CEUR-WS.org/Vol-2749/paper2.pdf

Exploiting Smart Contract Bytecode for Classification on Ethereum

Selin Sezer$^{1[0000-0003-3635-0160]}$, Clemens Eyhoff$^{1[0000-0002-5193-9494]}$, Wolfgang Prinz$^{1,2[0000-0001-6846-5945]}$, and Thomas Rose$^{1,2[0000-0001-8728-826X]}$

[1] Fraunhofer Institute for Applied Information Technology FIT, Schloss Birlinghoven, 53754 Sankt Augustin, Germany
`{firstname.lastname}@fit.fraunhofer.de`
[2] RWTH Aachen University, Ahornstraße 55, 52074 Aachen, Germany

Abstract. Due to the increase in smart contracts in Ethereum, a need for proper classification has emerged. Although the smart contracts are accessible due to the open nature of the Blockchain, readability is still an issue with respect to the smart contract bytecode. We propose an automated approach for classifying smart contracts that utilize popular text classification methods on the opcode translation of the smart contract bytecode in order to overcome this limitation. Our experiments indicate that the decision-tree-based techniques like Random Forest and Xgboost outmatch the traditional classification tools like Naïve Bayes, Logistic Regression, and SVM once the opcode input is presented as n-gram tf-idf vectors.

Keywords: Blockchain · Text Classification · Smart Contract.

1 Introduction

Being one of the most famous platforms since 2015, Ethereum has enabled many decentralized applications built on top of smart contracts and has been home to roughly more than 20 million smart contracts and hundreds of millions of transactions between the members[1]. This large amount of data has made it hard to get an overview of this large amount of smart contracts and their services. However, to be able to identify similar contracts, with other words to be able to classify them properly, can ease this problem. Proper classification of smart contracts might also enhance the comprehension of the smart contract application trends by finding examples for different categories and therefore contribute to the enhancement of the smart contract programming for which efficiency is essentially needed, and work as a cross-validation tool by providing an overall sense if the smart contract that a user interacts with actually accomplishes what it ensures.

Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Classification of smart contracts is an interesting challange. The only accessible data with respect to smart contracts is the compiled contract bytecode since the smart contracts stay anonymous if the source code is not made public by the developers[18]. The number of open-source smart contracts remains considerably low compared to the ones without published source code and using the source code for classification gives only limited results for the entire network. Therefore, we suggest the *bytecode* of the smart contracts to be exploited for the classification instead.

With this work, we examine the probable use of text classification methods on the opcode translation of smart contracts to determine if the analogy between the instruction sequences in a smart contract opcode and word sequences in texts can help to represent the bytecode in machine learning applications. In that regard, we explain the pipeline of our approach and the results obtained by the application of this pipeline for some predefined classes.

2 Related Work

2.1 Smart Contract Classification

Automated classification means have only been sparsely used for smart contract analysis and comparison. Bartoletti and Pompianu[3] conducted one of the first studies in that regard by analyzing the published source codes of the 811 smart contracts from Ethereum and 23 from the Bitcoin network and creating five categories that represent their data as Financial, Notary, Game, Wallet, and Library. Closely related, Klein[14], and Wöhrer and Zdun[24] provide two studies on smart contract design patterns based on detailed research on literary work. While Klein identifies ten support and six application patterns, Wöhrer and Zdun suggest 18 patterns that are classified into five categories as Action and Control, Authorization, Lifecycle, Maintenance, and Security.

He et al.[11] present an analysis of the Ethereum network with a motivation to take a deeper look at the code reuse in smart contracts. They apply a customized fuzzy hashing logic to create smart contract fingerprints based on opcode and utilize the edit distance between fingerprints to calculate a similarity score that can be used for smart contract clustering. This process leads to four distinct clusters as ERC-20 Clusters, Game Contracts, ICO and AirDrop Contracts, and Other Contracts.

Tian et al.[23] offer a mixed approach based on Bi-LSTM model and Gaussian LDA to classify smart contracts accurately. They make use of the information extracted from smart contract source code like code comments in addition to the source code itself, the application labels fetched from www.stateofthedapps.com, and the account information. The proposed model is found to outperform alternative baseline models for the classification of smart contracts into one of the predefined categories like Entertainment, Tools, Management, Finance, Lottery, Internet of Things, and Others.

There have also been different inspirations that lead to other classification approaches. Security vulnerabilities of smart contracts have motivated Tann et al.[22] into classifying contracts according to having a code vulnerability. They translate the opcode of the smart contracts into one-hot vector sequences which are then used in an LSTM model with some case-dependent modifications. This approach has led to a tool with high detection accuracy that functions on the smart contract bytecode and identifies the security vulnerabilities of a smart contract. Another study led by Chen et al.[6] presented a comparison of different approaches in addition to different feature extraction techniques for detecting Ponzi schemes by using different characteristics of smart contracts coupled with the tf-idf measures of the smart contract opcode. They conclude that the Random Forest algorithm trained to solve a binary classification problem is found optimal for the detection of Ponzi schemes.

2.2 Text Analytics

One major application of machine learning has been text classification. The classification problem itself is defined by Aggarwal and Zhai[2] as the utilization of a set of training entries $D = \{X_1, ..., X_N\}$ with a label of a class picked from a set of k distinct entries with indexes $\{1, ..., k\}$ to build a classification model that maps the descriptive features of each entry to a class label. In order to adapt this definition into texts, texts and documents need to be transformed into a structured feature space to be able to profit from the mathematical components of a classifier model since texts and documents are originally unstructured datasets[15]. Many popular approaches like the bag-of-words model[13], TF-IDF[19] measure, and word embeddings like Word2Vec[17] have been suggested for this purpose.

Deep learning and machine learning techniques have been the primary means for traditional text classification tasks[23]. Logistic regression[12, 16] and the Naïve Bayes classifier[20], some of the earliest and straightforward classification tools, have been made use of to retrieve information. Some of the other methods that were investigated and utilized for texts include non-parametric algorithms like k-nearest neighbor[26], decision-tree-based classifiers such as Random Forest[4, 25] and Xgboost[5], and Support Vector Machine (SVM)[8][15]. Furthermore, the use of the neural network models like LSTM for modeling the complex relations with the data has also been motivated with the recent enhancements on deep learning techniques[23].

While some of the studies mentioned target classifying different smart contract applications, they do not make sense of the context through the smart contract bytecode. Furthermore, the studies that utilize the bytecode are built for a narrow use case and do not address different usage of smart contracts. In our study,we carry the use of bytecode to a broader perspective with the help of text analytics tools.

3 Methodology

The categories to be predicted in our study consist of the application patterns suggested by Klein[14] as *voting*, *auction*, *entity management*, *renting*, *trading*, and *track&trace* because of the extensive research comprehensive research conducted using the literature in addition to the descriptive and well-structured definitions provided for the patterns. Since a smart contract can belong to multiple categories, we decide to apply binary relevance to our problem that enables using separate data representations and algorithms for different categories and allows for the extension of the number of categories in the future work. Finally, our experiments are limited to five text classification algorithms, three of which appear traditional and easy to understand methods namely Naïve Bayes, Logistic Regression, and Support Vector Machine(SVM) and the remaining are decision-tree-based algorithms like Random Forest and Xgboost due to the recent popularity they have. Because of the small number of examples in the dataset, deep learning algorithms (e.g., LSTM) are excluded from the study.

3.1 Data Collection & Preprocessing

The public crypto_ethereum dataset in Google BigQuery[3], which gets updated daily through a fully synced node, is utilized to collect bytecode of all the smart contracts residing in Ethereum and the corresponding addresses. Out of 16 426 716 smart contracts deployed to Ethereum by the data collection day, 6th January 2020, only 283 898 of them have been found to have a unique bytecode. While 84 909 contracts are found to have a published source code after querying etherscan.io, a famous block explorer, the source code of 198 989 contracts remained unknown. Finally, the opcode representation of each smart contract bytecode is obtained using an evm bytecode disassembler based on Python named evmdasm[4] and the function names are extracted from the source code to ease the dataset creation process.

3.2 Dataset Creation

The class-specific datasets which are later used for training and testing are created using the smart contracts with the available source code. Because of the timing constraints of the study and the high number of smart contracts for labeling, a filtering process based on domain keywords is applied to the extracted function names to get potential positive and negative examples for a class dataset. Due to their achievable size, we look at the source code of the potential positives and label them manually according to the class descriptions. However, the high number of potential negatives hinder us from getting a manual investigation over the source code for labeling. Thus, the filter keywords for potential negatives are extended via recursive analysis of the randomly sampled function names and the filtering is kept as strict as possible to refrain from any mislabeled examples in the dataset.

To illustrate the filtering keyword choices better, the Voting class can be examined. The contracts that include directly use case related words *vote* or

[3] https://cloud.google.com/bigquery
[4] https://github.com/ethereum/evmdasm

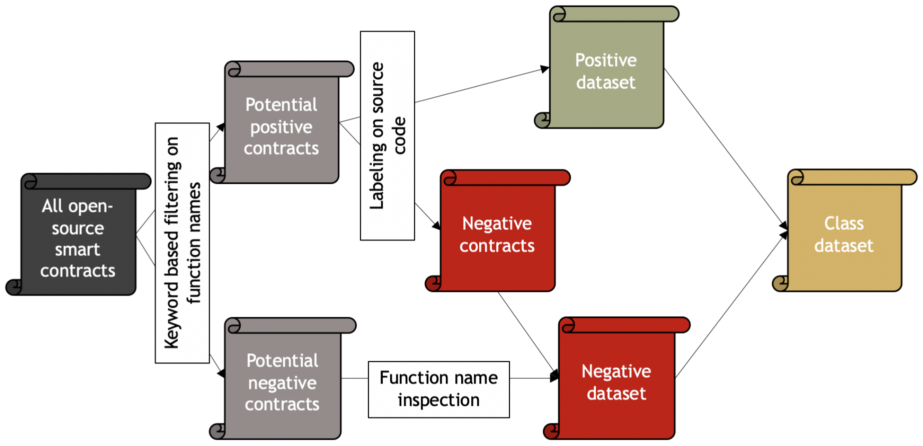
All open-source smart contracts
Keyword based filtering on function names
Potential positive contracts
Potential negative contracts
Labeling on source code
Function name inspection
Positive dataset
Negative contracts
Negative dataset
Class dataset

Fig. 1. Dataset creation sub-steps

ballot keywords in their function names are picked for potential positives. After randomly sampling function names, they are extended to include the smart contracts that have both or *support* and *against* at the same time. For the potential negatives, first a set of negative filtering keywords are created with random function name sampling to find voting use case related words and the contracts that do not have any of the keywords of *vote*, *ballot*, *voting*, *proposal*, *support*, and *against* are chosen as potential negatives.

In the end, the positive and negative examples are merged to create a dataset that represents the class well enough. A summary of this process can be seen in Fig.1. Since there is a different dataset obtained for each class, it is possible to find examples that are positive for one class but negative for another or even positive for multiple classes, e.g. some gaming contracts which apply both entity management and auctioning.

3.3 Model Selection

Initially, the class dataset is split into a larger training&validation set that is used for model training and a smaller test set to determine the generalization ability of the final model. The opcodes of the smart contracts inside the training&validation set are transformed into a feature space using three different vector representations including a count vector (e.g.[9]) which consists of the instruction frequencies inside an opcode, a tf-idf vector which indicates the tf-idf weights of instructions inside an opcode, and an n-gram tf-idf vector (e.g. [27]) that displays the tf-idf (term frequency-inverse document frequency) weights of n successive instructions inside an opcode. The default value for the n value in initial experiments are set as (9-10) as we find it convenient through our preliminary experiments. Afterward, experiments on the training&validation set

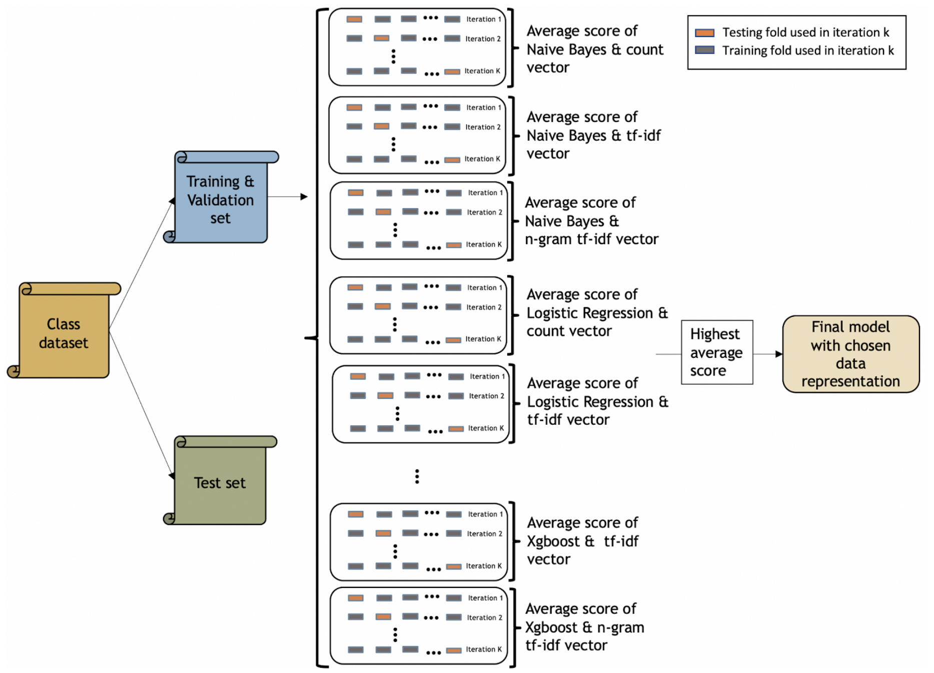
Class dataset
Training & Validation set
Test set
Testing fold used in iteration k
Training fold used in iteration k
Average score of Naive Bayes & count vector
Average score of Naive Bayes & tf-idf vector
Average score of Naive Bayes & n-gram tf-idf vector
Average score of Logistic Regression & count vector
Average score of Logistic Regression & tf-idf vector
Average score of Xgboost & tf-idf vector
Average score of Xgboost & n-gram tf-idf vector
Iteration 1
Iteration 2
Iteration K
Highest average score
Final model with chosen data representation

Fig. 2. Model selection sub-steps

are conducted to obtain an average score using the k-fold cross-validation technique about one of the five chosen algorithms (Naïve Bayes, Logistic Regression, SVM, Random Forest, and Xgboost) coupled with one of these vector representations. Although there are several evaluation metrics that suit highly imbalanced datasets, the comparison and final evaluation metric is chosen to be the Matthews Correlation Coefficient (MCC) as it is not independent from the true negatives[7] and in our case identifying both cases correctly is important. The definition of MCC can be seen in Equation 1.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)\,(TP+FN)\,(TN+FP)\,(TN+FN)}} \tag{1}$$

where TP, FP, TN, and FN indicate the number of true positive, false positive, true negative, and false negative predictions on the test set, respectively. It takes a value between -1 which indicates a complete disagreement between the real values and predictions and 1 which corresponds to a perfect classifier. A random classifier is expected to take a value around 0.

The highest scoring algorithm and data representation are then selected for additional improvements. A summary of the model selection phase can be seen in Fig. 2.

3.4 Model Improvement

Two potential spots for enhancement have been identified with regard to the selected algorithm and data representation. The first one is applicable if n-gram tf-idf vector is found to be the highest-scoring data representation by trying out different n-gram ranges. This is realized by following the same methodology as in the model selection step by testing out different n-gram ranges depending on how much it changes the average score obtained via the k-fold cross-validation technique to increase or decrease the range. As higher n values cause drastically increased computation power, the range is kept as small as possible if the increase in the scores is not meaningful. A summary of this step can be seen in Fig. 3.

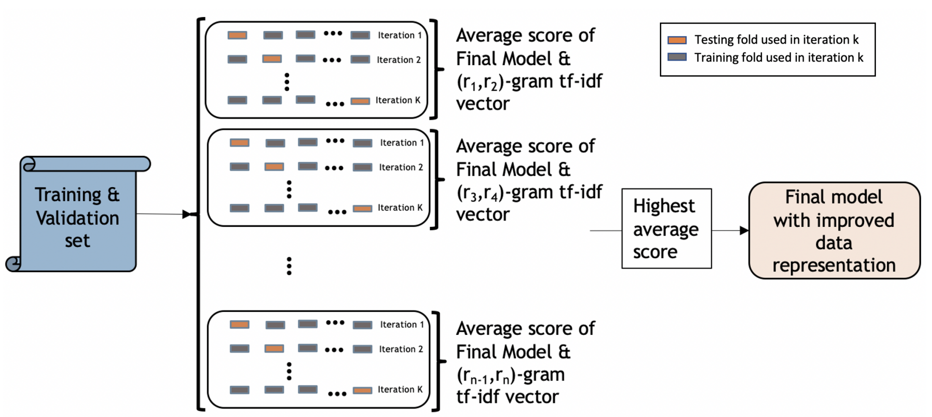
Training & Validation set
Iteration 1
Iteration 2
Iteration K
Average score of Final Model & (r_1,r_2)-gram tf-idf vector
Average score of Final Model & (r_3,r_4)-gram tf-idf vector
Average score of Final Model & (r_{n-1},r_n)-gram tf-idf vector
Testing fold used in iteration k
Training fold used in iteration k
Highest average score
Final model with improved data representation

Fig. 3. Model improvement via n-gram modifications

The second enhancement addresses the imbalance problem. It tests different resampling ratios for positive and negative examples within the training set and compares the results obtained on the validation set once the training&validation set is split into separate sets as a training set and validation set. For resampling, either the positive examples are increased by adding copies of randomly chosen positive examples, or the negative examples are decreased by the removal of some random ones to make the ratio of positive and negative examples better [10]. However, a drastic change is avoided to refrain from overfitting or loss of information. At the end of this process, if the test scores obtained via applying the model on the test set is compatible with the best validation scores, the final model is applied to all the smart contracts to get the final predictions for belonging to the trained class. A summary of this step can be seen in Fig. 4.

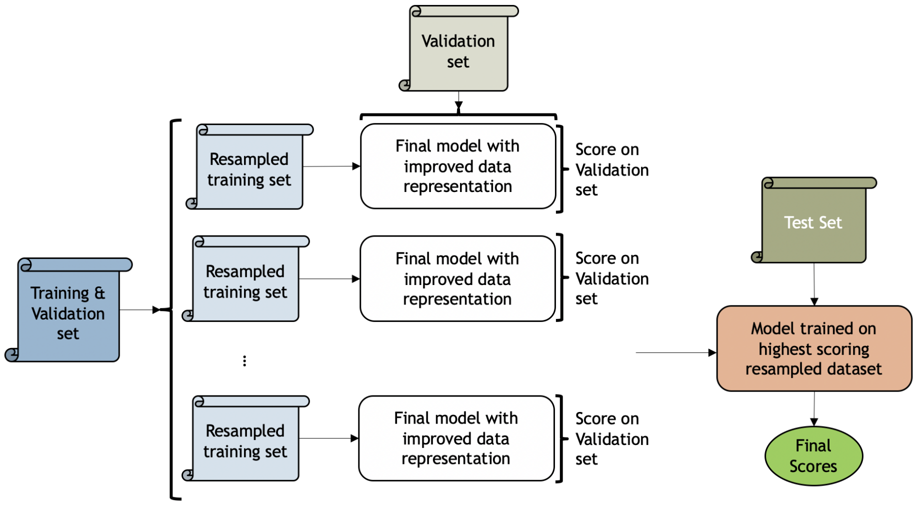
Validation set
Training & Validation set
Resampled training set
Resampled training set
Resampled training set
Final model with improved data representation
Final model with improved data representation
Final model with improved data representation
Score on Validation set
Score on Validation set
Score on Validation set
Test Set
Model trained on highest scoring resampled dataset
Final Scores

Fig. 4. Model improvement via resampling

4 Results

The dataset creation step creates class datasets with different distributions which are depicted in Table 1. The Track&Trace class is excluded from the study and left for future work as there were not any representative filtering domain words found. For the remaining classes, a high imbalance in favor of negative examples can be observed.

Table 1. Distributions of the Class Datasets

Class	Number of Positive Examples	Number of Negative Examples
Voting	779 (1%)	73162 (99%)
Auction	290 (4%)	6655 (96%)
Entity Management	495 (3%)	19274 (97%)
Renting	33 (1%)	6052 (99%)
Trading	738 (12%)	5658 (88%)

The three highest scoring algorithms and data representations are depicted in Table 2. While for some classes like Voting and Renting, the chosen method and data representation outperforms other options by a large margin, the score difference between the top models is not so large for the other classes. On the other hand, the success of decision-tree-based algorithms compared to other tested algorithms becomes apparent. Furthermore, n-gram tf-idf vector is found to be the most successful data structure to represent the smart contract opcode.

Table 2. Top Three Algorithms and Data Representations for Each Class

Class	Algorithm	Data Representation	MCC Score
Voting	**Random Forest**	**n-gram tf-idf**	**0.745190**
	Random Forest	count vector	0.626318
	Random Forest	tf-idf	0.584108
Auction	**Xgboost**	**n-gram tf-idf**	**0.828594**
	Random Forest	n-gram tf-idf	0.781024
	Xgboost	tf-idf	0.766158
Entity Management	**Xgboost**	**n-gram tf-idf**	**0.792610**
	Random Forest	count vector	0.763228
	Xgboost	count vector	0.749616
Renting	**Xgboost**	**n-gram tf-idf**	**0.533938**
	Xgboost	count vector	0.361356
	Random Forest	n-gram tf-idf	0.299207
Trading	**Xgboost**	**n-gram tf-idf**	**0.787698**
	Random Forest	n-gram tf-idf	0.751675
	Xgboost	count vector	0.750335

Table 3 shows the algorithm, data representation, and corresponding average or final MCC score by the application of the model after each step of the learning process for each class. Model Improvement I and Model Improvement II correspond to the testing of different n-gram ranges and resampling ratios, respectively. If there is no improvement observed for a step, no MCC score is given. However, it can already be observed that the effect of the model improvement phase is comparably small when the overall scores are considered. Finally, since there were no prior studies based on the same dataset, we compare our results to the results obtained by the application of a random predictor that randomly assigns 0 or 1 for belonging to the class. The MCC score comparisons of the test results of the final models and random model results show that the classifiers trained on the opcode representations of the smart contract bytecode outperform the random models for each class which is promising for further applications on the smart contract bytecode. Even though almost all the classifiers seem to do a decent job, further investigations are needed for the underperformance of the Renting classifier even though the fairly lower amount of positive examples in the dataset might be a factor. Further details about the methodology and results can be found in [21].

Threats to Validity

There have been two important issues found that might pose a threat to the validity of the test outcomes. While creating the class datasets, the filtering is kept very strict to remove the possible examples with false labels. For example, any smart contract with a *buy* function is discarded from the Trading class. This might cause the class dataset to be unrepresentative for all smart contracts with various application results. Hence, the results obtained on the test set, which is also derived from the class dataset, that shows how well the model generalizes might be inadequate. Another possibly problematic issue can be related to the disassembler used to get the opcode representations of the smart contracts. Some of the commands in smart contract bytecode were unknown and were discarded during the bytecode to opcode translation whose effects are not explored in our study.

Table 3. Learning Process Results of Each Class

Class	Step	Algorithm	Data Representation	MCC
Voting	Model Selection	Random Forest	(9, 10)-gram tf-idf	0.745190
	Model Improvement I	Random Forest	(9, 10)-gram tf-idf	-
	Model Improvement II	Random Forest	(9, 10)-gram tf-idf	0.749153
	Test Results	Random Forest	(9, 10)-gram tf-idf	0.780024
	Random Model Results	-	-	-0.012987
Auction	Model Selection	Xgboost	(9, 10)-gram tf-idf	0.828594
	Model Improvement I	Xgboost	(7, 12)-gram tf-idf	0.855746
	Model Improvement II	Xgboost	(7, 12)-gram tf-idf	-
	Test Results	Xgboost	(7, 12)-gram tf-idf	0.746421
	Random Model Results	-	-	0.038154
Entity Management	Model Selection	Xgboost	(9, 10)-gram tf-idf	0.792610
	Model Improvement I	Xgboost	(7, 15)-gram tf-idf	0.809968
	Model Improvement II	Xgboost	(7, 15)-gram tf-idf	0.876016
	Test Results	Xgboost	(7, 15)-gram tf-idf	0.768002
	Random Model Results	-	-	0.003240
Renting	Model Selection	Xgboost	(9, 10)-gram tf-idf	0.533938
	Model Improvement I	Xgboost	(7, 15)-gram tf-idf	0.567797
	Model Improvement II	Xgboost	(7, 15)-gram tf-idf	0.567797
	Test Results	Xgboost	(7, 15)-gram tf-idf	0.630497
	Random Model Results	-	-	-0.046200
Trading	Model Selection	Xgboost	(9, 10)-gram tf-idf	0.787698
	Model Improvement I	Xgboost	(7, 15)-gram tf-idf	0.806156
	Model Improvement II	Xgboost	(7, 15)-gram tf-idf	-
	Test Results	Xgboost	(7, 15)-gram tf-idf	0.782209
	Random Model Results	-	-	-0.022187

5 Conclusion & Future Work

So far, the classification of smart contracts with respect to application patterns has been based on code analysis. We employed text analytics to analyze smart contracts on the bytecode level. This analysis has allowed us to derive different smart contract use cases. This approach is easy to implement due to its few processing stages and applicable to a larger amount of data since it is not source-code dependent once it is trained. If there is enough data available for training, the final test results show a good rate of classification and makes it promising to treat the smart contract opcode like a text for future applications.

For future work, we plan to extend our approach with a better labeling process of the smart contracts to be able to tackle the representativeness problem of the class datasets. Furthermore, we intend to investigate the effect of the earlier elaboration of resampling in the learning process as well as the effects of other potential resampling techniques. Considering the final success of n-gram tf-idf vectors, another interesting investigation can be realized on performances of different algorithms with different n-gram ranges. Finally, improvements in the feature extraction is aimed with the help of additional smart contract related information.

References

1. Ethereum. https://ethereum.org/, accessed: 2020-10-25.
2. Aggarwal, C.C., Zhai, C.: A survey of text classification algorithms. In: Mining Text Data (2012)
3. Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: platforms, applications, and design patterns. CoRR **abs/1703.06322** (2017), http://arxiv.org/abs/1703.06322
4. Bouaziz, A., Dartigues-Pallez, C., da Costa Pereira, C., Precioso, F., Lloret, P.: Short text classification using semantic random forest. In: Bellatreche, L., Mohania, M.K. (eds.) Data Warehousing and Knowledge Discovery. pp. 288–299. Springer International Publishing, Cham (2014)
5. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 785–794. KDD '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939785, https://doi.org/10.1145/2939672.2939785
6. Chen, W., Zheng, Z., Ngai, E., Zheng, P., Zhou, Y.: Exploiting blockchain data to detect smart ponzi schemes on ethereum. IEEE Access **PP**, 1–1 (03 2019). https://doi.org/10.1109/ACCESS.2019.2905769
7. Chicco, D., Jurman, G.: The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. BMC Genomics **21** (1 2020). https://doi.org/10.1186/s12864-019-6413-7
8. Colas, F., Brazdil, P.: Comparison of svm and some older classification algorithms in text classification tasks. In: Bramer, M. (ed.) Artificial Intelligence in Theory and Practice. pp. 169–178. Springer US, Boston, MA (2006)
9. Erk, K.: Vector space models of word meaning and phrase meaning: A survey. Language and Linguistics Compass **6**(10), 635–653 (2012). https://doi.org/10.1002/lnco.362, https://onlinelibrary.wiley.com/doi/abs/10.1002/lnco.362
10. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering **21**(9), 1263–1284 (2009). https://doi.org/10.1109/TKDE.2008.239

11. He, N., Wu, L., Wang, H., Guo, Y., Jiang, X.: Characterizing code clones in the ethereum smart contract ecosystem (2019)
12. Ifrim, G., Bakir, G., Weikum, G.: Fast logistic regression for text categorization with variable-length n-grams. Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1401890.1401936, https://doi.org/10.1145/1401890.1401936
13. K, S., Joseph, S.: Text classification by augmenting bag of words (bow) representation with co-occurrence feature. IOSR Journal of Computer Engineering **16**, 34–38 (01 2014). https://doi.org/10.9790/0661-16153438
14. Klein, S.: Smart Contract Design Patterns to assist Blockchain Conceptualization. Master's thesis, University of Cologne, Cologne (2019)
15. Kowsari, K., Meimandi, K.J., Heidarysafa, M., Mendu, S., Barnes, L.E., Brown, D.E.: Text classification algorithms: A survey. ArXiv **abs/1904.08067** (2019)
16. Liu, B.: Learning with positive and unlabeled examples using weighted logistic regression. vol. 20, pp. 448–455 (01 2003)
17. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013)
18. Norvill, R., Fiz Pontiveros, B., State, R., Awan, I., Cullen, A.: Automated labeling of unknown contracts in ethereum. pp. 1–6 (07 2017). https://doi.org/10.1109/ICCCN.2017.8038513
19. Robertson, S.: Understanding inverse document frequency: On theoretical arguments for idf. Journal of Documentation - J DOC **60**, 503–520 (10 2004). https://doi.org/10.1108/00220410410560582
20. Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, Sung Hyon Myaeng: Some effective techniques for naive bayes text classification. IEEE Transactions on Knowledge and Data Engineering **18**(11), 1457–1466 (2006)
21. Sezer, S.: Automated Classification of Smart Contracts on Ethereum. Master's thesis, RWTH Aachen University, Aachen (2020)
22. Tann, W.J., Han, X.J., Gupta, S.S., Ong, Y.: Towards safer smart contracts: A sequence learning approach to detecting vulnerabilities. CoRR **abs/1811.06632** (2018), http://arxiv.org/abs/1811.06632
23. Tian, G., Wang, Q., Zhao, Y., Guo, L., Sun, Z., Lv, L.: Smart contract classification with a bi-lstm based approach. IEEE Access **8**, 43806–43816 (2020)
24. Wöhrer, M., Zdun, U.: Design patterns for smart contracts in the ethereum ecosystem. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). pp. 1513–1520 (2018)
25. Xu, B., Guo, X., Ye, Y., Cheng, J.: An improved random forest classifier for text categorization. JCP **7**, 2913–2920 (2012)
26. Yang, Y., Liu, X.: A re-examination of text categorization methods. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. p. 42–49. SIGIR '99, Association for Computing Machinery, New York, NY, USA (1999). https://doi.org/10.1145/312624.312647, https://doi.org/10.1145/312624.312647
27. Zhang, H., Xiao, X., Mercaldo, F., Ni, S., Martinelli, F., Sangaiah, A.K.: Classification of ransomware families with machine learning based on n-gram of opcodes. Future Generation Computer Systems **90**, 211 – 221 (2019). https://doi.org/https://doi.org/10.1016/j.future.2018.07.052, http://www.sciencedirect.com/science/article/pii/S0167739X18307325