

Convolution kernels for natural language (Collins and Duffy, 2001)

LING 572

Advanced Statistical Methods for NLP

February 28, 2019

Highlights

- Introduce a tree kernel
- Show how it is used for reranking

Reranking

Reranking

- Training data:
 $\{(x_i, y_i)\}$ and for each x_i , a set of candidates $\{y_{ij}\}$.
and one of y_{ij} is the same as y_i .
- Goal: create a module that reranks candidates
- The reranker is used as a post-processor.
- In this paper, build a reranker for parsing
 x_i is a sentence, y_{ij} is a parse tree.
Notation: $\{(s_i, t_i)\}$, $C(s_i) = \{x_{ij}\}$

Formulating the problem

$$\{(s_i, t_i)\}, C(s_i) = \{x_{ij}\}$$

$h(x_{ij})$ is the feature vector of candidate x_{ij} .

Let x_{i1} be the correct parse for s_i .

Training: calculate \vec{w}

Decoding: $x^* = \operatorname{argmax}_{x \in C(s)} \vec{w} \cdot h(x)$

Reranking: Training

Minimize $\|\vec{w}\|^2$ subject to the constraints

$$\vec{w} \cdot h(x_{i1}) \geq \vec{w} \cdot h(x_{ij}), \forall i, \forall j \geq 2$$



$$\vec{w} \cdot (h(x_{i1}) - h(x_{ij})) \geq 1, \forall i, \forall j \geq 2$$



Recall that in SVM $\vec{w} = \sum_i \alpha_i y_i \vec{x}_i$

$$\vec{w} = \sum_{(i,j)} \alpha_{ij} (h(x_{i1}) - h(x_{ij}))$$

$$f(x) = \vec{w} \cdot x = \sum_{ij} \alpha_{ij} (h(x_{i1}) \cdot h(x) - h(x_{ij}) \cdot h(x))$$



With the kernel trick

$$f(x) = \sum_{ij} \alpha_{ij} (K(x_{i1}, x) - K(x_{ij}, x))$$

Perceptron training

$$f(x) = \vec{w} \cdot x = \sum_{ij} \alpha_{ij} (h(x_{i1}) \cdot h(x) - h(x_{ij}) \cdot h(x))$$

$$\alpha_{i,j} = 0;$$

for each sentence i

for each $j > 1$

if $f(x_{i1}) < f(x_{ij})$ then $\alpha_{ij}++$;

Tree kernel

$$f(x) = \sum_{ij} \alpha_{ij} (K(x_{i1}, x) - K(x_{ij}, x))$$

$$K: X \times X \rightarrow R$$

Each member of X is a parse tree.

What is a good tree kernel?

A tree kernel

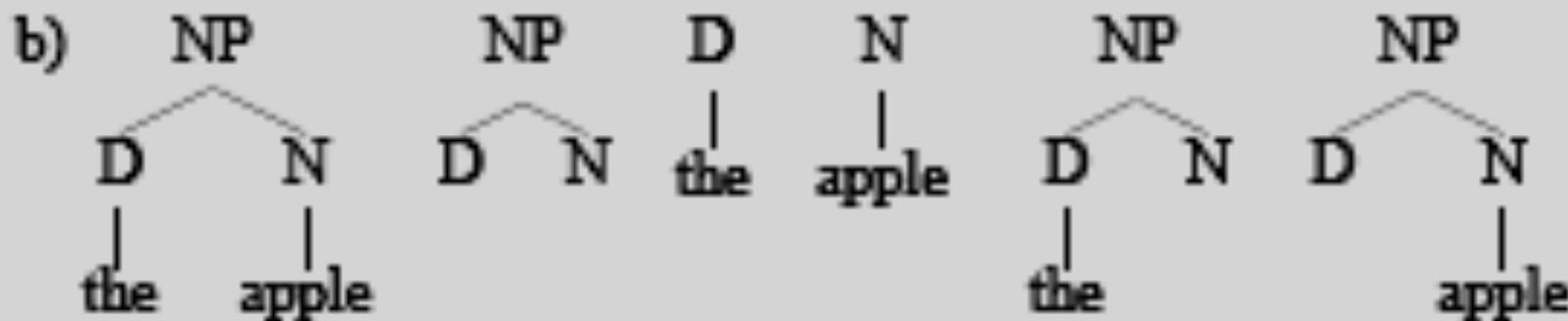
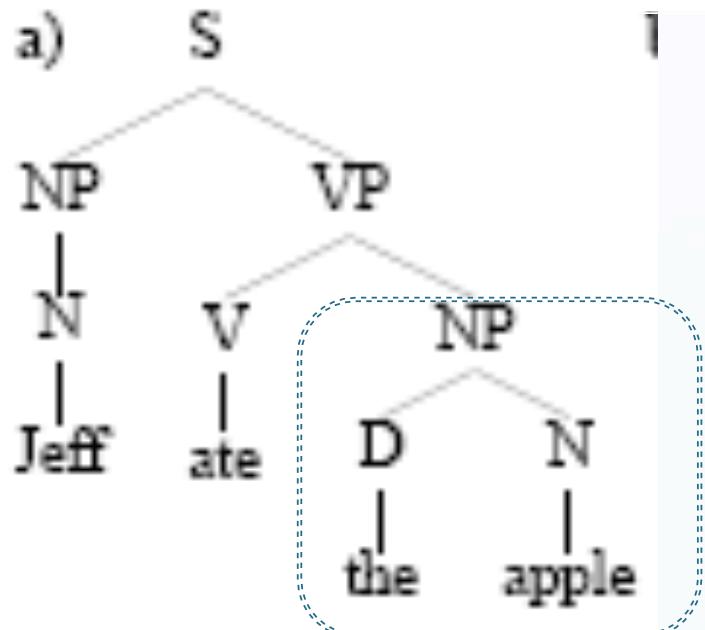
Intuition

- Given two trees T_1 and T_2 , the more subtrees T_1 and T_2 share, the more similar they are.
- Method:
 - For each tree, enumerate all the subtrees
 - Count how many are in common
- Do it in an efficient way

Definition of subtree

- A subtree is a subgraph which has **more than** one node, with the restriction that entire (not partial) rule productions must be included.
- “A subtree rooted at node n” means “a subtree whose root is n”.

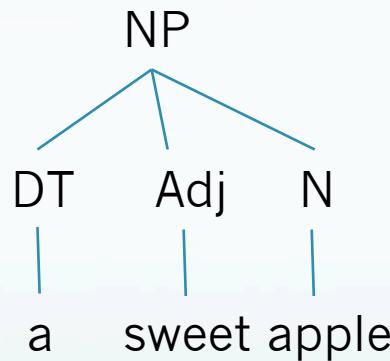
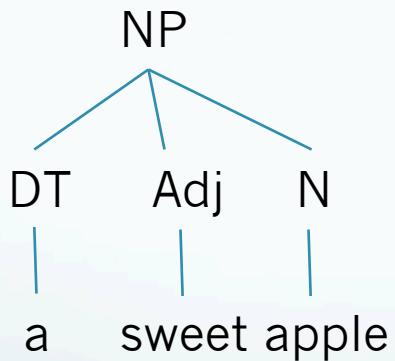
An example



$C(n_1, n_2)$

$C(n_1, n_2)$ counts the number of common subtrees rooted at n_1 and n_2 .

$$C(n_1, n_2) = ??$$



Calculating C(n1, n2)

If the productions at n1 and n2 are different

then $C(n1, n2) = 0$

else if n1 and n2 are pre-terminals

then $C(n1, n2) = 1$

else $C(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (1 + C(ch(n1, j), ch(n2, j)))$

Representing a tree as a feature vector

Let ST be the set of sub-trees in **any** tree

$$ST = \{s_1, s_2, \dots, s_n, \dots\}$$

Let $h_i(T)$ be the num of occurrences of s_i in T

$$h(T) = (h_1(T), h_2(T), \dots, h_n(T), \dots)$$

$$\begin{aligned} I_i(n) &= 1 && \text{if } s_i \text{ is a subtree rooted at } n. \\ &= 0 && \text{otherwise} \end{aligned}$$

$$h_i(T_1) = \sum_{n_1 \in N_1} I_i(n_1), \text{ where } N_1 \text{ is set of nodes in } T_1$$

$$h_i(T_2) = \sum_{n_2 \in N_2} I_i(n_2)$$

A tree kernel

$$\begin{aligned} h(T_1) \cdot h(T_2) &= \sum_i h_i(T_1)h_i(T_2) \\ &= \sum_i \left(\sum_{n_1 \in N_1} I_i(n_1) \right) * \left(\sum_{n_2 \in N_2} I_i(n_2) \right) \\ &= \sum_i \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} I_i(n_1)I_i(n_2) \\ &= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \sum_i I_i(n_1)I_i(n_2) \\ &= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} C(n_1, n_2) \end{aligned}$$

$K(T_1, T_2) = h(T_1) \cdot h(T_2)$ can be calculated in $O(|N_1||N_2|)$

Properties of this kernel

- The value of $K(T_1, T_2)$ depends greatly on the size of the trees T_1 and T_2 .

$$K'(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1)K(T_2, T_2)}}$$

- $K(T, T)$ would be huge. The output would be dominated by the most similar tree.
=> The model would behave like a nearest neighbor rule

Downweighting the contribution of large subtrees when calculating $C(n_1, n_2)$

If the productions at n_1 and n_2 are different

then $C(n_1, n_2) = 0$

else if n_1 and n_2 are pre-terminals

then $C(n_1, n_2) = \lambda$

else $C(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + C(ch(n_1, j), ch(n_2, j)))$

Experimental results

Experiment setting

- Data:
 - Training data: 800 sentences,
 - Dev set: 200 sentences
 - Test set: 336 sentences
 - For each sentence, 100 candidate parse trees
- Learner: voted perceptron
- Evaluation measure: 10 runs and report the average parse score
- Baseline (with PCFG): 74% (labeled f-score)

Results

With different max subtree size

Depth	1	2	3	4	5	6
Score	73 ± 1	79 ± 1	80 ± 1	79 ± 1	79 ± 1	78 ± 0.01
Improvement	-1 ± 4	20 ± 6	23 ± 3	21 ± 4	19 ± 4	18 ± 3

Summary

- Show how to use a SVM or a perceptron learner for the reranking task.
- Define a tree kernel that can be calculated in polynomial time.
 - Note: the number of features is infinite.
- The reranker improves parse score from 74% to 80%.