Tree mining I

Timo Petmanson
18. november 2009
Combinatorial Data Mining Algorithms (MTAT.03.249)
Abstract

• Applications of treemining
• Labeled, ordered, rooted trees.
• Embedded subtrees
• Counting support
• Candidate subtree generation
• Scope-lists
• The treeminer algorithm
• Extensions to the algorithm
Applications

• Web
  – Mining frequent user access paths on a website
• XML
  – Mining frequent subtrees in documents
• Chemistry
  – Most molecules can be represented as trees.
  – Given some unknown substance, one can get hints about the properties, based on similarities between known substances
• etc..

• Everything that can be represented as trees

It is a subject of frequent structure mining

Next: Labeled, ordered trees
Labeled, ordered, rooted trees

- A directed, acyclic, connected graph.
- One distinguished vertex as root.
- Unique path from root to every other vertex in the tree.
- Every vertex has a label associated
  - Let function $l(x)$ associate a label to vertex $x$
- If $x, y \in V$ and there is a path from $x$ to $y$, then $x$ is called an ancestor of $y$, denoted as $x \leq_p y$

$p$ is the length of the path
Labeled, ordered, rooted trees

- Every vertex \( x \) is synonymous with its position in the preorder traversal of the tree.
  - Note that this is not the same thing as with labels.
- The scope of vertex \( x \) is given as \( S(x) = [x, y] \)
Labeled, ordered, rooted trees

- Tree can be written as a string of labels
  - Start from root node
  - Add vertex labels in depth-first preorder traversal
  - If a backtrack is needed, write -1

String Encoding: 0 1 3 -1 2 -1 -1 -1 2 -1

Diagram of tree with labeled nodes.
Labeled, ordered, rooted trees

- Prefix of a tree is obtained by deleting rightmost leafs from it
  - Prefix obtained this way by deleting only 1 leaf is called an immediate prefix
- Trees $X$ and $Y$ are in same equivalence class, if they share a common immediate prefix

Next: Embedded subtrees
Embedded subtrees

$S = (V_S, E_S)$ is an embedded subtree of $T = (V_T, E_T)$, denoted as $S \subseteq_e T$, if there exists a 1-to-1 mapping $\varphi : V_S \rightarrow V_T$, such that

a) $(x, y) \in E_S$, if $\varphi(x) \leq_p \varphi(y)$

b) $l(x) = l(\varphi(x))$

Each occurrence of $S$ in $T$ can be identified by its unique match label: $\varphi(x_0)\varphi(x_1)\ldots\varphi(x_{|S|})$ where $x_i \in V_S$.
Embedded subtrees

$S = (V_S, E_S)$ is an embedded subtree of $T = (V_T, E_T)$, denoted as $S \subseteq_e T$, if there exists a 1-to-1 mapping $\varphi : V_S \rightarrow V_T$, such that

a) $(x, y) \in E_S$, if $\varphi(x) \leq_p \varphi(y)$

b) $l(x) = l(\varphi(x))$

Each occurrence of $S$ in $T$ can be identified by its unique match label: $\varphi(x_0)\varphi(x_1)…\varphi(x_{|S|})$ where $x_i \in V_S$
Embedded subtrees

$S = (V_S, E_S)$ is an embedded subtree of $T = (V_T, E_T)$, denoted as $S \subseteq_e T$, if there exists a 1-to-1 mapping $\varphi : V_S \rightarrow V_T$, such that

a) $(x, y) \in E_S$, if $\varphi(x) \leq_p \varphi(y)$

b) $l(x) = l(\varphi(x))$

Each occurrence of $S$ in $T$ can be identified by its unique match label: $\varphi(x_0)\varphi(x_1)\ldots\varphi(x_{|S|})$ where $x_i \in V_S$
Embedded subtrees

$S = (V_S, E_S)$ is an embedded subtree of $T = (V_T, E_T)$, denoted as $S \subseteq_e T$, if there exists a 1-to-1 mapping $\phi : V_S \to V_T$, such that

a) $(x, y) \in E_S$, if $\phi(x) \leq_p \phi(y)$

b) $l(x) = l(\phi(x))$

Each occurrence of $S$ in $T$ can be identified by its unique match label: $\phi(x_0)\phi(x_1)\ldots\phi(x_{|S|})$ where $x_i \in V_S$
Embedded subtrees

$S = (V_S, E_S)$ is an embedded subtree of $T = (V_T, E_T)$, denoted as $S \subseteq_e T$, if there exists a 1-to-1 mapping $\varphi : V_S \rightarrow V_T$, such that

a) $(x, y) \in E_S$, if $\varphi(x) \leq_p \varphi(y)$

b) $l(x) = l(\varphi(x))$

Each occurrence of $S$ in $T$ can be identified by its unique match label: $\varphi(x_0)\varphi(x_1)\ldots\varphi(x_{|S|})$ where $x_i \in V_S$
Embedded subtrees

$S = (V_S, E_S)$ is an embedded subtree of $T = (V_T, E_T)$, denoted as $S \subseteq_e T$, if there exists a 1-to-1 mapping $\varphi : V_S \rightarrow V_T$, such that

a) $(x, y) \in E_S$, if $\varphi(x) \leq_p \varphi(y)$

b) $l(x) = l(\varphi(x))$

Each occurrence of $S$ in $T$ can be identified by its unique match label: $\varphi(x_0)\varphi(x_1)\ldots\varphi(x_{|S|})$ where $x_i \in V_S$
Embedded subtrees

\[ S = (V_S, E_S) \] is an embedded subtree of \[ T = (V_T, E_T) \], denoted as \( S \subseteq_e T \), if there exists a 1-to-1 mapping \( \varphi : V_S \rightarrow V_T \), such that

a) \((x, y) \in E_S, \text{ if } \varphi(x) \leq_p \varphi(y)\)

b) \(l(x) = l(\varphi(x))\)

Each occurrence of \( S \) in \( T \) can be identified by its unique match label: \( \varphi(x_0)\varphi(x_1)\ldots\varphi(x_{|S|}) \) where \( x_i \in V_S \)

Next: Support
Support

Let $D$ be a forest, a database.

Let $S$ be a subtree candidate of $T$

Define $\delta_T(S)$ to denote the number of occurrences of $S$ in $T$.

Define $d_T(S) = 1$, if $\delta_T(S) > 0$ and $d_T(S) = 0$ otherwise

Support of $S$ in database $D$ is

$$\text{supp}(S) = \sum_{T \in D} d_T(S)$$

Weighted support is

$$\text{supp}_w(S) = \sum_{T \in D} \delta_T(S)$$
Support

\[ \text{supp}(S) = \sum_{T \in D} d_T(S) \]
\[ \text{supp}_w(S) = \sum_{T \in D} \delta_T(S) \]

T's String Encoding: 0 1 3 1 -1 2 -1 -1 -1 2 -1

Next: Candidate subtree generation
Candidate subtree generation

• If subtree $S$ has $k$ nodes, we will also call it a $k$-subtree
• We will use frequent $k$-subtrees to generate candidate $(k+1)$-subtrees
• Recall equivalence classes
  – Two $k$-subtrees $X,Y$ are in the same \textit{prefix equivalence class}, if they share a common prefix up to the $(k-1)'th$ node
  – \textit{Prefix} of some tree is also a tree
Candidate subtree generation

Let $P$ be a prefix subtree of size $k - 1$
Let $[P]_{k-1}$ refer to the equivalence class of $P$
Let tuple $(x, i) \in [P]_{k-1}$ be an element of $P$, where $x$ is a label of a node attached to a node at $i$'th position in depth-first preorder traversal in $P$
Candidate subtree generation

Let $P_x^i$ be an extension of $P$ with element $(x, i)$
Let $[P_x^i]$ denote the class of possible extensions of $P_x^i$
Candidate subtree generation

- Algorithm for extending equivalence classes
  Let us have a \((k-1)\)-subtree \(P\) as a prefix class
  Let \((x, i), (y, j) \in P\) be two elements of that class
  Case 1: \(i = j\)
    
    If \(P \neq \emptyset\), then add \((y, j)\) and \((y, n_i)\) to class \([P_x^i]\)
    
    \(n_i\) is the preorder index if \((x, i)\) in \(P_x^i\)
    
    If \(P = \emptyset\), then add \((y, j + 1)\) to class \([P_x^i]\)
  
  Case 2: \(i > j\)
    
    Add \((y, j)\) to class \([P_x^i]\)
  
  Case 3: \(i < j\)
    
    No new candidate is possible in that case

- All possible \((k+1)\)-subtrees with prefix \(P\) will be enumerated by applying described procedure to each ordered pair of elements \((x,i)\) and \((y,j)\)

Don’t worry, we’ll see an example on the blackboard at once ...
Candidate subtree generation

Equivalence Class
Prefix: 1 2
Element List: (3,1) (4,0)

Prefix: 1 2 3
Element List: (3,1) (3,2) (4,0)

Prefix: 1 2 4
Element List: (4,0) (4,1)

Next: Scope lists
Scope-lists

- We use *scope-lists* for fast support counting.
- There is one *scope-list* per candidate subtree.
- Basically contains tuples in format \((t, m, s)\)
  - \(t\) is a tree id where the subtree occurs
  - \(m\) is a match label of subtrees immediate prefix
  - \(s\) is the scope of the last item the rightmost node

Next: example of a scope-list
Scope-lists

- Scope-list for 1-subtrees:

Database D of 3 Trees

Tree T0
- n0, [0, 3]
- n1, [1, 1]
- n2, [2, 3]
- n3, [3, 3]

Tree T1
- n0, [0, 5]
- n1, [1, 3]
- n2, [2, 2]
- n3, [3, 3]
- n4, [4, 4]
- n5, [5, 5]

Tree T2
- n0, [0, 7]
- n1, [1, 2]
- n2, [2, 2]
- n3, [3, 7]
- n4, [4, 7]
- n5, [5, 5]
- n6, [6, 7]
- n7, [7, 7]

D in Horizontal Format: (tid, string encoding)
- (T0, 1 2 -1 3 4 -1 -1)
- (T1, 2 1 2 -1 4 -1 -1 2 -1 3 -1)
- (T2, 1 3 2 -1 5 1 2 -1 3 4 -1 -1 -1 -1)

D in Vertical Format: (tid, scope) pairs

<table>
<thead>
<tr>
<th>TID</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0, 3]</td>
</tr>
<tr>
<td>1</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>2</td>
<td>[0, 7]</td>
</tr>
<tr>
<td>3</td>
<td>[4, 7]</td>
</tr>
<tr>
<td>4</td>
<td>[5, 7]</td>
</tr>
</tbody>
</table>

NB! Last element is omitted from match label as it can be identified by scope.
Scope-lists

- Scope-list for 1-subtrees:

Database D of 3 Trees

D in Horizontal Format: (tid, string encoding)
(T0, 1 2 −1 3 4 −1 −1)
(T1, 2 1 2 −1 4 −1 −1 2 −1 3 −1)
(T2, 1 3 2 −1 5 1 2 −1 3 4 −1 −1 −1)

D in Vertical Format: (tid, scope) pairs

1 2 3 4 5

0, [0, 3] 0, [1, 1] 0, [2, 3] 0, [3, 3] 2, [3, 7]
1, [1, 3] 1, [0, 5] 1, [3, 3] 2, [7, 7]
2, [0, 7] 1, [2, 2] 1, [5, 5] 2, [6, 7]
2, [4, 7] 1, [4, 4] 2, [1, 2] 2, [7, 7]
Scope-lists

- Scope-list for 1-subtrees:

Database D of 3 Trees

Tree T0
- n0, [0,3]
  - 1
    - 2
      - n1, [1,1]
        - 4
          - n3, [3,3]

Tree T1
- n0, [0,5]
  - 1
    - n1, [1,3]
      - 2
        - n2, [2,2]
          - 4
            - n3, [3,3]
        - 3

Tree T2
- n0, [0,7]
  - 1
    - n1, [1,2]
      - 3
        - n2, [2,2]
          - 2
            - n5, [5,5]
          - 4
        - 5
          - n3, [3,7]
          - n4, [4,7]
          - n6, [6,7]
          - n7, [7,7]

D in Horizontal Format: (tid, string encoding)
- (T0, 1 2 −1 3 4 −1 −1)
- (T1, 2 1 2 −1 4 −1 −1 2 −1 3 −1)
- (T2, 1 3 2 −1 5 2 −1 3 4 −1 −1 −1 −1)

D in Vertical Format: (tid, scope) pairs

<table>
<thead>
<tr>
<th>Tid</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0, [0, 3]</td>
</tr>
<tr>
<td>1</td>
<td>1, [1, 3]</td>
</tr>
<tr>
<td>2</td>
<td>0, [2, 7]</td>
</tr>
<tr>
<td>2</td>
<td>2, [4, 7]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tid</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[1,1]</td>
</tr>
<tr>
<td>1</td>
<td>[0,5]</td>
</tr>
<tr>
<td>2</td>
<td>[2,2]</td>
</tr>
<tr>
<td>2</td>
<td>[4,4]</td>
</tr>
<tr>
<td>2</td>
<td>[2,2]</td>
</tr>
<tr>
<td>2</td>
<td>[5,5]</td>
</tr>
</tbody>
</table>
Scope-lists

- Scope-list for 1-subtrees:

Database D of 3 Trees

D in Horizontal Format: (tid, string encoding)
(T0, 1 2 -1 3 4 -1 -1)
(T1, 2 1 2 -1 4 -1 -1 2 -1 3 -1)
(T2, 1 3 2 -1 5 1 2 -1 3 4 -1 -1 -1)

D in Vertical Format: (tid, scope) pairs

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0, 3]</td>
<td>0</td>
<td>[2, 3]</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>[0, 7]</td>
<td>2</td>
<td>[2, 2]</td>
<td>2</td>
</tr>
</tbody>
</table>
Scope-lists

- Scope-list for 1-subtrees:

Database D of 3 Trees

Tree T0
1
  2
  n1, [1, 1]
  n0, [0.3]

2
 n2, [2, 3]
 n3, [3, 3]

Tree T1
2
  1
  n1, [1, 3]

3
 n1, [1, 3]
 n2, [2, 2]
 n3, [3, 3]

Tree T2
1
  3
  n1, [1, 2]

2
 n2, [2, 2]
 n5, [5, 5]
 n4, [4, 7]

3
 n3, [3, 7]
 n6, [6, 7]

4
 n7, [7, 7]

D in Horizontal Format: (tid, string encoding)
(T0, 1 2 1 3 4 1 1 1)
(T1, 2 1 2 1 4 1 1 2 1 3 1)
(T2, 1 3 2 1 5 1 2 1 3 4 1 1 1 1)

D in Vertical Format: (tid, scope) pairs

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0, 3]</td>
<td>0, [1, 1]</td>
<td>0, [2, 3]</td>
<td>0, [3, 3]</td>
<td>2, [3, 7]</td>
</tr>
<tr>
<td>1</td>
<td>[1, 3]</td>
<td>0, [1, 1]</td>
<td>1, [0, 5]</td>
<td>1, [3, 3]</td>
<td>2, [7, 7]</td>
</tr>
<tr>
<td>2</td>
<td>[0, 7]</td>
<td>1, [0, 5]</td>
<td>1, [2, 2]</td>
<td>2, [2, 2]</td>
<td>2, [5, 5]</td>
</tr>
</tbody>
</table>
Scope-lists

- Scope-list for 1-subtrees:

Database D of 3 Trees

Tree T0:
- n0, [0,3]
- n1, [1,1]
- n2, [2,3]
- n3, [3,3]

Tree T1:
- n0, [0,5]
- n1, [1,3]
- n2, [2,2]
- n3, [3,3]
- n4, [4,4]
- n5, [5,5]

Tree T2:
- n0, [0,7]
- n1, [1,2]
- n2, [2,2]
- n3, [3,7]
- n4, [4,7]
- n5, [5,5]
- n6, [6,7]
- n7, [7,7]

D in Horizontal Format: (tid, string encoding)
- (T0, 1 2 -1 3 4 -1 -1)
- (T1, 2 1 2 -1 4 -1 -1 2 -1 3 -1)
- (T2, 1 3 2 -1 5 1 2 -1 3 4 -1 -1 -1)

D in Vertical Format: (tid, scope) pairs

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, [0, 3]</td>
<td>0, [1, 1]</td>
<td>0, [2, 3]</td>
<td>0, [3, 3]</td>
<td>2, [3, 7]</td>
</tr>
<tr>
<td>1, [1, 3]</td>
<td>1, [0, 5]</td>
<td>1, [5, 5]</td>
<td>1, [3, 3]</td>
<td>2, [7, 7]</td>
</tr>
<tr>
<td>2, [0, 7]</td>
<td>1, [2, 2]</td>
<td>2, [1, 2]</td>
<td>2, [6, 7]</td>
<td>2, [5, 5]</td>
</tr>
</tbody>
</table>
Scope-lists

• Scope-list for 1-subtrees:

Database D of 3 Trees

Tree T0
n0, [0, 3]
  1
  2
  n1, [1, 1]
  2
  n2, [2, 3]
  4
  n3, [3, 3]

Tree T1
n0, [0, 5]
  2
  n1, [1, 3]
  2
  2
  n4, [4, 4]
  3
  n5, [5, 5]

Tree T2
n0, [0, 7]
  1
  3
  n1, [1, 2]
  3
  n2, [2, 2]
  2
  n3, [3, 7]
  5
  n4, [4, 7]
  4
  n5, [5, 5]
  6
  n6, [6, 7]
  7
  n7, [7, 7]

D in Horizontal Format: (tid, string encoding)
(T0, 1 2 1 3 4 1 1)
(T1, 2 1 2 1 4 1 1 1 2 1 3 1)
(T2, 1 3 2 1 5 1 2 1 3 4 1 1 1)

D in Vertical Format: (tid, scope) pairs

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, [0, 3]</td>
<td>0, [1, 1]</td>
<td>0, [2, 3]</td>
<td>0, [3, 3]</td>
<td>2, [3, 7]</td>
</tr>
<tr>
<td>1, [1, 3]</td>
<td>1, [0, 5]</td>
<td>1, [3, 3]</td>
<td>1, [3, 3]</td>
<td>2, [7, 7]</td>
</tr>
<tr>
<td>2, [0, 7]</td>
<td>1, [2, 2]</td>
<td>1, [5, 5]</td>
<td>1, [5, 5]</td>
<td></td>
</tr>
<tr>
<td>2, [4, 7]</td>
<td>1, [4, 4]</td>
<td>2, [1, 2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n4, [4, 4]</td>
<td>2, [2, 2]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n5, [5, 5]</td>
<td>2, [5, 5]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scope-lists

- Scope-list for 1-subtrees:

**Database D of 3 Trees**

**Tree T0**
- n0, [0, 3]
- n1, [1, 1]
- n2, [2, 3]
- n3, [3, 3]

**Tree T1**
- n0, [0, 5]
- n1, [1, 3]
- n2, [2, 2]
- n3, [3, 3]
- n4, [4, 4]
- n5, [5, 5]

**Tree T2**
- n0, [0, 7]
- n1, [1, 2]
- n2, [2, 2]
- n3, [3, 7]
- n4, [4, 7]
- n5, [5, 5]
- n6, [6, 7]
- n7, [7, 7]

**D in Horizontal Format:** (tid, string encoding)
- (T0, 1 2 -1 3 4 -1 -1)
- (T1, 2 1 2 -1 4 -1 -1 2 -1 3 -1)
- (T2, 1 3 2 -1 5 1 2 -1 3 4 -1 -1 -1)

**D in Vertical Format:** (tid, scope) pairs

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0, 3]</td>
<td>0, [1, 1]</td>
<td>0, [2, 3]</td>
<td>0, [3, 3]</td>
<td>2, [3, 7]</td>
</tr>
<tr>
<td>1</td>
<td>[1, 3]</td>
<td>1, [0, 5]</td>
<td>1, [2, 3]</td>
<td>1, [3, 3]</td>
<td>2, [7, 7]</td>
</tr>
<tr>
<td>2</td>
<td>[0, 7]</td>
<td>2, [2, 2]</td>
<td>2, [5, 5]</td>
<td>2, [6, 7]</td>
<td></td>
</tr>
</tbody>
</table>

Next: Scope-list joins
Scope-list joins

• Given two elements ($k$-subtrees) in equivalence prefix class $[P]$, we can compute the scope-list for a candidate ($k+1$)-subtrees obtained by equivalence class extension algorithm discussed before.

• For that we have 2 methods:
  – In-Scope Test
  – Out-Scope Test
Scope-list joins

\[ \forall \text{tuple } (t_x, m_x, s_x = [l_x, u_x]) \text{ in scope - list of element } (x,i) \]

\[ \forall \text{tuple } (t_y, m_y, s_y = [l_y, u_y]) \text{ in scope - list of element } (y,j) \]

If \( t_x = t_y \) and \( m_x = m_y \):

If \( i = j \) and \( s_y \subseteq s_x \):  // in - scope test

\[ \text{add triple } (t_y, \{ m_y \cup l_x \}, s_y) \text{ to the scope list of}(y,j+1) \]

in \( [P^i_x] \)

Else if \( s_x < s_y \):  // out - scope test

\[ \text{add triple } (t_y, \{ m_y \cup l_x \}, s_y) \text{ to the scope list of}(y, j) \]

in \( [P^i_x] \)

Do not worry, we’ll see an example on the blackboard at once ..
Scope-list joins

Prefix = {}
Elements = (1,−1), (2,−1), (3,−1), (4,−1)

1
0, [0, 3]
1, [1, 3]
2, [0, 7]
2, [4, 7]

2
0, [1, 1]
1, [0, 5]
1, [2, 2]
2, [4, 4]
2, [2, 2]

3
0, [2, 3]
1, [5, 5]
2, [1, 2]
2, [6, 7]

4
0, [3, 3]
1, [3, 3]
2, [7, 7]

Infrequent Elements
(5,−1): 5

Prefix = 1
Elements = (2,0), (4,0)

1

2
0, 0, [1, 1]
1, 1, [2, 2]
2, 0, [2, 2]
2, 0, [5, 5]
2, 4, [5, 5]

4
0, 0, [3, 3]
1, 1, [3, 3]
2, 0, [7, 7]
2, 4, [7, 7]

Infrequent Elements
(1,0) : 1 1 −1
(3,0) : 1 3 −1

Prefix = 12
Elements = (4,0)

1

2
0, 01, [3, 3]
1, 12, [3, 3]
2, 02, [7, 7]
2, 05, [7, 7]
2, 45, [7, 7]

4

Infrequent Elements
(2,0) : 1 2 −1 2
(2,1) : 1 2 2 −1 −1
(4,1) : 1 2 4 −1 −1

Database on next slide ...
Scope-list joins

Database $D$ of 3 Trees

Tree $T_0$
- $n_0, [0,3]$
- $n_1, [1,1]$
- $n_2, [2,3]$
- $n_3, [3,3]$

Tree $T_1$
- $n_0, [0,5]$
- $n_1, [1,3]$
- $n_2, [2,2]$
- $n_3, [3,3]$

Tree $T_2$
- $n_0, [0,7]$
- $n_1, [1,2]$
- $n_2, [2,2]$
- $n_3, [3,7]$
- $n_4, [4,7]$
- $n_5, [5,5]$
- $n_6, [6,7]$
- $n_7, [7,7]$

$D$ in Horizontal Format: (tid, string encoding)
- $T_0$, 1 2 1 3 4 1 1
- $T_1$, 2 1 2 1 4 1 1 2 1 3 1
- $T_2$, 1 3 2 1 5 1 2 1 3 4 1 1 1 1

$D$ in Vertical Format: (tid, scope) pairs

Example on previous slide ..
Next: The Treeminer algorithm
Treeminer Algorithm

\textsc{TreeMiner} \ (D, \ \text{mins}\up{u}p): \\
\quad F_1 = \{ \ \text{frequent 1-subtrees} \}; \\
\quad F_2 = \{ \ \text{classes } [P]_1 \text{ of frequent 2-subtrees} \}; \\
\quad \text{for all } [P]_1 \in E \ \text{do} \ \text{Enumerate-Frequent-Subtrees}([P]_1); \\

\textsc{Enumerate-Frequent-Subtrees}([P]): \\
\quad \text{for each element } (x, i) \in [P] \ \text{do} \\
\quad \quad [P^i_x] = \emptyset; \\
\quad \quad \text{for each element } (y, j) \in [P] \ \text{do} \\
\quad \quad \quad R = \{(x, i) \otimes (y, j)\}; \\
\quad \quad \quad \mathcal{L}(R) = \{\mathcal{L}(x) \cap \otimes \mathcal{L}(y)\}; \\
\quad \quad \quad \text{if for any } R \in R, \ R \text{ is frequent then} \\
\quad \quad \quad \quad [P^i_x] = [P^i_x] \cup \{R\}; \\
\quad \quad \text{Enumerate-Frequent-Subtrees}([P^i_x]);
Treeminer Algorithm

TREEMINER (D, \textit{minsop}):
\begin{align*}
F_1 &= \{ \text{frequent 1-subtrees} \}; \\
F_2 &= \{ \text{classes } [P]_1 \text{ of frequent 2-subtrees} \}; \\
\textbf{for all } [P]_1 \in E \textbf{ do } Enumerate-Frequent-Subtrees([P]_1);
\end{align*}

\textbf{Enume}rate-Frequent-Subtrees([P]):
\begin{align*}
\textbf{for each element } (x, i) \in [P] \textbf{ do} \\
\ [P^i_x] &= \emptyset; \\
\textbf{for each element } (y, j) \in [P] \textbf{ do} \\
R &= \{(x, i) \otimes (y, j)\}; \\
\mathcal{L}(R) &= \{\mathcal{L}(x) \cap_{\otimes} \mathcal{L}(y)\}; \\
\textbf{if for any } R \in R, R \text{ is frequent then} \\
\ [P^i_x] &= [P^i_x] \cup \{R\}; \\
\text{Enumerate-Frequent-Subtrees([P^i_x])};
\end{align*}

Simplest step, we count occurrences of each element in each tree. This takes $O(n)$ time per tree, where $n$ is the number of nodes in that tree.
Treeminer Algorithm

TREEMINER (D, \textit{minsup}):  
\[ F_1 = \{ \text{frequent 1-subtrees} \}; \]
\[ F_2 = \{ \text{classes } [P]_1 \text{ of frequent 2-subtrees} \}; \]
\[ \text{for all } [P]_1 \in E \text{ do } \text{Enumerate-Frequent-Subtrees}([P]_1); \]

ENUMERATE-FREQUENT-SUBTREES([P]):  
\[ \text{for each element } (x, i) \in [P] \text{ do} \]
\[ [P^i_x] = \emptyset; \]
\[ \text{for each element } (y, j) \in [P] \text{ do} \]
\[ R = \{(x, i) \otimes (y, j)\}; \]
\[ \mathcal{L}(R) = \{\mathcal{L}(x) \cap \mathcal{L}(y)\}; \]
\[ \text{if for any } R \in R, R \text{ is frequent then} \]
\[ [P^i_x] = [P^i_x] \cup \{R\}; \]
\[ \text{Enumerate-Frequent-Subtrees}([P^i_x]); \]

We use frequent 1-subtrees as prefix classes and tuples \((j,0)\) as their elements where \(j \geq i\). String representations for candidate subtrees are in form \((i j -1)\). This step takes \(O(n^*n)\) time.
Treeminer Algorithm

**TreeMiner** $(D, \text{minsup})$:

- $F_1 = \{\text{frequent 1-subtrees}\}$;
- $F_2 = \{\text{classes } [P]_1 \text{ of frequent 2-subtrees}\}$;
- **for all** $[P]_1 \in E$ **do** $\text{Enumerate-Frequent-Subtrees}([P]_1)$;

**Enumerate-Frequent-Subtrees**$([P])$:

- **for each** element $(x, i) \in [P]$ **do**
  - $[P_x^i] = \emptyset$;
  - **for each** element $(y, j) \in [P]$ **do**
    - $R = \{(x, i) \otimes (y, j)\}$;
    - $\mathcal{L}(R) = \{\mathcal{L}(x) \cap \mathcal{L}(y)\}$;
    - **if** for any $R \in R$, $R$ is frequent **then**
      - $[P_x^i] = [P_x^i] \cup \{R\}$;
      - $\text{Enumerate-Frequent-Subtrees}([P_x^i])$;

For all prefix classes $[P]_1$, enumerate it’s frequent subtrees. Recall that we use the elements of given prefix class to do that.
Treeminer Algorithm

\textsc{TreeMiner} (D, \textit{minsdp}):\n\begin{align*}
F_1 &= \{ \text{frequent 1-subtrees } \}; \\
F_2 &= \{ \text{classes } [P]_1 \text{ of frequent 2-subtrees } \}; \\
\text{for all } [P]_1 \in E \text{ do } \text{Enumerate-Frequent-Subtrees}([P]_1); \\
\end{align*}

\textsc{Enumerate-Frequent-Subtrees}([P]):
\begin{align*}
\text{for each element } (x, i) \in [P] \text{ do } \\
[P^i_x] &= \emptyset; \\
\text{for each element } (y, j) \in [P] \text{ do } \\
R &= \{(x, i) \otimes (y, j)\}; \\
\mathcal{L}(R) &= \{\mathcal{L}(x) \cap \otimes \mathcal{L}(y)\}; \\
\text{if for any } R \in R, R \text{ is frequent then } \\
[P^i_x] &= [P^i_x] \cup \{R\}; \\
\text{Enumerate-Frequent-Subtrees}([P^i_x]);
\end{align*}
Treeminer Algorithm

**TreeMiner** (D, minsup):

\[ F_1 = \{ \text{frequent 1-subtrees} \}; \]

\[ F_2 = \{ \text{classes } [P]_1 \text{ of frequent 2-subtrees} \}; \]

**for** all \([P]_1 \in E** do** Enumerate-Frequent-Subtrees([P]_1);

**Enumerate-Frequent-Subtrees**([P]):

**for each** element \((x, i) \in [P] \) **do**

\[ [P^i_x] = \emptyset; \]

**for each** element \((y, j) \in [P] \) **do**

\[ R = \{(x, i) \otimes (y, j)\}; \]

\[ \mathcal{L}(R) = \{\mathcal{L}(x) \cap_{\otimes} \mathcal{L}(y)\}; \]

**if** for any \( R \in R \), \( R \) is frequent **then**

\[ [P^i_x] = [P^i_x] \cup \{R\}; \]

Enumerate-Frequent-Subtrees([P^i_x]);
Treeminer Algorithm

TREE_MINER \((D, \text{minsup})\):

\(F_1 = \{ \text{frequent 1-subtrees} \};\)
\(F_2 = \{ \text{classes} [P]_1 \text{ of frequent 2-subtrees} \};\)

for all \([P]_1 \in E\) do Enumerate-Frequent-Subtrees([P]_1);

ENUMERATE-FREQUENT-SUBTREES([P]):

for each element \((x, i) \in [P]\) do

\([P_x^i] = \emptyset;\)

for each element \((y, j) \in [P]\) do

\(R = \{(x, i) \otimes (y, j)\};\)
\(\mathcal{L}(R) = \{\mathcal{L}(x) \cap_{\otimes} \mathcal{L}(y)\};\)

if for any \(R \in R, R\) is frequent then

\([P_x^i] = [P_x^i] \cup \{R\};\)

Enumerate-Frequent-Subtrees([P_x^i]);
Treeminer Algorithm

**TREEMINER** (D, $\minsup$):

$F_1 = \{ \text{frequent 1-subtrees} \}$;
$F_2 = \{ \text{classes } [P]_1 \text{ of frequent 2-subtrees} \}$

**for all** $[P]_1 \in E$ **do** Enumerate-Frequent-Subtrees($[P]_1$);

**Enumerate-Frequent-Subtrees**($[P]$):

**for each** element $(x, i) \in [P]$ **do**

$[P^i_x] = \emptyset$;

**for each** element $(y, j) \in [P]$ **do**

$R = \{(x, i) \otimes (y, j)\}$;
$\mathcal{L}(R) = \{\mathcal{L}(x) \cap \otimes \mathcal{L}(y)\}$;

**if** for any $R \in R$, $R$ is frequent **then**

$[P^i_x] = [P^i_x] \cup \{R\}$;

Enumerate-Frequent-Subtrees($[P^i_x]$);

R denotes the two possible candidate subtrees, that are enumerated from equivalence class $P$ and it’s elements $(x, i)$ and $(y, j)$
Treeminer Algorithm

\[ \text{TREE} \text{MINER (D, minsup):} \]
\[ F_1 = \{ \text{frequent 1-subtrees} \}; \]
\[ F_2 = \{ \text{classes [P]_1 of frequent 2-subtrees} \}; \]
\[ \text{for all [P]_1 \in E do Enumerate-Frequent-Subtrees([P]_1);} \]

\[ \text{ENUMERATE-FREQUENT-SUBTREES([P]):} \]
\[ \text{for each element (x, i) \in [P] do} \]
\[ [P^i_x] = \emptyset; \]
\[ \text{for each element (y, j) \in [P] do} \]
\[ R = \{(x, i) \otimes (y, j)\}; \]
\[ \mathcal{L}(R) = \{\mathcal{L}(x) \cap_{\otimes} \mathcal{L}(y)\}; \]
\[ \text{if for any } R \in R, R \text{ is frequent then} \]
\[ [P^i_x] = [P^i_x] \cup \{R\}; \]
\[ \text{Enumerate-Frequent-Subtrees([P^i_x])}; \]

Join the scope-lists of elements x and y.
Treeminer Algorithm

\textsc{TreeMiner} (D, \textit{minsup}):
\begin{align*}
F_1 &= \{ \text{frequent 1-subtrees } \}; \\
F_2 &= \{ \text{classes } [P]_1 \text{ of frequent 2-subtrees } \}; \\
\text{for all } [P]_1 \in E &\text{ do } \text{Enumerate-Frequent-Subtrees}([P]_1);
\end{align*}

\textsc{Enumerate-Frequent-Subtrees}([P]):
\begin{align*}
\text{for each element } (x, i) \in [P] &\text{ do } \\
[P_x^i] &= \emptyset; \\
\text{for each element } (y, j) \in [P] &\text{ do } \\
R &= \{(x, i) \otimes (y, j)\}; \\
\mathcal{L}(R) &= \{\mathcal{L}(x) \cap \mathcal{L}(y)\}; \\
\text{if for any } R \in R, R \text{ is frequent } &\text{ then } \\
[P_x^i] &= [P_x^i] \cup \{R\}; \\
\text{Enumerate-Frequent-Subtrees}([P_x^i]);
\end{align*}

If any of the possible two candidate subtrees is frequent, then ...
Treeminer Algorithm

\textsc{TreeMiner} (D, \textit{minsup}): 
\begin{align*} 
F_1 &= \{ \text{frequent 1-subtrees} \}; \\
F_2 &= \{ \text{classes } [P]_1 \text{ of frequent 2-subtrees} \}; \\
& \text{for all } [P]_1 \in E \text{ do } \text{Enumerate-Frequent-Subtrees}([P]_1); 
\end{align*}

\textsc{Enumerate-Frequent-Subtrees}([P]): 
\begin{align*} 
& \text{for each element } (x, i) \in [P] \text{ do} \\
& \quad [P^i_x] = \emptyset; \\
& \quad \text{for each element } (y, j) \in [P] \text{ do} \\
& \quad \quad R = \{(x, i) \otimes (y, j)\}; \\
& \quad \quad \mathcal{L}(R) = \{\mathcal{L}(x) \cap \otimes \mathcal{L}(y)\}; \\
& \quad \quad \text{if for any } R \in R, \ R \text{ is frequent then} \\
& \quad \quad \quad [P^i_x] = [P^i_x] \cup \{R\}; \\
& \quad \text{Enumerate-Frequent-Subtrees}([P^i_x]); 
\end{align*}
**Treeminer Algorithm**

\[ \text{TREEMINER (D, minsup):} \]
- \[ F_1 = \{ \text{frequent 1-subtrees} \} \];
- \[ F_2 = \{ \text{classes } [P]_1 \text{ of frequent 2-subtrees} \} \];
- \text{for all } [P]_1 \in E \text{ do Enumerate-Frequent-Subtrees([P]_1);} \]

**Enumerate-Frequent-Subtrees([P]):**
- \text{for each element } (x, i) \in [P] \text{ do}
  - \[ [P_x^i] = \emptyset; \]
  - \text{for each element } (y, j) \in [P] \text{ do}
    - \( R = \{(x, i) \otimes (y, j)\}; \)
    - \( \mathcal{L}(R) = \{\mathcal{L}(x) \cap \otimes \mathcal{L}(y)\}; \)
    - \text{if for any } R \in R, R \text{ is frequent then}
      - \[ [P_x^i] = [P_x^i] \cup \{R\}; \]
      - Enumerate-Frequent-Subtrees([P_x^i]);

Continue the whole procedure with frequent elements of this equivalence class.
Treeminer Algorithm: an example
Extending the Treeminer algorithm

- By modifying the *scope-lists* used by Treeminer algorithm we can count distinct occurrences of subtrees instead of *weighted support*.
- We can also extend Treeminer to discover *sub-forests*.
- We can modify Treeminer to discover frequent *free (unordered) trees*.
Questions?
The end.