Dependencias, parsing

Aprendizaje basado en historia

GFLN

InCo

2016

Algoritmo basado en transiciones

Algoritmo basado en transiciones

- Algoritmo de tipo shift-reduce

Algoritmo basado en transiciones

- Algoritmo de tipo shift-reduce
- Utiliza un stack y una cola

Algoritmo basado en transiciones

- Algoritmo de tipo shift-reduce
- Utiliza un stack y una cola
- Completamente determinista, tiene una guía que le permite decidir la mejor transición

Algoritmo basado en transiciones

- Algoritmo de tipo shift-reduce
- Utiliza un stack y una cola
- Completamente determinista, tiene una guía que le permite decidir la mejor transición

Parse$(x = (w_1, ..., w_n))$
  $c \leftarrow ((\mathrm{R}oot), (1, ..., n), h_0, d_0)$
  while $c = (\rho, \tau, h, d)$ no terminal
    $c \leftarrow [\mathrm{ORACULO}(c, x)](c)$
  $\mathrm{G} \leftarrow (\mathrm{V}_x, \mathrm{E}_c, \mathrm{L}_c)$
return G

Algoritmo basado en transiciones

- Algoritmo de tipo shift-reduce
- Utiliza un stack y una cola
- Completamente determinista, tiene una guía que le permite decidir la mejor transición

Parse$(x = (w_1, ..., w_n))$

$c \leftarrow ((\mathrm{R}oot), (1, ..., n), h_0, d_0)$

while $c = (\rho, \tau, h, d)$ no terminal

$c \leftarrow [\mathrm{ORACULO}(c, x)](c)$

$\mathrm{G} \leftarrow (\mathrm{V}_x, \mathrm{E}_c, \mathrm{L}_c)$

return G

ORACULO(c,x) realiza una de 4 transiciones posibles, generando una nueva configuración.

Algoritmo, transiciones

Transición

Una transición es una función parcial $\mathrm{C}^n \rightarrow \mathrm{C}$, de alguno de los siguientes tipos:

1. STACK-HIJO(r):
   $(\rho|i,j|\tau,h,d) \rightarrow (\rho,j|\tau,h(i)=j,d(i)=r)$, si $h(i)=\mathrm{R}oot$
2. STACK-PADRE(r):
   $(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h(j)=i,d(j)=r))$, si $h(j)=\mathrm{R}oot$
3. REDUCE:
   $(\rho|i,\tau,h,d) \rightarrow (\rho,\tau,h,d)$ si $h(i)\neq \mathrm{R}oot$
4. SHIFT:
   $(\rho,i|\tau,h,d) \rightarrow (\rho|i,\tau,h,d)$

Algoritmo, transiciones

Transición

Una transición es una función parcial $\mathrm{C}^n \rightarrow \mathrm{C}$, de alguno de los siguientes tipos:

1. STACK-HIJO(r):
   $(\rho|i,j|\tau,h,d) \rightarrow (\rho,j|\tau,h(i)=j,d(i)=r)$, si $h(i)=\mathrm{R}oot$
2. STACK-PADRE(r):
   $(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h(j)=i,d(j)=r))$, si $h(j)=\mathrm{R}oot$
3. REDUCE:
   $(\rho|i,\tau,h,d) \rightarrow (\rho,\tau,h,d)$ si $h(i)\neq \mathrm{R}oot$
4. SHIFT:
   $(\rho,i|\tau,h,d) \rightarrow (\rho|i,\tau,h,d)$

Nota: En las configuraciones nuevas, las funciones h y d modifican el valor especificado, quedando igual en el resto de valores.

Algoritmo, transiciones

1. Dado un grafo, se obtiene la secuencia de transiciones que construye el grafo.
2. Para todo grafo proyectivo de dependencias existe una secuencia de transiciones que conduce a una configuración final.
3. El algoritmo es lineal en función del largo de la tira de entrada.

Aprendizaje

El objetivo es aprender el oráculo que guía el parsing.

Parse$(x = (w_1, ..., w_n))$

$c \leftarrow ((\mathrm{R}oot), (1, ..., n), h_0, d_0)$

while $c = (\rho, \tau, h, d)$ no terminal

$c \leftarrow [\mathrm{ORACULO}(c, x)](c)$

$\mathrm{G} \leftarrow (\mathrm{V}_x, \mathrm{E}_c, \mathrm{L}_c)$

return G

Aprendizaje

El objetivo es aprender el oráculo que guía el parsing.

Parse$(x = (w_1, ..., w_n))$

$c \leftarrow ((\mathrm{R}oot), (1, ..., n), h_0, d_0)$

while $c = (\rho, \tau, h, d)$ no terminal

$c \leftarrow [\mathrm{ORACULO}(c, x)](c)$

$\mathrm{G} \leftarrow (\mathrm{V}_x, \mathrm{E}_c, \mathrm{L}_c)$

return G

Se hace por aprendizaje supervisado, basado en un corpus anotado con árboles de dependencias.

Nivre propone un modelo de inferencia basado en historia.

Aprendizaje

- Se define un modelo de atributos (*features*) para enriquecer cada instancia de aprendizaje (cada configuración), que incluye palabras, POS-tags y dependencias de varios elementos del stack y de la cola de entrada.

Aprendizaje

- Se define un modelo de atributos (*features*) para enriquecer cada instancia de aprendizaje (cada configuración), que incluye palabras, POS-tags y dependencias de varios elementos del stack y de la cola de entrada.
- El conjunto de entrenamiento para el aprendizaje consiste en pares $(\mathrm{F}(c), t)$, en donde $\mathrm{F}(c)$ son los atributos correspondientes a una configuración c y t es la transición que se debe aplicar dada la configuración c.

Aprendizaje

- Se define un modelo de atributos (*features*) para enriquecer cada instancia de aprendizaje (cada configuración), que incluye palabras, POS-tags y dependencias de varios elementos del stack y de la cola de entrada.
- El conjunto de entrenamiento para el aprendizaje consiste en pares $(\mathrm{F}(c), t)$, en donde $\mathrm{F}(c)$ son los atributos correspondientes a una configuración c y t es la transición que se debe aplicar dada la configuración c.
- Estos datos se generan a partir de un treebank de árboles de dependencias (gold standard), reconstruyendo la secuencia de transiciones que genera cada árbol.

Aprendizaje

- Dado que la salida es discreta (una de las cuatro transiciones posibles), planteamos el problema directamente como un **problema de clasificación**.

Aprendizaje

- Dado que la salida es discreta (una de las cuatro transiciones posibles), planteamos el problema directamente como un **problema de clasificación**.
- Existen distintos métodos que se podrían usar:
  - SVM (se usa en algunas versiones de MaltParser)
  - redes neuronales
  - k-nearest neighbour(k-NN)

Aprendizaje

- Dado que la salida es discreta (una de las cuatro transiciones posibles), planteamos el problema directamente como un **problema de clasificación**.
- Existen distintos métodos que se podrían usar:
  - SVM (se usa en algunas versiones de MaltParser)
  - redes neuronales
  - k-nearest neighbour(k-NN)
- Se propone **memory-based learning (MBL)**, una variante de k-NN.
  Se le puede llamar también aprendizaje basado en ejemplos, o basado en casos.

k-NN para parsing basado en historia

La formulación es la siguiente:

- Inducción de un clasificador $\hat{g} : \mathrm{S} \rightarrow \mathrm{T}$ a partir de un conjunto de instancias de entrenamiento $\mathrm{D}_t = \{(s_1, t_1), ..., (s_n, t_n)\}$, donde $s_i \in \mathrm{S}$ es una instancia y t_i es su clase.

k-NN para parsing basado en historia

La formulación es la siguiente:

- Inducción de un clasificador $\hat{g} : \mathrm{S} \rightarrow \mathrm{T}$ a partir de un conjunto de instancias de entrenamiento $\mathrm{D}_t = \{(s_1, t_1), ..., (s_n, t_n)\}$, donde $s_i \in \mathrm{S}$ es una instancia y t_i es su clase.
- En nuestro caso, los s_i son los atributos asociados a la entrada y a la configuración.

k-NN para parsing basado en historia

La formulación es la siguiente:

- Inducción de un clasificador $\hat{g} : \mathrm{S} \rightarrow \mathrm{T}$ a partir de un conjunto de instancias de entrenamiento $\mathrm{D}_t = \{(s_1, t_1), ..., (s_n, t_n)\}$, donde $s_i \in \mathrm{S}$ es una instancia y t_i es su clase.
- En nuestro caso, los s_i son los atributos asociados a la entrada y a la configuración.
- El valor de la clase es la mejor transición en la situación dada.

k-NN para parsing basado en historia

La formulación es la siguiente:

- Inducción de un clasificador $\hat{g} : \mathrm{S} \rightarrow \mathrm{T}$ a partir de un conjunto de instancias de entrenamiento $\mathrm{D}_t = \{(s_1, t_1), ..., (s_n, t_n)\}$, donde $s_i \in \mathrm{S}$ es una instancia y t_i es su clase.
- En nuestro caso, los s_i son los atributos asociados a la entrada y a la configuración.
- El valor de la clase es la mejor transición en la situación dada.
- Aprender es almacenar el conjunto D_t en memoria (no hay parámetros, como en los algoritmos clásicos de aprendizaje).

k-NN para parsing basado en historia

La formulación es la siguiente:

- Inducción de un clasificador $\hat{g} : \mathrm{S} \rightarrow \mathrm{T}$ a partir de un conjunto de instancias de entrenamiento $\mathrm{D}_t = \{(s_1, t_1), ..., (s_n, t_n)\}$, donde $s_i \in \mathrm{S}$ es una instancia y t_i es su clase.
- En nuestro caso, los s_i son los atributos asociados a la entrada y a la configuración.
- El valor de la clase es la mejor transición en la situación dada.
- Aprender es almacenar el conjunto D_t en memoria (no hay parámetros, como en los algoritmos clásicos de aprendizaje).
- La clasificación de una nueva instancia r se realiza en dos pasos:
  - Se compara r (sus atributos) con todas las instancias almacenadas s_i, mediante una función de distancia $\Delta(r, s_i)$
  - Se toma la clase mayoritaria entre los k más cercanos como la clase de r

k-NN para parsing basado en historia

La formulación es la siguiente:

- Inducción de un clasificador $\hat{g} : \mathrm{S} \rightarrow \mathrm{T}$ a partir de un conjunto de instancias de entrenamiento $\mathrm{D}_t = \{(s_1, t_1), ..., (s_n, t_n)\}$, donde $s_i \in \mathrm{S}$ es una instancia y t_i es su clase.
- En nuestro caso, los s_i son los atributos asociados a la entrada y a la configuración.
- El valor de la clase es la mejor transición en la situación dada.
- Aprender es almacenar el conjunto D_t en memoria (no hay parámetros, como en los algoritmos clásicos de aprendizaje).
- La clasificación de una nueva instancia r se realiza en dos pasos:
  - Se compara r (sus atributos) con todas las instancias almacenadas s_i, mediante una función de distancia $\Delta(r, s_i)$
  - Se toma la clase mayoritaria entre los k más cercanos como la clase de r
- Se utilizó el paquete TiMBL

Elementos variables a definir en MBL

1. Cuál es el k?
2. Qué función de distancia Δ se usa?
3. Cómo se resuelve un empate entre 2 valores?

MBL, k utilizado

En los experimentos reportados en (Nivre, 2005) se utiliza siempre $k = 5$.

O sea, se compara con las 5 instancias más próximas.

MBL, k utilizado

En los experimentos reportados en (Nivre, 2005) se utiliza siempre $k = 5$.

O sea, se compara con las 5 instancias más próximas.

Valores chicos de k manifiestan mejores particularidades locales, y valores grandes una mejor decisión global.

MBL, k utilizado

En los experimentos reportados en (Nivre, 2005) se utiliza siempre $k = 5$.

O sea, se compara con las 5 instancias más próximas.

Valores chicos de k manifiestan mejores particularidades locales, y valores grandes una mejor decisión global.

El valor de k influye en la eficiencia del algoritmo.

MBL, función de distancia Δ

Un modo simple de comparar 2 instancias es contar cuántos atributos coinciden.
Esta métrica ha sido llamada *métrica de Overlap, distancia de Hamming, métrica Manhattan*
La distancia entre 2 instancias $r = (r_1, ..., r_n)$ y $s = (s_1, ..., s_n)$ es simplemente:

MBL, función de distancia Δ

Un modo simple de comparar 2 instancias es contar cuántos atributos coinciden.
Esta métrica ha sido llamada *métrica de Overlap, distancia de Hamming, métrica Manhattan*
La distancia entre 2 instancias $r = (r_1, ..., r_n)$ y $s = (s_1, ..., s_n)$ es simplemente:

$$\Delta(r, s) = \sum_{i=1}^{n} \delta(r_i, s_i) \text{ con } \delta(i, j) = 1 \text{ si } i \neq j, \ 0, \text{ si no}$$

MBL, función de distancia Δ, ponderación de atributos

Variantes de esta medida consideran pesos para ponderar de modo distinto la coincidencia en distintos atributos.
Ponderación de atributos:

$$\Delta(r, s) = \sum_{i=1}^{n} w_i \delta(r_i, s_i)$$

MBL, resolución de empate

En TiMBL se usa el siguiente método:

1. Incrementar k en 1 y tomar la clase mayoritaria, si existe.
2. Si no existe, tomar la clase mayoritaria en todo el conjunto de entrenamiento.
3. Si no existe, tomar la clase t_1 de la primera instancia (s_1, t_1) del conjunto de entrenamiento.

Atributos

Se utilizan como atributos

- palabras
- POS-tags
- relación de dependencia con el padre

de algunos elementos del stack y de la cola de entrada.

La elección de los atributos es configurable.

Atributos

Conjunto de atributos utilizado en Nivre, 2004:

TOP	Token tope del stack
TOP.POS	Part-of-speech de TOP
TOP.DEP	Etiqueta de dependencia de TOP
TOP.LEFT	Etiqueta de dependencia del hijo izquierdo de TOP
TOP.RIGHT	Etiqueta de dependencia del hijo derecho de TOP
NEXT	Próximo token
NEXT.POS	Part-of-speech de NEXT
NEXT.LEFT	Etiqueta de dependencia del hijo izquierdo de NEXT
LOOK.POS	Part-of-speech del siguiente de NEXT

Atributos

Conjunto de atributos utilizado en Nivre 2007:

- Se incluyen los dos primeros elementos del stack (no solo TOP).
- Se incluyen los cuatro primeros elementos de la cola de entrada (no solo NEXT y LOOK).
- Se incluyen las mismas dependencias que en 2004.

Medidas de evaluación

- Por token:
  - UAS Unlabeled Accuracy Score
    Porcentaje de tokens con head bien.
  - LAS Labeled Accuracy Score
    Porcentaje de tokens con head y etiqueta bien.
  - LA Label Accuracy
    Porcentaje de tokens con etiqueta bien.
- Por oración:
  - LCM Labeled Complete Match
    Porcentaje de oraciones con grafo etiquetado correcto.
  - UCM Unlabeled Complete Match
    Porcentaje de oraciones con grafo correcto, sin considerar etiquetas.

Medidas de evaluación

- Por token:
  - UAS Unlabeled Accuracy Score, 93%
    Porcentaje de tokens con head bien.
  - LAS Labeled Accuracy Score, 88%
    Porcentaje de tokens con head y etiqueta bien.
  - LA Label Accuracy
    Porcentaje de tokens con etiqueta bien.
- Por oración:
  - LCM Labeled Complete Match
    Porcentaje de oraciones con grafo etiquetado correcto.
  - UCM Unlabeled Complete Match, 47%
    Porcentaje de oraciones con grafo correcto, sin considerar etiquetas.

MALTPARSER

- Parser que implementa los algoritmos vistos.
- Estructura modular, permite cambiar fácilmente:
  - El método de aprendizaje (por otro del mismo estilo)
  - Los atributos para el aprendizaje

MALTPARSER

- Parser que implementa los algoritmos vistos.
- Estructura modular, permite cambiar fácilmente:
  - El método de aprendizaje (por otro del mismo estilo)
  - Los atributos para el aprendizaje
- Existen implementaciones para más de 10 idiomas.

MALTPARSER

- Parser que implementa los algoritmos vistos.
- Estructura modular, permite cambiar fácilmente:
  - El método de aprendizaje (por otro del mismo estilo)
  - Los atributos para el aprendizaje
- Existen implementaciones para más de 10 idiomas.
- UAS casi 90%, LAS casi 88%, en el mejor caso.

MALTPARSER

- Parser que implementa los algoritmos vistos.
- Estructura modular, permite cambiar fácilmente:
  - El método de aprendizaje (por otro del mismo estilo)
  - Los atributos para el aprendizaje
- Existen implementaciones para más de 10 idiomas.
- UAS casi 90%, LAS casi 88%, en el mejor caso.
- Extremadamente rápido.

Referencias

- Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S. and Marsi, E. (2007) MaltParser: A language-independent system for data-driven dependency parsing. Natural Language Engineering, 13(2), 95-135
- Nivre, J. (2008) Algorithms for Deterministic Incremental Dependency Parsing. Computational Linguistics 34(4), 513-553
- Ballesteros, M. and Nivre, J. (2013) Going to the Roots of Dependency Parsing. Computational Linguistics 39(1): 5-13