0.1 Strictly k-Local Treesets

Trees are like strings in that they are recursive structures. Informally, trees are structures with a single 'root' which dominates a sequence of trees.

0.1.1 Defining Trees and Treesets

Formally, trees extend the dimensionality of string structures from 1 to 2 (Rogers, 2003). Like strings, we assume a set of symbols Σ. This is often partioned into symbols of different types depending on whether the symbols can only occur at the leaves of the trees or not. We don't make any such distinction here.

Definition 1 (Trees)**.**

Base Case: If $a \in \Sigma$ then $a[\,]$ is a tree.

Inductive Case: If $a \in \Sigma$ and $t_1, t_2, \ldots t_n$ is a string of trees of length n then $a[t_1 t_2 \ldots t_n]$ is a tree.

Also, a tree $a[]$ is called a *leaf*. We denote set of all possible trees with $\mathbb{T}^2_\Sigma$. A *treeset* T is a subset of $\mathbb{T}^2_\Sigma$.

(This notation follows Rogers (2003) wherein $\mathbb{T}^d_\Sigma$ denotes tree-like structures with Σ being the set of labels and d being the dimensionality. Since strings are of dimension 1, this means the set of all strings Σ^* is equivalent to $\mathbb{T}^1_\Sigma$.)

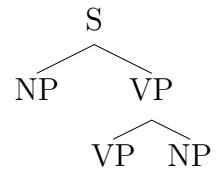
Here are some examples.

Example 1. Let $\Sigma = \{\text{NP, VP, S}\}$. Then the following are trees.

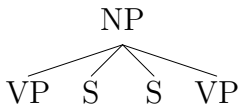
1. S[NP[ ] VP[VP[ ] *NP*[ ]]]
2. NP[VP[ ] S[ ] S[ ] VP[ ]]
3. NP[NP[NP[ ] VP[ ] S[ ]]]

We might draw these structures as follows.

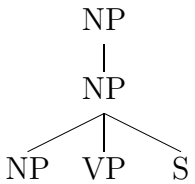
(1)



(2)



(3)



Note that the expression "a string of trees" in the definition of trees implies that our alphabet for strings is all the trees. Since we are defining trees, this may seem a bit circular. The key to resolving this circularity is to interleave the definition of the alphabet of the strings with the definition of trees in a zig-zag fashion. First we apply the inductive case for trees *once*, then we use those trees as an alphabet to define some strings of trees. Then we

go back to trees and apply the inductive case again, which yields more trees which we can use to enlarge our set of strings and so on. While we do not go through the details here, this method essentially provides a way to enumerate the set of all possible trees.

Here are some useful definitions which give us information about trees.

Definition 2.

1. The *root* of a tree $a[t_1 \dots t_n]$ is a.
2. The *size* of a tree t, written $|t|$, is defined as follows. If $t = a[\ ]$, its size is 1. If not, then $t = a[t_1 t_2 \dots t_n]$ where each t_i is a tree. Then $|t| = 1 + |t_1| + |t_2| + \dots + |t_n|$.
3. The *depth* of a tree t, written $\texttt{depth}(t)$, is defined as follows. If $t = a[\ ]$, its depth is 1. If not, then $t = a[t_1 t_2 \dots t_n]$ where each t_i is a tree. Then $\texttt{depth}(t) = 1 + \texttt{max}\big\{\texttt{depth}(t_1), \texttt{depth}(t_2), \dots, \texttt{depth}(t_n)\big\}$ where $\texttt{max}$ takes the largest number in the set.
4. The *yield* of a tree t, written $\texttt{yield}(t)$, maps a tree to a string of its leaves as follows. If $t = a[\ ]$ then $\texttt{yield}(t) = a$. If not, then $t = a[t_1 t_2 \dots t_n]$ where each t_i is a tree. Then $\texttt{yield}(t) = \texttt{yield}(t_1) \cdot \texttt{yield}(t_2) \cdot \dots \cdot \texttt{yield}(t_n)$.
5. Tree t is a *subtree* of $t' = a[t_1 \dots t_n]$ provided there is i such that either $t = t_i$ or t is a subtree of t_i.
6. A tree $t = a[a_1[\ ] \dots a_n[\ ]\ ]$ is a *2-local tree* of tree t' ($t \trianglelefteq t'$) provided there exists a subtree s of t' such that $s = a[t_1 \dots t_n]$ and the root of each t_i is a_i.
7. A 1-*treetop* of $t = a[t_1 t_2 \dots t_n]$ is $a[\ ]$.
8. A k-*treetop* of $t = a[t_1 t_2 \dots t_n]$ is $a[s_1 s_2 \dots s_n]$ where each s_i is a $(k-1)$-treetop of t_i.
9. A tree $t = a[t_1 \dots t_n]$ is a k-*local tree* of tree t' ($t \trianglelefteq t'$) provided
   (a) t is of depth k
   (b) there exists a subtree s of t' such that $s = a[s_1 \dots s_n]$ and for each i, t_i is the $(k-1)$ treetop of s_i.

0.1.2 Strictly Local Treesets

The notion of k-*local tree* ($\trianglelefteq$) is integral to Strictly Local treesets.

As with SL stringsets, we define a function $\texttt{factor}_k$, which extracts the k-local trees present in a tree (or set of trees). If a tree t is of depth less than k then $\texttt{factor}_k$ just returns $\{t\}$.

Formally, let $\texttt{factor}_k(t)$ equal $\{s \mid s \trianglelefteq t, \texttt{depth}(s) = k\}$ whenever $k \leq \texttt{depth}(t)$ and let $\texttt{factor}_k(t) = \{t\}$ whenever $\texttt{depth}(t) < k$. We expand the domain of this function to include sets of strings as follows: $\texttt{factor}_k(T) = \bigcup_{t \in T} \texttt{factor}_k(t)$.

Then a SL_k treeset grammar $G = (\Sigma_0, \Sigma_\ell, F_k)$, which will be interpreted as the symbols permissible as the roots, the symbols permissible as the leaves, and the permissible k-Local trees. So $\Sigma_0, \Sigma_\ell \subseteq \Sigma$ and F_k is a finite subset of $\texttt{factor}_k(\mathbb{T}^2_\Sigma)$.

The "language of the grammar" $L(G)$ is defined as the treeset

$$L\big((\Sigma_0, \Sigma_\ell, F_k)\big) \stackrel{\text{def}}{=} \{t \mid \texttt{root}(t) \in \Sigma_0, \texttt{yield}(t) \in \Sigma_\ell^*, \texttt{factor}_k(t) \subseteq F_k, \} \ .$$

We are going to be interested in the collection of treesets SLT_k, defined as those treesets generated from grammars G where the depth of the largest permissible local tree is k. Formally,

$$\text{SLT}_k \stackrel{\text{def}}{=} \{T \mid \exists G = (\Sigma_0, \Sigma_\ell, F_k), \Sigma_0 \subseteq \Sigma,\ \Sigma_\ell \subseteq \Sigma,\ F_k \subseteq \texttt{factor}_k(\mathbb{T}^2_\Sigma),\ F_k \text{ finite},\ L(G) = T\} \ .$$

This is the collection C of learning targets.

Theorem 1. *For each k, the class k-SLT is identifiable in the limit from positive data.*

To make this result clear, let us remind ourselves what a positive presentation of data is. It means for each $t \in T$ there is some time point i such that $\phi(i) = t$. So the evidence here are tree structures, not the yields of tree structures.

Exercise 1. Prove the theorem.

0.1.3 Context-Free Grammars

Definition 3. A *context-free grammar* is a tuple $\langle T, N, S, \mathcal{R}\rangle$ where

- $\mathcal{T}$ is a nonempty finite alphabet of symbols. These symbols are also called the *terminal* symbols, and we usually write them with lowercase letters like $a, b, c, \ldots$
- $\mathcal{N}$ is a nonempty finite set of *non-terminal* symbols, which are distinct from elements of $\mathcal{T}$. These symbols are also called *category* symbols, and we usually write them with uppercase letters like $A, B, C, \ldots$
- S is the *start* category, which is an element of $\mathcal{N}$.
- A finite set of *production rules* $\mathcal{R}$. A production rule has the form

  $$A \rightarrow \beta$$

  where β belongs to $(\mathcal{T} \cup \mathcal{N})^*$ and $A \in \mathcal{N}$. So β are strings of non-terminal and terminal symbols and A is a non-terminal.

Example 2. Consider the following grammar G_1:

- $\mathcal{T} = \{\textbf{\textit{john}}, \textbf{\textit{laughed}}, \textbf{\textit{and}}\}$;

- $\mathcal{N} = \{\text{S, VP1, VP2}\}$; and
- $$\mathcal{R} = \left\{ \begin{array}{l} \text{S} \rightarrow \textbf{\textit{john}}\ \text{VP1} \\ \text{VP1} \rightarrow \textbf{\textit{laughed}} \\ \text{VP1} \rightarrow \textbf{\textit{laughed}}\ \text{VP2} \\ \text{VP2} \rightarrow \textbf{\textit{and laughed}}\ \text{VP2} \\ \text{VP2} \rightarrow \textbf{\textit{laughed}} \end{array} \right\}$$

Example 3. Consider the following grammar G_2:

- $T = \{\text{a, b}\}$;
- $V = \{\text{S}\}$; and
- The production rules are
$$\mathcal{R} = \left\{ \begin{array}{l} \text{S} \rightarrow \text{aSb} \\ \text{S} \rightarrow ab \end{array} \right\}$$

The language of a context-free grammar (CFG) is defined recursively below.

Definition 4. The (partial) *derivations* of a CFG $G = \langle \mathcal{T}, \mathcal{N}, S, \mathcal{R} \rangle$ is written $D(G)$ and is defined recursively as follows.

1. *The base case:* S belongs to $D(G)$.
2. *The recursive case:* For all $A \rightarrow \beta \in \mathcal{R}$ and for all $\gamma_1, \gamma_2 \in (\mathcal{T} \cup \mathcal{N})^*$, if $\gamma_1 A \gamma_2 \in D(G)$ then $\gamma_1 \beta \gamma_2 \in D(G)$.
3. Nothing else is in $D(G)$.

Then the language of the grammar $L(G)$ is defined as

$$L(G) = \big\{ w \in \mathcal{T}^* \mid w \in D(G) \big\}.$$

Exercise 2. How does G_1 generate ***John laughed and laughed and laughed***?

Exercise 3. What language does G_2 generate?

Theorem 2. *The languages generated by context-free grammars are exactly the yields of the Strictly 2-Local treesets.*

Exercise 4. Explain how the 2-local trees in a tree relate to the production rules of a context-free grammar.

References

Rogers, James. 2003. wMSO theories as grammar formalisms. *Theoretical Computer Science* 293:291–320.