Parseador de dependencias para el castellano usando técnicas de "machine learning"

David Herrero Marco
LTH, Lund University, Sweden
CPS, Zaragoza University, Spain
davidherrerin@gmail.com

Resumen

Este artículo muestra los resultados obtenidos al realizar un parseador de dependencias gramaticales. La idea es crear un modelo que pueda ser utilizado en Prolog para parsear un texto y extraer dependencias gramaticales construyendo un árbol sintáctico de una oración. La medida de rendimiento será mejor, cuanto más se asemeje el árbol que extrae el modelo, al árbol solución. Los resultados obtenidos muestran una evolución según se van aplicando diferentes metodologías, en las que se ve claramente que los resultados se incrementan sustancialmente, hasta obtener una tasa de aciertos del 70%.

1 Créditos

Este documento es el resultado de la realización del proyecto de la asignatura "Procesamiento de lenguajes y lingüísticas computacionales" que ha sido tutorado por Pierre Nugues, en el LTH de Suecia. El trabajo parte de las premisas y de las ideas ya desarrolladas por Joakim Nivre en su parseador de sueco. Para la creación del modelo se ha utilizado la herramienta de minería de datos Weka, desarrollada en la misma Universidad de Lund. Y para el entrenamiento del modelo y su posterior validación se ha utilizado el corpus CliC-TALP del Centre de Llenguatge i Computació, España.

2 Introducción

El objetivo por tanto del proyecto es adaptar parte del parseador de Nivre para el castellano. Más concretamente, para el formato del Corpus CliC-TALP. Con este parseador, obtendríamos una lista de acciones, extraídas del corpus. Si pasamos todas estas acciones a una herramienta de minería de datos como Weka, deberíamos extraer estadísticamente un modelo con unos patrones y unas reglas que se suelen cumplir. Al aplicar este modelo sobre una oración posteriormente, extraemos su árbol sintáctico. Comparándolo con el árbol sintáctico que nos debería haber salido, se compara el rendimiento del modelo y variando diferentes parámetros obtenemos como resultados un 70% de acierto, dejando abiertos nuevos caminos para mejorar este dato. Un ejemplo de árbol de dependencias sería el siguiente:

Figura 1: Ejemplo de árbol de dependecias

La estructura que va a seguir este documento es la siguiente: Primero una breve explicación del parseador usado por Nivre. Seguido de una introduc-

ción del Corpus CliC-TALP y su formato. Luego se irán explicando los diferentes experimentos aplicados y los resultados obtenidos, para finalizar con algunas conclusiones y posibles vías para mejorar los resultados.

3 El parseador de Nivre

Joakim Nivre diseñó un parseador de dependencias para sueco y obtuvo los mejores sultados hasta la fecha.

3.1 Clasificación gramatical

Para empezar, hay que obtener un corpus, y para cada palabra del corpus clasificarla gramaticalmente. Se han utilizado principalmente las anotaciones morfológicas que venían ya con el corpus y podemos clasificarlos en dos grupos, detallados y no detallados. Así, por ejemplo, no es lo mismo decir que "casa" es un nombre (clasificación no detallada), que decir que "casa" es un nombre común (clasificación detallada). Una descripción más detallada de estas abreviaturas, viene definida en el apartado dedicado al Corpus.

3.2 Las estructuras de datos

Básicamente se van a utilizar dos estructuras de datos muy básicas:

- Una pila: que contiene los tokens parcialmente procesados.
- Una cola: que contiene los tokens de entrada que faltan.

3.3 Las acciones

Nivre aplica una variación de la técnica shift and reduce que implementa un parseador bottom-up (de abajo a arriba), ya que las gramáticas de dependencias no tienen símbolos no-terminales. El algoritmo construlle un grafo de dependencias en una única pasada de izquierda a derecha sobre los tokens de entrada (coste lineal) añadiendo arcos, usando 4 acciones elementales:

- Shift (sh), la cual introduce el siguiente token de la cola encima de la pila
- Reduce (re), que elimina la palabra de encima de la pila
- Left arc (la), que añade un arco desde el próximo token de entrada al token de encima de la pila, además de eliminarlo de la pila.
- Right arc (ra), que añade un arco desde el token de encima de la pila hasta el próximo token de entrada, además de eliminarlo de la pila.

Según Nivre, es posible determinar una secuencia de acciones correcta parseando cualquier oración ordenada de un corpus anotado de la forma que hemos visto anteriormente. De esta forma, obtenemos los árboles gramaticales de las oraciones.
Realmente, se podría decir que estamos definiendo un clasificador de acciones. El clasificador puede predecir cuál va a ser la siguiente acción según se cumplen ciertas restricciones. Un ejemplo de restricción sería, que un determinante tiene su "cabeza" sintáctica a continuación. En el sintagma "el coche", la palabra "el" tiene una flecha de dependencia con origen en "coche". Al llegar la palabra "el" la apilaríamos en la pila. Y al llegar la palabra "coche" se aplicaría la acción Left arc.

El siguiente paso por tanto, es entrenar este clasificador, de forma que funcione de la mejor forma posible. Para eso se va a usar Weka, como herramienta de minería de datos, para extraer un modelo desde un corpus anotado.

3.4 El Algoritmo

Inicialización: (nil, W, ϕ);
Terminación: (S, nil, A);

- **Left arc (la):** (n | S, n' | I, A) --> (S, n' | I, A ∪ {(n', n)}) LEX(n) <-- LEX(n') ∈ R, - ∃n''(n'',n) ∈ R

- **Right arc (ra):** (n | S, n' | I, A) --> (n', n |S, I, A ∪ {(n', n)}) LEX(n) --> LEX(n') ∈ R, -∃n''(n'',n') ∈ A
- **Reduce (re):** (n | S, I, A) --> (S, I, A) ∃ n' {n', n} ∈ R
- **Shift (sh):** (S,n | I, A) --> (n | S, I, A)

Para el algoritmo a continuación vamos a considerar TOP como lo más alto de la pila, FIRST el primer token de la cola de entrada y G el grafo de dependencia.

```
  if arc(TOP, FIRST)  ∈ G, then ra;
  else if arc(FIRST, TOP) ∈ G, then
la;
  else if  ∃k  ∈ Stack, (arc(FIRST,
k)  ∈ G, or arc(k, FIRST) ∈ G), then
re;
  else sh;
```

4 El Corpus

4.1 Descripción del Corpus

El corpus CLiC TALP consta de 1 millón de palabras y proviene de dos fuentes diferentes. Por una parte recoge una muestra representativa (de 500.000 palabras) de un corpus de prensa de 7 millones de palabras cedido por el periódico La Vanguardia. Por otra, recoge una muestra (también de 500.000 palabras) del corpus LexEsp (Léxico informatizado del español). Es representativo del español estándar escrito porque presenta varios estilos narrativos, procedentes de distintas fuentes (literatura, pensa, etc.) e incluye también muestras tanto del español peninsular como del de América. Recoge un número reducido de palabras por obra y no más de tres obras por autor.

Estilo	Porcentaje
Narrativa	40%
Divulgación científica	10%
Ensayo	10%
Prensa	25%
Semanarios	10%
Prensa deportiva	5%

Tabla 1: Fuentes del Corpus LexEsp

4.2 Abreviaturas usadas

El corpus viene incluído con una notación detallada de clasificación gramatical para cada palabra. Como ya se ha mencionado, hay dos tipos de notación que vamos a usar principalmente. La detallada y la no detallada.

- **No detallada:**

Abreviatura	Significado
a	Adjetivo
r	Adverbio
d	Determinante
n	Nombre
v	Verbo
i	Interjección
y	Abreviaturas
s	Adposición
p	Pronombres
z	Caracteres numéricos y matemáticos
w	Horas y fechas
c	Conjunción
f	Signos de puntuación
x	Elementos desconocidos

Tabla 2. Acrónimos no detallados.

- **Detallada:**

Abreviatura	Significado
aq	Adjetivo calificativo
ao	Adjetivo ordinal
rg	Adverbio general

rn	Adverbio negativo
dd	Determinante demostrativo
dp	Determinante posesivo
dt	Determinante interrogativo
de	Determinante exclamativo
di	Determinante indefinido
da	Determinante artículo
dn	Determinante numeral
nc	Nombre común
np	Nombre propio
vm	Verbo principal
va	Verbo auxiliar
vs	Verbo semiauxiliar
i	Interjección
y	Abreviaturas
sp	Preposición
pp	Pronombre personal
pd	Pronombre demostrativo
px	Pronombre posesivo
pt	Pronombre interrogativo
pr	Pronombre relativo
pi	Pronombre indefinido
pn	Pronombre numeral
pe	Pronombre exclamativo
zm	Cantidades monetarias
w	Horas y fechas
cc	Conjunción coordinada
cs	Conjunción subordinada
Fc	“,”
Fs	“...”
Fx	“;”
Fg	“-”
Fe	“ ' “, “ “ “, “ ` “
Fpt	“)”
Faa	“¡”
Fia	“¿”
Fat	“!”
Fla	“{“
Frc	“>>”
Fp	“.”
Fd	“:”
Ft	“%”
Fh	“/”
Fpa	“(“
Fit	“?”
Fca	“[“
Flt	“}”
Fct	“]”
Fra	“<<”
Fz	Otros caracteres de puntuación
x	Elementos desconocidos

Tabla 3. Acrónimos detallados.

4.3 Características añadidas

El corpus además de clasificar las palabras, viene con caractarísticas añadidas para cada palabra como podrían ser género y número para los nombres, o persona para los pronombres. Estas características son utilizadas sólamente en el último experimento a modo ilustrativo de hasta dónde llegan los límites del sistema que estamos usando. El corpus también tiene definidos los árboles gramaticales “solución” con los que podremos comparar nuestros resultados.

5 Los experimentos

A continuación se van a describir cinco experimentos que se han realizado. Todos han seguido el mismo patrón de funcionamiento.

Para cada oración que tenga un grafo de dependencias hay una secuencia de acciones que permiten al parseador de Nivre generar este grafo,
Esta secuencia de acciones puede ser entrenado con un corpus como el que tenemos, o más concretamente, la próxima acción puede ser deducida del contexto. Para poder predecir cuál va a ser la siguiente acción, debemos también extraer unos vectores característicos en cada paso del proceso de parseo.

5.1 Experimento 1

El contexto más obvio y más sencillo para empezar sería tener en cuenta lo que tenemos encima de la pila y cuál es la siguiente palabra a tratar. Así, con la última palabra o token tratado y con la siguiente que viene, se le asigna una acción de las mecionadas (la, ra, sh o re). Además, a la herramienta de minería de datos se le van a pasar 3 parametros más booleanos.

- can_do_leftarc, que indica si con lo que hay encima de la pila y la siguiente palabra, se puede aplicar la acción "la".
- can_do_rightarc, que indica si con lo que hay encima de la pila y la siguiente palabra, se puede aplicar la acción "ra".
- can_reduce, que indica si con lo que hay encima de la pila y la siguiente palabra, se puede aplicar la acción "re".

Esta información se almacena en un vector, por lo que a Weka, sólo queda indicarle el tamaño del vector, es decir: lo que hay en cada posición, los posibles valores que puede haber en cada posición y la lista de vectores extraída del corpus. Los valores que pueden tomar lo que hay en lo alto de la pila y la primera posición de la lista de palabras a tratar corresponden a los de la Tabla 2, de acrónimos no detallados. El fichero de extensión .arff queda algo así:

```
@relation parse_action

@attribute can_do_leftarc  {no, yes}
@attribute can_do_rightarc {no, yes}
@attribute can_reduce {no, yes}

@attribute first_stack_pos { NOTHING,
a, r, d, n, v, i, y, Y, s, p, Z, z,
w, c, F, x, X}

@attribute first_token_pos { a, r, d,
n, v, i, y, Y, s, p, Z, z, w, c, F,
x, X}

@attribute action { la, ra, re, sh }



@data

no, no, no, NOTHING, d, sh
--aquí seguirían todas las reglas
--generadas por el parseador...
```

Weka con esta información produce un clasificador de 4 clases (las 4 acciones que tenemos), usando técnicas de data mining. El resultado es un árbol de decisión. Por lo que el último paso es embeber este clasificador en el parseador para que lo llame cada vez que tiene que escoger una acción (la, ra, sh, re) para un contexto determinado. Una vez puesto a prueba este clasificador con un corpus de prueba y comprobándolo con los resultados finales, obtenemos un resultado del 60% de aciertos! Por tanto, de aquí en adelante, ésta va a ser la cifra a mejorar en los siguientes experimentos.

5.2 Experimento 2

Si en el primer experimento se ha probado con la notación sencilla, el siguiente paso inmediato es usar la notación especificada en la tabla 3, usando terminología más detallada. El fichero .arff, quedaría bastante similar, solo que first_stack_pos y first_token_pos pueden tener los valores de la notación detallada. A priori, se podría pensar que

vamos a conseguir mejores resultados, ya que cuanta más información dispongamos de los tokens que vamos a tratar, el clasificador es más preciso. De hecho, al hacer exactamente lo mismo que en el experimento 1, pero usando la información detallada de las palabras, se obtiene una mejora sustancial del resultado del 5%, es decir, una tasa de aciertos del 65%.

5.3 Experimento 3

El tercer experimento siguiendo la misma idea, de obtener más información acerca del contexto, consiste en no tener sólamente en cuenta la primera posición de la pila y el primer token de la lista de entrada, sino en tener en cuenta las dos primeras posiciones de la pila y los dos tokens siguientes de entrada. De esta forma, al incrementarse la información, también estamos complicando el clasificador, pero también deberían mejorar los resultados. El fichero .arff que se le pasa a Weka, quedaría algo así, con los nuevos parámetros:

```
@relation parse_action


@attribute can_do_leftarc {no, yes }
@attribute can_do_rightarc {no, yes }
@attribute can_reduce {no, yes }

@attribute first_stack_pos { NOTHING,
a, r, d, n, v, i, y, Y, s, p, Z, z,
w, c, F, x, X,END_OF_SENTENCE}

@attribute first_token_pos {NOTHING,
a, r, d, n, v, i, y, Y, s, p, Z, z,
w, c, F, x, X,END_OF_SENTENCE}

@attribute second_stack_pos { NOTH-
ING, a, r, d, n, v, i, y, Y, s, p, Z,
z, w, c, F, x, X,END_OF_SENTENCE}

@attribute second_token_pos { NOTH-
ING, a, r, d, n, v, i, y, Y, s, p, Z,
z, w, c, F, x, X,END_OF_SENTENCE}

@attribute action { la, ra, re, sh }

@data

no,no,no, NOTHING, NOTHING, a, r, sh
--aquí seguirían todas las reglas
--generadas por el parseador...
```

En este tercer experimento, se ha usado la notación sencilla de la tabla 2, y el resultado obtenido ha sido del 68%.

5.4 Experimento 4

El cuarto experimento, es idéntico al experimento 3, pero usando la notación detallada de la tabla 3, con lo cual los valores que pueden tener first_token_pos,first_stack_pos, second_token_pos y second_stack_pos se adaptan a ella. Así se obtiene el mejor resultado de todos, obteniendo un 70%.

5.5 Experimento 5

Viendo que se han conseguido mejorar los resultados usando una notación detallada tanto si usábamos sólo la primera posición de la pila y de la lista de entrada como las dos primeras posiciones, lo lógico sería pensar que cuanta más información tengamos sobre cada palabra para realizar el clasificador, mejor. El corpus como hemos dicho antes, para cada palabra da mucha más información como puede ser género, número, persona, tiempo... Entonces, el último experimento consiste en resumir en un acrónimo todas las cualidades de cada palabra. Por ejemplo, el determinante artículo masculino singular “el” quedaría en “damc”. El problema, es que se amplía la lista de acrónimos por encima de los 2000, y al hacer el clasificador con Weka, el árbol crece muchísimo y se vuelve bastante intratable. Para entrenar este clasificador, habría que entrenarlo con un corpus bastante más grande, por lo que los resultados obtenidos son de un 27% de aciertos.

6 Cómo seguir mejorando

Es probable, que incrementando algo la información de cada palabra, se puedan obtener algo mejores los resultados. Quizás sólamente añadiendo el género y el número el modelo mejore. Pero queda demostrado en el experimento 5, que introducir más características tiene un límite.

El siguiente paso por tanto es hacer lo que se ha hecho en el experimento 3 y 4, a diferencia del 1 y el 2. Añadir más parámetros. Quizás ampliando a 3 el número de tokens a tener en cuenta en la pila y en la lista de entrada, también se obtenga alguna mejora. Pero el límite, comparando con otros lenguajes estará por ahí, como mucho en 5.

El camino más esperanzador y que resultaría más interesante, sería añadir dós parámetros nuevos, "left most probably" y "right most probably" que marquen cuál es el token más probable que aparezca tanto a la izquierda como a la derecha del token analizado.

Otra forma de mejorar el resultado sería teniendo en cuenta las dependencias non-projective. En estos experimentos no se han tenido en cuenta porque no son muy relevantes en el castellano (apenas un 2%) e incrementaban la complejidad del sistema, pero es evidente que para alcanzar un buen resultado, habría que tenerlas en cuenta. Nivre, provee en su parseador soluciones para este problema.

Agradecimientos

Agradecer especialmente a Pierre Nugues, profesor de la asignatura y tutor en este proyecto el tiempo dedicado, la paciencia y el saber hacer demostrado durante el tiempo en que se ha desarrollado estas prácticas. También agradecer a Richard Johansson, el profesor de apoyo de la asignatura, que aunque de forma breve, ha influído también de alguna manera, al haber recibido alguna clase magistral y alguna práctica también de parte suya.

Referencias

Nugues, Pierre M. : *An Introduction to Language Processing with Perl and Prolog*. 2006, ISBN: 978-3-540-25031-9

Palomar, M., M. Civit, A. Díaz, L. Moreno, E. Bisbal, M. Aranzabe, A. Ageno, M.A. Martí y B. Navarro (2004) 3LB: *Construcción de una base de datos de árboles sintáctico-semánticos para el catalán, euskera y español.* XX Congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN), pp. 81-88. Barcelona, Julio de 2004 ISSN 1135-5948

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryigit, Svetoslav Marinov: *Labeled Pseudo-Projective Dependency Parsing*

Christopher D. Manning and Hinrich Schütze, *Foundations of Statistical Natural Language Processing,* MIT Press, 1999

Daniel Jurafsky, James H. Martin, Keith Vander Linden, Andrew Kehler, and Nigel Ward, *Speech and Language Processing*, Prentice Hall, 2000.