Clustering and Classification of UVM Test Failures Using Machine Learning Techniques

Andy Troung, Daniel Hellström, Harry Duque, Lars Viklund

Axis Communications AB, Lund, Sweden

AXIS® COMMUNICATIONS

accellera SYSTEMS INITIATIVE

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Problem Statement

- Applying coverage-driven constrained random verification
  - Test suites often consists of a large number of test invocation
  - Same base tests run many times
    - Different seeds
    - Adjusted random distributions
  - One problem often results in many test invocations failing
- Analysis of test failures
  - Which failures have a common cause?
  - What is the type of root cause for each failure?
- Can this analysis be automated using machine learning?

Input Data

- Simulation log files from UVM test benches
- Semi-structured text
- Consists mostly of UVM report messages
- Dataset used for evaluation:
  - 12500 samples labeled with failure root cause
  - Originated from 29 UVM test benches

Features

- Convert input into numerical feature vectors
- Examples of information extracted:
  - UVM test name and UVM configuration settings from command line
  - UVM report messages
  - Simulator warnings and error messages
- Some information is abstracted
- Mostly the frequency of occurrence is used as the feature
- Original set consists of 616 features

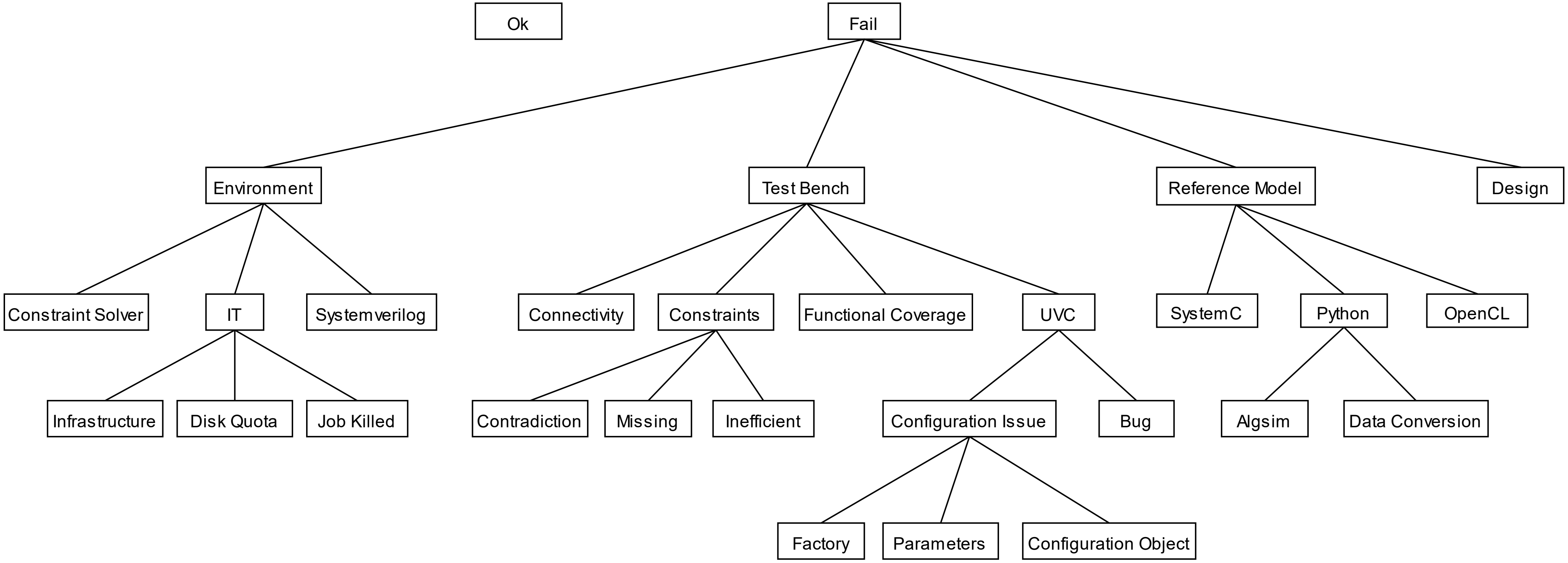
Dimensionality Reduction

- Feature selection
  - Remove less relevant features
  - Manual or automatic
- Feature extraction
  - Merge existing features
- Applied manual feature selection
  - Based on domain knowledge
  - Reduced set down to 287 features
  - Baseline used for evaluation
- Evaluated algorithms for automatic selection and extraction

Classification

- Supervised learning problem
- Training using a dataset associating input with output
  - Input is simulation log file
  - Output is failure root cause
- Algorithm creates a model that predicts output for new input data

Taxonomy of Test Failure Root Causes



Classification Algorithms

- Nine classification algorithms evaluated
  - Selected because they use different strategies
- Three dimensionality reduction algorithms evaluated
  - Two feature selection algorithms
  - One feature extraction algorithm
- Implementations from *scikit-learn*
- Initial evaluation using default hyperparameter settings

Classification Algorithm Evaluation

- Dataset divided into a training set and a test set
  - 10000 samples for the training set
  - 2500 samples for the test set
- Evaluation steps:
  - Initial evaluation using training set and *k*-fold cross validation
  - Optimization of the most promising algorithms
  - Final evaluation using test set

Confusion Matrix

		Predicted	
		YES	NO
Actual	YES	True Positive	False Negative
	NO	False Positive	True Negative

Classification Metrics

- Accuracy
  - How often is the classifier correct?
  - (TP + TN) / Total
- Precision
  - When it predicts yes, how often is it correct?
  - TP / (TP + FP)
- Recall
  - When it is actually yes, how often does it predict yes?
  - TP / (TP + FN)
- F_1-score
  - Harmonic mean of precision and recall

Initial Classification Results

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

Summary of Initial Classification Evaluation

- Three algorithms performed better than the others
  - Random forest
  - Decision tree
  - *k*-nearest neighbors
- Impact of dimensionality reduction
  - Slightly lower scores
  - Significantly reduced computation time

Optimization of Classification Algorithms

- Tune hyperparameters
- Small hyperparameter space
  - Decision tree and *k*-nearest neighbor
  - Exhaustive search
- Large hyperparameter space
  - Random forest
  - Random search

Classification Results After Optimization

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%

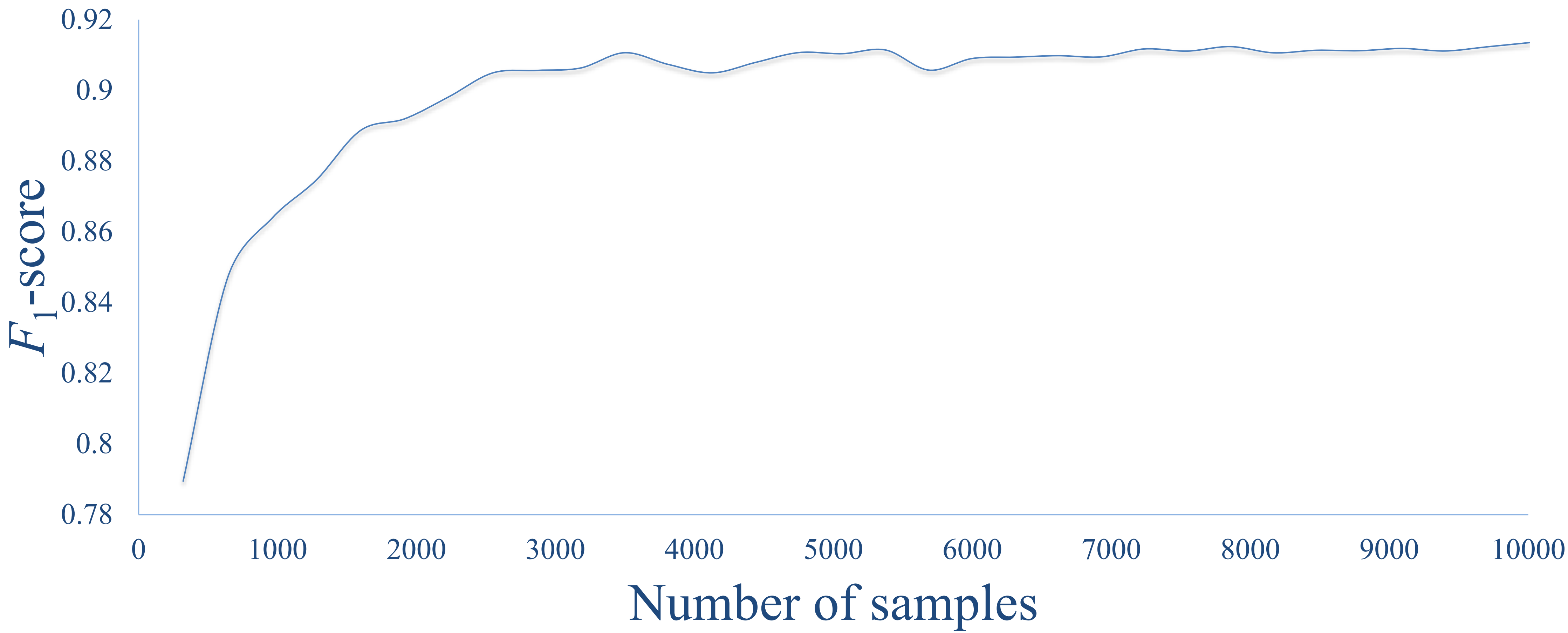
Final Classification Results

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

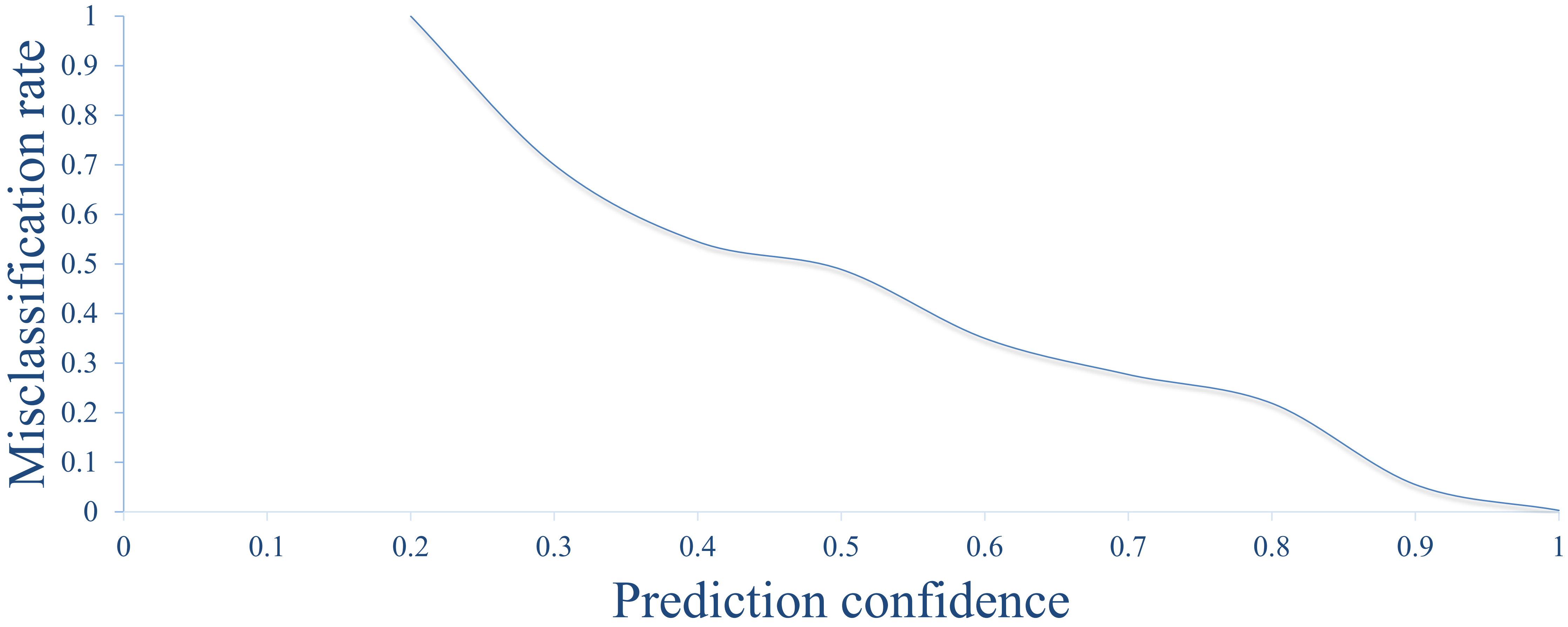
Summary of Classification Evaluation

- Random forest performs best
- Algorithms generalizes well to new data
- Learning and classification run-times are reasonable
- Most common misclassification is design bug vs. reference model bug

Significance of Data



Prediction Confidence



Clustering

- Unsupervised learning problem
- Divide the dataset into clusters (groups)
- Clusters will contain samples that are similar
  - In this case same failure root cause
- Different types of algorithms:
  - Prototype-based
  - Density-based
  - Graph-based

Clustering Algorithms

- Three different clustering algorithms evaluated
  - One of each of the three main types
- One dimensionality reduction algorithms evaluated
- Two different visualization algorithms evaluated
- Implementations from *scikit-learn*
- Hyperparameter settings were optimized

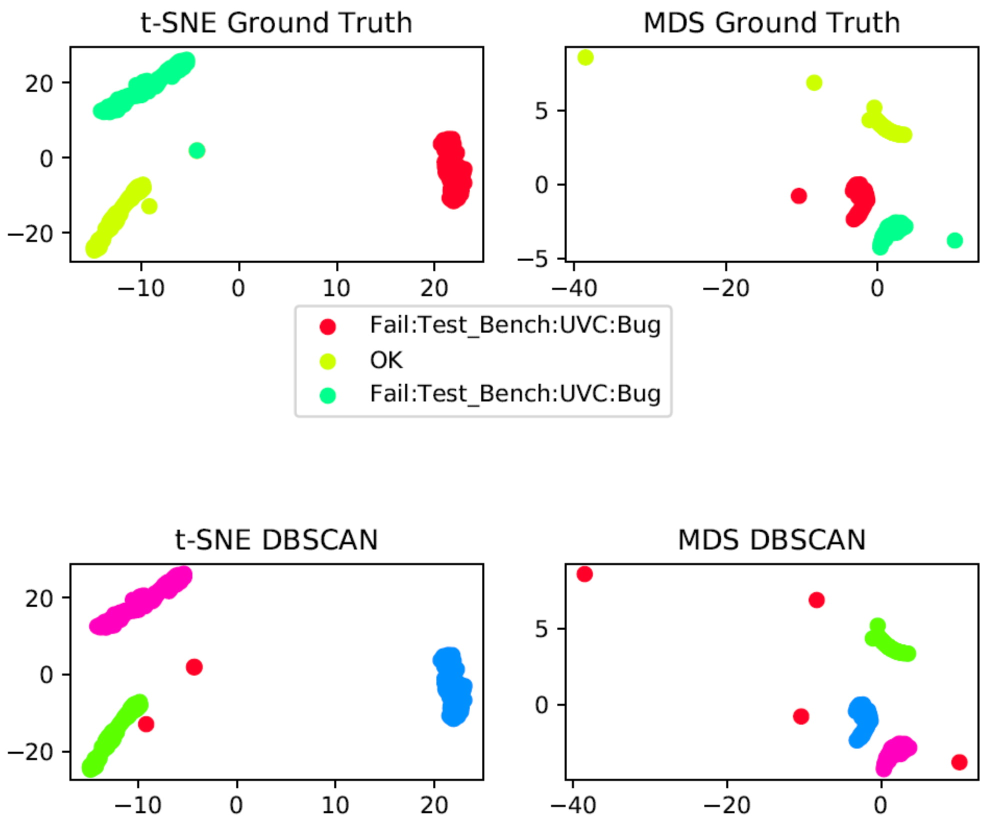
Clustering Algorithm Evaluation

- Metrics used for classification are not applicable
- No risk of overfitting since there is no training
  - No need for a separate test set
  - No need for cross-validation
- Measure similarities between clusters while ignoring permutations
- Adjusted Rand Index (ARI)
  - Based on pair-counting
  - Range [-1, 1]
- Adjusted Mutual Information (AMI)
  - Based on information theory
  - Range [0, 1]

Clustering Results

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

Visualization of Clustering



Tool Implementation

- Prototype tool based on the most suitable algorithms
- Implemented in Python using *scikit-learn*
- Pre-processing of log file using regular expressions
- Interactive visualization of clustering results

Future Work

- Investigate dependency on test bench coding style
- Investigate dependency on report verbosity level
- Use other data in addition to log files
- Improve performance of clustering algorithms
  - More careful feature engineering
  - Tuning of hyperparameters

Conclusions

- Machine learning can effectively be applied to classify UVM test failures
  - Random forest yielded an accuracy of 0.907 and an F_1-score of 0.913
- Machine learning for clustering UVM test failures was less convincing
  - DBSCAN yielded an AMI score of 0.593 and an ARI score of 0.545
- Clustering algorithms can provide a good overview when used in combination with visualization algorithms
- The investigated algorithms show promise as tools to reduce the time invested in analyzing failures in large test suites

Questions