1. Recapitulación

- La Gramática de Dependencias es un marco teórico desarrollado principalmente por Tesnière (1959)
- Su implementación computacional se dio principalmente por medio de *parsers*, basados en grafos o basados en transiciones, que constan entre sus componentes de algún modelo de aprendizaje automático entrenado en tareas supervisadas.

2. Gramática de dependencias como cuádruplas

En los años '60, los trabajos de Hays (1964) y Gaifman (1965) exploraron las propiedades matemáticas de las gramáticas de dependencias y demostraron su equivalencia débil a a las gramáticas independientes de contexto. En estos trabajos, la gramática de dependencias se formaliza como una cuádrupla de la siguiente manera:

(1) $\langle T, A, FA, RD \rangle$

Los cuatro elementos que conforman la cuádrupla representan respectivamente lo siguiente:

- **Símbolos Terminales (T)**: son los elementos mínimos que aparecen efectivamente en las cadenas/oraciones.
- **Símbolos Auxiliares (A)**: son los nombres o símbolos que tipifican los símbolos terminales, como, por ejemplo, N, V, Adj, etc.
- **Función de Asignación (FA)**: establece a qué categoría (extraída del conjunto de los símbolos preterminales) pertenece determinada palabra (extraída de los símbolos terminales). Esto es, las reglas de asignación son un conjunto de pares ordenados de modo tal que para toda palabra haya al menos una categoría y para toda categoría haya al menos una palabra.

- **Reglas de Dependencia (RD)**: establecen las relaciones de dependencia entre distintos símbolos auxiliares. Existen tres tipos:

    1. $X_i(Y_{j_1}, X_{j_2}, ..., *, .., Y_{j_n})$: relaciona un símbolo auxiliar (X_i) con un número finito de auxiliares ($Y_{j_1}, .., Y_{j_n}$), que dependen de aquel. De X_i diremos que es el regidor y de $Y_{j_1}, .., Y_{j_n}$, sus dependientes. * simboliza la posición lineal de X en la cadena.
    2. $X_i(*)$: indica que el símbolo auxiliar X_i no tiene dependencias.
    3. $*(X_i)$: el símbolo auxiliar x_i puede ocurrir sin ser regido; *i.e.* esta unidad puede funcionar como raíz (*root*) o nodo principal (*main*) de una cadena bien formada/oración.

La regla de dependencia 1 se relaciona directamente con la noción de **valencia** de una unidad sintáctica. Siguiendo la notación, la valencia de X_i está dada por el conjunto de sus dependientes, aquellas unidades con las que se relaciona direcamtente.

Como ya hemos visto anteriormente, una gramática formal, del tipo que sea, deberá contar con un procedimiento que genere todas las oraciones del lenguaje que esa gramática describe (procedimiento generativo), así como un mecanismo de decisión que le permita, dada una cadena, decidir si pertenece o no al conjunto de cadenas que la gramática describe (procedimiento de decisión).

3. Procedimiento generativo

Ver ejercicio en 6.1

Si tomamos la cuádrupla de elementos que describe Hays y construimos con ella un mecanismo que nos permita generar oraciones de un lenguaje, el procedimiento tendría dos etapas. La primera etapa consitiría en una serie de pasos de los cuales todos, a excepción del primero, ocurrirían en el mismo plano. El primer paso en cuestión es elegir el elemento central de la cadena. Es decir, aplicar la regla de dependencia 3 (reescrita en 2):

(2) $*(X_i)$

El segundo paso es elegir entre una regla del tipo 2 (resscrita en 3) donde se terminaría la primera etapa del procedimiento de generación, o una de tipo 1 (en 4)

(3) $(X_i(*))$

(4) $(X_i(Y_{j_1}, X_{j_2}, ..., *, .., Y_{j_n}))$

Si se elige una regla de tipo 1, cada uno de los dependiente de X será agregado a la construcción y se volverán a aplicar reglas de tipo 1 o 2 sobre ese elemento agregado, que a su vez podrá agregar otro dependientes, y así sucesivamente hasta finalizar la construcción, cuando cada símbolo auxiliar contenga un término parentético del tipo (*) después de sí.

Hasta este punto del procedimiento hemos hecho uso de elementos del vocabulario auxiliar. La segunda etapa remplaza los elementos auxiliares por los terminales mediante la función de asignación.

Repasemos estos pasos con un ejemplo un poco más concreto. El primer paso, consistirá en establecer cuál será elemento raíz de nuestra gramática. Asumamos que este elemento es el verbo:

(5) *(V)

Si quisiéramos que nuestra gramática genere oraciones que solo estén conformadas por un verbo (*¡Huyamos!*), en el segundo paso debemos seleccionar una regla de dependencia que indique que el verbo no tiene dependientes y, así, terminar el procedimiento:

(6) *(V (*))

Si, en cambio, queremos poder generar oraciones ditransitivas (como, por ejemplo, *La chica compró regalos para su madre*), debemos poder indicar cuáles serán los dependientes del verbo:

(7) *(V (N, * ,N, P))

En la regla 9, ya se ha determinado cuáles son las dependencias de V, pero no del resto de los símbolos auxiliares, si es que acaso las tienen. Si queremos que los nombres puedan recibir un determinante, deberemos indicarlo:

(8) *(V (N (D, *), *, N (*), P))

Y, así, deberemos ir indicando cuáles serán los dependientes de cada auxiliar:

(9) *(V (N (D (*), *) , *, N (*) , P (*, N (D (*), *)))))

Ver ejercicio en 6.2

Observemos que esta regla, así expresada, no nos permite ver claramente el orden de los caracteres auxiliares, para ello, debemos numerar los dependientes:

1.1 D

1.2 N

2 V

3 N

4.1 P

4.2.1 D

4.2.2 N

4. Procedimiento de decisión

El procedimiento de decisión trabajará "al revés" del de generación. Su primer paso será listar los símbolos auxiliares que corresponden a Los símbolos terminales presentes en la cadena sobre la que se está aplicando el mecanismo. Luego, considerará todas las reglas de la gramática que coincidan con los caracteres auxiliares listados. Para ello, comenzará con las reglas del tipo 3, donde la unidad en consideración no tiene dependientes. Luego seguirá con los símbolos de los que depende un elementos, y así sucesivamente hasta listar la regla que incluye todas las unidades auxiliares de la cadena.

Veamos un ejemplo. Supongamos que tenemos la gramática definida en 10 y queremos saber si acepta la oración *Mi hermano rindió un examen difícil el martes*. Esta oración podría representarse de la siguiente manera (ver 11)

(10)

- **Símbolos Terminales:** { mi, el, su, un, hermano, hermana, novio, examen, parcial, lunes, martes, rindió, aprobó, difícil, fácil }
- **Símbolos Auxiliares:** { D, N, V, Adj }
- **Función Asignación:** { ⟨mi, D⟩, ⟨el, D⟩, ⟨su, D⟩, ⟨un, D⟩ ⟨hermano, N⟩, ⟨hermana, N⟩, ⟨novio, N⟩, ⟨examen, N⟩, ⟨parcial, N⟩, ⟨lunes, N⟩, ⟨martes, N⟩, ⟨rindió, V⟩, ⟨aprobó, V⟩, ⟨difícil, Adj⟩, ⟨fácil, Adj⟩ }
- **Reglas de Dependencia:** { *(V), D(*), Adj(*), N(D, *), N(D, *, Adj), V(N, *, N), V(N, *, N, N) }

(11) *(V (N (D (*), *) , *, N (D (*), *, Adj (*)) , N (D (*), *)))

(12)

D	*N*	*V*	*D*	*N*	*Adj*	*D*	*N*
1.1.	1.2	2	3.1	3.2	3.3	4.1	4.2

Primero, generamos la tabla con las unidades sintácticas mínimas y sus correspondientes caracteres auxiliares. Luego, iremos aplicando las reglas de la gramática que conicidan con estos caracteres. El orden en el que las aplicaremos será de menor a mayor cantidad de dependientes (empezando por los auxiliares que no tienen depedientes alguno):

Unidades sintácticas	1	2	3	4
Caracteres auxilires	D_{11} N_{12}	V_2	D_{31} N_{32} Adj_{33}	D_{41} N_{42}
Rgl. de un caracter	$D_{11}(*)$		$D_{31}(*)$ $Adj_{33}(*)$	$D_{41}(*)$
Rgl. de 2 caracteres	$N_{12}(D_{11}(*),*)$			$N_{42}(D_{41}(*),*)$
Rgl. de 3 caracteres	$N_{32}(D_{31}(*),*,Adj(*))$			

Ver ejercicio en 6.3

Por motivos de extensión, no hemos completado todas las reglas que permiten procesar la oración en la tabla. Pero, si combinamos la última regla expresada en 10 con los dependientes pertinentes, obtendremos la regla detallada en 11.

5. Implementaciones

- Los ***parsers* de dependencias proyectivos y no proyectivos de NLTK** están basados en reglas en un modo similar a los parsers de CFG. Vamos a ver cómo usarlos en la notebook.

- **Stanza** es el nombre que recibe la librería de python para procesamiento de lenguaje del Stanford NLP Group. Dentro de esa librería, hay disponible un parser de dependencias basado en transiciones (con un modelo de redes neuronales como base). En la página del grupo se puede encontrar una descripción del parser que originalmente fue escrito en Java pero que en la notebook vamos a ver implementado en python. Stanza también cuenta con una demo para probar el pipeline de NLU en distintos idiomas.

- **Malt parser** Malt Parser es más un proyecto para desarrollar parsers que un parser único en sí mismo. Está desarrollado en Java y se puede visitar la página oficial para buscar más información. En la notebook vamos a ver una implementación del malt parser a través de NLTK, lo que nos va a permitir usarlo con Python.

- **spaCy** es una librería *open source* de Python para procesamiento de lenguaje natural (como NLTK). Entre las herramientas que provee la librería (NER, PoS tag, etc.) hay un parser de dependencias basado en transiciones pero que permite una transformación pseudo proyectiva basada en Nivre y Nilsson (2005) para predecir árboles no proyectivos. Pasemos a ver el parser de spaCy en acción en la notebook de esta clase.

- **Freeling** es también una librería open source en C++ que tiene una implementación en Python llamada pyfreeling. Se puede acceder a su:

- Página
- GitHub
- Demo

En la demo pueden ver que FreeLing tiene distintos tipos de parsers, entre los que se incluye uno de CFG (full parser), un shallow parser (que devuelve los *chunks*) sin relaciones profundas, y un dependency parser. Pero internamente, FreeLing provee de tres DP diferentes: uno basado en reglas (Txala), uno estadístico y uno basado en un modelo de redes neuronales. DP en el Manual de Usuario.

- **SintaxNet (deprecado)** En 2016, Google abrió el código y presentó el parser "más preciso del mundo" basado en un modelo de redes neuronales y transiciones. La publicación de presentación explica de un modo bastante claro cómo funciona este parser.

6. Ejercitación

6.1. Definición de cuádrupla

Viene de la página 2

Generá la cuádrupla para una gramática sencilla que describa las siguientes oraciones, incluyendo las reglas de dependencias necesarias:

- Alquilo un departamento.
- El departamento es chico.
- Pienso mudarme nuevamente.

Cuádrupla:

6.2. Orden de términos auxiliares

Viene de la página 4

Ordená y numerá los términos auxilires de la siguiente regla de dependencias:
(X$_i$, (X$_{j_1}$ (X$_{k_1}$ (, X$_{l_1}$ (*)), *), X$_{j_2}$ (*), *, X$_{j_3}$ (*, X$_{m_1}$ (*))))

6.3. Decisión de aceptación

Viene de la página 6

¿Qué otras oraciones acepta la gramática definida en la sección 4? ¿Acepta oraciones que agramaticales?

Referencias

Gaifman, H. (1965). Dependency systems and phrase structure systems. *Information and Control*, (8):304–337.

Hays, D. G. (1964). Dependency theory: A formalism and some observations. *Language*, 40(4):511–525.

Nivre, J. y Nilsson, J. (2005). Pseudo-projective dependency parsing. En *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pp. 99–106, Ann Arbor, Michigan. Association for Computational Linguistics.

Tesnière, L. (1959). *Eléments de syntaxe structurale*. Klincksieck, Paris.