technische universität
dortmund

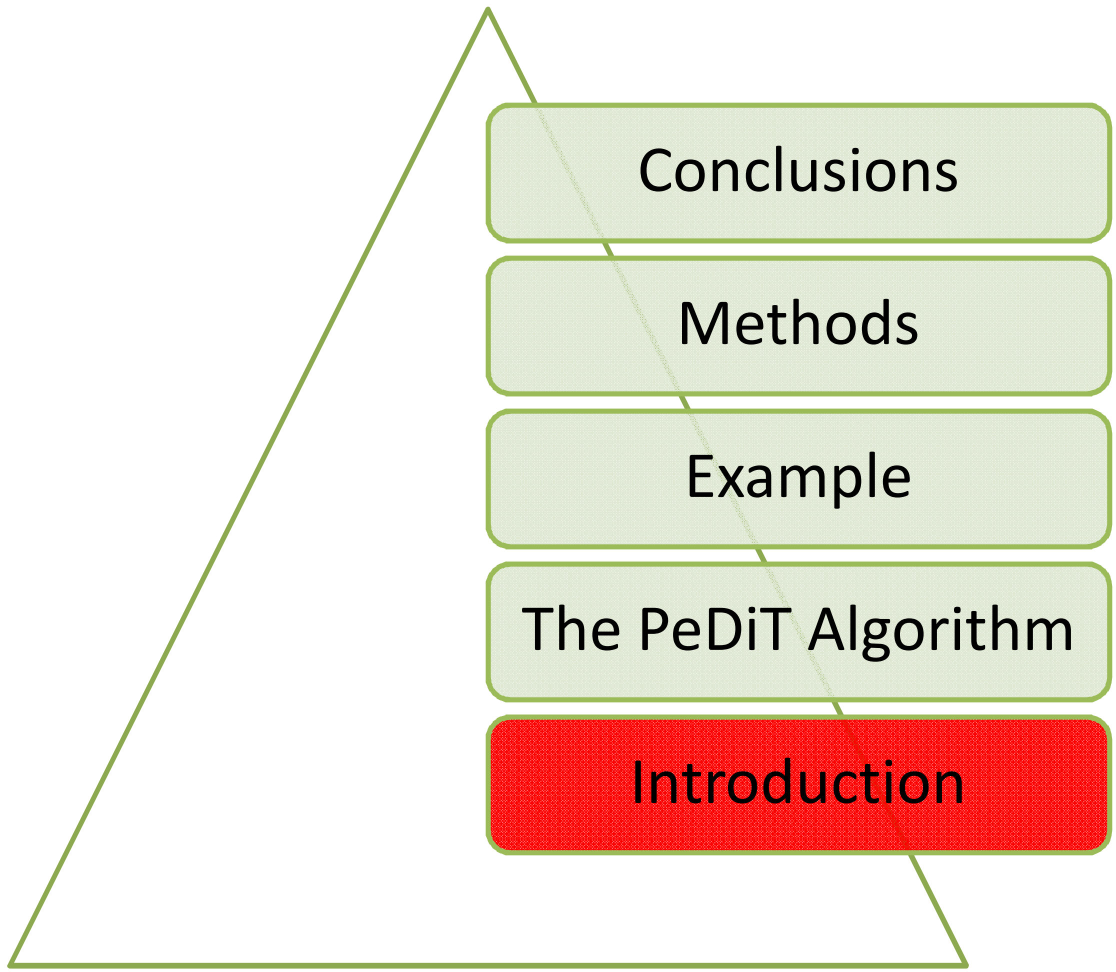
Distributed Decision Tree Induction in Peer-to-Peer Systems

Kanishka Bhaduri * Ran Wolff * Chris Giannella *Hillol Kargupta

Aktuelle Arbeiten des Data Mining

Student : Simona Florescu 26.05.2009

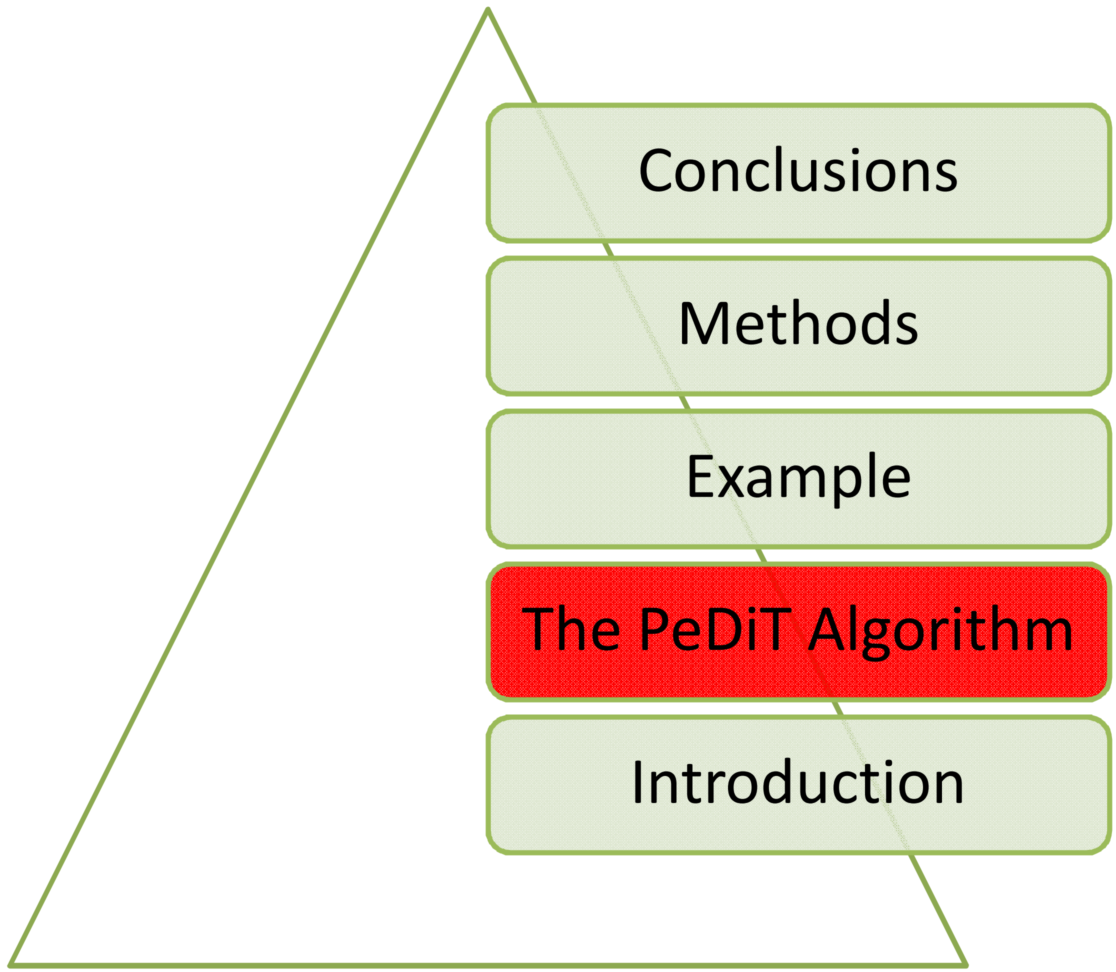
Table of Contents



Introduction

- Motivation
- Goal
- Algorithm Overview

Table of Contents



PeDiT Algorithm

- Goal : to select the best attribute --> an ad-hoc decision tree with active nodes + developing of the peer-to-peer decision tree.
- Active node : the root and any node whose parent is split by the ad-hoc attribute value computed by the P2P misclassification minimization (P2PMM) .
- Inactive : the rest of the nodes.

PeDiT Algorithm

Input: S — a set of learning examples, τ — mitigation delay
Initialization:
Create a root leaf and let $root.S \leftarrow S$. Set $nodes \leftarrow \{root\}$. Push $root$ to $queue$
Send BRANCH message to self with delay τ

- As an input we take the training samples and a τ – the time interval for the further development.
- We create an empty queue where we store all the new created nodes.

PeDiT Algorithm

On BRANCH message:

Send BRANCH message to self with delay τ

For ($i \leftarrow 0$, $\ell \leftarrow null$; $i < queue.length$ and not active(ℓ); i++)

  Pop head of queue into ℓ

  If not active(ℓ)

    enqueue ℓ

  If active(ℓ)

  Let A^j be the ad-hoc solution of P^2MM for ℓ

  call **Branch**(ℓ, j)

- We find the next active node and call the Branch procedure for that new node.

PeDiT Algorithm

On data message $\langle n, data \rangle$**:**
If $n \notin nodes$
store $\langle n, data \rangle$ in $out - of - context$
Else
Transfer the $data$ to the P^2MM instance of n
If active(n) then
Process(n)

- All the messages who come in the context of a not yet developed node are stored into a out-of-context Queue.
- Later, when that node will be new created it will look up in the out-of-context Queue to check for its messages and process them.

PeDiT Algorithm

Procedure Active(n)**:**
  If $n = null$ or $n = root$
    return true
  Let A^j be the ad-hoc solution for P^2MM for $n.parent$
  If $n \notin n.parent.sons[j]$
    return false
  Return Active$(n.parent)$

- The procedure checks whether a node is active or not.

PeDiT Algorithm

Procedure Process(n)**:**

Perform tests required by P^2MM for n and send any resulting messages
Let A^j be the ad-hoc solution for P^2MM for n
If $n.sons\,[j]$ is not empty
  for each $m \in n.sons\,[j]$
    call Process(m)
Else
  push n to the tail of the queue

- All the precedent nodes who are not active are inserted at the tail of the Queue

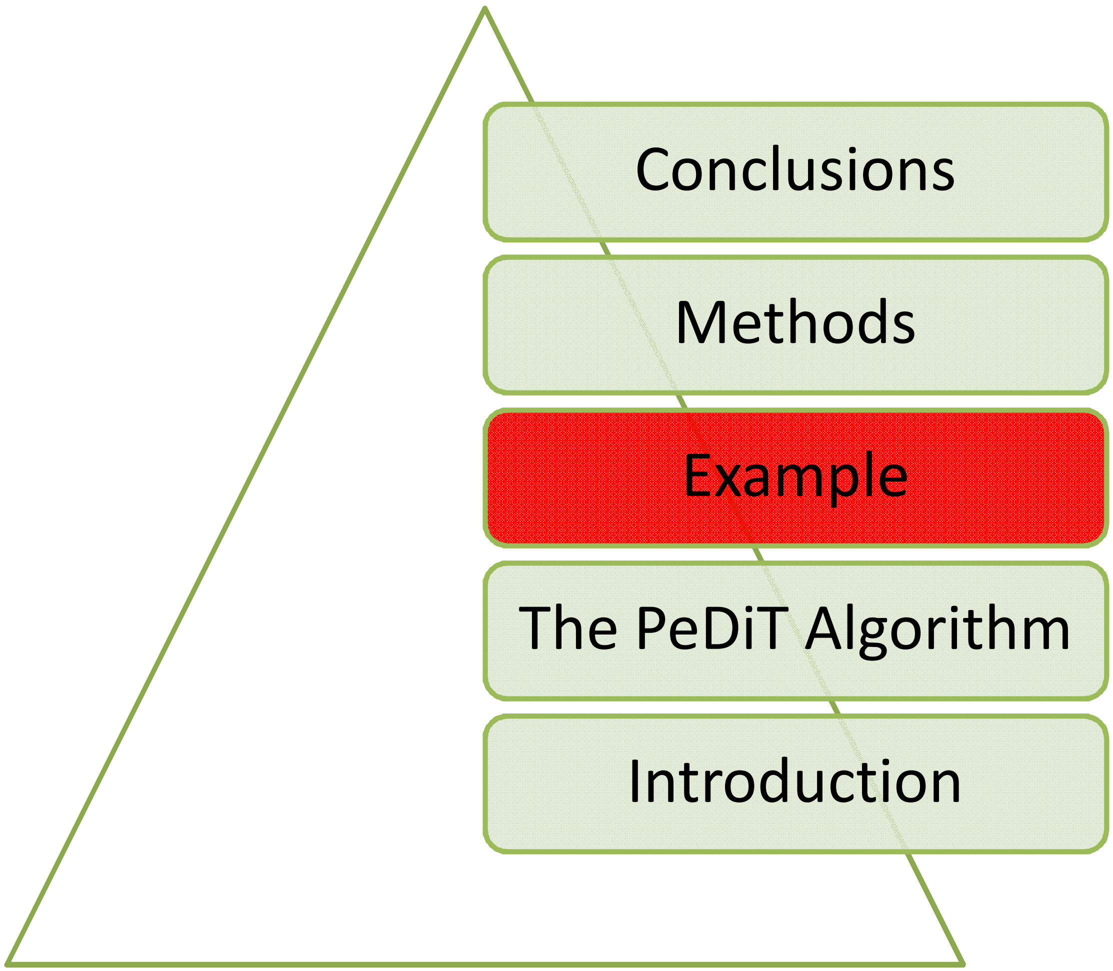
PeDiT Algorithm

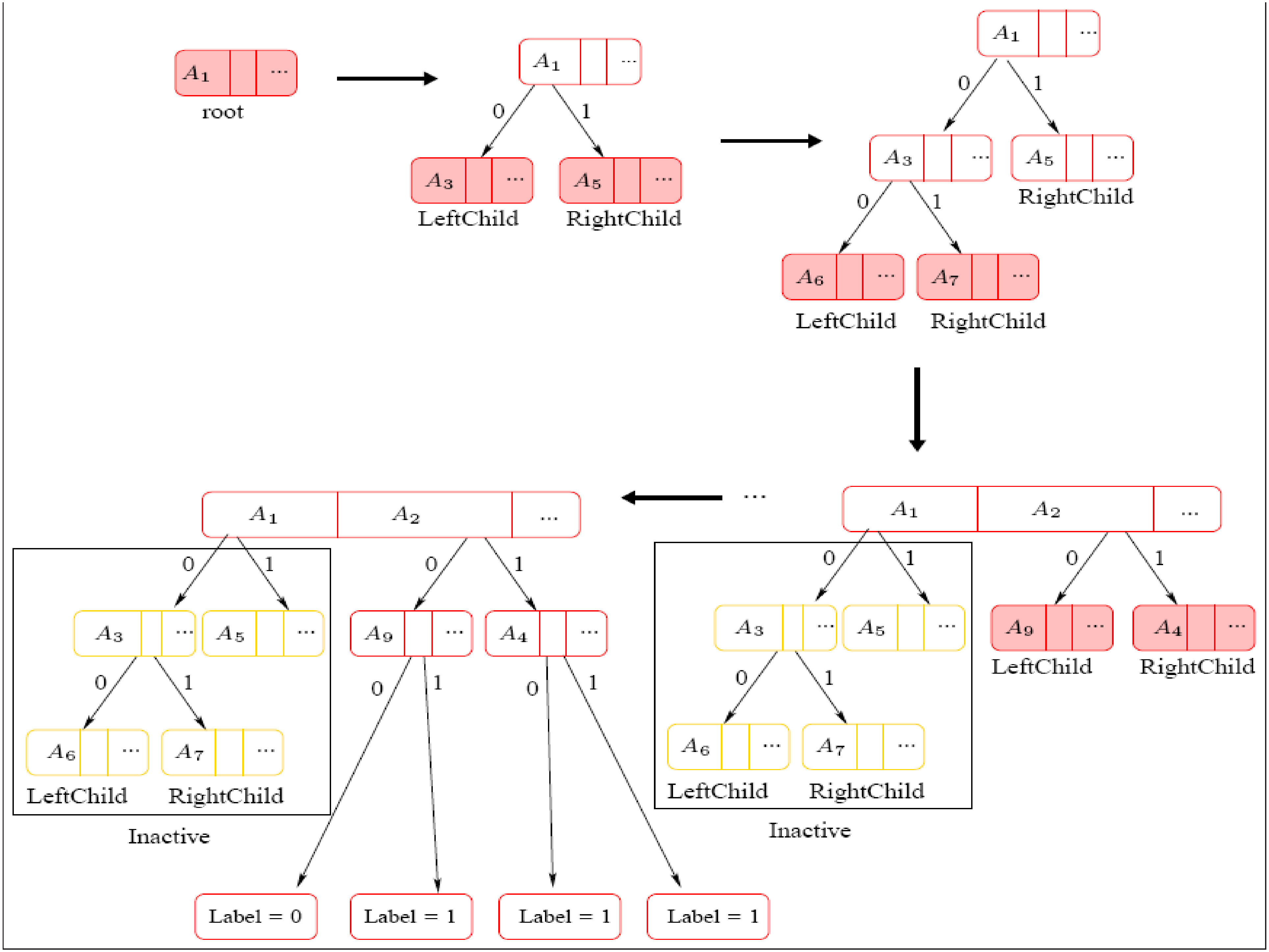
Procedure Branch$(\ell, , j)$**:**
Create two new leaves ℓ_0 and ℓ_1
Set $\ell_0.parent \leftarrow \ell$, $\ell_1.parent \leftarrow \ell$
Set $\ell_0.S \leftarrow \{s \in \ell.S : s[j] = 0\}$ and $\ell_1.S \leftarrow \{s \in \ell.S : s[j] = 1\}$
Remove from $out-of-context$ messages intended for ℓ_0 and ℓ_1 and deliver the data to the respective instance of P^2MM
Set $\ell.sons[j] = \{\ell_0, \ell_1\}$, add ℓ_0, ℓ_1 to nodes and push ℓ_0 and ℓ_1 to the tail of the queue

---

- It develops the tree with the new root and it checks for messages belonging to the respective node and it process them.
- It pushes the precedent sons of the node into the tail of the queue.

Table of Contents





A_1 ...
root
A_1 ...
0 1
A_3 ...
A_5 ...
LeftChild
RightChild
A_1 ...
0 1
A_3 ...
A_5 ...
RightChild
0 1
A_6 ...
A_7 ...
LeftChild
RightChild
...
A_1 A_2 ...
A_3 ...
A_5 ...
A_6 ...
A_7 ...
LeftChild
RightChild
Inactive
A_9 ...
A_4 ...
LeftChild
RightChild
Label = 0
Label = 1
Label = 1
Label = 1

Table of Contents

P2P misclassification minimization

- Returns the "AD-HOC" attribute with the highest misclassification gain
- Input : we consider as an input only the direct neighbors of a peer and the learning examples.
- Strategy : We compute the best attribute using the peer information and the misclassification gain and pivoting method.

P2P misclassification minimization

•The algorithm takes as an input the peer k and its direct neighbors and the set of samples.

Input variables of peer k: the set of neighbors — N_k, the set of examples — S_k
Output variables of peer k: the attribute A^{pivot}
Initialization:

- For every A^i in $A^1 \ldots A^d$ initialize two instances of LSD-Majority with inputs $x^i_{k,00} - x^i_{k,01}$ and $x^i_{k,10} - x^i_{k,11}$. Denote these instances by M^i_0 and M^i_1 respectively and let $M^i_0.\Delta_k$ and $M^i_1.\Delta_k$ denote the knowledge of those two instances. Further, for every $\ell \in N_k$, let $M^i_0.\Delta_{k,\ell}$ and $M^i_1\Delta_{k,\ell}$ be their agreement.

•For every attribute Ai we denote 2 instances of LSD(large-scale distributed) Majority : in order to determine s_0^i and s_1^i
•Their agreements are obtained by multiplying them with the exchanged information between 2 nodes, $\Delta_{k,l}$

P2P misclassification minimization

Initialization : Step 2

- For every $a, b, c, d \in \{-1, 1\}$ and every $A^i, A^j \in [A^1 \ldots A^d]$ initialize an instance of LSD-Majority with input $\delta_k^{i,j}|abcd$. Denote these instances by $M_{abcd}^{i,j}$. Let $M_{abcd}^{i,j}.\Delta_k$ and $M_{abcd}^{i,j}\Delta_{k,\ell}$ $(\forall \ell \in N_k)$ be the knowledge and agreement of the $M^{i,j}$ instance, respectively. Specifically denote $M^{i,j}.\Delta_k$ and $M^{i,j}\Delta_{k,\ell}$ the instance with a, b, c, and d equal to $s_{k,0}^i$, $s_{k,1}^i$, $s_{k,0}^j$, and $s_{k,1}^j$, respectively.

Secondly, we initialize the sixteen possible combinations from the values s_0^i, s_1^i, s_0^j, s_1^j for every pair i<j ∈ {1,..d}

P2P misclassification minimization

On any event:

- For $A^i \in \{A^1 \ldots A^d\}$ and every $\ell \in N_k$
  - If not $M_0^i.\Delta_k \leq M_0^i.\Delta_{k,\ell} < 0$ and not $M_0^i.\Delta_k \geq M_0^i.\Delta_{k,\ell} \geq 0$ call $Send(M_0^i, \ell)$
  - If not $M_1^i.\Delta_k \leq M_1^i.\Delta_{k,\ell} < 0$ and not $M_1^i.\Delta_k \geq M_1^i.\Delta_{k,\ell} \geq 0$ call $Send(M_1^i, \ell)$

After the initialization the algorithm takes the following cases into consideration (events) (DMV):

- k experiences a data change or a change of its neighborhood
- k receives a message from a neighbor

- If the message – condition (**) is not satisfied then it calls the send message function.

P2P misclassification minimization

Next, the pivoting method is used to reduce complexity.

- Do
  - Let $pivot = \arg\max_{i \in [1 \ldots d]} \left\{ \max_{\ell \in N_k, j<i, m>i} \left\{ M^{j,i}.\Delta_{k,\ell}, -M^{i,m}.\Delta_{k,\ell} \right\} \right\}$
  - For $A^i \in \{A^1 \ldots A^{pivot-1}\}$ and every $\ell \in N_k$
    - $*$ If not $M^{i,pivot}.\Delta_k \leq M^{i,pivot}.\Delta_{k,\ell} < 0$ and not $M^{i,pivot}.\Delta_k \geq M^{i,pivot}.\Delta_{k,\ell} \geq 0$ call $Send\left(M^{i,pivot}, \ell\right)$
  - For $A^i \in \left\{A^{pivot+1} \ldots A^d\right\}$ and every $\ell \in N_k$
    - $*$ If not $M^{pivot,i}.\Delta_k \leq M^{pivot,i}.\Delta_{k,\ell} < 0$ and not $M^{pivot,i}.\Delta_k \geq M^{pivot,i}.\Delta_{k,\ell} \geq 0$ call $Send\left(M^{pivot,i}, \ell\right)$
- While $pivot$ changes

• The chosen pivot is the attribute with the largest $M^{j,i}$ value for j<i or the smallest $M^{i,m}$ for i<m.
• If the pivoting condition fails, then it is called the Send function.

P2P misclassification minimization

On message (id, δ) **from** ℓ**:**

- Let M be a majority voting instance with $M.id = id$
- Set $M.\delta_{\ell,k}$ to δ

Procedure Send(M, ℓ)**:**

- $M.\delta_{k,\ell} = \alpha M.\Delta_k + M.\delta_{\ell,k}$
- Send to ℓ $(M.id, M.\delta_{k,\ell})$

- In the Send procedure the $\Delta_{k,l}$ becomes $\alpha\Delta_k$ where α is set by default to 1/2

Distributed majority voting

- Goal : to decide when a peer must send a message to a neighbor after detecting an event.
- Each peer k contains a real number : δ^k
- The latest message sent from a neighbor l to k : δ^{lk}
- => $\Delta^k = \delta^k + \sum_{l \in N_k} \delta^{lk}$
- All the exchanged information between k and a neighbor l : Δ^{kl}

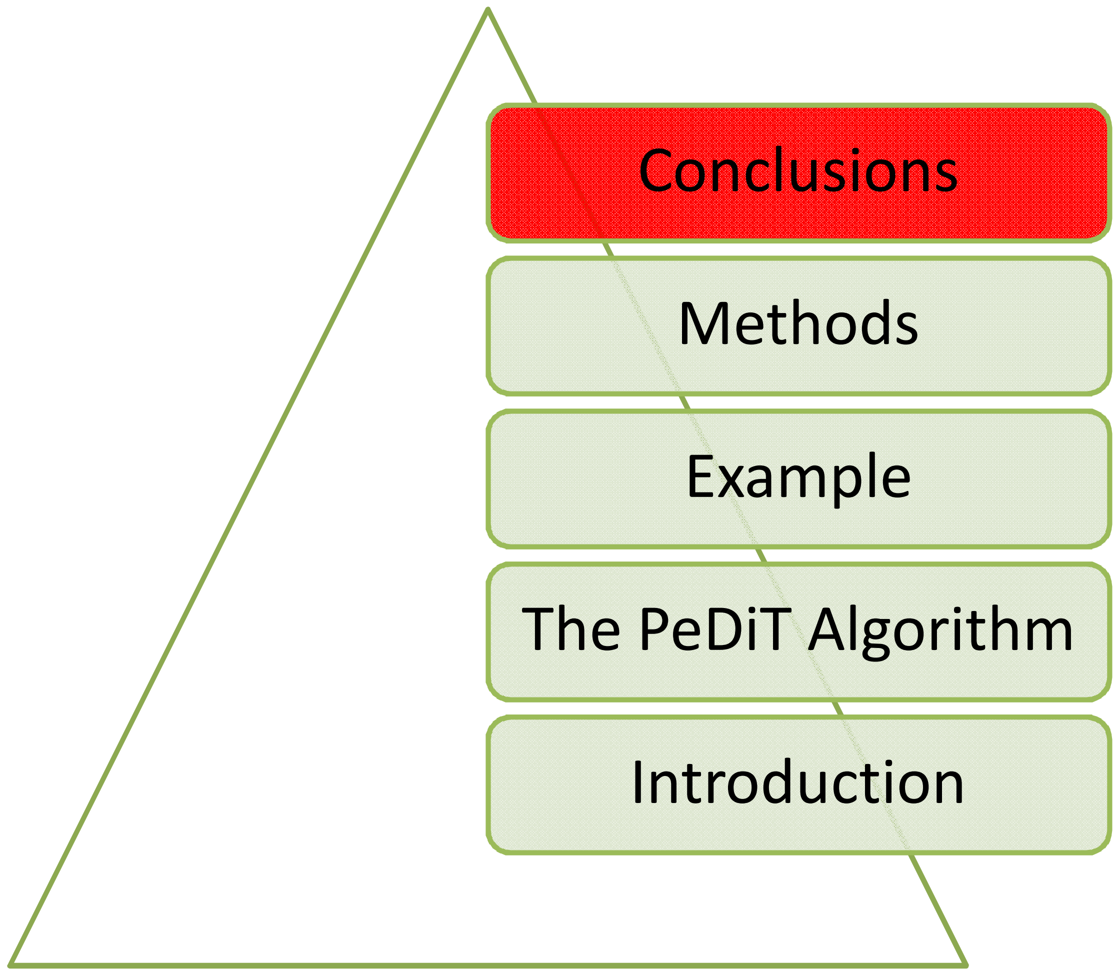
Distributed majority voting

- The condition when k would send a message to l :

$$(\Delta^{kl} \geq 0 \wedge \Delta^{kl} > \Delta^{k}) \vee (\Delta^{kl} < 0 \wedge \Delta^{kl} < \Delta^{k}) \quad (*)$$

- When a message is sent : $\Delta^{kl} = \alpha\Delta^{k}$ where α is a parameter between 0 and 1 set by default to ½.
- Leaky bucket mechanism : it introduces time space between messages sending.

Table of Contents



Conclusions

- The PeDiT Algorithm derivates from the standard decision tree induction algorithm except that it uses a misclassification gain as a splitting criteria and it uses a stopping rule the depth of the tree .
- *Experiments show :*
- a modest accuracy loss of the misclassification gain compared to Entropy criteria.
- the depth could decrease the efficiency of the algorithm but a depth of 3 it is an optimal choice.

Conclusions

- The PeDiT Algorithm is suitable for networks with millions of peers.
- Even if the number of attributes is increased, the algorithm remains moderate.
- With a sufficient given time the algorithm obtains from a P2P network the same result tree if given all the data of all the peers.

Thank you !

THE END