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
Introduction

- Motivation
- Goal
- Algorithm Overview
Table of Contents

- Introduction
- The PeDiT Algorithm
- Example
- Methods
- Conclusions
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 misclassfication minimization (P2PMM).
- **Inactive**: the rest of the nodes.
PeDiT Algorithm

**Input:** $S$ — a set of learning examples, $\tau$ — mitigation delay

**Initialization:**
- Create a root leaf and let $\text{root}.S \leftarrow S$. Set $\text{nodes} \leftarrow \{\text{root}\}$. Push $\text{root}$ to queue.
- Send BRANCH message to self with delay $\tau$

- As an input we take the training samples and a $\tau$ — 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 $\tau$
For $(i \leftarrow 0, \ell \leftarrow null; i < \text{queue.length} \text{ and not active}(\ell); i++)$
  Pop head of queue into $\ell$
  If not active($\ell$)
    enqueue $\ell$
  If active($\ell$)
Let $A^j$ be the ad-hoc solution of $P^2MM$ for $\ell$
call Branch($\ell, j$)

• We find the next active node and call the Branch procedure for that new node.
PeDiT Algorithm

On data message $(n, data)$:
   If $n \notin \text{nodes}$
       store $(n, data)$ 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
devolved node are stored into an out-of-context Queue.
• Later, when that node will be newly created it will look up in
the out-of-context Queue to check for its messages and
process them.
PeDiT Algorithm

Procedure Active\((n)\):

\begin{align*}
\text{If } n &= \textit{null} \text{ or } n = \textit{root} \\
\text{    return true} \\
\text{Let } A^i \text{ be the ad-hoc solution for } P^2MM \text{ for } n.parent \\
\text{If } n \notin n.parent.sons[j] \\
\text{    return false} \\
\text{Return Active}(n.parent)
\end{align*}

• 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 $A_i$ 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 $\ell_0$ and $\ell_1$

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 $\ell_0$ and $\ell_1$ and deliver the data to the respective instance of $P^2MM$

Set $\ell.sons[j] = \{\ell_0, \ell_1\}$, add $\ell_0, \ell_1$ to nodes and push $\ell_0$ and $\ell_1$ 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.
The diagram illustrates a binary tree structure with nodes labeled "A1", "A2", "A3", "A4", "A5", "A6", etc. Each node has two child nodes labeled "LeftChild" and "RightChild". The tree structure is hierarchical, with each node having a label of either 0 or 1. The labels indicate the decision-making process at each node, with 0 and 1 representing different paths or outcomes. The tree branches out to represent decision points, leading to further nodes or leaf nodes labeled "Label = 0" or "Label = 1". The diagram shows how the tree structure is used to make decisions or categorizations.
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 $A^{\text{pivot}}$

**Initialization:**
- For every $A^i$ in $A^1 \ldots A^d$ initialize two instances of LSD-Majority with inputs $x_{k,00}^i - x_{k,01}^i$ and $x_{k,10}^i - x_{k,11}^i$. Denote these instances by $M_0^i$ and $M_1^i$ respectively and let $M_0^i \Delta_k$ and $M_1^i \Delta_k$ denote the knowledge of those two instances. Further, for every $\ell \in N_k$, let $M_0^i \Delta_{k,\ell}$ and $M_1^i \Delta_{k,\ell}$ be their agreement.

- For every attribute $A_i$ 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, $\Delta_{k,\ell}$
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 $\delta^i,j \mid abcd$. Denote these instances by $M^{i,j}_{abcd}$. Let $M^{i,j}_{abcd} \Delta_k$ and $M^{i,j}_{abcd} \Delta_{k,\ell}$ ($\forall \ell \in N_k$) 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 $a, b, c, d$ equal to $s^i_{k,0}, s^i_{k,1}, s^j_{k,0}$, and $s^j_{k,1}$, respectively.

Secondly, we initialize the sixteen possible combinations from the values $s^i_0, s^i_1, s^j_0, s^j_1$ for every pair $i < j \in \{1,..d\}$
P2P misclassification minimization

On any event:

- For $A^i \in \{A^1 \ldots A^d\}$ and every $\ell \in N_k$
  - If not $M^i_0.\Delta_k \leq M^i_0.\Delta_{k,\ell} < 0$ and not $M^i_0.\Delta_k \geq M^i_0.\Delta_{k,\ell} \geq 0$ call $Send(M^i_0, \ell)$
  - If not $M^i_1.\Delta_k \leq M^i_1.\Delta_{k,\ell} < 0$ and not $M^i_1.\Delta_k \geq M^i_1.\Delta_{k,\ell} \geq 0$ call $Send(M^i_1, \ell)$

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 \( \text{pivot} = \arg \max_{i \in [1..d]} \left\{ \max_{\ell \in N_k, j < i, m > i} \{ M^{j,i} \Delta_{k,\ell}, -M^{i,m} \Delta_{k,\ell} \} \right\} \)
  - For \( A^i \in \{ A^1, \ldots A^{\text{pivot}-1} \} \) and every \( \ell \in N_k \)
    * If not \( M^{i,\text{pivot}} \Delta_{k} \leq M^{i,\text{pivot}} \Delta_{k,\ell} < 0 \) and not \( M^{i,\text{pivot}} \Delta_{k} \geq M^{i,\text{pivot}} \Delta_{k,\ell} \geq 0 \) call \( \text{Send} \left( M^{i,\text{pivot}}, \ell \right) \)
  - For \( A^i \in \{ A^{\text{pivot}+1} \ldots A^d \} \) and every \( \ell \in N_k \)
    * If not \( M^{\text{pivot},i} \Delta_{k} \leq M^{\text{pivot},i} \Delta_{k,\ell} < 0 \) and not \( M^{\text{pivot},i} \Delta_{k} \geq M^{\text{pivot},i} \Delta_{k,\ell} \geq 0 \) call \( \text{Send} \left( M^{\text{pivot},i}, \ell \right) \)
- While \( \text{pivot} \) 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 \((id, \delta)\) from \(\ell\):
- Let \(M\) be a majority voting instance with \(M.id = id\)
- Set \(M.\delta_{\ell,k}\) to \(\delta\)

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

In the Send procedure the \(\Delta_{k,\ell}\) becomes \(\alpha \Delta_k\) where \(\alpha\) 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: $\delta^k$
- The latest message sent from a neighbor $l$ to $k$: $\delta^{lk}$
- $\Delta^k = \delta^k + \sum_{l \in N_k} \delta^{lk}$
- All the exchanged information between $k$ and a neighbor $l$: $\Delta^{kl}$
Distributed majority voting

• The condition when k would send a message to l:
  \[(\Delta^{kl} \geq 0 \land \Delta^{kl} > \Delta^k) \lor (\Delta^{kl} < 0 \land \Delta^{kl} < \Delta^k) \] (*)

• When a message is sent: \(\Delta^{kl} = \alpha \Delta^k\) where \(\alpha\) is a parameter between 0 and 1 set by default to \(\frac{1}{2}\).

• Leaky bucket mechanism: it introduces time space between messages sending.
Table of Contents

- Introduction
- The PeDiT Algorithm
- Example
- Methods
- Conclusions
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