Sharpening Ponzi Schemes Detection on Ethereum with Machine Learning

Letterio Galletta and Fabio Pinelli
DLT 2023
Bologna, May 25th 2023
What is a Ponzi scheme?

- It is an investment fraud
- It requires a constant flow of money from new investors to continue
- It inevitably collapses, when it becomes difficult to recruit new investors

There are smart contracts on Ethereum that are Ponzi Schemes.
Our main contributions:

- A **reusable and publicly available data set** that collects 4422 real-world smart contracts (3749 not Ponzi, and 673 Ponzi);
- A **binary classifier** to detect smart Ponzi contracts that performs better than classifiers proposed in the literature;
- The study of the impact of such features on the classification using **eXplainable AI techniques**;
- A small **set of features** that ensures a good classification quality.

https://github.com/fpinell/ponzi_ml
The dataset

- Datasets with 4422 real-world smart contracts (3749 not Ponzi, and 673 Ponzi)
- Enriched the set of features starting from different sets available in the literature
- Collect the data for all the smart contracts using https://etherscan.io

<table>
<thead>
<tr>
<th>Feature</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Balance</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Lifetime</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tx_in</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tx_out</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Investment_in</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Payment_out</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>#addresses_paying_contract</td>
<td>✓</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>#addresses_paid_by_contract</td>
<td>✓</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Mean_v1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mean_v2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sdev_v1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sdev_v2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Paid_rate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Paid_one</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Known_rate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>N_maxpayment</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Skew_v1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Skew_v2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Investment_in/tx_in</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Payment_out/tx_out</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Percentage_some_tx_in</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Sdev_tx_in</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Percentage_some_tx_out</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Sdev_tx_out</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Initiator_gets_eth_Wo_investing</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Initiator_gets_eth_investing</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Initiator_no_eth</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
**RQ1**: Does the new features improve classification?

**RQ2**: What is the smallest and best set of features?

**RQ3**: What are the most important features and what is their contribution?
RQ1: Does the new features improve classification?

The best classifier is LGBM and it performs better on dataset D1 than D2.
RQ2: What is the smallest and best set of features?

<table>
<thead>
<tr>
<th>Feature</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Balance</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Lifetime</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tx_in</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tx_out</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Investment_in</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payment_out</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#addresses_paying_contract</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#addresses_paid_by_contract</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean_v1</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Mean_v2</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Sdev_v1</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Sdev_v2</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Paid_rate</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Paid_one</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Known_rate</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>N_maxpayment</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Skew_v1</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Skew_v2</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Investment_in/tx_in</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Payment_out/tx_out</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Percentage_some_tx_in</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Sdev_tx_in</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Percentage_some_tx_out</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Sdev_tx_out</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Initiator_gets_eth_Wo_investing</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Initiator_gets_eth_investing</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Initiator_no_eth</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

D3 is quite similar to D1: we remove three features that may mislead the classifier.
**RQ3:** What are the most important features?

- SHAP values (SHapley Additive exPlanations) is a method based on **cooperative game theory**

- SHAP shows the **contribution** or the importance of each feature on the prediction of the model

- The **beeswarm plot** is designed to display an information-dense summary of how the top features in a dataset impact the model’s output

- 3 new features are in the set of the 10 most important features
RQ3: What are the most important features?

Dependence plots between pair of features:

- Investment-in/Tx-in and Paid-one
- Tx-in and Lifetime

- The shap values (y-axis) varying as the colours of the second feature
• A **reusable and publicly available data set**
• A **binary classifier** to detect smart Ponzi contracts
• A small **set of features** that ensures a good classification quality
• The study of the impact of features using **eXplainable AI techniques**

Ongoing/future research activities:
• Identify features of bytecode
• Can we use only bytecode features?
• Study the robustness of the classifier
Sharpening Ponzi Schemes Detection on Ethereum with Machine Learning

Letterio Galletta

letterio.galletta@imtlucca.it
contract Multiplier {
    // Address of the "promoter" of the contract: she receives fee for each transaction
    address constant private PROMO = 0x5d5fe29339592eeb51c43e54f0a81c7642b6d2b;
    // Percent received by the "promoter"
    uint constant public PROMO_PERCENT = 7;
    // How many percent for your deposit to be multiplied
    uint constant public MULTIPLIER = 121;

    // The deposit structure holds all the info about the made deposits
    struct Deposit {
        address depositor; // The depositor address
        uint128 deposit;   // The deposit amount
        uint128 expect;    // How much we should pay out (initially it is 121% of deposit)
    }

    Deposit[] private queue; // The queue of investors
    uint public currentHead = 0; // The index of the first depositor in the queue.

    // This function receives all the deposits stores them in the queue and make immediate payouts
    function () public payable {
        if(msg.value > 0){
            // Add the investor into the queue. Mark that he expects to receive 121% of deposit back
            queue.push(Deposit(msg.sender, uint128(msg.value), uint128(msg.value*MULTIPLIER/100)));

            // Send some promo to enable this contract to leave long-long time
            uint promo = msg.value*PROMO_PERCENT/100;
            PROMO.send(promo);

            // Pay to first investors in line
            pay();
        }
    }
}
contract Multiplier {
  address constant private PROMO = 0x505fe29339592eeb51c43e54f0a81ca7642b6d2b;
  uint constant public PROMO_PERCENT = 7;
  uint constant public MULTIPLIER = 121;

  struct Deposit {
    address depositor; //The depositor address
    uint128 deposit; //The deposit amount
    uint128 expect; //How much we should pay out (initially it is 121% of deposit)
  }

  Deposit[] private queue; //The queue of investors
  uint public currentHead = 0; //The index of the first depositor in the queue.

  function () public payable {
    if(msg.value > 0){
      //Add the investor into the queue. Mark that he expects to receive 121% of deposit back
      queue.push(Deposit(msg.sender, uint128(msg.value), uint128(msg.value* MULTIPLIER/100)));

      //Send some promo to enable this contract to leave long-long-long time
      uint promo = msg.value*PROMO_PERCENT/100;
      PROMO.send(promo);

      //Pay to first investors in line
      pay();
    }
  }
}
contract Multiplier {
    // Address of the "promoter" of the contract: she receives fee for each transaction
    address constant private PROMO = 0x5D5fe29339592eEb51c43E54F0a81cA7642B6d2b;
    // Percent received by the "promoter"
    uint constant public PROMO_PERCENT = 7;
    // How many percent for your deposit to be multiplied
    uint constant public MULTIPLIER = 121;

    // The deposit structure holds all the info about the made deposits
    struct Deposit {
        address depositor; // The depositor address
        uint128 deposit; // The deposit amount
        uint128 expect; // How much we should pay out (initially it is 121% of deposit)
    }

    Deposit[] private queue; // The queue of investors
    uint public currentHead = 0; // The index of the first depositor in the queue.

    // This function receives all the deposits stores them in the queue and make immediate payouts
    function () public payable {
        if(msg.value > 0){
            // Add the investor into the queue. Mark that he expects to receive 121% of deposit back
            queue.push(Deposit(msg.sender, uint128(msg.value), uint128(msg.value*MULTIPLIER/100)));

            // Send some promo to enable this contract to leave long-long time
            uint promo = msg.value*PROMO_PERCENT/100;
            PROMO.send(promo);

            // Pay to first investors in line
            pay();
        }
    }
}
contract Multiplier {
    // Address of the "promoter" of the contract: she receives fee for each transaction
    address constant private PROMO = 0x5D5fe29339592eFh51c4354F0a81cA7642B6d2b;
    // Percent received by the "promoter"
    uint constant public PROMO_PERCENT = 7;
    // How many percent for your deposit to be multiplied
    uint constant public MULTIPLIER = 121;

    // The deposit structure holds all the info about the made deposits
    struct Deposit {
        address depositor;  // The depositor address
        uint128 deposit;    // The deposit amount
        uint128 expect;     // How much we should pay out (initially it is 121% of deposit)
    }

    Deposit[] private queue;  // The queue of investors
    uint public currentHead = 0; // The index of the first depositor in the queue.

    // This function receives all the deposits stores them in the queue and make immediate payouts
    function () public payable {
        if (msg.value > 0) {
            // Add the investor into the queue. Mark that he expects to receive 121% of deposit back
            queue.push(Deposit(msg.sender, uint128(msg.value), uint128(msg.value * MULTIPLIER / 100)));

            // Send some promo to enable this contract to leave long-long-time
            uint promo = msg.value * PROMO_PERCENT / 100;
            PROMO.send(promo);

            // Pay to first investors in line
            pay();
        }
    }
}
contract Multiplier {
    // Address of the "promoter" of the contract: she receives fee for each transaction
    address constant private PROMO = 0x5D5fe29339592eEb51c43E54F0a81cA7642B6d2b;
    // Percent received by the "promoter"
    uint constant public PROMO_PERCENT = 7;
    // How many percent for your deposit to be multiplied
    uint constant public MULTIPLIER = 121;

    // The deposit structure holds all the info about the made deposits
    struct Deposit {
        address depositor; // The depositor address
        uint128 deposit; // The deposit amount
        uint128 expect; // How much we should pay out (initially it is 121% of deposit)
    } Deposit[] private queue; // The queue of investors
    uint public currentHead = 0; // The index of the first depositor in the queue.

    // This function receives all the deposits stores them in the queue and make immediate payouts
    function () public payable {
        if(msg.value > 0){
            // Add the investor into the queue. Mark that he expects to receive 121% of deposit back
            queue.push(Deposit(msg.sender, uint128(msg.value), uint128(msg.value*MULTIPLIER/100)));

            // Send some promo to enable this contract to leave long-long time
            uint promo = msg.value*PROMO_PERCENT/100;
            PROMO.send(promo);

            // Pay to first investors in line
            pay();
        }
    }
}
contract Multiplier {
    //Address of the "promoter" of the contract: she receives fee for each transaction
    address constant private PROMO = 0x5d5fe29339592eEe51c43E54F0a81cA7642B6d2b;
    //Percent received by the "promoter"
    uint constant public PROMO_PERCENT = 7;
    //How many percent for your deposit to be multiplied
    uint constant public MULTIPLIER = 121;

    //The deposit structure holds all the info about the made deposits
    struct Deposit {
        address depositor; //The depositor address
        uint128 deposit; //The deposit amount
        uint128 expect; //How much we should pay out (initially it is 121% of deposit)
    }

    Deposit[] private queue; //The queue of investors
    uint public currentHead = 0; //The index of the first depositor in the queue.

    //This function receives all the deposits stores them in the queue and make immediate payouts
    function () public payable {
        if (msg.value > 0) {
            //Add the investor into the queue. Mark that he expects to receive 121% of deposit back
            queue.push(Deposit(msg.sender, uint128(msg.value), uint128(msg.value * MULTIPLIER / 100)));

            //Send some promo to enable this contract to leave long-long time
            uint promo = msg.value * PROMO_PERCENT / 100;
            PROMO.send(promo);

            //Pay to first investors in line
            pay();
        }
    }
}
contract Multiplier {
    // Address of the "promoter" of the contract: she receives fee for each transaction
    address constant private PROMO = 0x5d5fe29339592eEb51c43E54F0a81cA7642B6d2b;
    // Percent received by the "promoter"
    uint constant public PROMO_PERCENT = 7;
    // How many percent for your deposit to be multiplied
    uint constant public MULTIPLIER = 121;

    // The deposit structure holds all the info about the made deposits
    struct Deposit {
        address depositor; // The depositor address
        uint128 deposit; // The deposit amount
        uint128 expect; // How much we should pay out (initially it is 121% of deposit)
    }

    Deposit[] private queue; // The queue of investors
    uint public currentHead = 0; // The index of the first depositor in the queue.

    // This function receives all the deposits stores them in the queue and make immediate payouts
    function () public payable {
        if(msg.value > 0){
            // Add the investor into the queue. Mark that he expects to receive 121% of deposit back
            queue.push(Deposit(msg.sender, uint128(msg.value), uint128(msg.value*MULTIPLIER/100)));

            // Send some promo to enable this contract to leave long-long time
            uint promo = msg.value*PROMO_PERCENT/100;
            PROMO.send(promo);

            // Pay to first investors in line
            pay();
        }
    }
}
contract Multiplier {
    // Address of the "promoter" of the contract: she receives fee for each transaction
    address constant private PROMO = 0x5D5fe29339592eEb51c43E54F0a81cA7642B6d2b;
    // Percent received by the "promoter"
    uint constant public PROMO_PERCENT = 7;
    // How many percent for your deposit to be multiplied
    uint constant public MULTIPLIER = 121;

    // The deposit structure holds all the info about the made deposits
    struct Deposit {
        address depositor; // The depositor address
        uint128 deposit;   // The deposit amount
        uint128 expect;   // How much we should pay out (initially it is 121% of deposit)
    }

    Deposit[] private queue; // The queue of investors
    uint public currentHead = 0; // The index of the first depositor in the queue.

    // This function receives all the deposits stores them in the queue and make immediate payouts
    function () public payable {
        if (msg.value > 0) {
            // Add the investor into the queue. Mark that he expects to receive 121% of deposit back
            queue.push(Deposit(msg.sender, uint128(msg.value), uint128(msg.value*MULTIPLIER/100)));

            // Send some promo to enable this contract to leave long-long time
            uint promo = msg.value*PROMO_PERCENT/100;
            PROMO.send(promo);

            // Pay to first investors in line
            pay();
        }
    }
}
contract Multiplier {
    //Address of the "promoter" of the contract: she receives fee for each transaction
    address constant private PROMO = 0x5d5fe29339592eEb51c43E54F0a81cA7642B6d2b;
    //Percent received by the "promoter"
    uint constant public PROMO_PERCENT = 7;
    //How many percent for your deposit to be multiplied
    uint constant public MULTIPLIER = 121;

    //The deposit structure holds all the info about the made deposits
    struct Deposit {
        address depositor; //The depositor address
        uint128 deposit;   //The deposit amount
        uint128 expect;   //How much we should pay out (initially it is 121% of deposit)
    }

    Deposit[] private queue; //The queue of investors
    uint public currentHead = 0; //The index of the first depositor in the queue.

    //This function receives all the deposits stores them in the queue and make immediate payouts
    function () public payable {
        if(msg.value > 0){
            //Add the investor into the queue. Mark that he expects to receive 121% of deposit back
            queue.push(Deposit(msg.sender, uint128(msg.value), uint128(msg.value*MULTIPLIER/100)));

            //Send some promo to enable this contract to leave long-long time
            uint promo = msg.value*PROMO_PERCENT/100;
            PROMO.send(promo);

            //Pay to first investors in line
            pay();
        }
    }
}
contract Multiplier {
    // Address of the "promoter" of the contract: she receives fee for each transaction
    address constant private PROMO = 0x5d5fe2933959eEb51c43E54F0a81cA7642B6d2b;
    // Percent received by the "promoter"
    uint constant public PROMO_PERCENT = 7;
    // How many percent for your deposit to be multiplied
    uint constant public MULTIPLIER = 121;

    // The deposit structure holds all the info about the made deposits
    struct Deposit {
        address depositor; // The depositor address
        uint128 deposit;   // The deposit amount
        uint128 expect;    // How much we should pay out (initially it is 121% of deposit)
    }

    Deposit[] private queue; // The queue of investors
    uint public currentHead = 0; // The index of the first depositor in the queue.

    // This function receives all the deposits stores them in the queue and make immediate payouts
    function () public payable {
        if(msg.value > 0) {
            // Add the investor into the queue. Mark that he expects to receive 121% of deposit back
            queue.push(Deposit(msg.sender, uint128(msg.value), uint128(msg.value*MULTIPLIER/100)));

            // Send some promo to enable this contract to leave long-long time
            uint promo = msg.value*PROMO_PERCENT/100;
            PROMO.send(promo);

            // Pay to first investors in line
            pay();
        }
    }
}
function pay() private {
    // The current balance of the contract
    uint128 money = uint128(address(this).balance);

    // Cycle on the queue
    for(uint i=0; i<queue.length; i++){

        uint idx = currentHead + i; // the index of the currently first investor
        Deposit storage dep = queue[idx]; // the info of the first investor

        if(money >= dep.expect){ // If we have enough money on the contract to fully pay to investor
            dep.depositor.send(dep.expect); // Send money to him
            money -= dep.expect; // update money left

            // The investor is fully paid and she is removed from the queue
            delete queue[idx];
        }else{
            // No enough money, so partially pay to investor
            dep.depositor.send(money); // Send to her the money left
            dep.expect -= money; // Update her expected amount
            break;
        }
    }

    currentHead += i; // Update the index of the current first investor
}
function pay() private {
    // The current balance of the contract
    uint128 money = uint128(address(this).balance);

    // Cycle on the queue
    for(uint i=0; i<queue.length; i++){
        uint idx = currentHead + i; // the index of the currently first investor
        Storage deposit = queue[idx]; // the info of the first investor

        if(money >= deposit.expect){ // If we have enough money on the contract to fully pay to investor
            deposit.depositor.send(deposit.expect); // Send money to him
            money -= deposit.expect; // update money left

            // the investor is fully paid and she is removed from the queue
            delete queue[idx];
        } else {
            // No enough money, so partially pay to investor
            deposit.depositor.send(money); // Send to her the money left
            deposit.expect -= money; // Update her expected amount
            break;
        }
    }

    currentHead += i; // Update the index of the current first investor
}
// Used to distribute all the money on contract to the first investors starting from the head of queue

```solidity
function pay() private {
    // The current balance of the contract
    uint128 money = uint128(address(this).balance);

    // Cycle on the queue
    for (uint i=0; i<queue.length; i++) {
        uint idx = currentHead + i; // the index of the currently first investor

        Deposit storage dep = queue[idx]; // the info of the first investor

        if (money >= dep.expect) {
            dep.depositor.send(dep.expect); // Send money to him
            money -= dep.expect; // Update money left

            // the investor is fully paid and she is removed from the queue
            delete queue[idx];
        } else {
            // No enough money, so partially pay to investor
            dep.depositor.send(money); // Send to her the money left
            dep.expect -= money; // Update her expected amount
            break;
        }
    }

    currentHead += i; // Update the index of the current first investor
}
```
function pay() private {
   // The current balance of the contract
   uint128 money = uint128(address(this).balance);

   // Cycle on the queue
   for(uint i=0; i<queue.length; i++) {

      uint idx = currentHead + i; // the index of the currently first investor

      Deposit storage dep = queue[idx]; // the info of the first investor

      if (money >= dep.expect) { // If we have enough money on the contract to fully pay to investor
         dep.depositor.send(dep.expect); // Send money to him
         money -= dep.expect; // Update money left

         // the investor is fully paid and she is removed from the queue
         delete queue[idx];
      } else { // No enough money, so partially pay to investor
         dep.depositor.send(money); // Send to her the money left
         dep.expect -= money; // Update her expected amount
         break;
      }
   }
}

currentHead += i; // Update the index of the current first investor
Some features