Computer Fraud and Security
ISSN (online): 1873-7056

A Hybrid Ensemble Machine Learning Model for Smart Contract Vulnerability Detection

Gurpreet Singh[1*], Amandeep Kaur[2]

[1,2]Department of Computer Science and Engineering, Punjabi University, Patiala

*Corresponding Author: Gurpreet Singh

Abstract

On Ethereum, a smart contract that has been deployed cannot be patched after it is live. It is therefore essential to identify vulnerabilities prior to deployment and not just as an afterthought. The recent literature focuses largely on complex neural architectures based on raw code or a code-to-graph conversion of a contract's code. These techniques are frequently correct. They also cost a lot of money to train and are not so practical when used routinely or frequently to screen newly deployed contracts. This research is an alternative that is lighter. Random Forest, XGBoost and Gradient Boosting are fused using a hard voting classifier, without using code or graphs directly. Before training, Principal Component Analysis (PCA) reduces a dataset with 10,475 smart contracts and 21 numeric features down to 12 components. Proposed ensemble is evaluated against the four standalone classifiers namely KNN, Random Forest, XGBoost and Gradient Boosting on the same train test split of 70:30. It outperforms all tested individual classifiers with an accuracy of 92.58%, precision of 89.2%, recall of 85.24% and an F1 score of 87.04%.

Keywords: Smart Contract Vulnerability Detection, Ensemble Learning, Random Forest, XGBoost, Gradient Boosting, Principal Component Analysis.

1. Introduction

The blockchain security literature reports numerous smart contract vulnerabilities. Most of the reported vulnerabilities were reentrancy, integer overflow, access control, timestamp dependency. Partly, this is due to a significant issue called code cloning. Developers frequently copy the code of functions and templates from public repositories instead of coding them from the scratch. So, a single vulnerability can spread into thousands of deployed smart contracts. Much recent detection research has shifted to graph neural networks and multi-branch deep architectures that directly process bytecode or opcode sequences, or even control flow graphs. These methods are usually high accuracy. But they also cost more to train and are not always feasible for regular, frequent scanning of new contracts that have been deployed.

This work takes a different, lighter route. Rather than working from raw code or a graph representation, it works from a set of static software complexity metrics: source lines of code, cyclomatic complexity, coupling between objects, depth of inheritance. These are cheap to compute for any contract. Earlier work in object-oriented software design has already shown they correlate with defect proneness in conventional programs [14]. The central question here is simple to state. Can a hybrid ensemble of well-established classifiers, namely Random Forest, XGBoost, and Gradient Boosting combined through voting and paired with a Principal Component Analysis step, detect smart contract vulnerabilities from this kind of feature set more reliably than any one of these classifiers used alone.

Given a feature set that any contract auditing pipeline can compute cheaply, without parsing bytecode or building a control flow graph, how much benefit does combining several classical classifiers actually buy over using the best of them alone. That is a narrower question than the one most graph and bytecode-based techniques set out to answer. But it bears directly on whether lightweight, easily deployed screening tools are worth building alongside the more elaborate methods already in use.

2. Related Work

Ensemble learning has already been applied to smart contract vulnerability detection by several groups, though generally in combination with richer, code level feature sets than the one used here. Sosu et al. [9] built SmartPol around a swarm optimisation-based feature extraction step followed by a semi supervised support vector machine. Tested on close to 49,500 contracts drawn from Etherscan, it reported 96% precision. Zhang et al. [10] took a different approach. These were pre-trained using an information graph created from contract source on seven different types of neural network

architecture before merging them together to form the SCVDIE ensemble. This combination model was more resilient compared to each individual constituent model although the improvement was not quantified by just a number for accuracy in any particular study. CBGRU [15] is another related work where the authors came up with a pipeline approach that used both FastText and Word2Vec along with various deep learning models to extract features. It was found to perform better than the single models evaluated against the SmartBugs Wild dataset. MANDO GURU model by Nguyen et al. [16] integrates control flow graphs with call graphs into a heterogeneous graph attention network, improving detection at the contract level by as much as 63.4% relative to prior graph-based methods.

What these studies share is a reliance on code level or graph level representations, whether raw source, bytecode, or a constructed contract graph. Xu et al. [11] moved closer to the structured feature approach taken here. They proposed a machine learning based analysis model built on features extracted from contract structure rather than raw bytecode, though with a different feature set and classifier selection than the one developed in this study. Table 1 summarises how the present work positions itself relative to this group of studies.

Table 1. Related ensemble based and structured feature based smart contract vulnerability detection studies

Study	Technique	Input representation	Reported strength
SmartPol, Sosu et al. [9]	PSOGSA feature optimisation + semi-supervised TSVM	Extracted contract features	96% precision, ~49,500 contracts
SCVDIE, Zhang et al. [10]	7-network ensemble, information graph pretraining	Information graph	More robust than any single constituent network
CBGRU, Zhang et al. [15]	FastText/Word2Vec + hybrid deep learning	Word embeddings of source	Outperforms single-model baselines
MANDO-GURU, Nguyen et al. [16]	Heterogeneous graph attention network	Control-flow + call graph	Up to 63.4% improvement, contract level
Xu et al. [11]	Machine learning based analysis model	Structured contract features	Not directly comparable; different metric set reported
Proposed	RF + XGBoost + Gradient Boosting, hard voting	Static software complexity metrics + PCA	92.58% accuracy, 87.04% F1

The comparison makes the positioning of this study fairly clear. Rather than adding another ensemble on top of code level or graph level features, the aim here is to test how much detection capability a considerably cheaper feature set retains once several classical classifiers are combined. That angle, to the extent the sources above capture the current state of the field, has received less direct attention than deep, code aware ensembles.

3. Problem Formulation

Smart contracts are increasingly being used in applications like finance, supply chains and identities, increasing the importance of identifying and eliminating vulnerabilities right from the start. After deployment, contracts cannot be patched. A vulnerability not detected by static or dynamic analysis can remain dormant and undetected for a long time. There are some known patterns that are captured by rule-based tools like Oyente, Mythril, and Slither. But they use hand written checks which don't generalise to new and mutated versions of an old bug. This encourages a data driven solution by training a classifier on a corpus of contracts, some of which are vulnerable and some of which are safe, so that the classifier can learn from the data, instead of simply from a fixed set of rules. A similar approach was taken by Xu et al. [11] but with different features and classifiers from the one that was constructed here.

Each of the classifiers has different assumptions about the data. Consequently, each classifier tends to produce different types of prediction errors. A voting ensemble addresses this limitation by combining the predictions of multiple classifiers, thereby reducing the impact of individual errors and improving overall predictive performance. Ensemble learning has been applied by Sosu et al. [9] and Zhang et al. [10] for smart contract vulnerability demonstrating its potential to improve detection accuracy by combining the predictions of multiple classifiers. This work extends this notion. It is based on three

base learners namely Random Forest, XGBoost, and Gradient Boosting and their predictions are combined using hard voting (majority).

This work aims at detecting vulnerable smart contracts given only their static software complexity metrics, without requiring solving their Solidity source code, Solidity bytecode or control flow graph. It explores if a hybrid ensemble of machine learning classifiers, based on dimensionality reduction, combined using a voting strategy, can obtain a higher accuracy, precision, recall and f1 score than the individual classifiers. The goal of the study is to prove that software complexity metrics have enough information to detect vulnerabilities and to prove that a combination of multiple classifiers can improve the performance of the detection as compared with using a single classifier.

4. Proposed Methodology

The general flow of the pipeline is given in Fig. 1. The data is then read and cleaned, and divided into training and testing sets. The four individual classifiers (KNN, random forest, XGBoost algorithm, and the gradient boosting algorithm) are trained on the reduced feature space obtained by Principal Component Analysis.

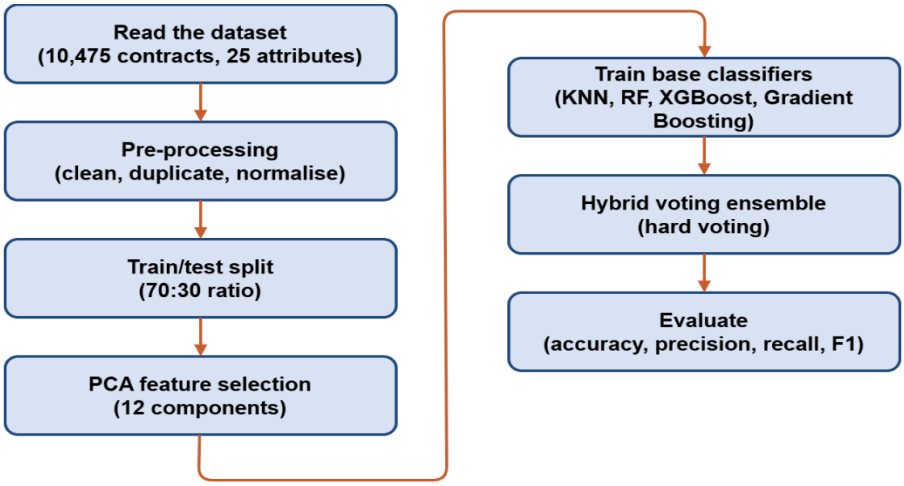

Fig. 1. Pipeline of the proposed methodology.

A random forest, XGBoost algorithm, and the gradient boosting algorithm are then used to form the proposed voting ensemble while KNN is evaluated.

4.1 Dataset Description

The data analysed in this study was compiled by Ajienka [1] and published on Figshare repository. It has 10,475 rows with 25 columns that describe smart contracts, each labelled individually. Each row represents one contract, and contains vulnerability information as well as software measurements like the number of source lines of code (SLOC), logical lines of code (LLOC), comment lines of code (CLOC), number of functions (NF), weighted methods per class (WMC), depth of inheritance tree (DIT), coupling between objects (CBO), and a number of average complexity measures (e.g. average McCabe cyclomatic complexity). Twenty one of the twenty-five columns are numeric and were used as model input. The other four are Solidity file name, contract name, vulnerability label, and contract type are categorical, and are used to label and bookkeep, but not as prediction features. Table 2 provides a list of all the twenty-one numeric attributes as represented in the source dataset.

Table 2. Numeric attributes in the smart contract dataset

Sr. No.	Attribute	Description
1	SLOC	Source lines of code
2	LLOC	Logical lines of code
3	CLOC	Comment lines of code
4	NFd	Number of functions

Sr. No.	Attribute	Description
5	WMC	Weighted methods per class
6	NL	Number of parameters in method
7	NLE	Number of lines in a method
8	NUMPAR	Number of parameters in class
9	NOS	Number of statements in method
10	DIT	Depth of inheritance tree
11	NOA	Number of attributes in a class
12	NOD	Number of dependencies of a class
13	CBO	Coupling between objects
14	NA	Number of aggregations
15	NOI	Number of inheritances
16	Avg. McCC	Average McCabe cyclomatic complexity
17	Avg. NL	Average nested level
18	Avg. NLE	Average number of lines in a method
19	Avg. NUMPAR	Average number of parameters in a method
20	Avg. NOS	Average number of statements in a method
21	Avg. NOI	Average number of inheritances

Several of these attributes are, by construction, close to redundant with one another. SLOC and LLOC in particular tend to move together. NOS and average NOS capture overlapping information at the method and class level. This redundancy is one reason Principal Component Analysis, described in Section 4.3, was applied before training rather than feeding all twenty one attributes directly into the base classifiers.

These metrics were originally developed in the software engineering literature to characterise the internal complexity of conventional object-oriented programs. The reasoning: classes and methods that are large, deeply nested, or tightly coupled to other components tend to be harder to reason about and harder to test thoroughly, and therefore more likely to harbour defects. The working assumption here is that this reasoning carries over to Solidity contracts. A contract with a high weighted methods per class figure, or unusually deep inheritance, is plausibly harder for a developer to audit by hand, and therefore more likely to contain an overlooked flaw. This is a different premise from most of the graph and bytecode based techniques in the wider detection literature, which learn directly from code tokens, bytecode, or a control flow graph. The fundamental premise is that structural complexity metrics, even in the absence of the smart contract's source code, contain sufficient discriminatory information for accurate vulnerability classification.

4.2 Data Preprocessing

The dataset was cleaned for missing data and duplicate rows prior to training. Numeric data columns were normalised to ensure that features with different ranges like SLOC (hundreds) and DIT (single digits) would not have a dominating influence in the calculations of distance based in KNN or bias the variance estimates used in PCA. Vulnerability was label encoded for use in the ensemble classifier. The cleaned data set was subsequently divided into 70% training set and 30% testing set, 7,332 contracts for training and 3,143 contracts for testing. In all experiments reported here this division is used.

4.3 PCA Based Feature Selection

The dataset contains twenty-one numeric features, and some of them are correlated. SLOC (source line of code) and LLOC (logical line of code) features capture similar code characteristics. For instance, NOS (number of statements in a method) is closely related to Average NOS (Average number of statements in a method). Principal Component Analysis was used to reduce the dimensionality of data while preserving the majority of the variance in the data [7]. PCA works by finding a small number of orthogonal directions, the principal components, that capture the largest share of variation in the original feature set, then projecting the data onto those directions. In this study the transformation was configured to retain 12 components. This cut training time for the ensemble classifier and reduced the risk of the model fitting noise associated with the more redundant original features.

4.4 Base Classifiers

KNN classifies a new instance by looking at the labels of its closest neighbours in the training set and assigning the majority class among them, using Euclidean distance to measure closeness [2]. It is a lazy learner: it does no explicit training beyond storing the data. Its accuracy depends heavily on the choice of k. A KNeighborsClassifier with 7 neighbours was used in this study, arrived at through iterative tuning.

Random Forest builds a large number of decision trees, each trained on a bootstrap sample of the training data with a random subset of features considered at each split. Their individual predictions are combined through majority voting [3]. This randomisation reduces the correlation between trees, and in turn the variance of the overall model relative to a single decision tree. Later work by Ren et al. [12] and Liu et al. [13] confirmed this property across a range of applied classification settings beyond the one Breiman originally studied. A RandomForestClassifier with 50 trees was used here.

XGBoost is a gradient boosted tree ensemble. It builds trees sequentially, with each new tree fitting the residual errors of the trees built so far. It incorporates regularisation terms and an efficient approximate split finding algorithm, which make it well suited to large, sparse, tabular datasets [4]. A DecisionTreeClassifier served as the weak learner underlying the boosting procedure in this implementation.

Gradient Boosting shares the same sequential, residual fitting logic as XGBoost. But it follows the more general formulation given by Friedman [5], where each new weak learner is fit to the negative gradient of the loss function with respect to the current ensemble's predictions. It is used here as the third base learner feeding into the proposed voting classifier.

4.5 Proposed Hybrid Voting Ensemble

The core contribution of this work is a voting classifier that combines Random Forest, XGBoost, and Gradient Boosting, illustrated in Fig. 2. Random Forest, XGBoost, and Gradient Boosting classifiers are trained independently on the PCA transformed training features. At prediction time, these three classifiers each cast a vote. The ensemble's final prediction is the class receiving the majority of votes, a scheme generally called hard voting. This differs from soft voting, where classifiers instead contribute predicted class probabilities that get averaged before a decision is made. Hard voting was chosen here for its simplicity. It also does not require the base classifiers' probability estimates to be well calibrated against one another, which is not guaranteed when combining tree based models trained with different objective functions.

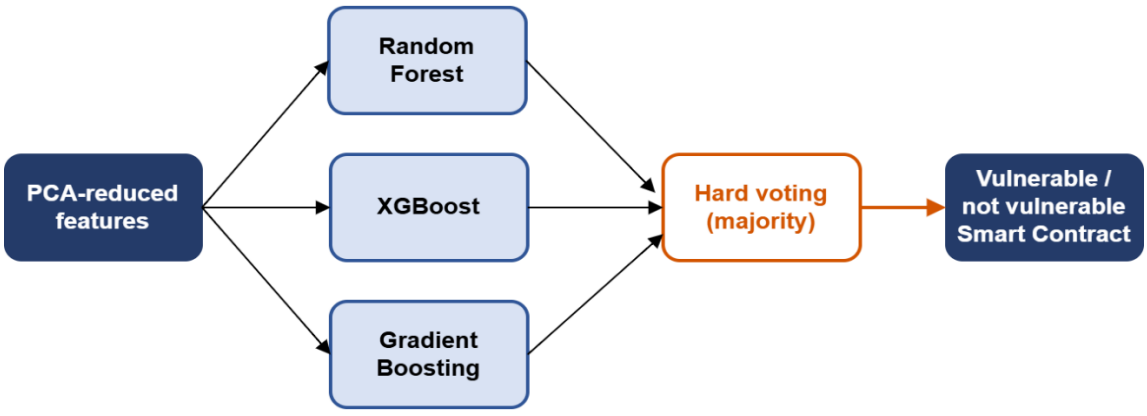

Fig. 2. Architecture of the proposed hybrid voting ensemble.

5. Experimental Setup

5.1 Tools and Environment

Experiments were run on a Windows 11 machine with an Intel Core i5 10th generation processor, 8 GB of RAM, and a 256 GB SSD. Python was used throughout, with Anaconda Navigator managing the environment and Jupyter Notebook used for interactive development. The scikit-learn library provided the implementations of KNN, Random Forest, PCA, the train test split utility, and the voting classifier used to combine the base learners [6]. Pandas and NumPy handled data loading and numeric preprocessing.

5.2 Performance Evaluation Metrics

Four standard classification metrics were used to compare the performance of the proposed and the baseline models are presented in the Table 3. These metrics are true positive, true negative, false positive and false negative. Accuracy is the percentage of correct predicted contracts over the total number of contracts. Precision is the percentage of contracts that are vulnerable and that are correctly predicted. Recall is the percentage of the truly vulnerable contracts that are accurately classified by the model. The F1 score is the harmonic average of precision and recall, and it is a balanced score for classification performance. It can be particularly helpful for binary classification tasks, like smart contract vulnerability detection, where false positives and false negatives are significant factors.

Table 3. Performance metrics

Metric	Formula
Accuracy	(TP + TN) / (TP + TN + FP + FN)
Precision	TP / (TP + FP)
Recall	TP / (TP + FN)
F1-Score	2 x (Precision x Recall) / (Precision + Recall)

6. Results and Discussion

The configuration of every component of the proposed model is summarised in table 4. Table 5 reports four performance metrics for the standalone classifiers (KNN, Random Forest, Gradient Boosting and XGBoost) and the proposed ensemble on the same held out test split, which is 30%. The same information is shown graphically in Fig. 3 for ease of comparison.

Table 4. Configuration parameters used in proposed model

Component	Setting
KNN	n_neighbors = 7
Random Forest	n_estimators = 50, random_state = 1
Gradient Boosting	n_estimators=100, learning rate =0.1
XGBoost	n_estimators=100, max_depth=3, learning rate =0.1
PCA	n_components = 12
Voting classifier	hard voting
Train/test split	70:30

Table 5. Comparative performance of the base classifiers and the proposed ensemble

Algorithm	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
KNN	68.37	46.30	40.52	43.20
Random Forest	74.64	67.18	61.35	64.10
Gradient Boosting	77.85	69.85	63.92	66.75
XGBoost	81.35	74.10	69.35	71.65
Proposed ensemble	92.58	89.20	85.24	87.04

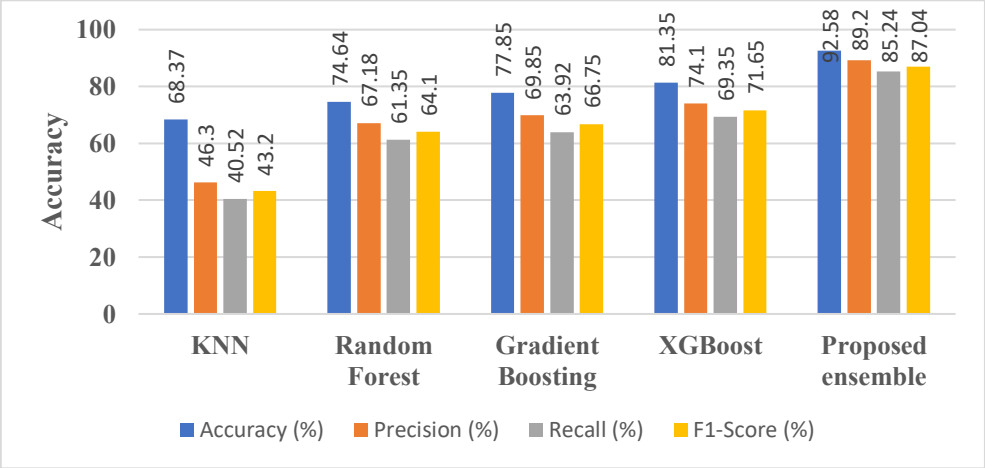

Fig. 3. Comparative performance of the base classifiers and the proposed ensemble.

6.1 Individual Classifier Performance

Random Forest reached 74.64% accuracy. But its recall was a comparatively modest 61.35%, meaning it missed roughly half of the genuinely vulnerable contracts in the test set, even though its overall accuracy looked reasonable. This is a reminder that accuracy alone can be a misleading single metric on a dataset where the vulnerable class is a minority. Gradient Boosting achieved an accuracy of 77.85%, with a precision of 69.85%, a recall of 63.92%, and an F1-score of 66.75%. In all evaluation metrics, it outperformed Random Forest and KNN, which shows that sequential boosting is effective in enhancing the vulnerability detection. It was still not as good as XGBoost and the proposed hybrid voting ensemble, though. KNN performed the weakest among four classifiers with 68.37% accuracy, and a recall of only 40.52%. This fits with the general expectation that distance based methods struggle once the number of features grows and the classes are not well separated in the raw feature space, a limitation PCA is specifically meant to address further down the pipeline. XGBoost produced the strongest results among the individual classifiers, at 81.35% accuracy, 69.35% recall, and an F1-score of 71.65%. The higher recall and F1-score indicate that gradient boosting was more effective at distinguishing vulnerable contracts than either Random Forest or KNN. However, despite outperforming the other standalone models, its performance remained below that of the proposed hybrid voting ensemble, suggesting that combining multiple complementary classifiers provides more robust and reliable vulnerability detection than relying on any single model alone.

6.2 Performance of the Proposed Ensemble

The proposed voting ensemble achieved the best results in all four parameters namely accuracy (92.58%), precision (89.2%), recall (85.24%) and F1 (87.04%) score. The recall gain is most important here, with the best single classifier (XGBoost) achieving 69.35% and the proposed ensemble obtaining a higher than 85.24% score. Even an accurate-looking detection tool that fails to detect 50 percent of vulnerabilities is not very secure. This is in line with the ensemble learning literature overall; specifically, it is known that the combination of classifiers that make partially independent errors generally reduces the variance of the final prediction compared to any individual model [8]. It also supports the specific result reported by Zhang et al. [10] comparing a seven network ensemble to its individual networks on a similar task of

smart contract vulnerability detection, which used a much larger collection of neural network architectures than the ensemble that has been proposed.

Table 6. Performance summary of the proposed ensemble

Metric	Value (%)
Accuracy	92.58
Precision	89.20
Recall	85.24
F1-Score	87.04
Specificity	95.66

Specificity is also increased compared to each individual baseline, with a best individual score of 88.43% for RF, and a score of 95.66%. The ensemble is not just giving up specificity for the sake of gaining recall. It enhances both simultaneously, the more challenging of the two.

6.3 Comparative Analysis

The values in table 7 are not raw values from the table 5, but rather the gains that the proposed ensemble performs against the best individual for each metric. This helps to make the size of the improvement more easily discernible at a glance.

Table 7. Improvement of the proposed ensemble over the best performing individual classifier

Metric	Best individual classifier	Best individual value (%)	Proposed ensemble (%)	Gain (percentage points)
Accuracy	XGBoost	81.35	92.58	+ 11.23
Precision	XGBoost	74.1	89.20	+ 15.10
Recall	XGBoost	69.35	85.24	+ 15.89
F1-Score	XGBoost	71.65	87.04	+ 15.39

Table 7 shows the improvement made by the proposed ensemble model compared to the best individual model on each evaluation measure. The ensemble makes consistent improvements in accuracy, precision, recall, and F1-score with the biggest improvement being the recall metric. This is especially crucial for smart contract vulnerability detection, where the ability to detect vulnerable contracts is more important than gaining a small boost in overall detection accuracy. The results show that the fusion of multiple classifiers by hard voting yields more reliable predictions than any single one of the classifiers.

6.4 Confusion Matrix Analysis

Each model was reported to have accuracy, precision, recall, and f1 score as reported in the source study. It was not reporting the actual number of true positives, true negatives, false positives or false negatives. Each of these four counts is related to the three reported metrics via the definitions given in Table 3, and the number of test set contracts is constant and equal to 3,143. This implies that the counts can be exactly retrieved, by solving the appropriate system of equations for each model, based on its reported accuracy, precision and recall. The results are in Tables 8 and Fig. 4. The following are included here because they provide a clearer pattern of errors than the overall number of errors. No new assumption is introduced in the derivation other than the reported metrics and the known size of the test set, with the reconstructed size of the contracts being read as close approximations of the originals, due to rounding to the nearest whole contract.

Table 8. Derived confusion matrix components (n = 3,143 test contracts)

Algorithm	TP	FN	FP	TN	Specificity (%)
K-Nearest Neighbors (KNN)	378	555	439	1771	80.14
Random Forest	713	449	348	1633	82.43
Gradient Boosting	699	395	302	1747	85.26
XGBoost	741	327	259	1816	87.52
Proposed Ensemble	793	137	96	2117	95.66

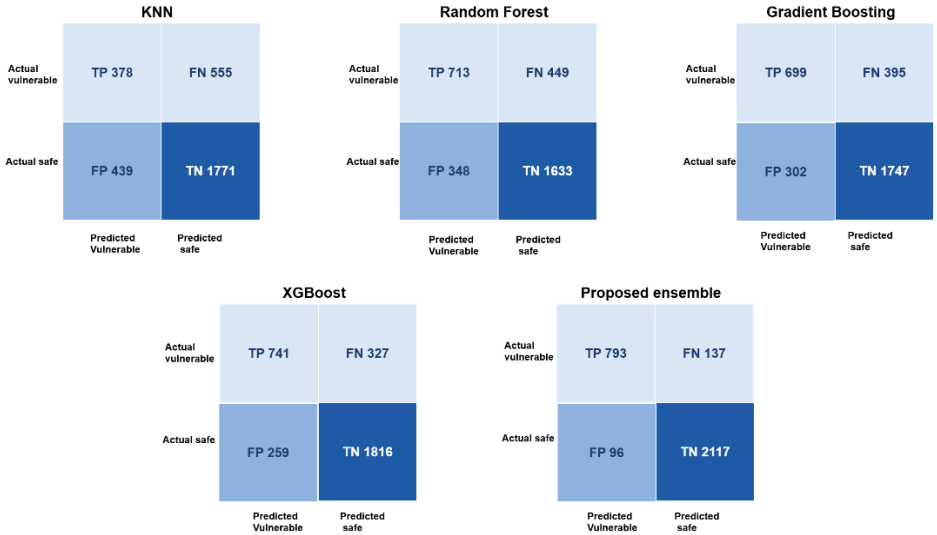

Fig. 4. Derived confusion matrices for the four models (n = 3,143 test contracts).

The reconstructed matrices make the earlier recall discussion concrete. Random Forest, Gradient Boosting, and XGBoost each leave several hundred genuinely vulnerable contracts unflagged, with 449, 395, and 327 false negatives, respectively. KNN misses more vulnerable contracts than it correctly identifies, recording 555 false negatives against 378 true positives. The proposed ensemble reduces false negatives to 137, less than half of the best individual classifier's figure, while also holding false positives to 96, the lowest among all models. Catching more true vulnerabilities while raising fewer false alarms is exactly the pattern one would want from a practical screening tool. It is also a more direct way of seeing the ensemble's advantage than the aggregate F1 score alone conveys.

6.5 Feature Importance Considerations

A natural follow up question is which of the twenty one software metrics contribute most to the ensemble's predictions. Answering this precisely would require access to the fitted models and the underlying feature importances scikit-learn's tree based classifiers expose [6]. That falls outside what the source study reported, and it is not something this discussion can reconstruct after the fact the way the confusion matrices above could be, since importance scores are not algebraically recoverable from aggregate accuracy figures. What can be said, drawing on the broader object oriented metrics literature, is this: coupling between objects and weighted methods per class have consistently been among the stronger defect proneness indicators in conventional software [14]. There is no obvious reason Solidity contracts would behave differently, since both metrics capture a similar notion of how much a single unit of code interacts with or is responsible for the surrounding system. Confirming whether this expectation holds for the specific ensemble proposed here, rather than assuming it carries over from conventional software, is left for further work rather than claimed as a result of the present study.

6.6 Threats to Validity

A few limitations deserve to be stated plainly. First, the dataset assembled by Ajienka [1] labels each contract with a single vulnerability category. The model reported here is therefore answering a binary or coarse grained multi class question, not flagging every distinct vulnerability type a contract might contain at once. A contract with more than one flaw is represented by whichever label the dataset assigns it, which understates the true complexity of the problem. Second, software complexity metrics such as SLOC and WMC are correlated with vulnerability in this dataset, but that correlation need not hold with the same strength on contracts written in a different style or a newer Solidity compiler version. The accuracy reported should not be interpreted as a universal accuracy, but as a characteristic of this dataset and this feature set. Third, the mixture of 70/30 split across the entire dataset was not repeated across multiple random seeds or folds, and therefore some sampling variance is associated with the numbers in Table 5 which could be more precisely determined in a k-fold cross validation study. None of these limitations invalidate the comparison between the base classifiers and the proposed ensemble, since all four models were evaluated under identical conditions. But they do mean the absolute performance figures should not be over interpreted as a general benchmark result.

7. Conclusion and Future Scope

This work proposed a hybrid voting ensemble combining Random Forest, XGBoost, and Gradient Boosting, with PCA applied beforehand to reduce a twenty one feature software metric set to twelve components. It was evaluated against standalone KNN, Random Forest, Gradient Boosting, and XGBoost classifiers on a labelled dataset of 10,475 smart contracts. The proposed ensemble reached 92.58% accuracy, 89.2% precision, 85.24% recall, and an F1 score of 87.04%, ahead of every individual classifier tested on the same split. The improvement in recall was the most practically meaningful part of that gain.

Several directions could extend this work. Testing the proposed ensemble against a bytecode or graph based dataset of the kind used in the wider detection literature would clarify how much of its apparent advantage is specific to the software metric feature set used here, rather than to the ensemble architecture itself. A soft voting variant, together with a systematic comparison of PCA against other dimensionality reduction techniques, would help establish whether the specific choices made in Section 4 are close to optimal or simply reasonable defaults. Extending the labelling scheme to distinguish between individual vulnerability types, reentrancy, integer overflow, and access control flaws separately, rather than a single binary vulnerable versus safe label, would also let the ensemble's strengths and weaknesses be examined at a finer grain. It would make its results more directly comparable to the vulnerability specific figures reported elsewhere in the literature. Finally, integrating a lightweight version of this ensemble into a continuous integration pipeline, so new contracts are screened automatically before deployment rather than audited after the fact, would be a natural next step toward making this kind of detection practically useful rather than purely a research result.

Considered alongside the wider detection literature, the picture that emerges is one of complementary rather than competing approaches. The graph based and bytecode based methods discussed earlier tend to reach for deeper structural or semantic understanding of a contract, at the cost of heavier computation. The ensemble proposed here trades some of that depth for a feature set that is fast to compute and a model that is fast to train, at the cost of working from a coarser signal. Neither is a replacement for careful manual audit. Neither should be treated as a final answer to smart contract security. Together they suggest that vulnerability detection in this domain is served by a range of tools operating at different points on the accuracy and cost trade off, and that a hybrid ensemble of the kind proposed here has a reasonable claim to a place near the fast, low cost end of that range.

8 References

[1] N. Ajienka, "Supervised machine learning for smart contract vulnerability prediction" (v2), figshare, 2020. [Online]. Available: https://doi.org/10.6084/m9.figshare.13417316.v2

[2] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21 to 27, 1967.

[3] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5 to 32, 2001.

[4] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785 to 794.

[5] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," Annals of Statistics, vol. 29, no. 5, pp. 1189 to 1232, 2001.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825 to 2830, 2011.

[7] I. T. Jolliffe, Principal Component Analysis, 2nd ed. New York: Springer, 2002.

[8] T. G. Dietterich, "Ensemble methods in machine learning," in Multiple Classifier Systems, Lecture Notes in Computer Science, vol. 1857, Springer, 2000, pp. 1 to 15.

[9] R. N. A. Sosu, J. Chen, W. Brown-Acquaye, E. Owusu and E. Boahen, "A vulnerability detection approach for automated smart contract using enhanced machine learning techniques," Research Square, preprint, 2022.

[10] L. Zhang, J. Wang, W. Wang, Z. Jin, C. Zhao, Z. Cai and H. Chen, "A novel smart contract vulnerability detection method based on information graph and ensemble learning," Sensors, vol. 22, no. 9, p. 3581, 2022.

[11] Y. Xu, G. Hu, L. You and C. Cao, "A novel machine learning based analysis model for smart contract vulnerability," Security and Communication Networks, vol. 2021, pp. 1 to 12, 2021.

[12] Q. Ren, H. Cheng and H. Han, "Research on machine learning framework based on random forest algorithm," in AIP Conference Proceedings, vol. 1820, no. 1, 2017, art. 080020.

[13] Y. Liu, Y. Wang and J. Zhang, "New machine learning algorithm: Random forest," in Information Computing and Applications, Springer, 2012, pp. 246 to 252.

[14] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476 to 493, 1994.

[15] L. Zhang, W. Chen, W. Wang, Z. Jin, C. Zhao, Z. Cai and H. Chen, "CBGRU: A detection method of smart contract vulnerability based on a hybrid model," Sensors, vol. 22, no. 9, p. 3577, 2022.

[16] H. H. Nguyen, N.-M. Nguyen, H.-P. Doan, Z. Ahmadi, T.-N. Doan and L. Jiang, "MANDO GURU: Vulnerability detection for smart contract source code by heterogeneous graph embeddings," in Proc. ACM Joint European Software Engineering Conf. and Symp. Foundations of Software Engineering (ESEC/FSE), 2022, pp. 504 to 515.