6.891: Lecture 24 (December 8th, 2003)

Kernel Methods

Overview

- Recap: global linear models
- New representations from old representations
- A computational trick
- Kernels for NLP structures
- Conclusions: 10 Ideas from the Course

Three Components of Global Linear Models

- $\mathbf{\Phi}$ is a function that maps a structure (x, y) to a **feature vector** $\mathbf{\Phi}(x, y) \in \mathbb{R}^d$

- $\mathbf{GEN}$ is a function that maps an input x to a set of **candidates** $\mathbf{GEN}(x)$

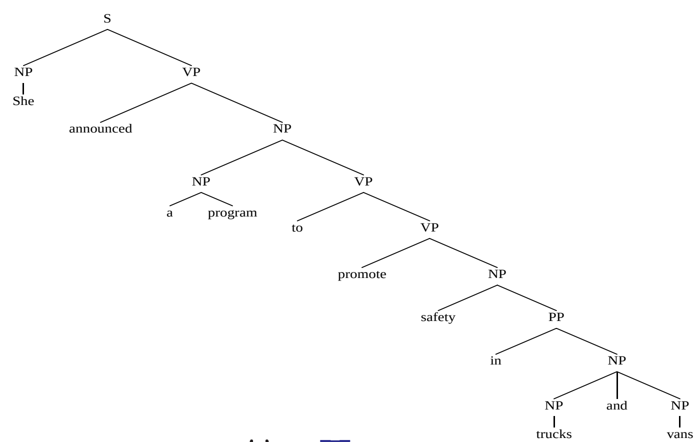
- $\mathbf{W}$ is a parameter vector (also a member of $\mathbb{R}^d$)

- Training data is used to set the value of $\mathbf{W}$

Component 1: Φ

- Φ maps a candidate to a **feature vector** $\in \mathbb{R}^d$

- Φ defines the **representation** of a candidate



$$\Downarrow \Phi$$

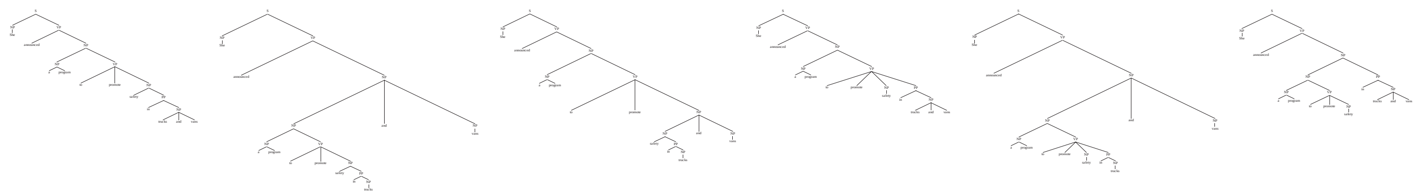
$$\langle 1, 0, 2, 0, 0, 15, 5 \rangle$$

Component 2: GEN

- **GEN** enumerates a set of **candidates** for a sentence

She announced a program to promote safety in trucks and vans

$\Downarrow$ **GEN**



Component 2: $\mathbf{GEN}$

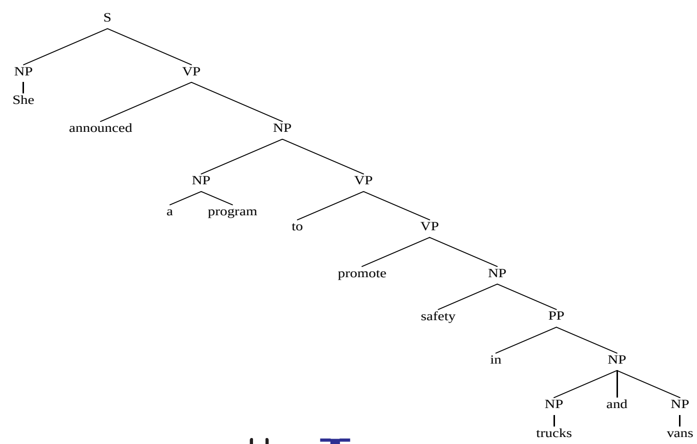
- $\mathbf{GEN}$ enumerates a set of **candidates** for an input x

- Some examples of how $\mathbf{GEN}(x)$ can be defined:
  - Parsing: $\mathbf{GEN}(x)$ is the set of parses for x under a grammar
  - Any task: $\mathbf{GEN}(x)$ is the top N most probable parses under a history-based model
  - Tagging: $\mathbf{GEN}(x)$ is the set of all possible tag sequences with the same length as x
  - Translation: $\mathbf{GEN}(x)$ is the set of all possible English translations for the French sentence x

Component 3: W

- **W** is a **parameter vector** $\in \mathbb{R}^d$
- $\mathbf{\Phi}$ and **W** together map a candidate to a real-valued score

---



$$\Downarrow \mathbf{\Phi}$$

$$\langle 1, 0, 2, 0, 0, 15, 5 \rangle$$

$$\Downarrow \mathbf{\Phi} \cdot \mathbf{W}$$

$$\langle 1, 0, 2, 0, 0, 15, 5 \rangle \cdot \langle 1.9, -0.3, 0.2, 1.3, 0, 1.0, -2.3 \rangle = 5.8$$

Putting it all Together

- $\mathcal{X}$ is set of sentences, $\mathcal{Y}$ is set of possible outputs (e.g. trees)

- Need to learn a function $F : \mathcal{X} \rightarrow \mathcal{Y}$

- $\mathbf{GEN}$, $\mathbf{\Phi}$, $\mathbf{W}$ define

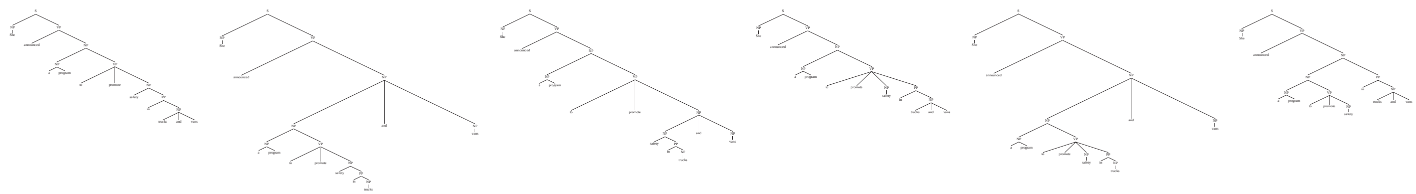
$$F(x) = \arg\max_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W}$$

Choose the highest scoring candidate as the most plausible structure

- Given examples (x_i, y_i), how to set $\mathbf{W}$?

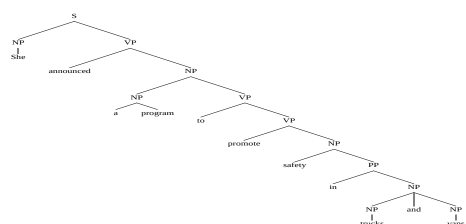
She announced a program to promote safety in trucks and vans

$\Downarrow$ **GEN**



$\Downarrow \mathbf{\Phi}$	$\Downarrow \mathbf{\Phi}$	$\Downarrow \mathbf{\Phi}$	$\Downarrow \mathbf{\Phi}$	$\Downarrow \mathbf{\Phi}$	$\Downarrow \mathbf{\Phi}$
$\langle 1, 1, 3, 5 \rangle$	$\langle 2, 0, 0, 5 \rangle$	$\langle 1, 0, 1, 5 \rangle$	$\langle 0, 0, 3, 0 \rangle$	$\langle 0, 1, 0, 5 \rangle$	$\langle 0, 0, 1, 5 \rangle$
$\Downarrow \mathbf{\Phi} \cdot \mathbf{W}$	$\Downarrow \mathbf{\Phi} \cdot \mathbf{W}$	$\Downarrow \mathbf{\Phi} \cdot \mathbf{W}$	$\Downarrow \mathbf{\Phi} \cdot \mathbf{W}$	$\Downarrow \mathbf{\Phi} \cdot \mathbf{W}$	$\Downarrow \mathbf{\Phi} \cdot \mathbf{W}$
13.6	12.2	12.1	3.3	9.4	11.1

$\Downarrow$ arg max



A Variant of the Perceptron Algorithm

Inputs: Training set (x_i, y_i) for $i = 1 \ldots n$

Initialization: $\mathbf{W} = 0$

Define: $F(x) = \operatorname{argmax}_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W}$

Algorithm: For $t = 1 \ldots T$, $i = 1 \ldots n$

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $\quad \mathbf{W} = \mathbf{W} + \mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, z_i)$

Output: Parameters $\mathbf{W}$

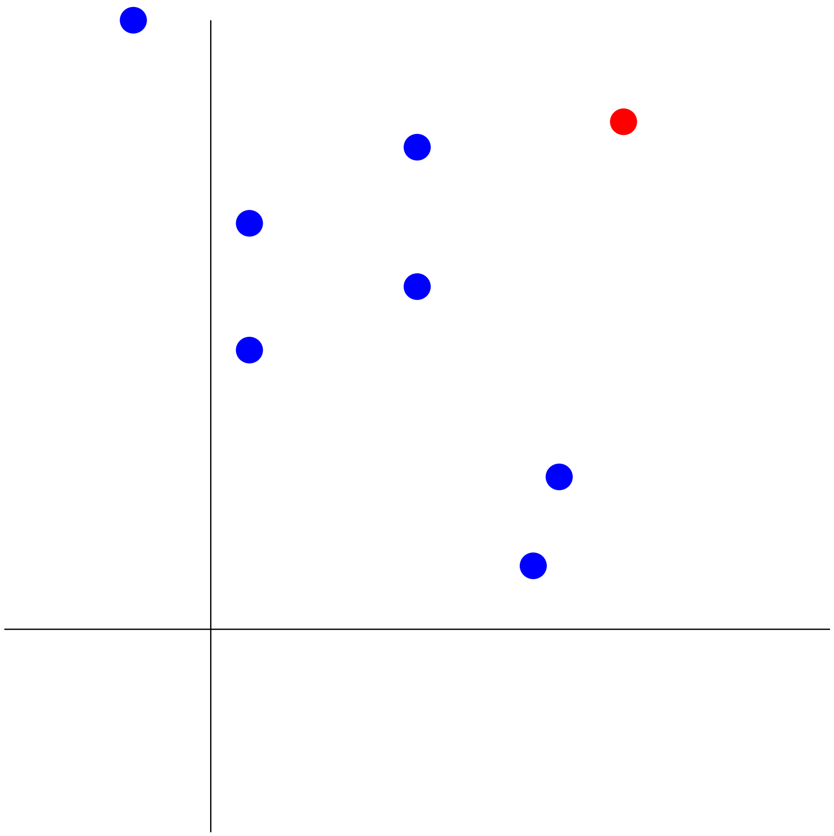
Theory Underlying the Algorithm

- **Definition:** $\overline{\mathbf{GEN}}(x_i) = \mathbf{GEN}(x_i) - \{y_i\}$

- **Definition:** The training set is **separable with margin** δ, if there is a vector $\mathbf{U} \in \mathbb{R}^d$ with $||\mathbf{U}|| = 1$ such that

$$\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i) \quad \mathbf{U} \cdot \mathbf{\Phi}(x_i, y_i) - \mathbf{U} \cdot \mathbf{\Phi}(x_i, z) \geq \delta$$

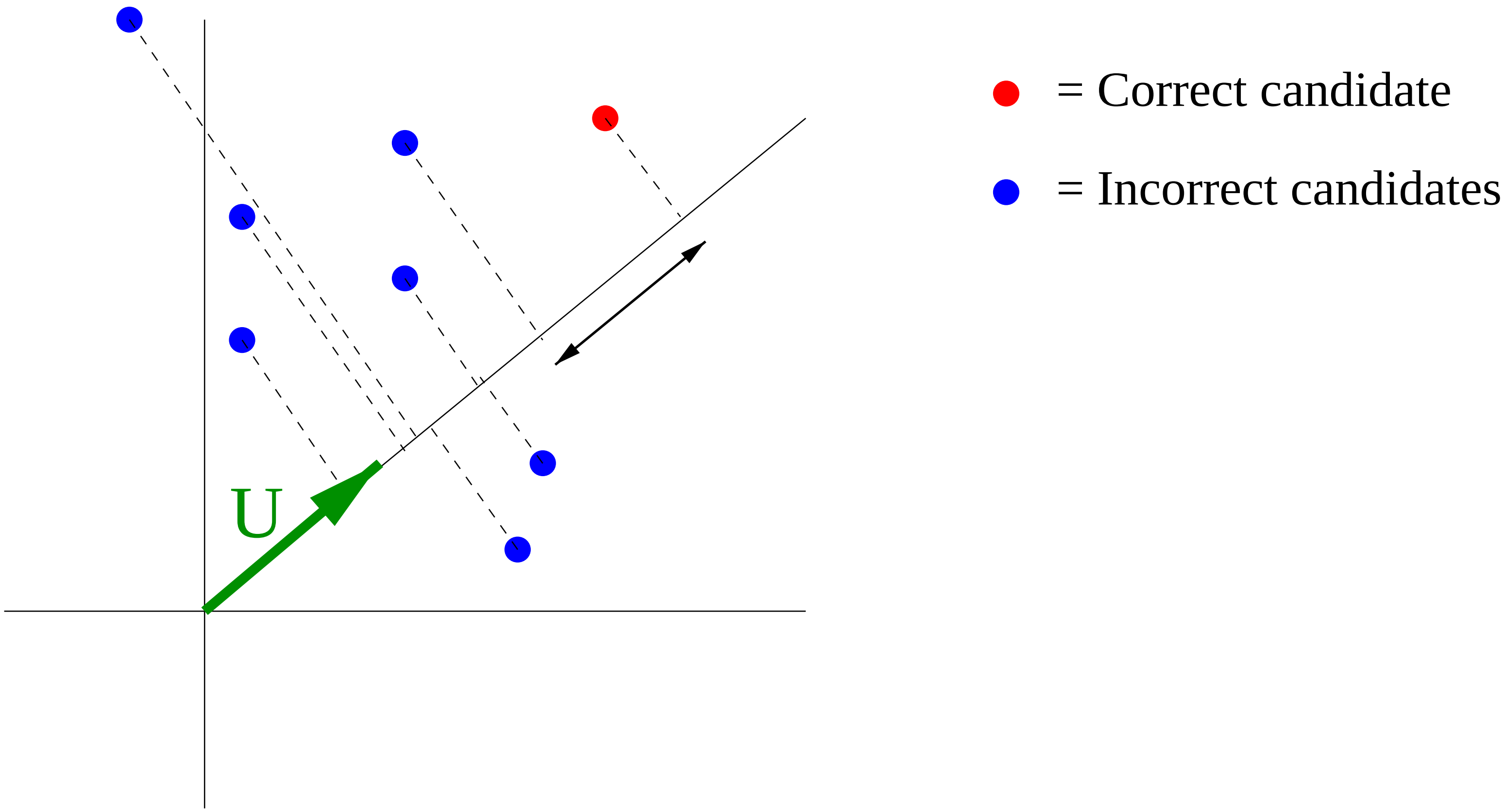
GEOMETRIC INTUITION BEHIND SEPARATION



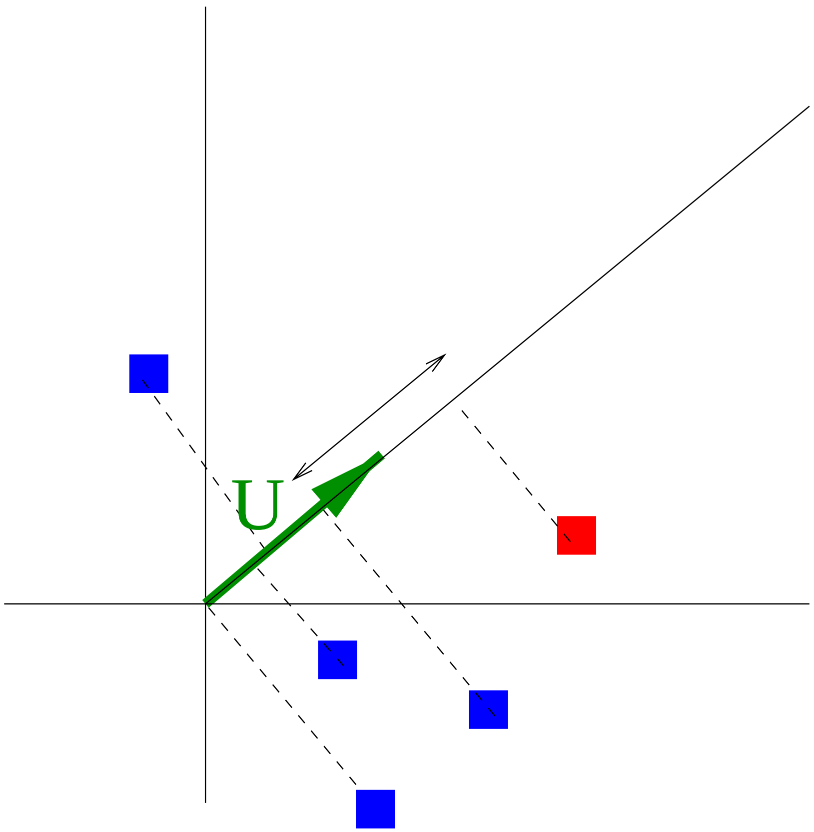
= Correct candidate

= Incorrect candidates

GEOMETRIC INTUITION BEHIND SEPARATION



ALL EXAMPLES ARE SEPARATED



= Correct candidate (2)

= Incorrect candidates (2)

THEORY UNDERLYING THE ALGORITHM

Theorem: For any training sequence (x_i, y_i) which is separable with margin δ, then for the perceptron algorithm

$$\text{Number of mistakes} \leq \frac{R^2}{\delta^2}$$

where R is a constant such that $\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i)$
$||\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, z)|| \leq R$

Proof: Direct modification of the proof for the classification case.

Proof:

Let $\mathbf{W}^k$ be the weights before the k'th mistake. $\mathbf{W}^1 = 0$
If the k'th mistake is made at i'th example,
and $z_i = \mathrm{argmax}_{y \in \mathbf{GEN}(x_i)} \mathbf{\Phi}(y) \cdot \mathbf{W}^k$, then

$$
\begin{aligned}
\mathbf{W}^{k+1} &= \mathbf{W}^k + \mathbf{\Phi}(y_i) - \mathbf{\Phi}(z_i) \\
\Rightarrow \mathbf{U} \cdot \mathbf{W}^{k+1} &= \mathbf{U} \cdot \mathbf{W}^k + \mathbf{U} \cdot \mathbf{\Phi}(y_i) - \mathbf{U} \cdot \mathbf{\Phi}(z_i) \\
&\geq \mathbf{U} \cdot \mathbf{W}^k + \delta \\
&\geq k\delta \\
\Rightarrow ||\mathbf{W}^{k+1}|| &\geq k\delta
\end{aligned}
$$

Also,

$$
\begin{aligned}
||\mathbf{W}^{k+1}||^2 &= ||\mathbf{W}^k||^2 + ||\mathbf{\Phi}(y_i) - \mathbf{\Phi}(z_i)||^2 + 2\mathbf{W}^k \cdot (\mathbf{\Phi}(y_i) - \mathbf{\Phi}(z_i)) \\
&\leq ||\mathbf{W}^k||^2 + R^2 \\
\Rightarrow ||\mathbf{W}^{k+1}||^2 &\leq kR^2 \\
\Rightarrow k^2\delta^2 &\leq ||\mathbf{W}^{k+1}||^2 \leq kR^2 \\
\Rightarrow k &\leq R^2/\delta^2
\end{aligned}
$$

Overview

- Recap: global linear models
- New representations from old representations
- A computational trick
- Kernels for NLP structures

New Representations from Old Representations

- Say we have an existing representation $\mathbf{\Phi}(x, y)$
- Our global linear model will learn parameters $\mathbf{W}$ such that

$$F(x) = \mathrm{argmax}_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W}$$

- This is a **linear** model:
  but perhaps the linearity assumption is bad?

New Representations from Old Representations

- Say we have an existing representation of size $d = 2$

$$\mathbf{\Phi}(x, y) = \{\mathbf{\Phi}_1(x, y), \mathbf{\Phi}_2(x, y)\}$$

- We define a new representation $\mathbf{\Phi}'(x, y)$ of dimension $d' = O(d^2)$, that contains every quadratic term in $\mathbf{\Phi}(x, y)$

$$\mathbf{\Phi}'(x, y) \quad = \quad \{\mathbf{\Phi}_1(x, y), \mathbf{\Phi}_2(x, y), \mathbf{\Phi}_1(x, y)^2, \mathbf{\Phi}_2(x, y)^2, \mathbf{\Phi}_1(x, y)\mathbf{\Phi}_2(x, y)\}$$

- A global linear model under representation $\mathbf{\Phi}'$ is **linear** in the new space $\mathbf{\Phi}'$, but **non-linear** in the old space $\mathbf{\Phi}$:

$$\mathbf{\Phi}'(x, y) \cdot \mathbf{W}' = \mathbf{W}'_1 \mathbf{\Phi}_1(x, y) + \mathbf{W}'_2 \mathbf{\Phi}_2(x, y) + \mathbf{W}'_3 \mathbf{\Phi}_1(x, y)^2 + \mathbf{W}'_4 \mathbf{\Phi}_2(x, y)^2 + \mathbf{W}'_5 \mathbf{\Phi}_1(x, y)\mathbf{\Phi}_2(x, y)$$

Basic idea: explicitly form new feature vectors $\mathbf{\Phi}'$ from $\mathbf{\Phi}$, and run the perceptron in the new space

More Generally

- Say we have an existing representation (writing $\mathbf{\Phi}_i$ instead of $\mathbf{\Phi}_i(x,y)$ for brevity):

$$\mathbf{\Phi}(x,y) = \{\mathbf{\Phi}_1, \mathbf{\Phi}_2, \ldots, \mathbf{\Phi}_d\}$$

- We define a new representation $\mathbf{\Phi}'(x,y)$ of dimension $d' = O(d^2)$, that contains every quadratic term in $\mathbf{\Phi}(x,y)$

$$\begin{aligned}
\mathbf{\Phi}'(x,y) &= \{\mathbf{\Phi}'_1, \mathbf{\Phi}'_2, \ldots, \mathbf{\Phi}'_{d'}\} \\
&= \{\mathbf{\Phi}_1, \mathbf{\Phi}_2, \ldots, \mathbf{\Phi}_d \\
&\quad \mathbf{\Phi}_1^2, \mathbf{\Phi}_2^2, \ldots, \mathbf{\Phi}_d^2, \\
&\quad \mathbf{\Phi}_1\mathbf{\Phi}_2, \mathbf{\Phi}_1\mathbf{\Phi}_3, \ldots, \mathbf{\Phi}_1\mathbf{\Phi}_d, \\
&\quad \mathbf{\Phi}_2\mathbf{\Phi}_1, \mathbf{\Phi}_2\mathbf{\Phi}_3, \ldots, \mathbf{\Phi}_2\mathbf{\Phi}_d, \\
&\quad \ldots \\
&\quad \mathbf{\Phi}_d\mathbf{\Phi}_1, \mathbf{\Phi}_d\mathbf{\Phi}_2, \ldots, \mathbf{\Phi}_d\mathbf{\Phi}_{d-1}, \}
\end{aligned}$$

Problem: size of $\mathbf{\Phi}'$ quickly gets very large
$\Rightarrow$ computational efficiency becomes a real problem

Overview

- Recap: global linear models
- New representations from old representations
- A computational trick
- Kernels for NLP structures

A Computational Trick (Part 1)

- Now, take feature vectors for a first example (x, y), and for a second example (v, w):

$$\mathbf{\Phi}(x, y) = \{\mathbf{\Phi}_1, \mathbf{\Phi}_2\} \qquad \mathbf{\Phi}(v, w) = \{\rho_1, \rho_2\}$$

- Consider a function K:

$$K((x, y), (v, w)) = (1 + \mathbf{\Phi}(x, y) \cdot \mathbf{\Phi}(v, w))^2 = (1 + \mathbf{\Phi}_1 \rho_1 + \mathbf{\Phi}_2 \rho_2)^2$$

- For example, if

$$\mathbf{\Phi}(x, y) = \{1, 3\} \qquad \mathbf{\Phi}(v, w) = \{2, 4\}$$

then

$$K((x, y), (v, w)) = (1 + 1 \times 2 + 3 \times 4)^2 = 225$$

A Computational Trick (Part 1)

- Consider a function K:

$$K((x,y),(v,w)) = (1 + \mathbf{\Phi}(x,y) \cdot \mathbf{\Phi}(v,w))^2 = (1 + \mathbf{\Phi}_1 \rho_1 + \mathbf{\Phi}_2 \rho_2)^2$$

- **Key point:** It can be shown that

$$K((x,y),(v,w)) = \mathbf{\Phi}'(x,y) \cdot \mathbf{\Phi}'(v,w)$$

where

$$\begin{array}{rcl} \mathbf{\Phi}'(x,y) & = & \{1, \sqrt{2}\mathbf{\Phi}_1, \sqrt{2}\mathbf{\Phi}_2, \mathbf{\Phi}_1^2, \mathbf{\Phi}_2^2, \sqrt{2}\mathbf{\Phi}_1\mathbf{\Phi}_2\} \\ \mathbf{\Phi}'(v,w) & = & \{1, \sqrt{2}\rho_1, \sqrt{2}\rho_2, \rho_1^2, \rho_2^2, \sqrt{2}\rho_1\rho_2\} \end{array}$$

So: K is an inner product in a new space that contains all quadratic terms in the original space $\mathbf{\Phi}$

Proof:

$$
\begin{aligned}
& K((x,y),(v,w)) \\
= \quad & (1 + \mathbf{\Phi}(x,y) \cdot \mathbf{\Phi}(v,w))^2 \\
= \quad & (1 + \mathbf{\Phi}_1 \rho_1 + \mathbf{\Phi}_2 \rho_2)^2 \\
= \quad & 1 + \mathbf{\Phi}_1^2 \rho_1^2 + \mathbf{\Phi}_2^2 \rho_2^2 + 2\mathbf{\Phi}_1 \rho_1 \mathbf{\Phi}_2 \rho_2 + 2\mathbf{\Phi}_1 \rho_1 + 2\mathbf{\Phi}_2 \rho_2 \\
= \quad & \{1, \sqrt{2}\mathbf{\Phi}_1, \sqrt{2}\mathbf{\Phi}_2, \mathbf{\Phi}_1^2, \mathbf{\Phi}_2^2, \sqrt{2}\mathbf{\Phi}_1 \mathbf{\Phi}_2\} \cdot \{1, \sqrt{2}\rho_1, \sqrt{2}\rho_2, \rho_1^2, \rho_2^2, \sqrt{2}\rho_1 \rho_2\}
\end{aligned}
$$

More Generally

- Say we have an existing representation

$$\mathbf{\Phi}(x,y) = \{\mathbf{\Phi}_1, \mathbf{\Phi}_2, \ldots, \mathbf{\Phi}_d\}$$

  and we take

$$K((x,y),(v,w)) = (1 + \mathbf{\Phi}(x,y) \cdot \mathbf{\Phi}(v,w))^2$$

- Then it can be shown that $K((x,y),(v,w)) = \mathbf{\Phi}'(x,y) \cdot \mathbf{\Phi}'(v,w)$ where

$$\begin{aligned}
\mathbf{\Phi}'(x,y) &= \{\mathbf{\Phi}'_1, \mathbf{\Phi}'_2, \ldots, \mathbf{\Phi}'_{d'}\} \\
&= \{1, \sqrt{2}\mathbf{\Phi}_1, \sqrt{2}\mathbf{\Phi}_2, \ldots, \sqrt{2}\mathbf{\Phi}_d, \\
&\quad \mathbf{\Phi}_1^2, \mathbf{\Phi}_2^2, \ldots, \mathbf{\Phi}_d^2, \\
&\quad \sqrt{2}\mathbf{\Phi}_1\mathbf{\Phi}_2, \sqrt{2}\mathbf{\Phi}_1\mathbf{\Phi}_3, \ldots, \sqrt{2}\mathbf{\Phi}_1\mathbf{\Phi}_d, \\
&\quad \sqrt{2}\mathbf{\Phi}_2\mathbf{\Phi}_1, \sqrt{2}\mathbf{\Phi}_2\mathbf{\Phi}_3, \ldots, \sqrt{2}\mathbf{\Phi}_2\mathbf{\Phi}_d, \\
&\quad \ldots \\
&\quad \sqrt{2}\mathbf{\Phi}_d\mathbf{\Phi}_1, \sqrt{2}\mathbf{\Phi}_d\mathbf{\Phi}_2, \ldots, \sqrt{2}\mathbf{\Phi}_d\mathbf{\Phi}_{d-1}, \}
\end{aligned}$$

A Variant of the Perceptron Algorithm

Inputs: Training set (x_i, y_i) for $i = 1 \ldots n$

Initialization: $\mathbf{W} = 0$

Define: $F(x) = \operatorname{argmax}_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W}$

Algorithm: For $t = 1 \ldots T$, $i = 1 \ldots n$

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $\quad \mathbf{W} = \mathbf{W} + \mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, z_i)$

Output: Parameters $\mathbf{W}$

A Computational Trick (Part 2)

- In standard perceptron, we store a parameter vector $\mathbf{W}$, and

$$F(x) = \arg\max_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W}$$

- In "dual form" perceptron, we store weights

$$\alpha_{i,y} \quad \text{for all } i\text{, and for all } y \in \mathbf{GEN}(x_i)$$

  and assume the equivalence:

$$\mathbf{W} = \sum_{i, z \in \mathbf{GEN}(x_i)} \alpha_{i,z} \mathbf{\Phi}(x_i, z)$$

- We then have

$$\begin{aligned} F(x) &= \arg\max_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(x, y) \cdot \mathbf{W} \\ &= \arg\max_{y \in \mathbf{GEN}(x)} \left(\sum_{i, z \in \mathbf{GEN}(x_i)} \alpha_{i,z} \mathbf{\Phi}(x_i, z) \cdot \mathbf{\Phi}(x, y) \right) \end{aligned}$$

Dual Form of the Perceptron Algorithm

Inputs: Training set (x_i, y_i) for $i = 1 \ldots n$

Initialization: $\alpha_{i,y} = 0$ for all i, for all $y \in \mathbf{GEN}(x_i)$

Define:

$$F(x) = \mathrm{argmax}_{y \in \mathbf{GEN}(x)} \left(\sum_{i,z \in \mathbf{GEN}(x_i)} \alpha_{i,z} \mathbf{\Phi}(x_i, z) \cdot \mathbf{\Phi}(x, y) \right)$$

Algorithm: For $t = 1 \ldots T$, $i = 1 \ldots n$

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $\quad \alpha_{i,y_i} = \alpha_{i,y_i} + 1$

$\alpha_{i,z_i} = \alpha_{i,z_i} - 1$

Output: Parameters $\alpha_{i,y}$

Equivalence: $\mathbf{W} = \sum_{i,z \in \mathbf{GEN}(x_i)} \alpha_{i,z} \mathbf{\Phi}(x_i, z)$

Original Form

Initialization: $\mathbf{W} = 0$

Define: $F(x) = \text{argmax}_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}(y) \cdot \mathbf{W}$

Algorithm: For $t = 1 \dots T$, $i = 1 \dots n$

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $\quad \mathbf{W} = \mathbf{W} + \mathbf{\Phi}(y_i) - \mathbf{\Phi}(z_i)$

Dual Form

Initialization: $\alpha_{i,y} = 0$ for all i, for all $y \in \mathbf{GEN}(x_i)$

Define: $F(x) = \text{argmax}_{y \in \mathbf{GEN}(x)} \left(\sum_{i,z \in \mathbf{GEN}(x_i)} \alpha_{i,z} \mathbf{\Phi}(x_i, z) \cdot \mathbf{\Phi}(x, y) \right)$

Algorithm: For $t = 1 \dots T$, $i = 1 \dots n$

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $\quad \alpha_{i,y_i} = \alpha_{i,y_i} + 1, \qquad \alpha_{i,z_i} = \alpha_{i,z_i} - 1$

Dual (Kernel) Form of the Perceptron Algorithm

Inputs: Training set (x_i, y_i) for $i = 1 \ldots n$

Initialization: $\alpha_{i,y} = 0$ for all i, for all $y \in \mathbf{GEN}(x_i)$

Define:

$$F(x) = \arg\max_{y \in \mathbf{GEN}(x)} \left(\sum_{i, z \in \mathbf{GEN}(x_i)} \alpha_{i,z} K((x_i, z), (x, y)) \right)$$

Algorithm: For $t = 1 \ldots T$, $i = 1 \ldots n$

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $\quad \alpha_{i,y_i} = \alpha_{i,y_i} + 1$

$\quad \alpha_{i,z_i} = \alpha_{i,z_i} - 1$

Output: Parameters $\alpha_{i,y}$

Dual (Kernel) Form of the Perceptron Algorithm

- For example, if we choose

$$K((x, y), (v, w)) = (1 + \mathbf{\Phi}(x, y) \cdot \mathbf{\Phi}(v, w))^2$$

then the kernel form learns a global linear model

$$F(x) = \arg \max_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}'(x, y) \cdot \mathbf{W}$$

where $\mathbf{\Phi}'$ is a representation that contains all quadratic terms of the original representation $\mathbf{\Phi}$

- The algorithm returns coefficients $\alpha_{i,y}$ which *implicitly* define

$$\mathbf{W} = \sum_{i,z \in \mathbf{GEN}(x_i)} \alpha_{i,z} \mathbf{\Phi}'(x_i, z)$$

and

$$\begin{aligned} F(x) &= \arg \max_{y \in \mathbf{GEN}(x)} \mathbf{\Phi}'(x, y) \cdot \mathbf{W} \\ &= \arg \max_{y \in \mathbf{GEN}(x)} \left(\sum_{i,z \in \mathbf{GEN}(x_i)} \alpha_{i,z} K((x_i, z), (x, y)) \right) \end{aligned}$$

We never have to manipulate parameter vectors $\mathbf{W}$ or representations $\mathbf{\Phi}'(x, y)$ directly: everything in training and testing is computed indirectly, through inner products or kernels

- Computational efficiency:
  - Say I is the time taken to calculate $K((x,y),(v,w))$
  - Say $N = \sum_i |\mathbf{GEN}(x_i)|$ is size of the training set
  - In taking T passes over the training set, at most $2Tn$ values of $\alpha_{i,y}$ can take values other than 0, $\rightarrow$

$$\sum_{i,z \in \mathbf{GEN}(x_i)} \alpha_{i,z} K((x_i,z),(x,y))$$

    takes $O(nTI)$ time
  - And T passes over the training set takes $O(nT^2IN)$ time

Kernels

- A kernel K is a function of two objects,

$$K((x_1, y_1), (x_2, y_2))$$

for example, two sentence/tree pairs (x_1, y_1) and (x_2, y_2)

- **Intuition:** $K((x_1, y_1), (x_2, y_2))$ is a measure of the similarity between (x_1, y_1) and (x_2, y_2)

- **Formally:** $K((x_1, y_1), (x_2, y_2))$ is a kernel if it can be shown that there is some feature vector mapping $\mathbf{\Phi}(x, y)$ such that

$$K((x_1, y_1), (x_2, y_2)) = \mathbf{\Phi}(x_1, y_1) \cdot \mathbf{\Phi}(x_2, y_2)$$

for all x_1, y_1, x_2, y_2

A (Trivial) Example of a Kernel

- Given an existing feature vector representation $\mathbf{\Phi}$, define

$$K((x_1, y_1), (x_2, y_2)) = \mathbf{\Phi}(x_1, y_1) \cdot \mathbf{\Phi}(x_2, y_2)$$

A More Interesting Kernel

- Given an existing feature vector representation $\mathbf{\Phi}$, define

$$K((x_1, y_1), (x_2, y_2)) = (1 + \mathbf{\Phi}(x_1, y_1) \cdot \mathbf{\Phi}(x_2, y_2))^2$$

This can be shown to be an inner product in a new space $\mathbf{\Phi}'$, where $\mathbf{\Phi}'$ contains all quadratic terms of $\mathbf{\Phi}$

- More generally,

$$K((x_1, y_1), (x_2, y_2)) = (1 + \mathbf{\Phi}(x_1, y_1) \cdot \mathbf{\Phi}(x_2, y_2))^p$$

can be shown to be an inner product in a new space $\mathbf{\Phi}'$, where $\mathbf{\Phi}'$ contains all polynomial terms of $\mathbf{\Phi}$ up to degree p

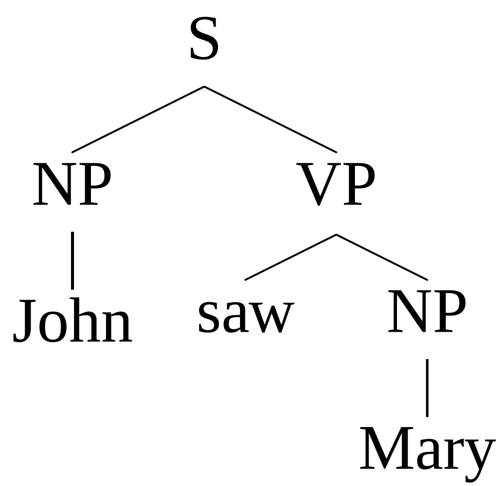
Question: can we come up with "specialized" kernels for NLP structures?

Overview

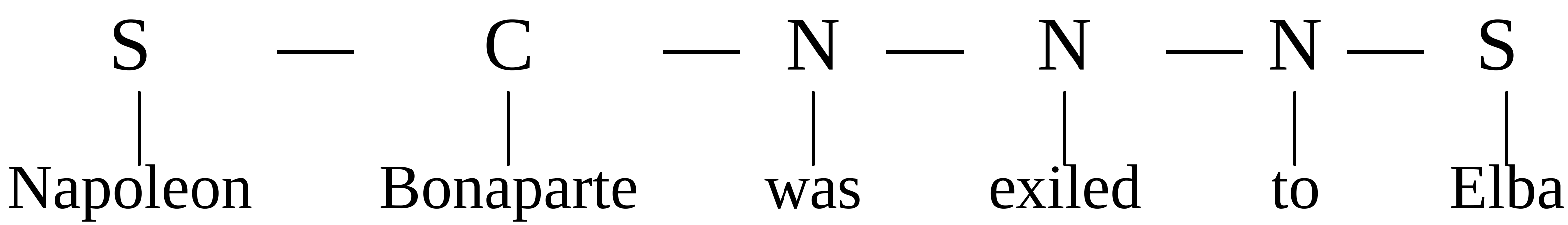
- Recap: global linear models
- New representations from old representations
- A computational trick
- Kernels for NLP structures

NLP Structures

- Trees



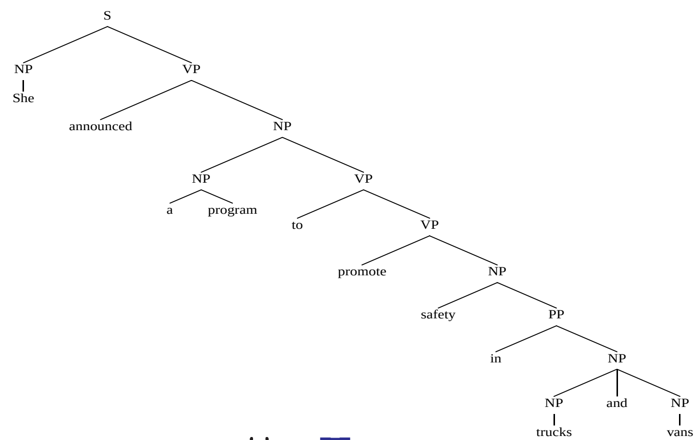
- Tagged sequences, e.g., named entity tagging



S = Start entity
C = Continue entity
N = Not an entity

Feature Vectors: $\mathbf{\Phi}$

- $\mathbf{\Phi}$ defines the **representation** of a structure
- $\mathbf{\Phi}$ maps a structure to a **feature vector** $\in \mathbb{R}^d$

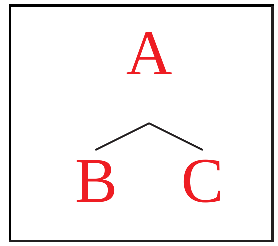


$$\Downarrow \mathbf{\Phi}$$

$$\langle 1, 0, 2, 0, 0, 15, 5 \rangle$$

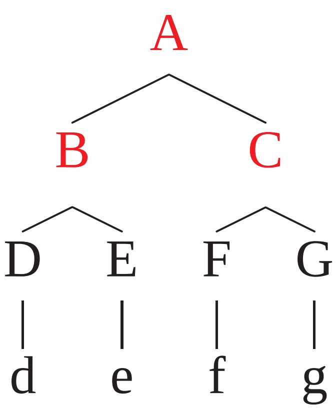
Features

- A "feature" is a function on a structure, e.g.,

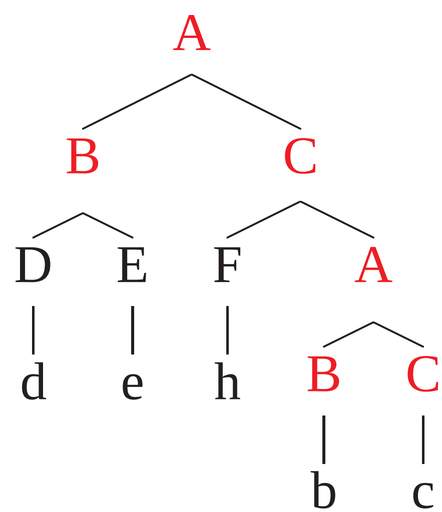
$h(x) =$ Number of times  is seen in x

---

T_1



T_2



$h(T_1) = 1$

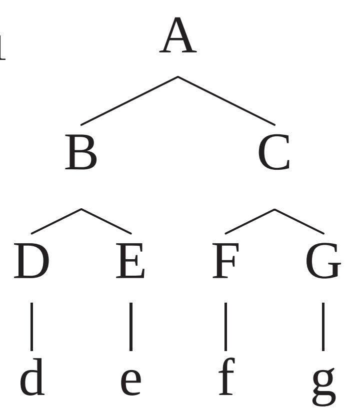
$h(T_2) = 2$

Feature Vectors

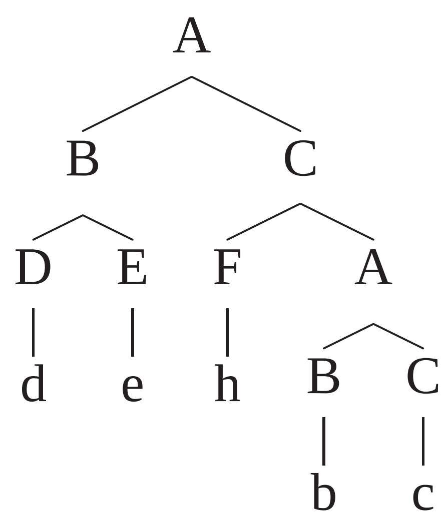
- A set of functions $h_1 \ldots h_d$ define a **feature vector**

$$\mathbf{\Phi}(x) = \langle h_1(x), h_2(x) \ldots h_d(x) \rangle$$

T_1



T_2

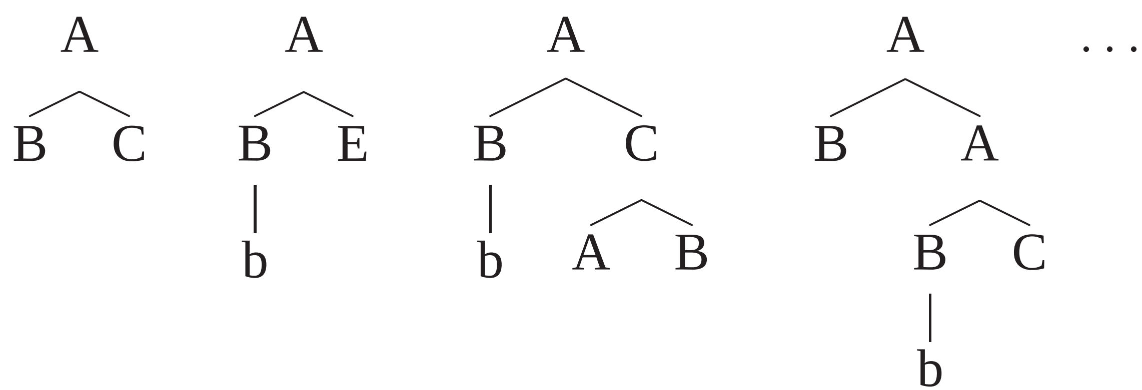


$$\mathbf{\Phi}(T_1) = \langle 1, 0, 0, 3 \rangle \qquad \mathbf{\Phi}(T_2) = \langle 2, 0, 1, 1 \rangle$$

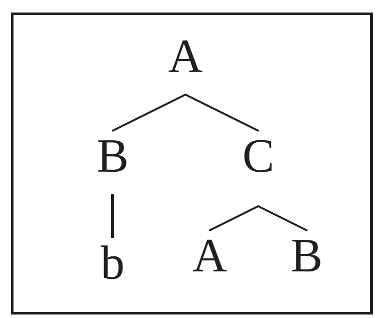
“All Subtrees” Representation [Bod, 1998]

- Given: Non-Terminal symbols $\{A, B, \ldots\}$
  Terminal symbols $\{a, b, c \ldots\}$

- An infinite set of subtrees



- An infinite set of features, e.g.,

$h_3(x, y) =$ Number of times  is seen in (x, y)

All Sub-fragments for Tagged Sequences

- Given: State symbols $\{S, C, N\}$
  Terminal symbols $\{a, b, c, \ldots\}$

- An infinite set of sub-fragments

$$\mathrm{S} \qquad \begin{array}{c} \mathrm{S} \\ | \\ \mathrm{a} \end{array} \qquad \mathrm{S} \text{ — } \mathrm{C} \qquad \begin{array}{ccc} \mathrm{S} & \text{—} & \mathrm{C} \\ & & | \\ & & \mathrm{b} \end{array} \qquad \ldots$$

- An infinite set of features, e.g.,

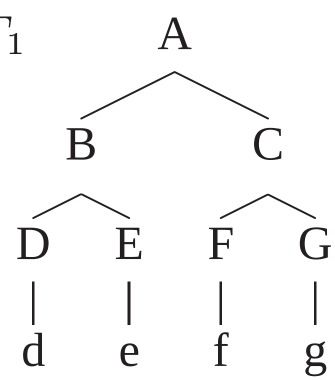
$$h_3(x) = \text{Number of times } \boxed{\begin{array}{ccc} \mathrm{S} & \text{—} & \mathrm{C} \\ & & | \\ & & \mathrm{b} \end{array}} \text{ is seen in } x$$

Inner Products

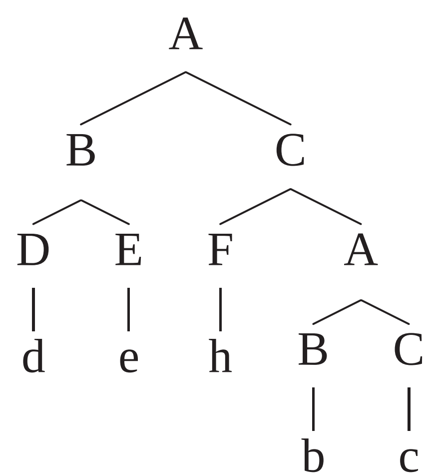
- $\mathbf{\Phi}(x) = \langle h_1(x), h_2(x) \ldots h_d(x) \rangle$
- Inner product (“**Kernel**”) between two structures T_1 and T_2:

$$\mathbf{\Phi}(T_1) \cdot \mathbf{\Phi}(T_2) = \sum_{i=1}^{d} h_i(T_1) h_i(T_2)$$

T_1



T_2

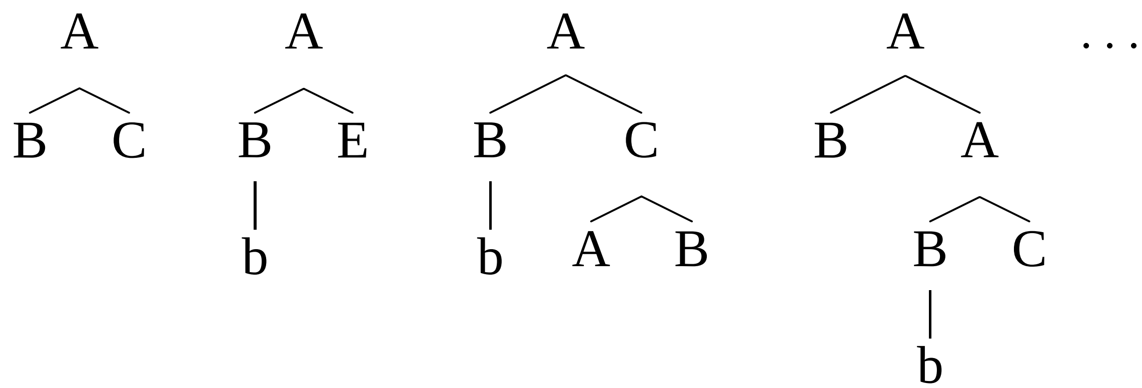


$\mathbf{\Phi}(T_1) = \langle 1, 0, 0, 3 \rangle$ $\mathbf{\Phi}(T_2) = \langle 2, 0, 1, 1 \rangle$

$$\mathbf{\Phi}(T_1) \cdot \mathbf{\Phi}(T_2) = 1 \times 2 + 0 \times 0 + 0 \times 1 + 3 \times 1 = 5$$

“All Subtrees” Representation

- Given: Non-Terminal symbols $\{A, B, \ldots\}$
  Terminal symbols $\{a, b, c \ldots\}$
- An infinite set of subtrees



- **Step 1:**
  Choose an (arbitrary) mapping from subtrees to integers

  $h_i(x) =$ Number of times subtree i is seen in x

  $\mathbf{\Phi}(x) = \langle h_1(x), h_2(x), h_3(x) \ldots \rangle$

All Subtrees Representation

- $\mathbf{\Phi}$ is now huge
- **But** inner product $\mathbf{\Phi}(T_1) \cdot \mathbf{\Phi}(T_2)$ can be computed efficiently using dynamic programming.

Computing the Inner Product

Define – N_1 and N_2 are sets of nodes in T_1 and T_2 respectively.

– $I_i(x) = \begin{cases} 1 \text{ if } i\text{'th subtree is rooted at } x. \\ 0 \text{ otherwise.} \end{cases}$

Follows that:

$h_i(T_1) = \sum_{n_1 \in N_1} I_i(n_1)$ and $h_i(T_2) = \sum_{n_2 \in N_2} I_i(n_2)$

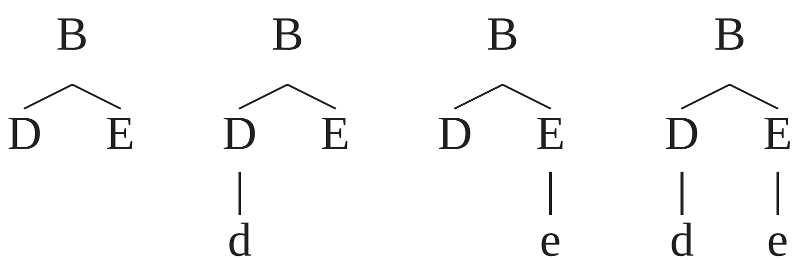
$$\begin{aligned}
\mathbf{\Phi}(T_1) \cdot \mathbf{\Phi}(T_2) = \sum_i h_i(T_1) h_i(T_2) &= \sum_i \left(\sum_{n_1 \in N_1} I_i(n_1)\right)\left(\sum_{n_2 \in N_2} I_i(n_2)\right) \\
&= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \sum_i I_i(n_1) I_i(n_2) \\
&= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \Delta(n_1,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

where $\Delta(n_1, n_2) = \sum_i I_i(n_1) I_i(n_2)$ is the number of common subtrees at n_1, n_2

An Example

$$\mathbf{\Phi}(T_1) \cdot \mathbf{\Phi}(T_2) = \Delta(A, A) + \Delta(A, B) \ldots + \Delta(B, A) + \Delta(B, B) \ldots + \Delta(G, G)$$

- Most of these terms are 0 (e.g. $\Delta(A, B)$).
- Some are non-zero, e.g. $\Delta(B, B) = 4$



Recursive Definition of $\Delta(n_1, n_2)$

- If the productions at n_1 and n_2 are different

$$\Delta(n_1, n_2) = 0$$

- Else if n_1, n_2 are pre-terminals,

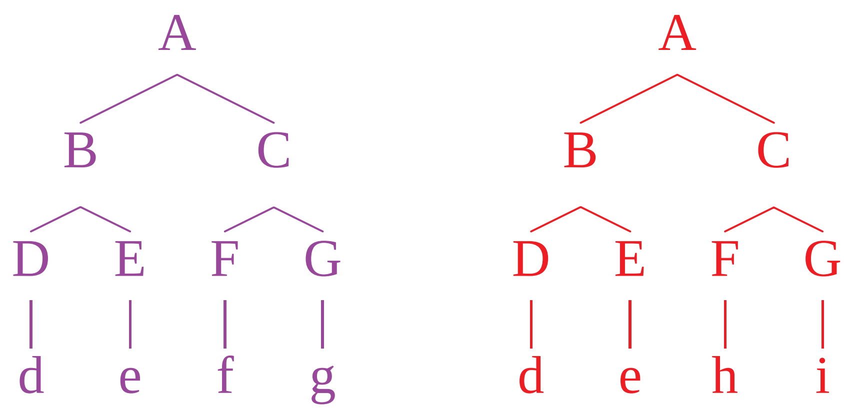
$$\Delta(n_1, n_2) = 1$$

- Else

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

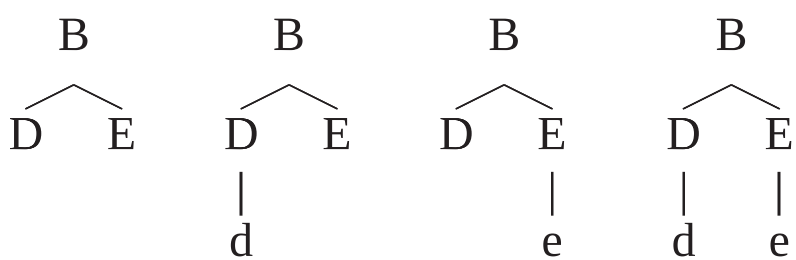
$nc(n_1)$ is number of children of node n_1;
$ch(n_1, j)$ is the j'th child of n_1.

Illustration of the Recursion

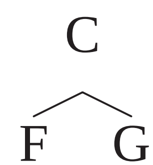


How many subtrees do nodes A and A have in common? i.e., What is $\Delta(A, A)$?

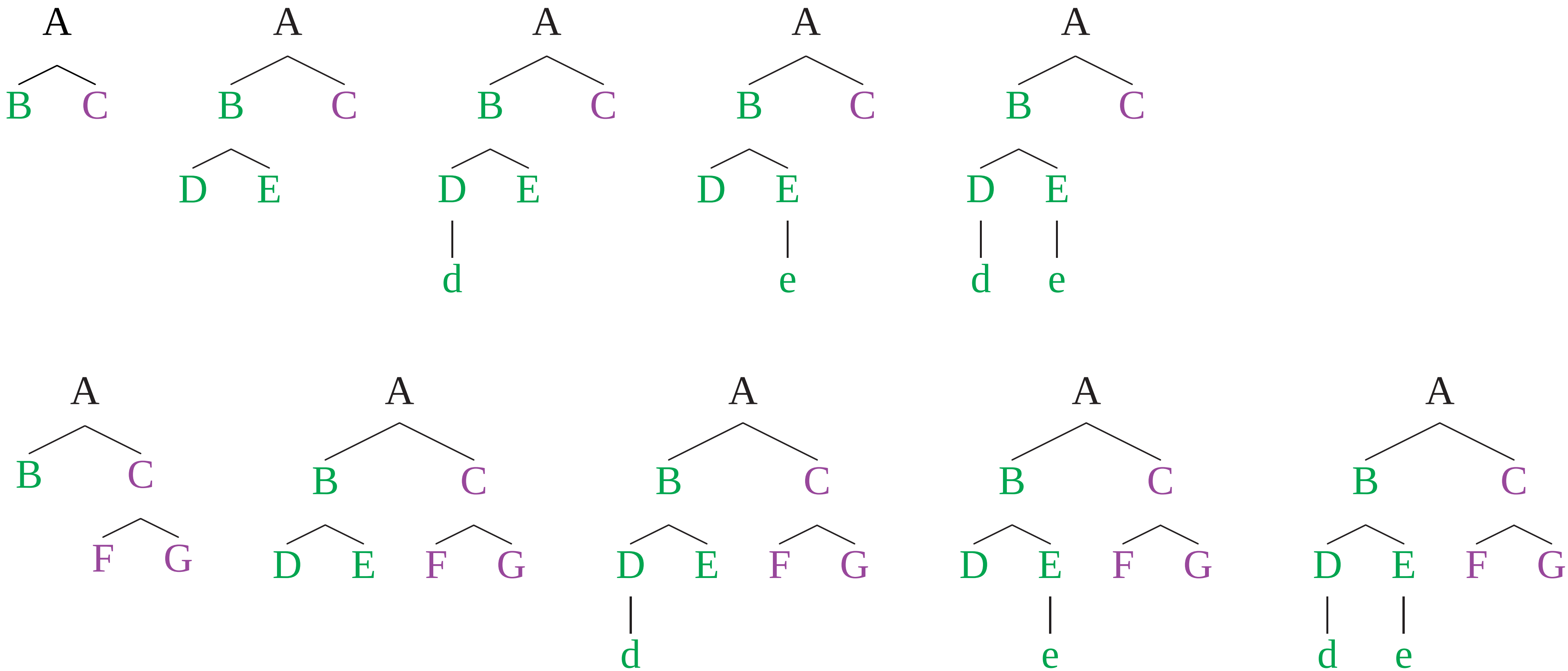
$$\Delta(B, B) = 4$$



$$\Delta(C, C) = 1$$



$$\boxed{\Delta(A, A) = (\Delta(B, B) + 1) \times (\Delta(C, C) + 1) = 10}$$


A B C
A B C D E
A B C D E d
A B C D E e
A B C D E d e
A B C F G
A B C D E F G
A B C D E F G d
A B C D E F G e
A B C D E F G d e

The Inner Product for Tagged Sequences

- Define N_1 and N_2 to be sets of states in T_1 and T_2 respectively.
- By a similar argument,

$$\mathbf{\Phi}(T_1) \cdot \mathbf{\Phi}(T_2) = \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \Delta(n_1, n_2)$$

where $\Delta(n_1, n_2)$ is number of common sub-fragments at n_1, n_2

$$\text{e.g., } T_1 = \begin{array}{ccccccc} \mathrm{A} & — & \mathrm{B} & — & \mathrm{C} & — & \mathrm{D} \\ | & & | & & | & & | \\ \mathrm{a} & & \mathrm{b} & & \mathrm{c} & & \mathrm{d} \end{array} \qquad T_2 = \begin{array}{ccccccc} \mathrm{A} & — & \mathrm{B} & — & \mathrm{C} & — & \mathrm{E} \\ | & & | & & | & & | \\ \mathrm{a} & & \mathrm{b} & & \mathrm{e} & & \mathrm{e} \end{array}$$

$$\mathbf{\Phi}(T_1) \cdot \mathbf{\Phi}(T_2) = \Delta(A, A) + \Delta(A, B) \ldots + \Delta(B, A) + \Delta(B, B) \ldots + \Delta(D, E)$$

e.g., $\Delta(B, B) = 4$,

$$\begin{array}{c} \mathrm{B} \\ \\ \\ \end{array} \qquad \begin{array}{c} \mathrm{B} \\ | \\ \mathrm{b} \end{array} \qquad \begin{array}{ccc} \mathrm{B} & — & \mathrm{C} \\ & & \\ & & \end{array} \qquad \begin{array}{ccc} \mathrm{B} & — & \mathrm{C} \\ | & & \\ \mathrm{b} & & \end{array}$$

The Recursive Definition for Tagged Sequences

- Define $N(n)$ = state following n, $W(n)$ = word at state n
- Define $\pi[W(n_1), W(n_2)] = 1$ iff $W(n_1) = W(n_2)$
- Then if labels at n_1 and n_2 are the same,

$$\Delta(n_1, n_2) = (1 + \pi[W(n_1), W(n_2)]) \times (1 + \Delta(N(n_1), N(n_2)))$$

---

e.g., $T_1 = \begin{array}{ccccccc} \mathrm{A} & — & \mathrm{B} & — & \mathrm{C} & — & \mathrm{D} \\ | & & | & & | & & | \\ \mathrm{a} & & \mathrm{b} & & \mathrm{c} & & \mathrm{d} \end{array}$ $T_2 = \begin{array}{ccccccc} \mathrm{A} & — & \mathrm{B} & — & \mathrm{C} & — & \mathrm{E} \\ | & & | & & | & & | \\ \mathrm{a} & & \mathrm{b} & & \mathrm{e} & & \mathrm{e} \end{array}$

$$\begin{aligned} \Delta(A, A) &= (1 + \pi[a, a]) \times (1 + \Delta(B, B)) \\ &= (1 + 1) \times (1 + 4) = 10 \end{aligned}$$

Refinements of the Kernels

- Include log probability from the baseline model:

  $\mathbf{\Phi}(T_1)$ is representation under all sub-fragments kernel

  $L(T_1)$ is log probability under baseline model

  New representation $\mathbf{\Phi}'$ where

  $$\mathbf{\Phi}'(T_1) \cdot \mathbf{\Phi}'(T_2) = \beta L(T_1) L(T_2) + \mathbf{\Phi}(T_1) \cdot \mathbf{\Phi}(T_2)$$

  (includes $L(T_1)$ as an additional component with weight $\sqrt{\beta}$)

- Allows the perceptron to use original ranking as default

Refinements of the Kernels

- Downweighting larger sub-fragments

$$\sum_{i=1}^{d} \lambda^{SIZE_i} h_i(T_1) h_i(T_2)$$

where $0 < \lambda \leq 1$,
$SIZE_i$ is number of states/rules in i'th fragment

- Simple modification to recursive definitions, e.g.,

$$\Delta(n_1, n_2) = (1+\pi[W(n_1), W(n_2)]) \times (1+\lambda\Delta(N(n_1), N(n_2))$$

Refinement of the Tagging Kernel

- Sub-fragments sensitive to spelling features (e.g., Capitalization)

  - Define $\pi[x, y] = 1$ if x and y are identical, $\pi[x, y] = 0.5$ if x and y share same capitalization features

$$\Delta(n_1, n_2) = (1+\pi[W(n_1), W(n_2)])\times(1+\lambda\Delta(N(n_1), N(n_2))$$

- Sub-fragments now include capitalization features

N — N — S
exiled to Elba

N — N — S
exiled to Cap

N — N — S
No cap to Cap

N — N — S
No cap No cap Cap

Experimental Results

Parsing Wall Street Journal

MODEL	$\leq$ 100 Words (2416 sentences)				
	LR	LP	CBs	0 CBs	2 CBs
CO99	88.1%	88.3%	1.06	64.0%	85.1%
VP	88.6%	88.9%	0.99	66.5%	86.3%

VP gives 5.1% relative reduction in error (CO99 = my thesis parser)

Named Entity Tagging on Web Data

	P	R	F
Max-Ent	84.4%	86.3%	85.3%
Perc.	86.1%	89.1%	87.6%
Improvement	10.9%	20.4%	15.6%

VP gives 15.6% relative reduction in error

Summary

- For any representation $\mathbf{\Phi}(x)$,
  Efficient computation of $\mathbf{\Phi}(x) \cdot \mathbf{\Phi}(y) \Rightarrow$
  Efficient learning through kernel form of the perceptron

- Dynamic programming can be used to calculate $\mathbf{\Phi}(x) \cdot \mathbf{\Phi}(y)$ under "all sub-fragments" representations

- Several refinements of the inner products:
  - Including probabilities from baseline model
  - Downweighting larger sub-fragments
  - Sensitivity to spelling features

<u>Conclusions: 10 Ideas from the Course</u>

1. Smoothed estimation
2. Probabilistic Context-Free Grammars, and history-based models
   $\Rightarrow$ lexicalized parsing
3. Feature-vector representations, and log-linear models
   $\Rightarrow$ log-linear models for tagging, parsing
4. The EM algorithm: hidden structure
5. Machine translation: making use of the EM algorithm
6. Global linear models: new representations (global features)
7. Global linear models: new learning algorithms (perceptron, boosting)
8. Partially supervised methods: applications to word sense disambiguation, named entity recognition, and relation extraction
9. Structured models for information extraction, and dialogue systems
10. A final representational trick, *kernels*