Evaluating Machine-Learning Techniques for Detecting Smart Ponzi Schemes

Giacomo Ibba, Giuseppe Antonio Pierro, Marco Di

Evaluating Machine-Learning Techniques for Detecting Smart Ponzi Schemes

Giacomo Ibba
Dep. of Mathematics and Computer Science
University Of Cagliari
Cagliari, Italy
g.ibba14@studenti.unica.it

Giuseppe Antonio Pierro
Centre Inria
Lille Nord Europe
Lille, France
giuseppe.pierro@inria.fr

Marco Di Francesco
Flosslab
Cagliari, Itay
marco.difrancesco@flosslab.com

***Abstract*—Ethereum is one of the most popular platforms for exchanging cryptocurrencies as well as the most established for peer to peer programming and smart contracts publishing [3]. The versatility of the Solidity language allows developers to program general-purpose smart contracts. Among the various smart contracts, there may be some fraudulent ones, whose purpose is to steal Ether from the network participants. A notorious example of such cases are Ponzi schemes, i.e. a financial frauds that require investors to be repaid through the investments of others who have just entered the scheme. Within the Ethereum blockchain, several contracts have been identified as being Ponzi schemes. The paper proposes a machine learning model that uses textual classification techniques to recognize contracts emulating the behavior of a Ponzi scheme. Starting from a contracts dataset containing exclusively Ponzi schemes uploaded between 2016 and 2018, we built models able to properly classify Ponzi schemes contracts. We tested several models, some of which returned an overall accuracy of 99% on classification. The best model turned out to be the linear Support Vector Machine and the Multinomial Naive Bayes model, which provides the best results in terms of metrics evaluation.**

***Keywords*—Blockchain, Smart Contract, Ethereum, Web Scraping, Ponzi Scheme, Fraud Detection, Transactions, Text Classification.**

I. Introduction

The Ethereum's network has millions of participants, and some of them could likely be malicious. Usually, a malicious user is defined as a user who abuses his privileges to harm other users. Since we are considering the Ethereum context, we refer to a malicious user as a network participant that tries to convince other participants to invest their money in high-risk contracts, in which there's no guarantee of getting back their Ethers. These types of contracts are obviously considered fraudulent contracts. A particular type of fraud is that of the Ponzi Scheme [1], a pyramidal model in which investors are recruited with the promise of easy earnings and high-interest rates relative to the initial investment. Other participants of the scheme can also recruit new investors to increase their earnings. This last point is interesting in the Ethereum context because many of the contracts identified as Ponzi Scheme require that a scheme participant recruits a new investor to get his money back [2]. Previously we said that a malicious user abuses his privileges. The main privileges provided by blockchain are [5]:

- **Anonimity**: thanks to inner properties of blockchain the head of the Ponzi Scheme can stay anonymous, and so the other participants of the scheme.
- **Ummutable and potentially unstoppable contracts**: Once a smart contract is deployed, it is not possible to modify it, therefore making impossible to break the scheme. Potentially, smart contract execution can't be stopped by any central authority though some Smart Ponzi put an exit condition, for which that scheme can not be executed anymore. This condition comes true when there are not sufficient funds to pay a participant. Obviously, a malicious user may want to exploit all the advantages provided by the blockchain technology, so this exit condition bring us to think that this types of contract are just either proof of concepts or experiments.
- **Trustworthiness**: Most of the time, a smart contract's source code is available on the blockchain. The availability and the transparency of the source code transmits trust to a user, since a potential participant is more enticed to invest money in a contract which clearly shows how it works instead of a contract which keeps hidden its inner properties.

Actually, some of the identified Smart Ponzi are just experiments [7] since the nature of the contract is explicitly explained by the contract's creator with comments or with the names of the variables and functions. Terms like participants, investments, payout, investors are quite common inside these contracts. Despite some of these contracts being just experiments, the problem of possible Ponzi Schemes scams is real, and it is important to prevent it from happening. To do so, one must identify Ponzi contracts and classify them; this is possible because a Ponzi Scheme contract is recognizable by its execution flow and some well-defined rules. The cyclic flow ensures that the head of the pyramid (who is the one that starts the scheme) finds potential investors. Likewise, investors recruit new participants. Funds from new participants are used to pay the previous investors that joined the scheme. The scheme goes on until it is broken. The general rules of a Smart Ponzi are:

1) A minimum investment is required to join the Ponzi scheme.

2) Investors starts to get paid only if funds are sufficient.
3) If new investors are not available anymore, the scheme is broken.
4) If funds are not sufficient to pay the investors, the scheme is broken.

These rules are common to all Smart Ponzi contracts despite some details that can be slightly different. For example, the minimum fee required to join the scheme differs from contract to contract. In addition to these rules, there are some other elements common to all Smart Ponzi. These are:

1) The contract's **owner**, which starts the scheme.
2) **Investors**, which are defined trough their address and their balance.
3) An array or mapping that stores all the scheme participants.
4) A function that allows a new participant to join the scheme.
5) A function to pay the participants.

It is also interesting to notice that there are different categories of Ponzi schemes contracts. Previous work [5] defined four categories and a taxonomy of Smart Ponzi. First, we have tree-shaped schemes that use a tree data structure to induce an ordering among users. In this particular category of schemes, the user who has just joined the tree must indicate the participant who invited him. The user who invited the new participant becomes his parent node in the tree. Another category is that of chain-shaped schemes, which are a special case of tree-shaped schemes. These types of schemes multiply the investment by a constant factor. Waterfall schemes have a different logic of money distribution. Here each new investment is poured along the chain of investors so that each can take their share. Eventually, we have Handover schemes, where the fee to join the scheme is determined by the contract. This fee gets an increased value each time a new investor joins the scheme.
Looking at the already classified Ponzi Schemes contracts on the Ethereum blockchain, we noticed that a lot of them were published between 2016 and 2017. We aim to build a classifier trained with all the contract that have been spotted in the past few years as well as some which don't belong to this group, as they are not Ponzi schemes. It is interesting to notice that a lot of the already known schemes are very similar between them, and it is clear that a lot of developers took inspiration from others' work. We can find also different versions of the same contract with slight differences (variable names or some unimportant operations).

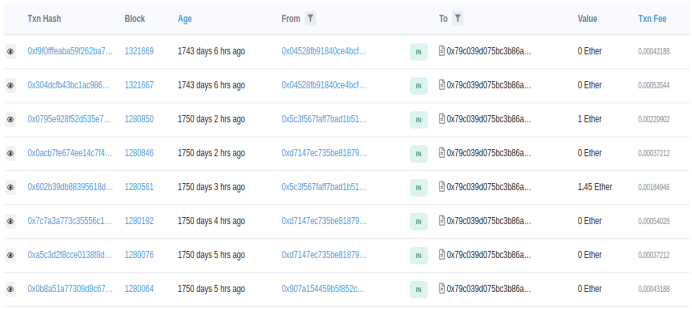
II. STATE OF ART

The first Ponzi scheme contracts date back to 2015, the year of the launch date of Ethereum, but almost all of the Smart Ponzi contracts of our dataset are dated between 2016-2018. Since they appeared in the blockchain (not only in that of Ethereum but also in that of Bitcoin) the Ponzi scheme and fraudulent contracts in general [11] have aroused great interest in the scientific community. This is because these schemes are potentially dangerous for participants who invest their money unconsciously and that the inherent properties and advantages offered by a blockchain make them even more effective. The main advantage that a fraudulent smart contract can take advantage of is that of the guarantee of anonymity offered by the blockchain. This makes a scam virtually impossible to track, which is why such schemes need to be spotted. Currently, several solutions have already been explored to identify Ponzi schemes within the blockchain, both in that of Bitcoin and in that of Ethereum. To recognize Smart Ponzi within Bitcoin, several approaches were used, including those oriented to data mining [8]. The idea is to apply data mining techniques to detect Bitcoin addresses related to Ponzi schemes, considering features like the lifetime of the address, the activity days, the sum of all the values transferred to the address and others; the problem is seen as a binary classification problem, in which a classifier must recognize between 'Ponzi' and 'non-Ponzi'. In the field of Bitcoin, using survival analysis, factors that affect scam persistence have been spotted [13]. Vasek and Moore found out that when the scammer interacts a lot with their victims, the scam's life is increased, and that scams are shorter-lived when the scammers register their account on the same day that they post about their scam.
The Ethereum blockchain has also aroused great interest in the search for Ponzi scheme contracts. Previous work [5] examined all the contracts with a certified source code, determining if they were implementing a Ponzi scheme. They also analyzed the source code of smart contracts (when available) and discovered that most of the contracts share common patterns. For the classification of Smart Ponzi on Ethereum, a proposed model classifies whether a contract implements a Ponzi scheme or not by using features extracted from the transaction history and from the opcodes of smart contracts [9]. In general, we can say that data science approaches are widely used not only to detect scams but honeypots in general [10]. Our approach is always based on solving a binary classification problem, but it differs as it is based on natural language processing techniques. In fact, we used the source code (when available) of smart contracts to create our features. The idea is that Smart Ponzi have a well-defined structure and use meaningful terms that identify them. It is very unlikely that there are Ponzi scheme contracts that do not make their source code clear, or that in any case try to make their operation ambiguous, because a participant is more inclined to invest his money in a contract that clearly shows their functioning, rather than in a contract whose functionality is not known. In the next sections we will see in detail our proposed approach for the classification of Smart Ponzi schemes.

III. SCRAPING

The use of machine learning concerning cryptocurrencies is becoming increasingly important, especially for the classification of smart contracts and predictions related to the trend in the currency market [12]. Our idea is to build a machine learning model taking advantage of Natural Language Processing (NLP) techniques. In particular, we want to exploit Text Classification techniques to classify Ponzi schemes contracts,

but first, we must determine what textual features will be part of our dataset. All Ponzi schemes contracts have a common execution flow and structure, so, we may take advantage of this information. The first text feature we identified is the source code of the smart contracts when available. We had this feature for the entire dataset. Another text feature is the contracts' Opcode. We collected Opcodes for the entire dataset since the source code of all the collected contracts was available. The idea is that since Smart Ponzi contracts have a similar structure, their Opcodes will be similar as well. The last text feature consists of pieces of information extracted from contracts' transactions. Again, we took advantage of the cyclic execution flow of a Ponzi scheme and key terms. The



Txn Hash	Block	Age	From	To	Value	Txn Fee
0xf9f0fffeaba59f262ba7...	1321669	1743 days 6 hrs ago	0x04528fb91840ce4bcf...	0x79c039d075bc3b86a...	0 Ether	0.00043188
0x304dcfb43bc1ac986...	1321667	1743 days 6 hrs ago	0x04528fb91840ce4bcf...	0x79c039d075bc3b86a...	0 Ether	0.00053544
0x0795e928f52d535e7...	1280850	1750 days 2 hrs ago	0x5c3f567faff7bad1b51...	0x79c039d075bc3b86a...	1 Ether	0.00220902
0x0acb7fe674ee14c7f4...	1280846	1750 days 2 hrs ago	0xd7147ec735be81879...	0x79c039d075bc3b86a...	0 Ether	0.00037212
0x602b39db88395618d...	1280561	1750 days 3 hrs ago	0x5c3f567faff7bad1b51...	0x79c039d075bc3b86a...	1.45 Ether	0.00184946
0x7c7a3a773c35556c1...	1280192	1750 days 4 hrs ago	0xd7147ec735be81879...	0x79c039d075bc3b86a...	0 Ether	0.00054028
0xa5c3d2f8cce01388d...	1280076	1750 days 5 hrs ago	0xd7147ec735be81879...	0x79c039d075bc3b86a...	0 Ether	0.00037212
0x0b8a51a77309d8c67...	1280064	1750 days 5 hrs ago	0x907a154459b5f852c...	0x79c039d075bc3b86a...	0 Ether	0.00043188

Fig. 1: Example of contracts transactions

flow consists in:

1) Contract creation by the pyramid leader.
2) Recruitment of investors.
3) The just recruited investors recruit new participants as well.
4) Payment of the investors.
5) Repeat from point 2 until the scheme is broken.

Considering this execution flow, it's reasonable to think that in Ponzi schemes contracts, we will find transactions performed to execute the function to allow a new participant to join the scheme and transactions that execute functions to pay the investors. Usually, these functions have well-defined names such as join, enter, pay, payout, and similar terms; the idea is to take advantage of these pieces of information to recognize a Smart Ponzi.

The next step in our recognition procedure is the feature extraction. All the text information is available on Smart-Corpus [6]. All one needs is the contract's address, and then scraping from the web page its relative source code, opcode, and transactions is straightforward. We started from all the collected contracts' addresses, and then we used Python to perform a web scraping task to retrieve all the needed information. In particular, we used the *beautifulsoup* module to perform web scraping and clean the text information from HTML tags and elements.

IV. CLASSIFICATION

We started from a dataset made of already known Ponzi schemes and of contracts that are not Ponzi schemes. We collected 171 Smart Ponzi contracts and 1500 contracts that are not Ponzi schemes, so our dataset is strongly unbalanced.

```
contract PiggyBank {

  struct InvestorArray {
      address etherAddress;
      uint amount;
  }

  InvestorArray[] public investors;

  uint public k = 0;
  uint public fees;
  uint public balance = 0;
  address public owner;

  // simple single-sig function modifier
  modifier onlyowner { if (msg.sender == owner)
      _; }

  // this function is executed at initialization
       and sets the owner of the contract
  function PiggyBank() {
    owner <= msg.sender;
  }

  // fallback function - simple transactions
       trigger this
  function() {
    enter();
  }

  function enter() {
    if (msg.value < 50 finney) {
        msg.sender.send(msg.value);
        return;
    }

    uint amount=msg.value;


    // add a new participant to array
    uint total_inv = investors.length;
    investors.length += 1;
    investors[total_inv].etherAddress = msg.
        sender;
    investors[total_inv].amount = amount;

    // collect fees and update contract balance

      fees = amount / 33;              // 3% Fee
      balance += amount;                //
          balance update


     if (fees != 0)
     {
      if(balance>fees)
{
        owner.send(fees);
        balance -= fees;                    //
            balance update
}
     }


   // 4% interest distributed to the investors
    uint transactionAmount;

    while (balance > investors[k].amount * 3/100
         && k<total_inv)  //exit condition to
        avoid infinite loop
    {
```

```
64
65      if(k%25==0 &&  balance > investors[k].
            amount * 9/100)
66      {
67       transactionAmount = investors[k].amount *
             9/100;
68       investors[k].etherAddress.send(
             transactionAmount);
69       balance -= investors[k].amount * 9/100;
                                  //balance update
70       }
71      else
72      {
73       transactionAmount = investors[k].amount
             *3/100;
74       investors[k].etherAddress.send(
             transactionAmount);
75       balance -= investors[k].amount *3/100;
                                      //balance
             update
76       }
77
78       k += 1;
79     }
80
81     //-----------------end enter
82   }
83
84
85
86   function setOwner(address new_owner) onlyowner
          {
87       owner = new_owner;
88   }
89 }
```

Listing 1: Example of a Smart Ponzi, the PiggyBank contract

We saw the Ponzi schemes contracts classification problem as a binary classification problem and we assigned the binary target variable with value 1 for a Smart Ponzi and 0 for other contracts. Section IV-C provides an overview about the models used for the Ponzi schemes contracts classification problem.

A. AST EXTRACTION

To have easy access to the contracts' variables, statements, conditions, and all the constructs in general, we extracted all the Abstract Syntax Trees (AST) from our samples. We used python as the programming language to build our solution. Extracting AST from contracts using python is a trivial task thanks to the *solidity_parser* module. The module only requires the user to load the desired contract for which the AST is then extracted in a JSON file after parsing the source code.

B. SEMANTIC INFORMATION EXTRACTION

To extract information from AST, we must parse it. First, to parse a Solidity contract AST we must know its structure, defined by Solidity's grammar. Checking grammar, we know how our AST fields are defined; some of the types defined are trivial, like *VariableDeclaration*, which contains only a variable name and its type. Other nodes could contain quite complex types and require a deep exploration of the subtree. Types like *BinaryOperation*, *MemberAccess*, and *IndexAccess* (for example) require a recursive exploration of the subtree because they could have operations of the same type (i.e. a BinaryOperation could have another BinaryOperation inside). Once we built our AST parser, we extracted all the potential information relevant from a semantic viewpoint. Variables names, struct names, conditions inside constructs (like for, if, while), and the operations performed inside contracts are all relevant because we saw previously that some terms and some operations are recurrent in Smart Ponzi schemes. To extract a semantic document from AST, we took all the labels previously listed and saved them in different structures. We tried to look for the extracted features as much as possible similar to something that suggests a natural language. To do this, we built a dictionary that replaces all the mathematics and logic operators with their semantic meaning (i.e. the symbol '=' is replaced with 'is assigned with value'). To add semantic to the resulting document, we added meaningful words related to the type of construct. For example, before a variable name or a struct name we added 'declaring', and after a struct name we added 'with members' followed by struct members name.Since this approach could sound tricky and probably not so easy to understand, we provide an example of the 'semantization' of the source code. Suppose to have a Solidity line of code like the one below.

```
1  if (msg.value < 50 finney)
```

The entire line of code is an 'IfStatement' field of the Solidity AST. In this specific case the expression inside this IfStatement, which is 'msg.value $<$ 50 finney', is a BinaryOperation. Our parser decomposes the expression as follows: the symbol '$<$' is the operator of the BinaryOperation, while 'msg.sender' and '50 finney' are respectively the left part of the operation and the right part of the operation. The right part is simple, since it includes just a numerical value and a Solidity's keyword. The left part is not trivial, because it is a MemberAccess expression and it needs further decomposition. In this case, the expression is not too complex, since it has only one identifier, which is 'sender', so exploring the subtree requires only one additional step. Now that we decomposed the IfStatement the parser will build the corresponding line in the semantic document.

```
1  if msg value is less than 50 finney
```

Following this approach for all contracts, we were able to extract semantic documents from contracts' AST and perform text classification.

C. MODELS

The models tested for the Smart Ponzi classification problem are:

- **Decision Tree**.
- **Support Vector Machine** (SVM).
- **Multinomial Naive Bayes**.

Decision Trees are among the simplest classifiers and can be used for both classification and regression problems. As the name suggests, they work building a tree representation starting from data. We choose to test the Decision Tree just because it is one of the simplest models and because any

missing values in the data do not affect the process of building a tree to any considerable extent. One possible problem is that this particular model is inadequate for applying regression and predicting continuous values (but this is not our case). Other problems consist in the average time required to train the model, which is very long, and that computations can get quite complex, especially when compared with other models. Consequently, despite being simple, the Decision Tree is quite an expensive model. The second model we tested is the SVM, which maps training examples to points in space. This model is memory efficient, and it works well when the two classes are linearly separated. The problem is that this model is not suitable for large datasets, and it is not easy to keep track of decisions taken by the SVM, and therefore, it is not simple to explain the obtained results. The Multinomial Naive Bayes consider a feature vector where a given term represents the number of times it appears or very often i.e. frequency. We choose to test this model because it has a low computational cost, can work with large datasets, and is well known to perform well in text classification problems, making it a perfect choice for our classification problem. The main disadvantage is that using a Naive Bayes is difficult to get the set of independent predictors for developing a model. Considering the models' inner properties, we can make some

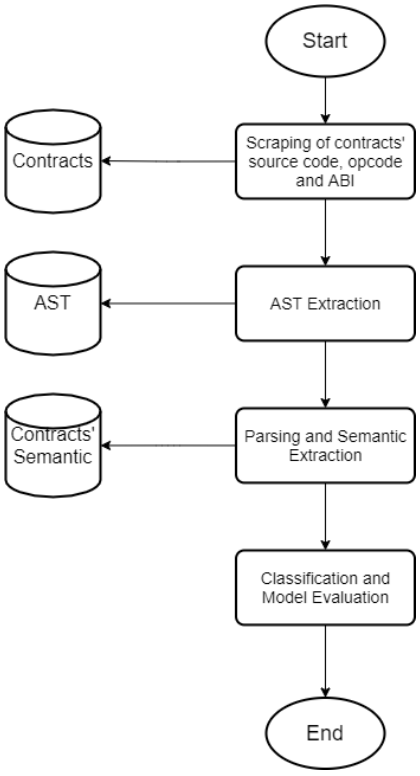


Fig. 2: Proposed approach's workflow

considerations about the expected results. In terms of time complexity, we expect the Decision Tree as the slowest model to train, and the SVM as the fastest. We expect to reach the best performances with the Multinomial Naive Bayes Model, and at least to reach similar results with the Support Vector Machine, which should behave as well in this particular task.

D. CLASSIFICATION METHODOLOGY

To perform the classification, we built two different CSV files. The first one has contracts' source code and opcode, while the second one has contracts' source code and transactions information. The idea is to test the models' performances making comparisons in terms of class metrics evaluation like accuracy, precision, recall, and f1-score. We want to check if these values are better with a model trained with source code and opcode or with a model trained with source code and transactions. First, we decided to use 80% of our dataset to build the training set and the remaining 20% to build the test set. Since we had fewer samples for the Ponzi schemes contracts, we decided not to use a validation set. Once we built our dataset, we dropped all the empty fields and possible duplicates, and then we converted our collection of documents to a matrix of token counts. Before training and testing the model, we normalized our data removing the words that commonly appear in the English language. We performed any character normalization, and we set a threshold to ignore terms having a document frequency lower than the given threshold.

E. EXPERIMENTAL SETUP USED

To perform our experiments we used a *MSI PL62 7RC* with the following tech specs:

- Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
- RAM: 8 GB
- Cores: 8
- Architecture:64-bits Ubuntu 18.04.4 desktop

We also used Google Colab to train and test our models. In particular, we used a VM with 25 GB of RAM.

V. RESULTS AND DISCUSSION

Table I resumes the results obtained with the Decision Tree model. We tested all the models with two different features. The first text feature joins the source code and the opcode, while the second one joins source code and transactions. Also, we tested our models with different parameters. For example, the *criterion* parameter allows to choose between the *Gini impurity* and *entropy* as functions to measure the quality of a split. Looking at results, we see that in terms of precision, recall, and f1-score, the decision tree behaved well with both the two different text features and with both Gini impurity and entropy. Choosing between the Gini impurity and the Entropy-based information gain doesn't make too much difference because they are pretty much the same. Anyway, selecting the Gini impurity would spare to compute logarithmic functions, which are computationally intensive. Anyway, we can see from the results that the best configuration is a Decision Tree trained with source code and transactions as a text feature. Table II shows results reached by a Multinomial Naive Bayes model. Again, the model behaved well and reached results not so

Features	Criterion	Target	Precision	Recall	F1-Score	Accuracy
Source code/Opcode	Gini	Not Ponzi	0.98	0.99	0.99	0.97833
	Gini	Ponzi	0.94	0.89	0.91	
Source code/Opcode	Entropy	Not Ponzi	0.98	1.00	0.99	0.98555
	Entropy	Ponzi	1.00	0.89	0.94	
Source code/Transactions	Gini	Not Ponzi	0.99	1.00	0.99	0.98916
	Gini	Ponzi	0.97	0.94	0.96	
Source code/Transactions	Entropy	Not Ponzi	0.99	1.00	0.99	0.98916
	Entropy	Ponzi	0.97	0.94	0.96	

TABLE I: Classification score for Decision Tree model

Features	Fit Prior	Target	Precision	Recall	F1-Score	Accuracy
Source code/Opcode	True	Not Ponzi	0.97	1.00	0.99	0.97472
	True	Ponzi	1.00	0.81	0.89	
Source code/Opcode	False	Not Ponzi	0.97	1.00	0.99	0.97472
	False	Ponzi	1.00	0.81	0.89	
Source code/Transactions	True	Not Ponzi	0.99	1.00	1.00	0.99277
	True	Ponzi	1.00	0.94	0.97	
Source code/Transactions	False	Not Ponzi	0.99	1.00	1.00	0.99277
	False	Ponzi	1.00	0.94	0.97	

TABLE II: Classification score for Multinomial Naive Bayes model

Features	Loss	Target	Precision	Recall	F1-Score	Accuracy
Source code/Opcode	Hinge	Not Ponzi	0.99	0.99	0.99	0.98555
	Hinge	Ponzi	0.94	0.94	0.94	
Source code/Opcode	Squared Hinge	Not Ponzi	0.99	0.99	0.99	0.98194
	Squared Hinge	Ponzi	0.92	0.93	0.94	
Source code/Transactions	Hinge	Not Ponzi	0.98	1.00	0.99	0.98194
	Hinge	Ponzi	0.97	0.89	0.93	
Source code/Transactions	Squared Hinge	Not Ponzi	0.99	1.00	0.99	0.98916
	Squared Hinge	Ponzi	0.97	0.94	0.96	

TABLE III: Classification score for Support Vector Machine model

far from Decision Trees's one, and sometimes slightly better. Again, the best configuration is a Multinomial Naive Bayes trained with source code and transactions. The parameter Fit Prior defines whether to learn class prior probabilities or not, but from the results, we can see that using a uniform prior or not doesn't make any difference. Table III shows results

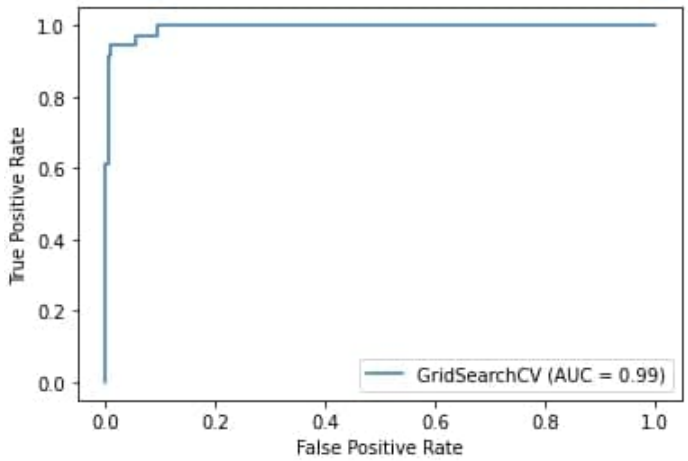


Fig. 3: Example of ROC curve showing the True Positive and False Positive rate reached by our SVM model

reached by a SVM model. The *loss* parameter allows to choose betweeen the *hinge* and *squared hinge* loss function. The SVM model behaved well both with source code and opcode and also with source code and transactions as text feature, despite the configuration trained with source code and transactions returns slightly better results.

Now that we showed all the results reached by all of the tested models, we can do further considerations. All the models have behaved well, and they were able to classify Ponzi schemes contracts. As we expected, the best models are the Multinomial Naive Bayes and the SVM, which obtained the best results in terms of precision, recall, and f1-score, reaching both an accuracy of 99pt%. The accuracy reached by our models is almost perfect as evidenced by the ROC curve plotted following a training of an SVM model shown in the Figure 3, but we must consider that this was also because Ponzi Schemes contracts have a well-defined structure, and many of them are similar in terms of code since a lot of Ethereum developers take inspiration from others' work.

VI. FUTURE WORK

It must be said that despite all the work that has already been done, Ethereum remains a relatively recent platform, and as a result, many vulnerabilities and honeypots have yet to be identified [14]. Given these premises, new scam schemes may be implemented by exploiting these vulnerabilities. All Ponzi scheme contracts used as dataset samples date back to 2016-2018, but new contracts may have been introduced in the meantime.We must consider that the Ponzi Schemes

contracts analyzed despite are included in a period of three years, they show no differences substantial in implementation. Although there have been several pragma and consequently some changes in the definition of functions and constructs are required, the flow of execution of a Smart Ponzi remains almost unchanged compared to the types already presented in section I.

The goal is to use our model to check whether new Ponzi schemes have actually been introduced into the Ethereum blockchain. The Ponzi scheme is just one of the possible scams that can be implemented against the participants of the Ehtereum network. In fact, other possibilities have been identified that allow taking advantage of the inner properties and possibilities offered by blockchain technology to implement scams. One possibility is to expand and make our model more comprehensive than the one presented in the paper in such a way as not only to recognize Ponzi schemes but also contracts that are potentially highly damaging, in economic terms, to the participants who use it.

REFERENCES

[1] M. Artzrouni "The Mathematics of Ponzi schemes," Mathematical Social Sciences, 2009, vol. 58, pp. 190-201

[2] N. Atzei, M. Bartoletti, and T. Cimoli "A Survey of Attacks on Ethereum Smart Contracts (SoK)", 2017, pp. 164-186

[3] Pinna, Andrea, et al. "Design of a Sustainable Blockchain-Oriented Software for Building Workers Management." Front. Blockchain 3: 38. doi: 10.3389/fbloc (2020).

[4] Marchesi, Michele, Lodovica Marchesi, and Roberto Tonelli. "An agile software engineering method to design blockchain applications." Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia. 2018.

[5] M. Bartoletti, S. Carta, T. Cimoli, and R. Saia "Dissecting Ponzi schemes on Ethereum: Identification, analysis, and impact," Future Generatio Computer Systems, August 2019, vol. 102

[6] Pierro GA, Tonelli R, Marchesi M. An Organized Repository of Ethereum Smart Contracts' Source Codes and Metrics. Future Internet. 2020; 12(11):197. https://doi.org/10.3390/fi12110197.

[7] Y. Wang, A. Bracciali, T. Li, F. Li, X. Cui, and M. Zhao, "Information Sciences," 2019, vol. 477, pp. 291-301

[8] M. Bartoletti, B. Pes, and S. Serusi "Data Mining for detecting bitcoin ponzi schemes", 2018 [Crypto Valley Conference on Blockchain Technology (CVCBT)], pp.75-84

[9] W. Chen, Z. Zheng, E. Ngai, P. Zheng, and Y. Zhou "Exploiting Blockchain Data to Detect Smart Ponzi Schemes on Ethereum," March 2019, vol. PP, pp. 1-1

[10] C. Ramiro, C. F. Torres, and R. State "A Data Science Approach for Honeypot Detection in Ethereum," 2019, CoRR

[11] C. F. Torres, M. Steichen, and R. State "The Art of The Scam: Demystifying Honeypots in Ethereum Smart Contracts," 2019

[12] N. Uras, L. Marchesi, M. Marchesi, and R. Tonelli "Forecasting Bitcoin closing price series using linear regression and neural networks models," PeerJ Computer Science 6, e279

[13] M. Vasek, and T. Moore "Analyzing the Bitcoin Ponzi Scheme Ecosystem," In A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, editors, Financial Cryptography and Data Security, 2019, pp. 101-112, Berlin, Hei-delberg. Springer Berlin Heidelberg

[14] H. Chen, M. Pendleton, L. Njilla, and S. Xu "A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses," June 2020, ACM Comput. Surv., vol. 53,

[15] Fenu, Gianni, et al. "The ICO phenomenon and its relationships with ethereum smart contract environment." 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). IEEE, 2018.