

Prédiction structurée et analyse syntaxique en dépendances

Joseph Le Roux

Université Paris 13

23 mars 2015

- 1 Introduction
- 2 Apprentissage en ligne et prédiction structurée
- 3 Analyse en dépendances
- 4 Conclusion

- 1 Introduction
- 2 Apprentissage en ligne et prédiction structurée
- 3 Analyse en dépendances
- 4 Conclusion

- Présentation des méthodes utilisées par l'analyseur MST [McDonald, 2005]
- Analyse syntaxique en dépendances (approche par construction de graphe)
- Utilisation de méthodes d'apprentissage pour inférer la bonne construction des graphes (prédiction structurée)

- 1 Introduction
- 2 Apprentissage en ligne et prédiction structurée**
- 3 Analyse en dépendances
- 4 Conclusion

Définition

Un champ de l'apprentissage artificiel où les étiquettes prédites ont une structure (séquence, arbre, graphe...)

Conséquence

L'ensemble des sorties (étiquettes) est *plus grand* que l'ensemble des entrées.

exemple :

séquences de mots → arbres

Modèle linéaire

- Soit une entrée $x \in \mathcal{X}$ (suites de mots)
- Soit une sortie $y \in \mathcal{Y}$ (arbres syntaxiques)

Modèle linéaire et fonction de score associée :

$$s(x, y) = w \cdot f(x, y) = \sum_{i=1}^d w_i \cdot f_i(x, y)$$

- $f(x, y) \in \mathbb{R}^d$ est un vecteur représentant une structure associant l'entrée et la sortie.
- $f(x, y)$ décompose cette structure en d parties élémentaires (traits ou features)
 - En pratique, $f(x, y) \in \{0, 1\}^d$
 - indique la présence/absence d'une partie possible
- $w \in \mathbb{R}^d$ est le vecteur de poids qui indique pour chaque trait si cette information est positive ou négative.

On veut que $s(x, y)$ donne

- un score élevé quand y est la sortie correcte pour x et
- un score bas dans le cas contraire

On pourra alors faire des **prédictions**

$$\hat{y} = \arg \max_{y \in \mathcal{Y}} s(x, y) = h(x)$$

Remarque : \mathcal{Y} est *grand* \rightarrow on ne peut pas énumérer (prg. dyn.)

Comment faire ?

- On suppose que le découpage des structures en d traits est *fixé*
- \Rightarrow c'est sur w qu'il faut jouer
- On veut apprendre w à partir d'exemples

Apprentissage en ligne

Apprentissage supervisé

- corpus d'entraînement $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$
- **but** apprendre w pour faire le moins possible d'erreurs sur \mathcal{T}
- On voit **un** exemple du corpus d'entraînement à la fois
- On met le vecteur de poids à jour après chaque exemple

Algorithme AeL générique

$$\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$$

$$w^{(0)} = 0; i = 0;$$

for $n = 1 \rightarrow N$ **do**

for $t = 1 \rightarrow T$ **do**

 ★ $w^{(i+1)}$ = mise à jour de $w^{(i)}$ en fonction de y_t et $h(x_t)$

$i = i + 1$

end for

end for

return $w^{(i)}$

Idée

Si la prédiction avec le modèle courant est incorrecte :

- donner plus de poids aux parties de la référence absents de la prédiction
- donner moins de poids aux parties de la prédiction absents de la référence

Mise à jour

$$\star \quad w^{(i+1)} = w^{(i)} + \underbrace{f(x_t, y_t)}_{\text{vecteur structure correcte}} - \underbrace{f(x_t, h(x_t))}_{\text{vecteur structure prédite}}$$

Remarque

Si la prédiction est correcte, pas de mise à jour

Perceptron (structuré) [Collins, 2002]

Le perceptron cherche un **hyperplan** dans \mathbb{R}^d qui mette

- les vecteurs d'exemples $\{f(x_t, y_t)\}_{i=1}^T$ d'un côté
- et les autres vecteurs $\{f(x_t, y) | y \neq y_t\}$ de l'autre.
- w est le vecteur normal à l'hyperplan

Si un tel plan existe (séparabilité linéaire)

- le perceptron le trouve
- sinon, *bon compromis*

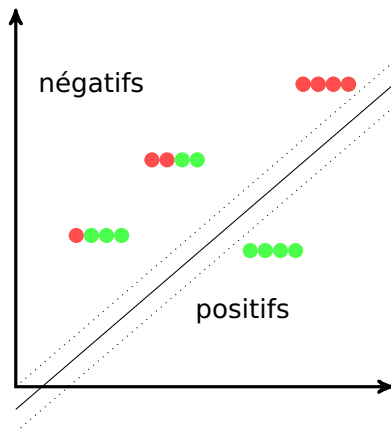
Avantage

Extrêmement simple

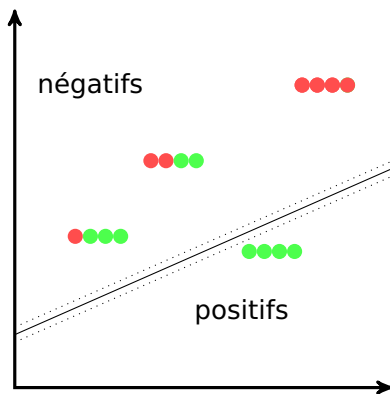
Inconvénient

Toutes les erreurs se valent, toutes les sorties différentes de l'exemple sont également fausses

Perceptron (illustration)



Perceptron (illustration 2)



Fonction de perte \mathcal{L}

- $\mathcal{L} : \mathcal{Y} \times \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$
- $\mathcal{L}(y, x, y')$: coût quand $h(x) = y'$ et la solution est y

Problèmes en TAL:

les fonctions de perte

- ne sont pas (toujours) bien définies formellement
- ne sont pas acceptées par tous
- ne sont pas uniques
- sont souvent extrinsèques
- changent
- requièrent le jugement d'êtres humains

Fonction de perte \mathcal{L}

- $\mathcal{L} : \mathcal{Y} \times \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$
- $\mathcal{L}(y, x, y')$: coût quand $h(x) = y'$ et la solution est y

Problèmes en TAL:

les fonctions de perte

- ne sont pas (toujours) bien définies formellement
- ne sont pas acceptées par tous
- ne sont pas uniques
- sont souvent extrinsèques
- changent
- requièrent le jugement d'êtres humains

Analyse en dépendances

la perte sera le nombre d'arcs différents entre y et y'

Margin Infused Relaxed Algorithm (MIRA) [Crammer and Singer, 2003]

On veut *corriger* le perceptron pour qu'il tienne compte de l'erreur commise pendant l'étape de mise à jour.

$$\min \|w\| \quad (1)$$

$$\text{t.q. } s(x_t, y_t) \geq s(x_t, y) + \mathcal{L}(y_t, x_t, y) \quad (2)$$

$$\forall (x_t, y_t) \in \mathcal{T} \text{ et } y \in \mathcal{Y} \quad (3)$$

- plus l'erreur prédiction est grande, plus le score doit être petit
- perceptron avec contraintes de marge

Margin Infused Relaxed Algorithm (MIRA) [Crammer and Singer, 2003]

$$\star \quad w^{(i+1)} = \arg \min_w \|w - w^{(i)}\| \quad (4)$$

$$\text{t.q. } s(x_t, y_t) \geq s(x_t, y) + \mathcal{L}(y_t, x_t, y) \quad (5)$$

$$\forall (x_t, y_t) \in \mathcal{T} \text{ et } y \in \mathcal{Y} \quad (6)$$

Margin Infused Relaxed Algorithm (MIRA) [Crammer and Singer, 2003]

Forme *close* (Algorithme de Hildreth à profondeur limitée)

$$\star \quad w^{(i+1)} = w^{(i)} + \sum_k \alpha(y_k)(f(x_t, y_t) - f(x_t, y_k)) \quad (7)$$

$$\alpha(y_k) = \max(0, \alpha(y_{k-1}) + \delta(y_k)) \quad (8)$$

$$\delta(y_k) = \frac{\mathcal{L}(y_t, x_t, y_k) - (s(x_t, y_t) - s(x_t, y_k))}{\|f(x_t, y_t) - f(x_t, y_k)\|} \quad (9)$$

- y_k est la k^e meilleure solution retournée par \bar{h}
- en pratique, pour avoir vraiment une forme close, on limite k

Margin Infused Relaxed Algorithm (MIRA) [Crammer and Singer, 2003]

Problème : on ne peut pas énumérer les y_i

on se limite aux k -meilleures solutions renvoyées ($k=1, k=5, k=10$)

$k = 1$

$$\star w^{(i+1)} = w^{(i)} + \alpha(y_1)(f(x_t, y_t) - f(x_t, y_1)) \quad (10)$$

$$\alpha(y) = \max(0, \delta(y)) \quad (11)$$

$$\delta(y) = \frac{\mathcal{L}(y_t, x_t, y) - (s(x_t, y_t) - s(x_t, y))}{\|f(x_t, y_t) - f(x_t, y)\|} \quad (12)$$

Avantage :

assez facilement calculable

Algorithme AeL moyenné

$$\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$$

$$w^{(0)} = 0; m^{(0)} = 0; i = 0;$$

for $n = 1 \rightarrow N$ **do**

for $t = 1 \rightarrow T$ **do**

 ★ $w^{(i+1)}$ = mise à jour de $w^{(i)}$ en fonction de (x_t, y_t)

$$m^{(i+1)} = m^{(i)} + w^{(i+1)}$$

$$i = i + 1$$

end for

end for

return $\frac{m^{(i)}}{i}$

Pourquoi ?

- marche mieux en pratique
- pas de preuve formelle ?

Une méthode d'apprentissage simple : le perceptron structuré

- méthode numérique pour la prédiction de structures discrètes
- ne tient pas compte des types d'erreur

Une méthode qui tient compte des marges : MIRA

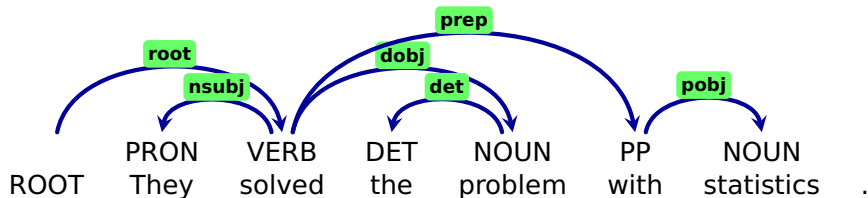
- aussi simple à mettre en place que le perceptron
- *tient compte* des types d'erreurs (du nombre)
 - pas de meilleure garantie formelle que le perceptron
 - en pratique : converge plus vite
 - en pratique : permet de faire de meilleures prédictions

Attention

Reposent sur la possibilité de générer le meilleur arbre en un temps *raisonnable* (cf. suite)

- 1 Introduction
- 2 Apprentissage en ligne et prédiction structurée
- 3 Analyse en dépendances**
- 4 Conclusion

Analyse en dépendances (Tesnière, 1959...)



Remarques

- Arbre projectif (pas de croisement)
 - CFG lexicalisée
- nœuds = mots (+ catégories)
- tête → dépendant

Décomposition en arcs

On suppose que le score / poids / probabilité d'un arbre peut se décomposer en fonction des arcs :

$$w(t) = \sum_{(i,j,k) \in t} a_{ij}^k$$

- a_{ij}^k est le score d'un arc entre les mots w_i et w_j étiqueté par l_k
- on suppose donc que chaque décision (juger la pertinence d'un arc) est indépendante.

Analyse en dépendances : Modèles 1er-ordre (*arc-factored*)

On suppose que le score / poids / probabilité d'un arbre peut se décomposer en fonction des arcs :

- $a_{ij}^k = s(i, j, k) = w_{i,j,k} \cdot f(i, j, k)$
- ce qui donne

$$w(x, t) = \sum_{(i,j,k) \in t} w_{i,j,k} \cdot f(x, i, j, k)$$

Bonne nouvelle

On peut apprendre ce modèle avec Perceptron/MIRA !

Nombre exponentiel d'arbres pour une phrase.

- on ne peut pas les énumérer pour ensuite décider lequel est le meilleur
- il faut pouvoir construire le(s) meilleur(s) arbre de manière efficace.

Arbre de recouvrement maximal (MST, Chu-Liu-Edmonds)

- on considère une phrase comme le graphe complet dont les mots sont les nœuds.
- on cherche l'arbre de poids maximal par l'algo MST

Pseudo-code

$G = (V, E)$

while true **do**

$E' = \{e_i | \exists v_i \in V \text{ et } e_i \text{ est l'arc entrant de score max pour } v_i\}$

if (V, E') est un arbre **then**

return (V, E')

else

$(V, E) = \text{contracte-cycle}(V, E', E)$

end if

end while

Arbre de recouvrement maximal (MST, Chu-Liu-Edmonds)

- on considère une phrase comme le graphe complet dont les mots sont les nœuds.
- on cherche l'arbre de poids maximal par l'algo MST

Avantages

- efficace $O(n^2)$
- extension pour obtenir les k meilleures analyses : $O(k^2 n^2)$ (en fait... non)

Inconvénients

- arbres non-projectifs
- difficile à adapter aux modèles d'ordre supérieur

Algorithme d'analyse : programmation dynamique

On peut aussi essayer de *recycler* un algorithme d'analyse plus classique [CKY, en $O(n^3)$] qui construit les arbres incrémentalement de bas en haut.

Avantage

- construction de la forêt (ensemble des arbres solutions)
- facilement adaptable aux modèles d'ordre supérieur

Inconvénients

- arbre projectifs seulement
- algorithme cubique ($O(k \log(k)n^3)$ pour les k meilleures analyses)

Algorithme d'analyse : programmation dynamique


ROOT PRON VERB DET NOUN PP NOUN
ROOT They solved the problem with statistics .

ROOT						
They						
solved						
the						
problem						
with						
statistics						

They solved the problem with statistics

Algorithme d'analyse : programmation dynamique

ROOT PRON VERB DET NOUN PP NOUN
They solved the problem with statistics .

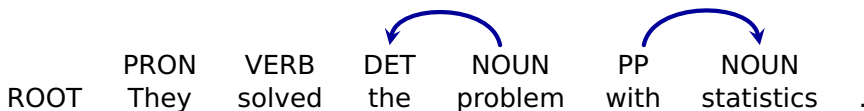


ROOT						
They						
solved						
the						
problem						
with						
statistics						

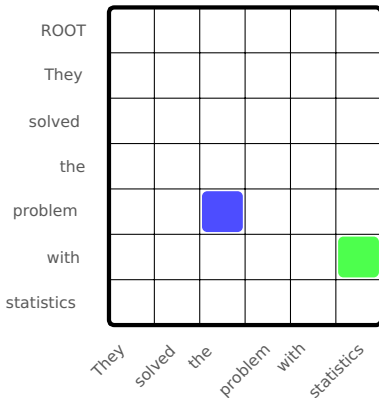
They solved the problem with statistics

Algorithme d'analyse : programmation dynamique

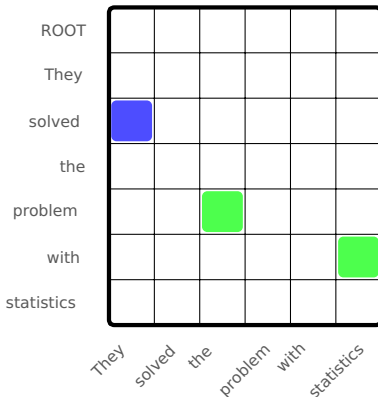
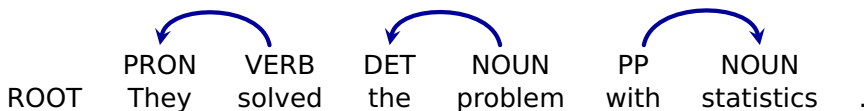
ROOT PRON VERB DET NOUN PP NOUN
They solved the problem with statistics .



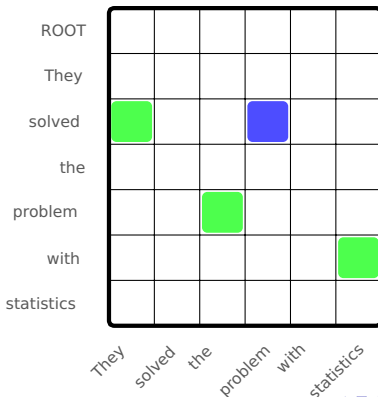
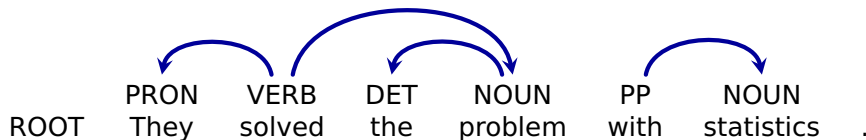
ROOT						
They						
solved						
the						
problem						
with						
statistics						



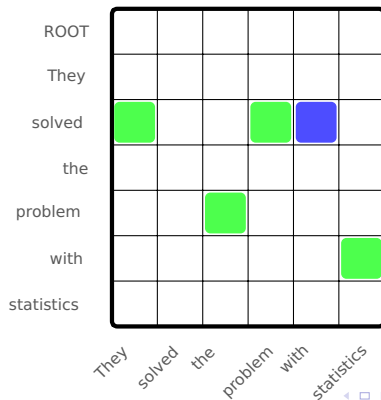
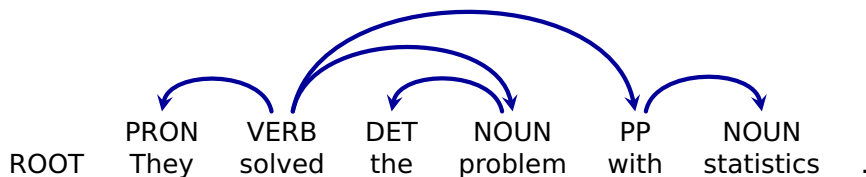
Algorithme d'analyse : programmation dynamique



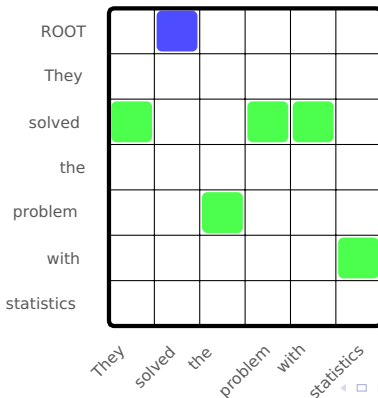
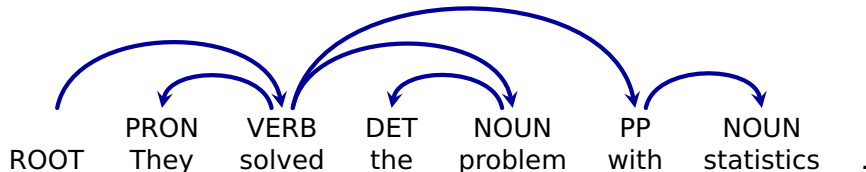
Algorithme d'analyse : programmation dynamique



Algorithme d'analyse : programmation dynamique

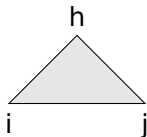


Algorithme d'analyse : programmation dynamique

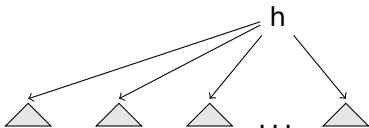


Algorithme d'analyse : programmation dynamique

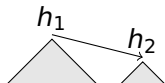
On veut construire une sous analyse de w_i à w_j de racine w_h



On peut décomposer cet arbres lui-même en sous-analyses



On peut exprimer cette décomposition en opérations binaires



Algorithme d'analyse : programmation dynamique

Analyse déductive

Une sous-analyse du mot w_i au mot w_j , de tête w_h et de score s se représente par un *item* $[h, i, j, s]$.

Spécification

but $[0, 0, n, s]$

initialisation $\frac{}{[i, i, i, s(i)]} \forall i \in [0; n]$

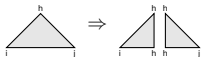
complète $\frac{[h_1, i, l, s_1][h_2, l + 1, j, s_2]}{[h_i, i, j, s_1 + s_2 + s(h_i, h_{\bar{i}})]}, i \in \{1, 2\}, \bar{i} \neq i$

Problème

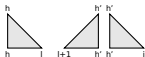
- complexité : $O(n^5)$
- (si on prend en compte les étiquettes $O(|L|n^5)$)

Programmation dynamique (Eisner, 96)

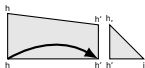
Sous-analyses orientées : tête toujours en périphérie.



On combine des sous-analyses opposées



On passe par un état incomplet (trapèze)



Qu'on recombine avec une sous-analyse orientée



Algorithme d'analyse : Eisner

ROOT PRON VERB DET NOUN PP NOUN
ROOT They solved the problem with statistics .

ROOT						
They						
solved						
the						
problem						
with						
statistics						

They solved the problem with statistics



Algorithme d'analyse : Eisner

ROOT PRON VERB DET NOUN PP NOUN
They solved the problem with statistics .

ROOT						
They						
solved	■					
the						
problem			■			
with					■	
statistics						

They solved the problem with statistics



Algorithme d'analyse : Eisner

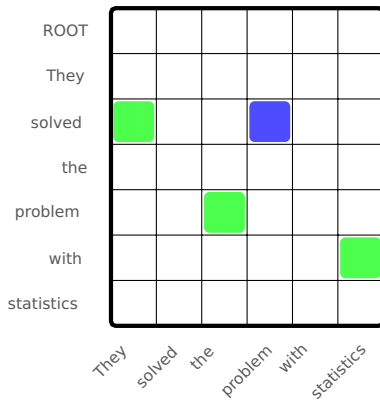
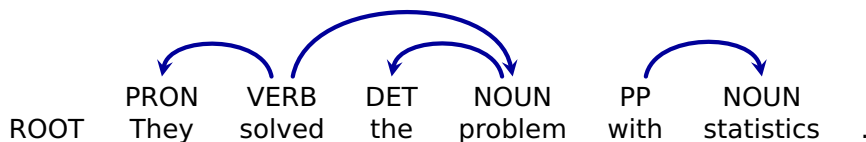
ROOT PRON VERB DET NOUN PP NOUN
They solved the problem with statistics .

ROOT						
They						
solved	■					
the						
problem			■			
with					■	
statistics						

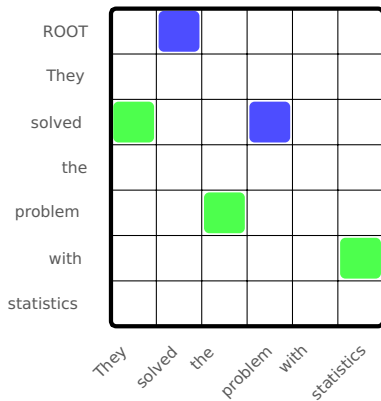
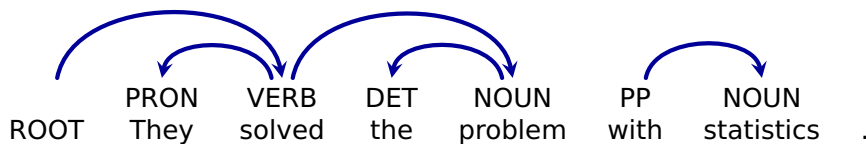
They solved the problem with statistics



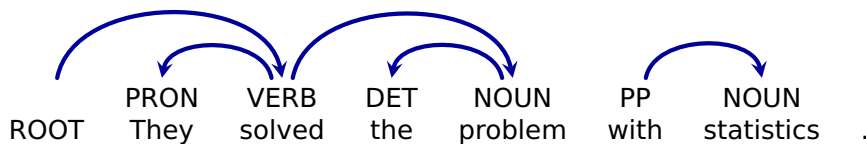
Algorithme d'analyse : Eisner



Algorithme d'analyse : Eisner



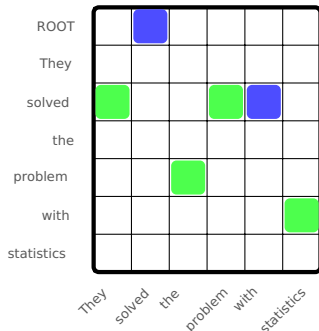
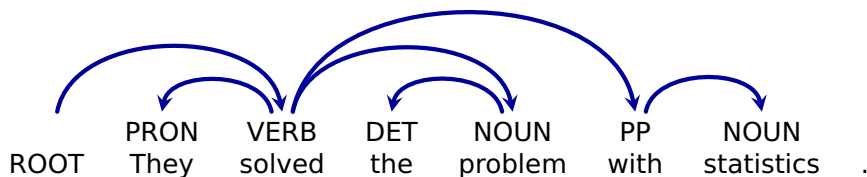
Algorithme d'analyse : Eisner



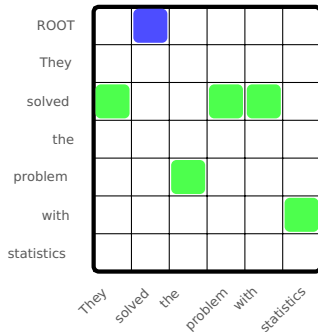
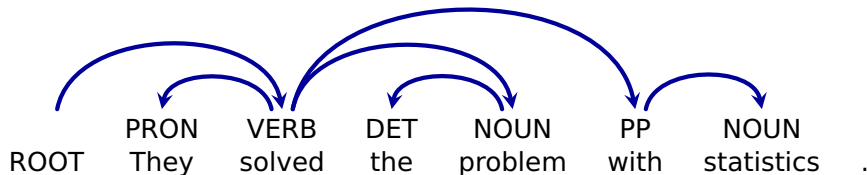
ROOT		ROOT				
They						
solved	SOLVED		SOLVED			
the						
problem		PROBLEM				
with					PP	
statistics						NOUN
	They	solved	the	problem	with	statistics



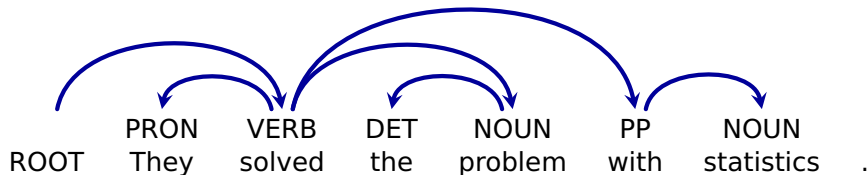
Algorithme d'analyse : Eisner



Algorithme d'analyse : Eisner



Algorithme d'analyse : Eisner



ROOT		■				
They						
solved	■			■	■	
the						
problem			■			
with						■
statistics						

They solved the problem with statistics



Pseudo-code

INIT: $C[s][s][d][c] = 0.0 \forall s, d, c$

for $k = 1 \rightarrow n$ **do**

for $s = 1 \rightarrow n$ **do**

$t = s + k$

if $t > n$ **then** break

end if

 % items incomplets (trapèzes)

$C[s][t][\leftarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(t, s))$

$C[s][t][\rightarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + s(s, t))$

 % items complets (triangles)

$C[s][t][\leftarrow][1] = \max_{s \leq r < t} (C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0])$

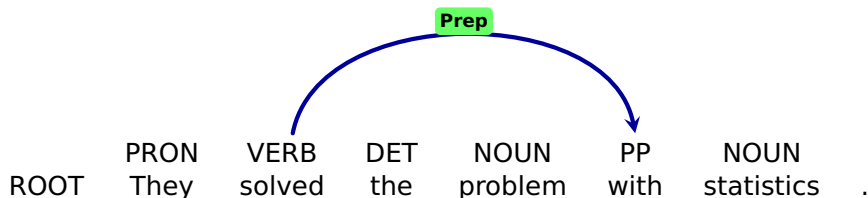
$C[s][t][\rightarrow][1] = \max_{s < r \leq t} (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1])$

end for

end for

Complexité

- $O(n^3)$
- avec les étiquettes $O(|L|n^3)$ (ou mieux $O(n^3 + |L|n^2)$ avec précalculs)



- mots w_i , w_j , et étiquette l_k
- étiquettes POS des mots w_i et w_j
- étiquettes des mots avant/après la tête
- étiquettes des mots avant/après le dépendant
- distance
- mots/étiquettes entre w_i et w_j
- + combinaisons des traits précédents

Complexité

On a supposé que les $s(i, j, k)$ sont calculables en $O(1)$



1^{er} ordre

chaque choix d'arc est indépendant

- hypothèse peu réaliste

2^e ordre (réduction d'erreur $\approx 10\%$)

chaque choix dépend d'un voisin

- un arc qui part de la même tête (2^e ordre "frère") 
- un arc qui arrive à la tête (2^e ordre "grand-père") 
- Dans les 2 cas, on peut trouver une décomposition des items pour rester en $O(n^3)$

3^e ordre (réduction d'erreur $\approx 10\%$)

- combinaison 2^e ordre "frère" + 2^e ordre "grand-père" : $O(n^4)$

Le calcul des traits devient le goulot d'étranglement

- Approches par raffinement
 - On calcule un ensemble de meilleures solutions en premier ordre
 - On applique l'algorithme de 2e ou 3e ordre

Cas non-projectif NP-difficile en ordre supérieur

- Méthodes efficaces par approximation (décomposition duale)

- 1 Introduction
- 2 Apprentissage en ligne et prédiction structurée
- 3 Analyse en dépendances
- 4 Conclusion

Conclusion

Méthodes efficaces pour calculer le(s) meilleur(s) arbre(s)

- polynomiales

Compatibles avec un apprentissage type Perceptron

- simples à mettre en œuvre

Questions

- Comment tenir compte des interactions entre les arcs ?
 - ordre supérieur généralisé (*cube pruning*)
 - calcul des traits efficace/paresseux
- Cas non-projectif en programmation dynamique ?
 - langages faiblement contextuels (*grammaires d'arbres adjoints*)