The algorithms provide hyperparameters for tuning their behavior. For the initial comparison default hyperparameters as provided by scikit-learn were used.

The results of the initial algorithm comparison are presented in Table 1. For each algorithm, the accuracy, precision, recall, F_1-score, training time and prediction time were calculated using 5-fold cross-validation. Each algorithm was first evaluated using the baseline feature set, and then once using each of the dimensionality reduction methods. The classification algorithms that yielded the highest F_1-score in the initial evaluation were random forest, decision tree and *k*-nearest neighbors. Dimensionality reduction had a negative impact on the classification metrics of most algorithms. However, dimensionality reduction reduced training and prediction times for all algorithms except the random forest and decision tree algorithms.

Based on the results, the three algorithms that appeared to be best suited were selected and optimized by tuning hyperparameters. Random forest, decision tree and *k*-nearest neighbors were chosen since there was a significant difference between these top three and the remaining algorithms. The *k*-nearest neighbors classifier yielded slightly lower scores when dimensionality reduction using principal component analysis (PCA) was applied, but the computation time was significantly shorter. Therefore, it was decided to use *k*-nearest neighbors with PCA for the optimization.

Table 1: Results of initial evaluation of classification algorithms

Classifier	Accuracy	Precision	Recall	F_1-score	Train (s)	Predict (s)
Baseline feature set						
Random forest	0.899	0.907	0.904	0.905	0.277	0.132
SVC poly	0.556	0.864	0.560	0.609	52.655	56.315
SVC rbf	0.806	0.845	0.800	0.813	17.311	36.422
LinearSVC	0.851	0.856	0.852	0.852	72.463	0.184
Decision tree	0.892	0.901	0.899	0.899	0.342	0.067
Logistic regression	0.841	0.851	0.840	0.842	62.498	0.191
K-neighbors	0.883	0.890	0.887	0.888	0.522	56.618
Naïve Bayes	0.643	0.763	0.652	0.607	0.180	0.933
Dimensionality reduction using PCA						
Random forest	0.891	0.899	0.896	0.897	0.513	0.100
SVC poly	0.760	0.851	0.756	0.779	9.809	15.468
SVC rbf	0.808	0.844	0.803	0.814	5.971	12.239
LinearSVC	0.779	0.801	0.768	0.778	37.075	0.189
Decision tree	0.885	0.893	0.891	0.892	0.806	0.058
Logistic regression	0.794	0.817	0.785	0.793	31.105	0.127
K-neighbors	0.882	0.890	0.885	0.887	0.111	1.683
Naïve Bayes	0.518	0.725	0.510	0.514	0.082	0.338
Dimensionality reduction using SFM SVC						
Random forest	0.893	0.901	0.899	0.900	0.133	0.096
SVC poly	0.650	0.810	0.643	0.675	7.732	12.853
SVC rbf	0.747	0.791	0.752	0.757	4.520	10.773
LinearSVC	0.786	0.790	0.791	0.786	26.048	0.156
Decision tree	0.887	0.897	0.895	0.896	0.121	0.057
Logistic regression	0.752	0.775	0.757	0.757	21.339	0.195
K-neighbors	0.877	0.882	0.883	0.882	0.270	16.335
Naïve Bayes	0.536	0.608	0.579	0.509	0.059	0.234
Dimensionality reduction using SFM RF						
Random forest	0.896	0.904	0.902	0.902	0.126	0.101
SVC poly	0.704	0.820	0.688	0.719	3.813	6.410
SVC rbf	0.781	0.820	0.776	0.788	2.912	6.284
LinearSVC	0.786	0.798	0.784	0.786	19.937	0.136
Decision tree	0.891	0.899	0.897	0.898	0.099	0.058
Logistic regression	0.772	0.786	0.766	0.769	15.400	0.181
K-neighbors	0.878	0.884	0.883	0.883	0.112	4.668
Naïve Bayes	0.575	0.677	0.610	0.534	0.044	0.192

The decision tree and *k*-nearest neighbors algorithms have small hyperparameter spaces and an exhaustive search of all hyperparameter combinations was feasible. For the random forest algorithm, which had a larger hyperparameter space, a random search was used. The results after optimization of the classification algorithms are presented in Table 2. Optimizing the algorithms slightly increased their scores for the classification metrics. Optimization increased the computation time for the random forest and the decision tree classifier whereas it decreased for the *k*-neighbors classifier. The random forest classifier responded best to optimization with regard to the classification metrics.

Table 2: Results and relative change after optimization of classification algorithms

Classifier	Accuracy	Precision	Recall	F_1-score	Train (s)	Predict (s)
Random forest	0.907 +0.9%	0.915 +0.9%	0.913 +1.0%	0.913 +0.9%	8.410 +2936.1%	2.074 +1471.2%
Decision tree	0.896 +0.4%	0.904 +0.3%	0.902 +0.3%	0.902 +0.3%	0.385 +12.6%	0.113 +68.7%
K-neighbors	0.885 +0.3%	0.891 +0.1%	0.891 +0.7%	0.891 +0.5%	0.101 -9.0%	0.994 -41.0%

The final evaluation was performed by training the algorithms on the entire training set and then predicting failure root causes for the test set. Table 3 shows the results for the test set. These indicate that the random forest algorithm is best suited for the problem at hand. Compared to the results obtained by using cross validation the

results of the final evaluation were only marginally changed, which indicates that all three classifiers generalize well to new data.

Table 3: Results of final evaluation of classification algorithms

Classifier	Accuracy	Precision	Recall	F_1-score	Train (s)	Predict (s)
Random forest	0.907	0.916	0.912	0.913	5.344	0.182
Decision tree	0.895	0.902	0.900	0.900	0.111	0.002
K-neighbors	0.880	0.888	0.886	0.886	0.059	0.113

Analysis of the confusion matrix for the random forest classifier shows that the root causes that were most often confused were different types of reference model bugs and RTL bugs. This is expected since an algorithmic bug in a reference model in some cases is indistinguishable from an RTL bug, without referring to some other specification source. Since there were other types of reference model bugs included in the dataset, the overall performance was still quite good.

Figure 2 shows how the performance of the random forest algorithm depends on the number of samples used for training. When the dataset reached 3000 samples the performance gain from adding more data diminished. The F_1-score fluctuates when the number of samples is between 3000 and 6000 but stabilizes after that.

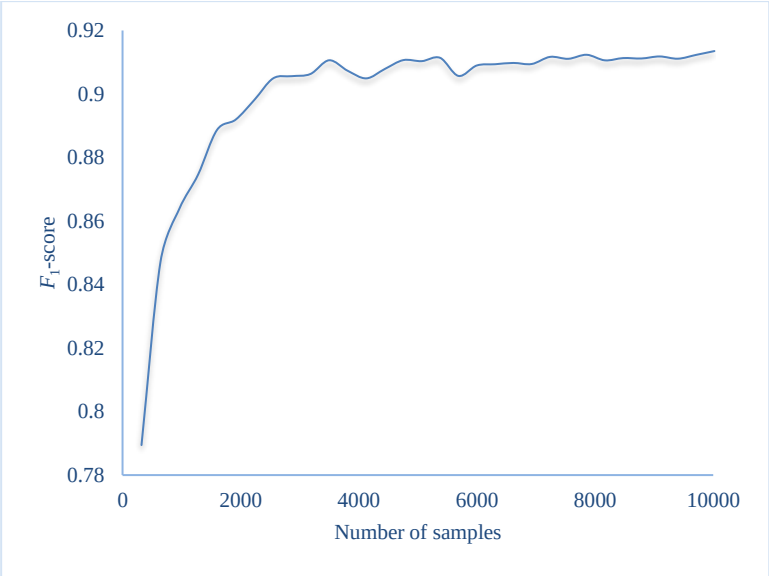

Figure 2. Significance of data

B. Clustering

The evaluation of clustering algorithms is not as straightforward as for classification algorithms. Metrics that compare the quality of the clusters, rather than the labeling of samples, are required. One method of measuring the performance of clustering is to use the adjusted Rand index (ARI) which measures the similarity between two data clusters while ignoring permutations [27]. Another way of measuring the similarity between two data clusters is to use the information theoretic concept of adjusted mutual information (AMI) [28].

Since there is no training and predicting involved in clustering there is no risk of overfitting on the data. This means that a separate test set and cross-validation is not required.

After optimization of hyperparameters, the clustering computed by all algorithms were compared to the correct clustering, the ground truth, using the AMI and ARI metrics. Note that since the goal of clustering is to cluster tests that failed due to the same root cause, as opposed to the classification goal of determining the type of the root cause, the labels used for classification are not sufficient to define the correct clustering. Since the dataset was generated by injecting one bug at a time, a cluster was considered correct when it contained all failing tests created together.

Each of the clustering algorithms was evaluated with and without dimensionality reduction (PCA). The results are presented in Table 4. All algorithms performed better when dimensionality reduction was applied. DBSCAN in conjunction with PCA yielded the highest AMI and ARI scores.

Table 4: Results for clustering algorithms

Algorithm	AMI	ARI	Computation time (s)
Baseline feature set			
K-means	0.505	0.480	0.079
DBSCAN	0.568	0.530	0.086
Agglomerative Clustering	0.540	0.515	0.036
Dimensionality reduction using PCA			
K-means	0.543	0.513	0.041
DBSCAN	0.593	0.545	0.007
Agglomerative Clustering	0.543	0.519	0.006

VII. TOOL IMPLEMENTATION

The most suitable clustering algorithm, DBSCAN, and the most suitable classification algorithm, random forest, were used to implement a tool for automatic clustering and classification of UVM test failures. The implementation is based on Python and scikit-learn. Pre-processing of the log files is implemented using Python regular expressions.

A. *Visualization of Clustering*

Clustering of test failures is combined with the visualization algorithms MDS and t-SNE. Figure 3 shows an example where the clustering computed by the DBSCAN algorithm is compared to the ground truth. In this example there is one set of passing tests as well as two sets of tests that fail due to two separate UVC bugs. The clustering produced by DBSCAN is almost perfect and yielded an AMI score of 0.940 and an ARI score of 0.980. As can be observed DBSCAN identifies a fourth cluster, the single dots, appear to differ from the clusters they belong to. This observation would not have been possible without the visualization algorithms, indicating their usefulness.

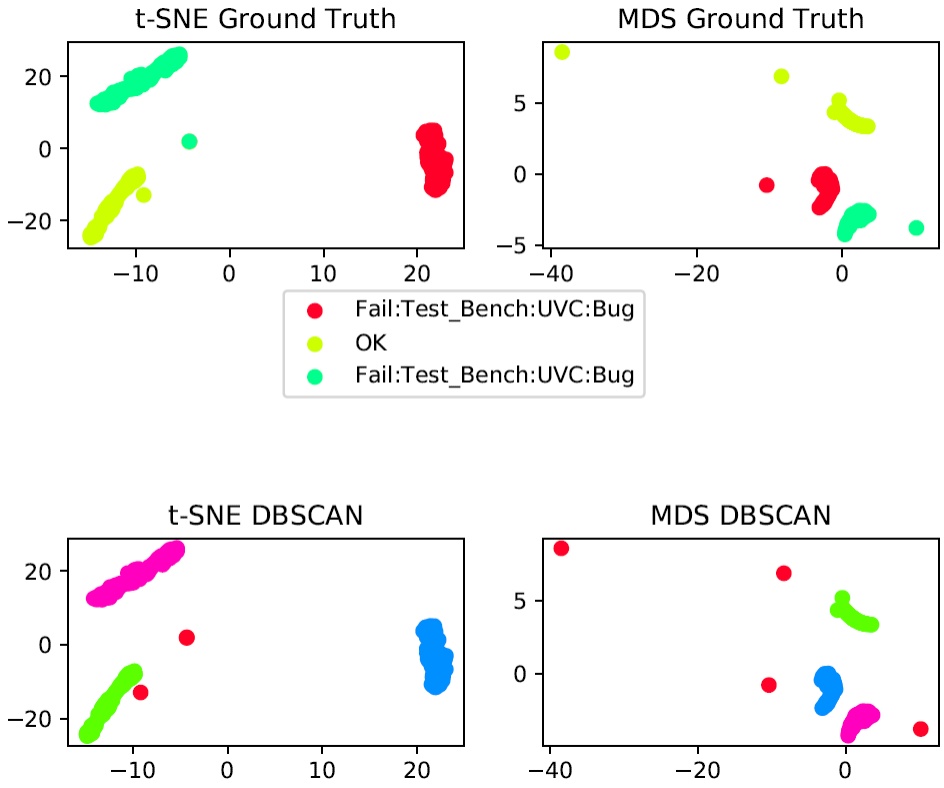

Figure 3. Visualization of clustering of test failures

The example in Figure 3 does, however, show an unusually good clustering. The average AMI and ARI scores obtained by the evaluation were 0.593 and 0.545 respectively.

B. Classification Prediction Confidence

The random forest classifier calculates its confidence about each prediction. Figure 4 shows the relationship between confidence and misclassification rate. The misclassification rates were obtained by using the random forest algorithm trained on the training set to predict outputs for the test set. The rates for each confidence level were then calculated by dividing the number of misclassifications made at that confidence level by the total number of classifications at that level. The results suggest that the confidence can be used to indicate how well a prediction can be trusted. Thus, the tool uses the confidence level to order the predications.

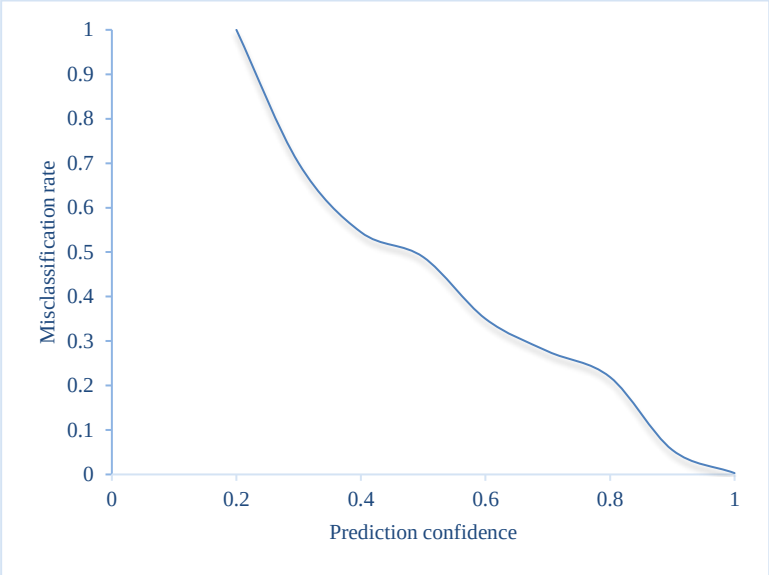

Figure 4. Dependency between predication confidence and misclassification rate

VIII. FUTURE WORK

The results from the final evaluation show that the investigated classifiers generalize well to new data. The used dataset originated from twenty-nine different UVM test benches. However, all these test benches follow the same general structure and style, derived from Axis' UVM and coding guidelines. How well models trained on data from such test benches would generalize to other UVM test benches remains to be investigated. Note that while re-training may be required to improve performance on UVM test benches from some other code base, the pre-processing of the log files is, to a large extent, applicable to any UVM test bench. This is because the pre-processing mostly relies on the structure of UVM report messages.

This investigation used log files from simulation as the input data to clustering and classification. To what extent the verbosity of the UVM test benches affects the results remains to be investigated. Another potential extension would be to investigate the use of other data produced by the simulation, e.g. waveforms or transaction recordings.

While the results of classification were quite good the clustering algorithms did not perform as well as desired. It is possible that the clustering is more sensitive to which features are extracted from the data and that the results can be improved by more careful feature engineering. Also, while basic optimization of the hyperparameters of the algorithms has been performed it is possible that the results can be improved by further tuning.

IX. CONCLUSIONS

Coverage driven constrained random verification typically implies test suites consisting of many invocations of the same or similar test cases. A small number of bugs in the design or verification environment often results in many test failures that need to be clustered and classified. The performance of applying machine learning to these problems has been evaluated.

The result of the evaluation show that machine learning can be effectively applied to classify the root cause of test failures in UVM test benches. The best performing classification algorithm evaluated was random forest with an accuracy of 0.907 and an F_1-score of 0.913.

Applying machine learning to the problem of clustering test failures was less effective. The best performing clustering algorithm evaluated was DBSCAN with the dimensionality reduction method PCA, which yielded an AMI score of 0.593 and an ARI score of 0.545. These results suggest that clustering algorithms, while better than a random process, may not be accurate enough to be completely relied upon. However, when used in combination with visualization the clustering algorithms can provide a good overview of which test failures are caused by a common root cause.

Machine learning classification algorithms and visualized machine learning clustering algorithms therefore show promise as tools for verification engineers to reduce the time invested in analyzing failures in large test suites.

REFERENCES

[1] P. Norvig and S. Russel, Artificial Intelligence: A Modern Approach, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2016.

[2] IEEE Standard for Universal Verification Methodology Language Reference Manual, IEEE Standard 1800.2-2017, 2017.

[3] M. Chen, A. X. Zheng, J. Lloyd, M. Jordan and E. Brewer, "Failure Diagnosis Using Decision Trees," in Proc. 1st Int. Conf. Autonomic Computing, pp. 36–43. Washington, DC: IEEE Computer Society, 2004.

[4] H. Lal and G. Pahwa, "Root cause analysis of software bugs using machine learning techniques," in 7th Int. Conf. Cloud Computing, Data Science and Engineering, pp. 105–111. Washington, DC: IEEE Computer Society, 2017.

[5] F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu, "Adaptive board-level functional fault diagnosis using decision trees," in Proc. IEEE 21st Asian Test Symp., pp. 202–207. Washington, DC: IEEE Computer Society, November 2012.

[6] Z. Zhang, X. Gu, Y. Xie, Z.Wang, Z.Wang, and K. Chakrabarty, "Diagnostic system based on support-vector machines for board-level functional diagnosis," in Proc. 17th IEEE European Test Symposium, pp. 1–6. Washington, DC: IEEE Computer Society, May 2012.

[7] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B.Wang. "Automated support for classifying software failure reports," in Proc. 25th Int. Conf. on Software Engineering, pp. 465–475. Washington, DC: IEEE Computer Socienty, May 2003.

[8] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: a review and new perspectives," IEEE Trans. Pattern Anal. Mach. Intell., vol. 35, no. 8, pp. 1798–1828, August 2013.

[9] J. Wu, Advances in K-means Clustering: A Data Mining Thinking. Berlin: Springer, 2012.

[10] S. Lloyd, "Least squares quantization in PCM," IEEE Trans. Inf. Theory, vol. 28, no. 2, pp. 129–137, March 1982.

[11] M. Ester, H.P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining, pp. 226–231. AAAI Press, 1996.

[12] J. H. Ward, "Hierarchical grouping to optimize an objective function," J. Am. Stat. Assoc., vol. 58, no. 301, pp. 236–244, March 1963.

[13] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," J. Mach. Learn. Res., vol. 9, pp. 2579–2605, November 2008.

[14] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a non-metric hypothesis," Psychometrika, vol. 29, no. 1, pp. 1–24, February 1964.

[15] A. Rocha and S.K. Goldenstein, "Multiclass from binary: expanding one-versus-all, one-versus-one and ECOC-based approaches," IEEE Trans. Neural Netw. Learn. Syst., vol. 25, no. 2, pp. 289–301, February 2014.

[16] D. Cox, "The regression analysis of binary sequences," J. Royal Stat. Soc., vol. 20, no. 2, pp. 215–242, 1958.

[17] C. Cortes and V. Vapnik, "Support-vector networks," Mach. Learn., vol. 20, no. 3, pp. 273–297, September 1995.

[18] R.O. Duda and P.E. Hart, Pattern Classification and Scene Analysis. New York, NY: Wiley, 1973.

[19] W. Y. Loh, "Fifty years of classification and regression trees," Int. Stat. Rev., vol. 82, no. 3, pp. 329–348, December 2014.

[20] N. J. Nilsson, Learning Machines: Foundations of Trainable Pattern-Classifying Systems. New York, NY: McGraw-Hill, 1965.

[21] L. Breiman, "Random forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, October 2001.

[22] F. Sebastiani, "Machine learning in automated text categorization," ACM Comput. Surv., vol. 34, no. 1, pp. 1–47, March 2002.

[23] F. Pedregosa et al., "Scikit-learn: machine learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, November 2011.

[24] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in Proc. 14th Int. Joint Conf. Artificial Intelligence, vol. 2, pp. 1137–1145. San Francisco, CA: Morgan Kaufmann Publishers Inc., 1995.

[25] O. Caelen, "A Bayesian interpretation of the confusion matrix," Ann. Math. Artif. Intell.., vol. 81, no. 3–4, pp. 429–450, December 2017.

[26] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures forclassification tasks," Inf. Process. Manag., vol 45, no 4, pp. 427–437, July 2009.

[27] L. Hubert and P. Arabie, "Comparing partitions," J. Classification, vol. 2, no. 1, pp. 193–218, December 1985.

[28] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for chance," J. Mach. Learn. Res., vol 11, pp. 2837–2854, October 2010.