Capítulo 6

Inferencia de lenguajes racionales de árboles

Bajo la bóveda verde de los árboles gigantes, un obstáculo: la raíz de uno de ellos cerrándonos el paso.
Pablo Neruda, Para nacer he nacido.

En este capítulo se aborda el problema del aprendizaje de gramáticas independientes del contexto a partir de datos estructurales estocásticos. Para ello, se desarrolla el algoritmo `tlips` que identifica cualquier conjunto racional de árboles a partir de muestras aleatorias y evalúa la distribución de probabilidad de los árboles del lenguaje. El procedimiento seguido se basa en la identificación de subárboles equivalentes en la muestra y requiere un tiempo que crece linealmente con el número de ejemplos.

6.1 Antecedentes

El aprendizaje de lenguajes independientes del contexto es más difícil que el de los lenguajes regulares, aunque se han producido algunos

avances recientemente. Por ejemplo, Sakakibara (1992) ha abordado un problema relacionado con el anterior, el aprendizaje de gramáticas independientes del contexto a partir de muestras positivas con descripciones estructurales, esto es, muestras formadas por frases que contienen información sobre la forma en que se han obtenido a partir de la gramática (esencialmente, el esqueleto del árbol de derivación). En la práctica, encontramos descripciones estructurales cuando las cadenas se presentan en forma parentizada: por ejemplo, expresiones aritméticas totalmente parentizadas. Todo conjunto de descripciones estructurales es un lenguaje de árboles racional, es decir, un lenguaje que puede ser reconocido por un autómata finito de árboles. Por ello, la identificación de un lenguaje independiente del contexto queda reducido, cuando los ejemplos contienen descripciones estructurales, a un problema de identificación de lenguajes racionales de árboles.

En el trabajo de Sakakibara (1992), se demuestra que la subclase de lenguajes de *árboles reversibles* se puede aprender en tiempo polinómico a partir de muestras positivas. Los lenguajes reversibles de árboles son la extensión natural de los lenguajes regulares reversibles estudiados por Angluin (1982) y forman un subconjunto propio de la clase de lenguajes racionales, a pesar de que la condición de reversibilidad puede ser considerada como una normalización de las gramáticas independientes del contexto. Es decir, toda gramática independiente del contexto puede reescribirse como gramática reversible —de forma que la nueva gramática genera exactamente el mismo lenguaje. Sin embargo, no hay ningún motivo para suponer que un conjunto de descripciones estructurales ha sido generado mediante una gramática reversible. Si este no es el caso, el procedimiento no identificará correctamente la gramática. De hecho, la clase de los lenguajes racionales de árboles (al igual que la clase de los lenguajes regulares) no es identificable en el límite a partir de muestras positivas.

Un algoritmo más general ha sido propuesto por Oncina y García (1994). Su algoritmo:

- identifica en el límite cualquier lenguaje racional de árboles;
- presenta el resultado en tiempo polinómico con el tamaño del conjunto de muestra;
- utiliza ejemplos y contraejemplos durante el período de aprendizaje.

Si bien las dos primeras características son deseables, la última limita el rango de aplicaciones posible, debido a las dificultades de obtener muestras completas auténticamente representativas del lenguaje (sobre todo desde el punto de vista de los contraejemplos). Por este motivo, en este capítulo se presenta un algoritmo que puede ser entrenado con muestras positivas aleatorias generadas según un esquema probabilístico. Una vez que el lenguaje racional de árboles de derivación ha sido identificado, existe siempre una gramática determinista hacia atrás (Aho y Ullman, 1972) equivalente que genera el mismo lenguaje racional de árboles. Las probabilidades de generación asociadas a este gramática pueden ser calculadas aproximadamente a partir de la muestra, y la precisión puede ser aumentada si se utilizan muestras más grandes.

La notación que utilizaremos es muy semejante a la de Sakakibara (1992) y se presenta en la sección 6.2. El algoritmo es descrito en la sección 6.3. Su versión probabilística se introduce en la sección 6.4 y un ejemplo de aplicación se estudia en la sección 6.5.

6.2 Formalismo

Sea $\mathbb{N}$ el conjunto de los números naturales, $\mathbb{N}^*$ el monoide libre generado por $\mathbb{N}$ mediante la operación "." con λ como elemento neutro. Es posible definir la *longitud* de la cadena $x \in \mathbb{N}^*$ como el número de naturales en x, pero de forma más rigurosa se utiliza una definición recursiva:

$$\begin{aligned} |\lambda| &= 0 \\ |x.i| &= |x| + 1 \ (\forall x \in \mathbb{N}^*, i \in \mathbb{N}) \ . \end{aligned} \tag{6.1}$$

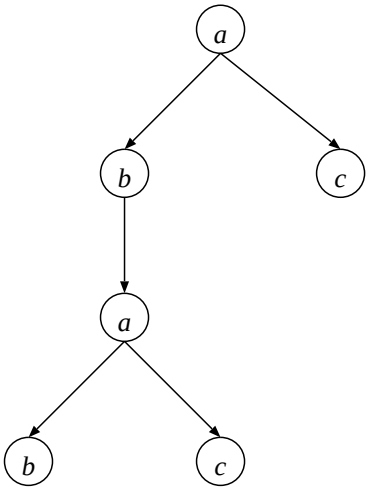

Figura 6.1: Un ejemplo de árbol

Para definir un árbol vamos a separar su estructura (dominio) de las etiquetas de los nodos de la siguiente forma. Llamaremos *dominio de árbol* a cualquier subconjunto $D \subseteq \mathbb{N}^*$ que satisface para cualquier $x, y \in \mathbb{N}^*$ y para todos los $i, j \in \mathbb{N}$

$$\begin{array}{rcl} x.y \in D & \Rightarrow & x \in D \\ x.i \in D \wedge j \leq i & \Rightarrow & x.j \in D \end{array} \tag{6.2}$$

La primera condición garantiza que todos los antecesores de un *nodo* (esto es, de un elemento del dominio) están también en el dominio y la segunda que los descendientes de un nodo están numerados correlativamente. Por la propia definición, es evidente que λ siempre pertenece al dominio, y se le denomina *raíz* del árbol. Además, la numeración establece un orden entre los descendientes de cada nodo, por lo que los árboles que se definan a partir del dominio serán árboles ordenados. En el ejemplo de la figura 6.1, el dominio es $D = \{\lambda, 0, 1, 0.0, 0.0.0, 0.0.1\}$.

Llamaremos *rango* de un nodo x al número $\rho(x)$ de descendientes

del nodo, es decir,

$$\rho(x) = \min\{i \in \mathbb{N} : x.i \notin \mathrm{dom}(t)\}. \tag{6.3}$$

Al rango máximo de los nodos de un árbol se le llama *anchura* del árbol.

Las etiquetas de los árboles serán símbolos de un alfabeto V. A veces, es conveniente considerar que el número posible de descendientes del nodo es una característica de la etiqueta. En lo que sigue, salvo que se especifique lo contrario, todas las etiquetas puede aparecer en nodos de rango arbitrario.

Con todo esto, pasamos a definir la estructura de árbol. Un *árbol finito* sobre un alfabeto con rango V es una función $t : \mathrm{dom}(t) \to V$ en la que $\mathrm{dom}(t)$ es un dominio de árbol finito. En nuestro ejemplo, la función t es $t(\lambda) = a$, $t(0) = b$, $t(1) = c$, $t(0.0) = a$, $t(0.0.0) = b$ y $t(0.0.1) = c$, mientras que $V = \{a, b, c\}$.

La *profundidad* del árbol el la longitud de su rama más larga, o de forma más rigurosa la longitud máxima de las cadenas del dominio:

$$\mathrm{depth}(t) = \max\{|x| : x \in \mathrm{dom}(t)\}. \tag{6.4}$$

En el ejemplo anterior, la profundidad del árbol es 3. La raíz del árbol es el nodo λ y las *hojas* del árbol son las cadenas del dominio de longitud máxima, es decir, $x \in D$ tales que no existe $x.y$ en el dominio con $y \neq \lambda$.

Representaremos como V^T al conjunto de árboles construibles sobre el alfabeto V. Cada nodo de un árbol puede entenderse como una función de nombre igual a su etiqueta que toma como argumentos a cada uno de los descendientes. Por ejemplo, el árbol de la figura 6.1 puede representarse como $a(b(a(bc))c)$. En lo que sigue, escribiremos los árboles en esta notación funcional $t = f(t_1, ..., t_k)$, siendo f un símbolo de V. Merece la pena destacar que V es isomorfo al subconjunto de árboles de V^T cuyo dominio es λ o, dicho de otra forma, el subconjunto de árboles de profundidad cero. Escribiremos, por tanto, $V \subset V^T$.

Pasamos a continuación a definir una máquina de estados finitos que puede operar sobre árboles. Un *autómata determinista de árboles* (DTA) se define como un cuarteto $A = (Q, V, \delta, F)$, formado por:

- un conjunto finito de estados Q;
- un alfabeto finito V;
- un subconjunto de estados $F \subset Q$ llamados estados de aceptación;
- un conjunto de funciones de transición $\delta = \{\delta_0, \ldots, \delta_n\}$.

Las funciones de transición operan sobre nodos de rango k de la siguiente forma

$$\delta_k : V \times Q^k \to Q. \tag{6.5}$$

y permiten definir la actuación del El DTA sobre árboles de la siguiente manera:

$$\delta(t) = \begin{cases} \delta_k(f, \delta(t_1), \ldots, \delta(t_k)) & \text{si} \quad t = f(t_1, \ldots, t_k) \in (V^T - V) \\ \delta_0(a) & \text{si} \quad t = a \in V \end{cases} \tag{6.6}$$

Cada DTA define un *lenguaje racional de árboles* (RTL) de la siguiente manera:

$$L(A) = \{t \in V^T : \delta(t) \in F\} \tag{6.7}$$

Debe destacarse que, a diferencia de los autómatas finitos que operan sobre cadenas de izquierda a derecha y procesan un símbolo en cada paso, aquí hay un estado asociado a cada nodo del árbol que se obtiene en función de la etiqueta del nodo y de los estados de los descendientes. Por ello, es necesario comenzar por las hojas y el estado de cada subárbol es utilizado por el DTA para evaluar el resultado en el antecesor hasta llegar a la raíz. Si el resultado de operar sobre el árbol al llegar a la raíz es un estado de aceptación, el árbol pertenece al lenguaje. Por convención $t \notin L(A)$ si $\delta(t)$ está indefinido. Por tanto, ningún árbol de anchura mayor que n es aceptado por el autómata.

Supongamos que tenemos una gramática independiente del contexto $G = (N, \Sigma, P, S)$ sin *producciones vacías*, es decir, sin reglas de producción en P del tipo $A \to \lambda$. En dicha gramática, cada proceso de producción de una cadena de terminales tiene asociado un árbol de derivación. Podemos definir (recursivamente) el conjunto de árboles enraizados en $X \in (N \cup \Sigma)$ de la siguiente forma:

$$D_X(G) = \begin{cases} \{X\} & \text{si } X \in \Sigma \\ \{X(t_1, ..., t_k) : & X \to X_1 ... X_k \in P \wedge \\ & \wedge\, t_i \in D_{X_i}(G), 1 \leq i \leq k\} \quad \text{si } X \in N \end{cases} \tag{6.8}$$

En particular, los árboles $D_S(G)$ reciben el nombre de *árboles de derivación* de G.

El esqueleto de un árbol $t \in V^T$, puede entenderse como ese mismo árbol pero despojado de las etiquetas en todos sus nodos interiores (aquellos que no son hojas). Una forma rigurosa de definir el esqueleto es:

$$\text{sk}(t) = \begin{cases} a & \text{si} \quad t = a \in V \\ \sigma(\text{sk}(t_1), ..., \text{sk}(t_k)) & \text{si} \quad t = f(t_1, ..., t_k) \in (V^T - V) \end{cases} \tag{6.9}$$

donde σ es un símbolo especial que no pertenece a V y que admite cualquier rango excepto el cero. De esta forma, los nodos interiores del esqueleto están etiquetados con σ, mientras que las hojas contienen símbolos de Σ.

La gramática $G = (N, \Sigma, P, S)$ es determinista hacia atrás si por cada cadena w de $(N \cup \Sigma)^*$ existe como máximo una producción en P tal que su parte derecha coincida con w, esto es, existe como máximo una variable $A \in N$ que produce $A \to w$. Para este tipo de gramáticas, tanto el conjunto de árboles de derivación $D_S(G)$ como el de sus esqueletos $\text{sk}(D_S(G))$ son conjuntos racionales de árboles. Por ejemplo,

se puede definir el aceptor $A = (Q, V, \delta, F)$, como:

$$\begin{array}{rcll} Q & = & N \cup \Sigma & \\ V & = & \{\sigma\} \cup \Sigma & \\ \delta_0(a) & = & a & \forall a \in \Sigma \\ \delta_k(\sigma, X_1, ..., X_k) & = & A & \text{si } A \to X_1...X_k \in P \end{array} \tag{6.10}$$

que reconoce el conjunto $\mathrm{sk}(D_S(G))$. Nótese que sólo si la gramática es determinista hacia atrás la función δ puede ser definida de forma determinista. De forma análoga, puede definirse un DTA que reconoce $D_S(G)$, aunque en este caso no es preciso que la gramática sea determinista hacia atrás.

A continuación vamos a definir los lenguajes estocásticos de árboles y otros elementos probabilísticos que utilizaremos en su estudio. Un *lenguaje de árboles estocástico* T se define mediante un a distribución de probabilidad sobre V^T, que denotaremos como $p(t|T)$. A su vez, la probabilidad de un subconjunto de árboles $S \subset V^T$ viene dada por

$$p(S|T) = \sum_{t \in S} p(t|T). \tag{6.11}$$

Es importante destacar que la *identidad* de lenguajes estocásticos sólo de produce si todas las probabilidades asignadas coinciden:

$$T_1 = T_2 \Leftrightarrow p(t|T_1) = p(t|T_2) \ \ \forall t \in V^T. \tag{6.12}$$

La distribución de probabilidad para los árboles puede generarse mediante un autómata de un tipo especial de la siguiente forma. Se define un *autómata estocástico determinista de árboles* $A = (Q, V, \delta, p, r)$, como:

- un conjunto finito de estados Q;
- un alfabeto finito V;
- un conjunto de funciones de transición $\delta = \{\delta_0, ..., \delta_n\}$, análogas a las de un DTA;

- una función $r : Q \to [0,1]$ que asigna a cada estado la probabilidad de que aparezca asociado a la raíz de un árbol del lenguaje;

- un conjunto de distribuciones de probabilidad $p = \{p_0, \ldots, p_n\}$ del tipo $p_k : V_k \times Q^k \to [0,1]$, que asignan una probabilidad a cada transición del autómata.

Las probabilidades $r(q)$ deben satisfacer la normalización:

$$\sum_{q \in Q} r(q) = 1 \tag{6.13}$$

mientras que las probabilidades p_k satisfacen para todos los estados $q \in Q$

$$\sum_{k=1}^{n} \sum_{f \in V} \sum_{\substack{q_1, \ldots q_k \in Q : \\ \delta(f, q_1, q_2, \ldots, q_k) = q}} p_k(f, q_1, \ldots, q_k) = 1 \tag{6.14}$$

El autómata estocástico define un lenguaje estocástico de árboles dado por:

$$p(t|A) = r(\delta(t)) p(t|\delta(t)) \tag{6.15}$$

donde la probabilidad de generación del subárbol $t = f(t_1, .., t_k)$ a partir del estado $\delta(t)$ es

$$p(t|\delta(t)) = p_k(f, t_1, \ldots, t_k)\ p(t_1|\delta(t_1)) \cdots p(t_k|\delta(t_k))\ . \tag{6.16}$$

Es preciso introducir un nuevo símbolo $\$ \notin V$ que sólo puede etiquetar nodos cuyo rango es cero. Este nuevo símbolo permite construir $V_{\$}^T$, el conjunto de árboles de $(V \cup \{\$\})^T$ que contienen exactamente un símbolo \$. Dado un árbol $s \in V_{\$}^T$, y otro $t \in V^T$, se define la *sustitución* del \$, $s\#t$ como

$$s\#t(x) = \begin{cases} s(x) & \text{si } x \in \mathrm{dom}(s) \wedge s(x) \neq \$ \\ t(z) & \text{si } x = y.z \wedge s(y) = \$ \end{cases} \tag{6.17}$$

y $x \notin \mathrm{dom}(s\#t)$ en cualquier otro caso.

Para cada lenguaje de árboles estocástico T y para cada $t \in V^T$, el *cociente* $t^{-1}T$ es un nuevo lenguaje estocástico sobre $V_\$^T$ definido por las probabilidades

$$p(s|t^{-1}T) = \frac{p(s\#t|T)}{p(V_\$^T\#t|T)} \ . \tag{6.18}$$

En caso de que s no sea un árbol de $V_\$^T$, entonces $p(s|t^{-1}T) = 0$. Además, si $p(V_\$^T\#t|T) = 0$, el cociente (6.18) no está definido. En ese caso, escribiremos por convención $t^{-1}T = \emptyset$, y todas las probabilidades del tipo $p(s|t^{-1}T)$ son nulas.

De forma análoga a como se hizo en la sección 5.2, el teorema de Myhill y Nerode para lenguajes racionales puede ser generalizado para lenguajes racionales de árboles estocásticos. Si T es un RTL estocástico, el número de conjuntos $t^{-1}T$ distintos es finito y se puede definir el *generador canónico* $M = (Q^M, V, \delta^M, F^M)$ de la siguiente forma:

$$\begin{aligned} Q^M &= \{t^{-1}T \neq \emptyset : t \in V^T\} \\ \delta_k^M(f, t_1^{-1}T, \ldots, t_k^{-1}T) &= f(t_1, \ldots, t_k)^{-1}T \\ r^M(t^{-1}T &= p(\Delta_t|T) \\ p_k^M(f, t_1^{-1}T, ..., t_k^{-1}T) &= \frac{p(V_\$^T\#t)}{p(V_\$^T\#\Delta_t)} \end{aligned} \tag{6.19}$$

siendo $\Delta_t = \{s \in V^T : \delta^M(s) = t^{-1}T)$.

Una muestra estocástica S del lenguaje T es una secuencia infinita de árboles que ha sido generada de acuerdo con la distribución de probabilidad $p(t|T)$. Los n primeros árboles de esta secuencia forman forman la subsecuencia S_n. Esta subsecuencia S_n contiene árboles repetidos y $c_n(t)$ representa el número de veces que aparece el árbol t en S_n. A su vez, para un subconjunto $X \subset V^T$,

$$c_n(X) = \sum_{t \in X} c_n(t) \ . \tag{6.20}$$

Si se conoce la estructura de M, es decir, los estados que lo componen y sus respectivas funciones de transición, podemos calcular las

funciones de probabilidad del DTA estocástico a partir de los ejemplos contenidos en S_n

$$r(t^{-1}T) \simeq \frac{c_n(\Delta_t)}{n} \tag{6.21}$$

$$p_k(f, t_1^{-1}T, \ldots, t_k^{-1}T) \simeq \frac{c_n(V_\$^T \# t)}{c_n(V_\$^T \# \Delta_t)} \, . \tag{6.22}$$

La precisión de estos valores estimados mejora según es mayor el valor de n. En la sección siguiente presentamos un algoritmo que identifica la estructura del generador canónico M.

6.3 Algoritmo de inferencia

En lo que sigue utilizaremos una relación de orden total arbitraria definida en el conjunto de árboles V^T. Cualquier relación es válida, siempre y cuando los árboles de menor profundidad precedan a los de profundidad mayor, es decir,

$$t_1 \leq t_2 \Leftrightarrow \mathrm{depth}(t_1) \leq \mathrm{depth}(t_2). \tag{6.23}$$

Como siempre, si $t_1 \leq t_2$ y $t_1 \neq t_2$, entonces escribiremos $t_1 < t_2$.

Para identificar el generador canónico definimos los siguientes conjuntos (no estocásticos):

- El *conjunto de subárboles* de T son los subárboles t con probabilidad no nula de aparición en T,

$$\mathrm{Sub}(T) = \{t \in V^T : t^{-1}T \neq \emptyset\} \, . \tag{6.24}$$

- Los *subárboles superficiales*

$$\mathrm{SSub}(T) = \{t \in \mathrm{Sub}(T) : s^{-1}T = t^{-1}T \Rightarrow s \geq t\} \tag{6.25}$$

son los menores subárboles entre los que generan el mismo lenguaje cociente.

- Los subárboles superficiales más los que se pueden crear directamente a partir de ellos forman el *kernel*:

$$K(T) = \{f(t_1, ..., t_k) \in \mathrm{Sub}(T) : t_1, ..., t_k \in \mathrm{SSub}(T)\} \ . \quad (6.26)$$

- Aquellos subárboles añadidos por el kernel componen la *frontera*

$$F(T) = K(T) - \mathrm{SSub}(T) \ . \quad (6.27)$$

Debe observarse que cada árbol contenido en $\mathrm{SSub}(T)$ es representante de un estado $t^{-1}T$ del generador canónico M definido en (6.19). Esto nos permite construir las funciones de transición de M en la forma $\delta_k(f, t_1, t_2, ..., t_k) = f(t_1, t_2, .., t_k)$ siempre que este último sea un árbol de $\mathrm{SSub}(T)$. En algunos casos, $f(t_1, t_2, ..., t_k)$ es un árbol de $F(T)$ y la transición deberá definirse de otra forma, tal y como veremos inmediatamente. Nótese que tanto $\mathrm{SSub}(T)$ como $K(T)$ son conjuntos finitos.

Vamos a utilizar la siguiente función lógica $\mathtt{equiv_T} : K(T) \times K(T) \to \{\mathrm{TRUE}, \mathrm{FALSE}\}$, cuyo resultado viene dado por

$$\mathtt{equiv_T}(t_1, t_2) = \mathrm{TRUE} \Leftrightarrow t_1^{-1}T = t_2^{-1}T. \quad (6.28)$$

El lema siguiente se obtiene inmediatamente:

Lema 4 *Dados* $\mathrm{SSub}(T)$, $F(T)$ *y* $\mathtt{equiv_T}$, *la estructura del aceptor canónico* M *es isomorfa a:*

$$\begin{aligned} Q &= \mathrm{SSub}(T) \\ \delta_k(f, t_1, ..., t_k) &= t \end{aligned} \quad (6.29)$$

donde t *es el único árbol en* $\mathrm{SSub}(T)$ *tal que* $\mathtt{equiv_T}(t, f(t_1, ..., t_k))$

Demostraci'on. Sea $\Phi : Q \to Q^M$ la función definida como $\Phi(t) = t^{-1}T$. La función Φ es un isomorfismo si

$$\Phi\delta_k(f, t_1, t_2, ..., t_k) = \delta_k^M(f, \Phi(t_1), \Phi(t_2), ..., \Phi(t_k)) \ ,$$

esto es,

$$\delta_k(f, t_1, t_2, ..., t_k)^{-1}T = f(t_1^{-1}T, t_2^{-1}T, ..., t_k^{-1}T)^{-1}T \ .$$

En otras palabras, $\delta_k(f, t_1, t_2, ..., t_k)$ es un elemento $t \in \mathrm{SSub}(T)$ que satisface $\mathtt{equiv}_\mathtt{T}(t, f(t_1, ..., t_k))$. De acuerdo con la definición (6.25), este elemento es único. ■

El siguiente teorema es la base del algoritmo de inferencia.

Teorema 5 *El algoritmo de la figura 6.2 encuentra la estructura del generador canónico de cualquier lenguaje racional de árboles estocástico T, además de* $\mathrm{SSub}(T)$ *y* $F(T)$*, si dispone como entrada de la función* $\mathtt{equiv}_\mathtt{T}$ *más cualquier subconjunto* $A \subset \mathrm{Sub}(T)$ *que incluya al kernel* $K(T)$*.*

Demostraci'on. Por inducción en el número de iteraciones i, se observa trivialmente que los resultados después de i iteraciones satisfacen $\mathrm{SSub}^{[i]} \subset \mathrm{SSub}(T)$, $F^{[i]} \subset F(T)$ y $W^{[i]} \subset K(T)$. Por otro lado, si $t \in K(T)$ entonces $t \in A$ y otro razonamiento de inducción en la profundidad del árbol demuestra que t debe aparecer eventualmente en $W^{[i]}$. Si $t = f(t_1, t_2, ..., t_k) \in \mathrm{SSub}(t)$, entonces $t = \delta_k(f, t_1, t_2, ..., t_k)$, de acuerdo con el lema 4. En cambio, si $t \in F(T)$, entonces existe un único $s \in \mathrm{SSub}(T)$ tal que $\mathtt{equiv}_\mathtt{T}(t, s)$ y $s = \delta_k(f, t_1, t_2, ..., t_k)$. ■

En el algoritmo 6.2, es posible utilizar, por ejemplo, $A = \mathrm{Sub}(S_n) \subset \mathrm{Sub}(T)$, ya que para un valor de n suficientemente grande ocurrirá que $K(T) \subset \mathrm{Sub}(S_n)$. Por otro lado, es conveniente destacar que el algoritmo nunca utiliza $\mathtt{equiv}_\mathtt{T}$ fuera del dominio en el que ha sido definida, $K(T)$, y además el número de llamadas a esta función está acotado por $|K(T)|^2$. Como consecuencia de lo anterior, la complejidad global del algoritmo es el producto de $\mathcal{O}(|K(T)|^2)$ por la complejidad de la función $\mathtt{equiv}_\mathtt{T}$.

6.4 Inferencia probabilística

En la práctica, el lenguaje desconocido T será sustituido por la muestra estocástica S y el test de equivalencia $\mathtt{equiv}_\mathtt{T}(x, y)$ será reempla-

zado por una función probabilística $\texttt{comp}_\texttt{n}(x, y)$ que dependerá de los n primeros árboles en la muestra S, esto es, de S_n. El algoritmo obtendrá como resultado el DTA correcto en el límite en tanto en cuanto $\texttt{comp}_\texttt{n}$ tienda a $\texttt{equiv}_\texttt{T}$ en el límite de n grande.

De acuerdo con (6.28), $\texttt{equiv}_\texttt{T}(x, y) = \text{TRUE}$ significa que $x^{-1}T = y^{-1}T$ en el sentido dado por (6.12). Por tanto, para todos los árboles $s \in V_\$^T$ se satisface $p(s|x^{-1}T = p(s|y^{-1}T)$. El caso $s = \$$ (único árbol en $V_\$^T$ de profundidad cero) corresponde a la comparación de $r(\delta(x))$ y $r(\delta(y))$:

$$\frac{p(x|T)}{p(V_\$^T(x|T)} = \frac{p(y|T)}{p(V_\$^T y|T)} \; . \tag{6.30}$$

Los árboles s de profundidad mayor o igual que uno pueden descomponerse como $s = t\#z$, donde t es de profundidad 1 y z de profundidad arbitraria. Esto permite escribir que para todos los $t \in V_\$^T$ cuya profundidad es 1 se satisface:

$$\frac{p^M(V_\$^T \#t\#z\#x|T)}{p^M(V_\$^T \#x|T)} = \frac{p^M(V_\$^T \#t\#z\#y|T)}{p^M(V_\$^T \#y|T)}, \tag{6.31}$$

condición asociada a la comparación de las probabilidades del tipo $p_k(t_1, ..., t\#x, ..., t_k)$ y $p_k(t_1, ..., t\#y, ..., t_k)$. Para comprobar (6.30) y (6.31), $\texttt{comp}_\texttt{n}$ utiliza un test estadístico que se aplica a la diferencia entre los términos que deben ser comparados:

$$\frac{c_n(x)}{c_n(V_\$^T \#x)} - \frac{c_n(y)}{c_n(V_\$^T \#y)}, \tag{6.32}$$

y (siempre que $t\#z\#x \in S_n$ o $t\#z\#y \in S_n$):

$$\frac{c_n(V_\$^T \#t\#z\#x)}{c_n(V_\$^T \#x)} - \frac{c_n(V_\$^T \#t\#z\#y)}{c_n(V_\$^T \#y)}, \tag{6.33}$$

donde c_n cuenta el número de apariciones en S_n de los árboles del conjunto argumento de la función. Hemos elegido un test basado en el límite de Hoeffding (1963) descrito en la sección 3.5 y adaptado según la ecuación (5.29) y representado en la figura (5.3). Este test proporciona la respuesta correcta con probabilidad mayor que $(1-\alpha)^2$,

siendo α el nivel de significación, un parámetro real arbitrario de valor tan pequeño como se desee. En consecuencia, el algoritmo $\texttt{comp}_\texttt{n}$, tal y como aparece representado en la figura 6.3 devuelve el valor correcto con probabilidad mayor que $(1-\alpha)^{2r}$, donde r es el número de árboles diferentes en $x^{-1}S_n \cup y^{-1}S_n$. Como r aumenta ligeramente según crece n, debemos permitir que el parámetro α dependa de r. De hecho, si α decrece más rápido que $1/r$, entonces $(1-\alpha)^r$ tiende a cero y $\texttt{comp}_\texttt{n}(x, y) = \texttt{equiv}_\texttt{T}(x, y)$ en el límite de n grande.

Por último, la complejidad de $\texttt{comp}_\texttt{n}$ es, en el peor de los casos, $\mathrm{O}(n)$. Como $|K(T)|$ no depende de S_n, entonces la complejidad global del algoritmo es $\mathrm{O}(n)$, es decir es lineal con el tamaño de la muestra utilizada.

6.5 Resultados

La gramática independiente del contexto estocástica de la figura 6.4 genera estructuras condicionales en un lenguaje de programación. Las variables aparecen en cursiva, los terminales en negrita y la probabilidad de cada regla como un número entre paréntesis. El número promedio de reglas en la hipótesis producida por el algoritmo `tlips` para esta gramática en función del número de ejemplos en la muestra aparece representado en la gráfica 6.6. Allí podemos observar que cuando la muestra es pequeña, se obtienen gramáticas más bien pequeñas y el algoritmo tiende a la generalización excesiva. Según crece el número de ejemplos, el algoritmo tiende a producir una gramática del tamaño correcto y para muestras aún más grandes (a partir de 250 ejemplos) siempre encuentra la gramática correcta.

En la figura 6.5 se representa la entropía relativa $H(T, M)$ entre el lenguaje T y el modelo M propuesto por `tlips` siguiendo el procedimiento descrito en la sección 3.4. Como referencia se representa también la entropía relativa $H(T, S_n)$ entre el lenguaje y la muestra aleatoria S_n. Como se puede observar en la gráfica, esta última proporciona resultados mucho mayores, lo que indica que está más alejada

del modelo correcto.

De nuevo, al igual que en el caso de los lenguajes de cadenas, la identificación de la estructura del generador canónico reduce el número de probabilidades que se deben estimar, que pasa a ser un número finito, lo que conduce a una convergencia mucho más rápida.

Además, la implementación del algoritmo realizada consumía alrededor de 15 milisegundos por árbol en la muestra, cuando fue ejecutado en un ordenador Hewlett-Packard 715 de 40 MIPS. Este coste temporal resulta competitivo para aplicaciones prácticas, sobre todo teniendo en cuenta que es un coste lineal con el tamaño de la muestra.

El comportamiento del algoritmo `tlips` ha sido estudiado también con la gramática del la figura 6.7, que genera expresiones regulares, esto es, expresiones numéricas ligadas por operadores de suma, concatenación y clausura de Kleene:

Los resultados para esta gramática aparecen representados en las figuras 6.8 y 6.9. El comportamiento del algoritmo es también satisfactorio en este caso, y las mismas propiedades se ponen de manifiesto con esta gramática.

```
algorithm tlips
input: A ⊂ Sub(T) tal que K(T) ⊂ A
output: SSub (subárboles superficiales)
        F (frontera)
begin algorithm
  SSub = F = ∅
  W = V_0 ∩ A
  do ( while W ≠ ∅ )
     x = f(t_1, ..., t_k) = min W
     W = W − {x}
     if ∃y ∈ SSub : equiv_T(x, y) then
        F = F ∪ {x}
        y = δ_k(f, t_1, ..., t_k)
     else
        SSub = SSub ∪ {x}
        W = W ∪ {f(t_1, ..., t_k) ∈ A : t_1, ..., t_k ∈ SSub}
        x = δ_k(f, t_1, ..., t_k)

     endif
  end do
end algorithm
```

Figura 6.2: Algoritmo `tlips`.

```
algorithm comp_n
input: x, y ∈ V^T, S_n
output: boolean
begin algorithm
  if different(c_n(x), c_n(V_$^T#x), c_n(y), c_n(V_$^T#y), α) then
     return FALSE
  endif
  do ( ∀t, z : depth(t) = 1 ∧ (t#z#x ∨ t#z#y) ∈ S_n )
     if different(c_n(V_$^T#t#z#x), c_n(V_$^T#y),
     c_n(V_$^T#t#z#y), c_n(V_$^T#y), α) then
        return FALSE
     endif
  end do
  return TRUE
end algorithm
```

Figura 6.3: Algoritmo $\texttt{comp}_\texttt{n}$.

statement → **if** *expression* **then** *statement* **else** *statement* **fi**	(0.2)
statement → **if** *expression* **then** *statement* **fi**	(0.3)
statement → **print** *expression*	(0.5)
expression → *expression* **operator** *term*	(0.4)
expression → *term*	(0.6)
term → **number**	(1.0)

Figura 6.4: Gramática que genera expresiones condicionales.

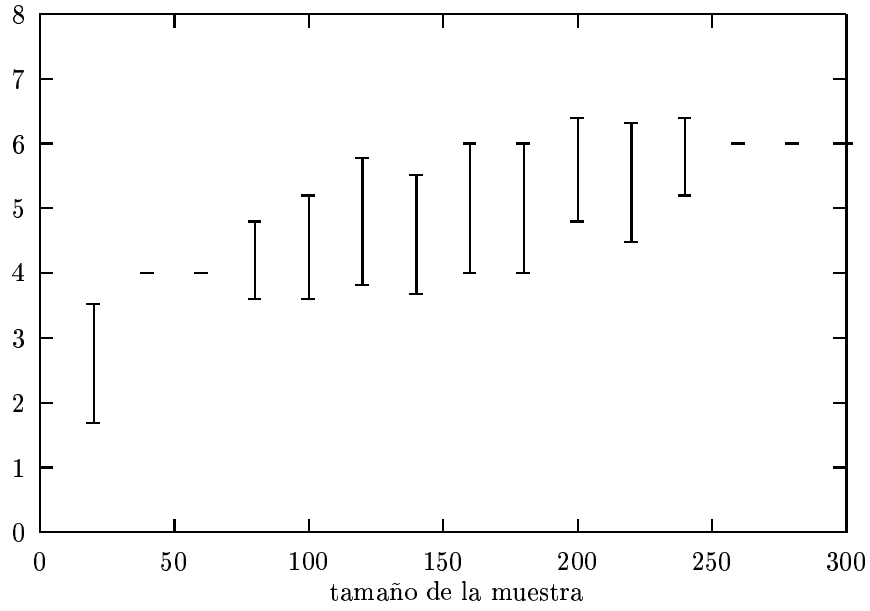

Figura 6.5: Número de reglas de la hipótesis en función del número de ejemplos. La gramática correcta 6.4 contiene 6 reglas.

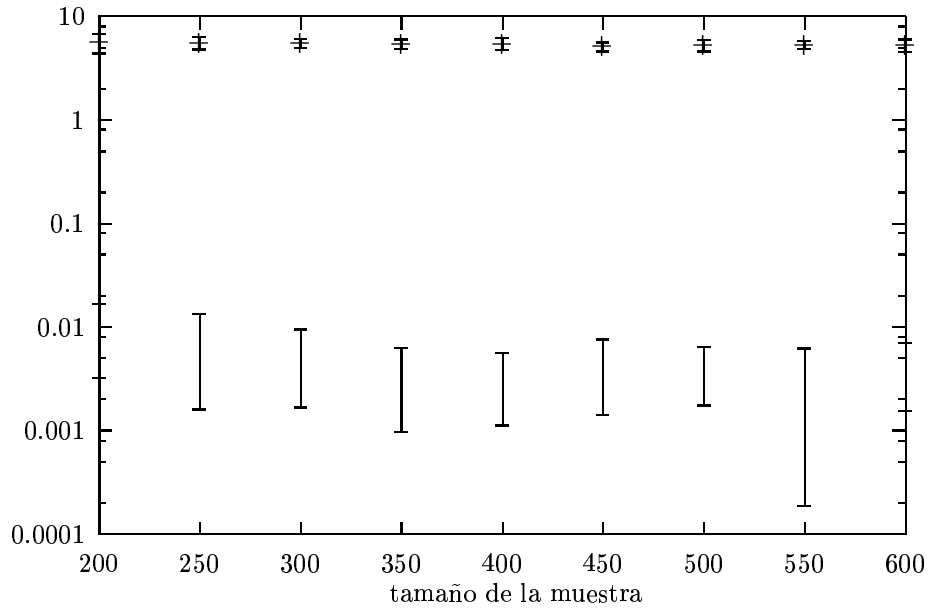

Figura 6.6: Línea inferior: entropía relativa (en bits) entre la gramática 6.4 y el modelo propuesto por `tlips` en función del número de ejemplos. Línea superior: entropía relativa entre la gramática y la muestra.

expression	$\rightarrow$	*term*	(0.5)
expression	$\rightarrow$	*expression* + *term*	(0.5)
term	$\rightarrow$	*factor*	(0.8)
term	$\rightarrow$	*term factor*	(0.2)
factor	$\rightarrow$	*factor* *	(0.4)
factor	$\rightarrow$	*element*	(0.6)
element	$\rightarrow$	**number**	(0.7)
element	$\rightarrow$	(*expression*)	(0.3)

Figura 6.7: Gramática que genera expresiones regulares.

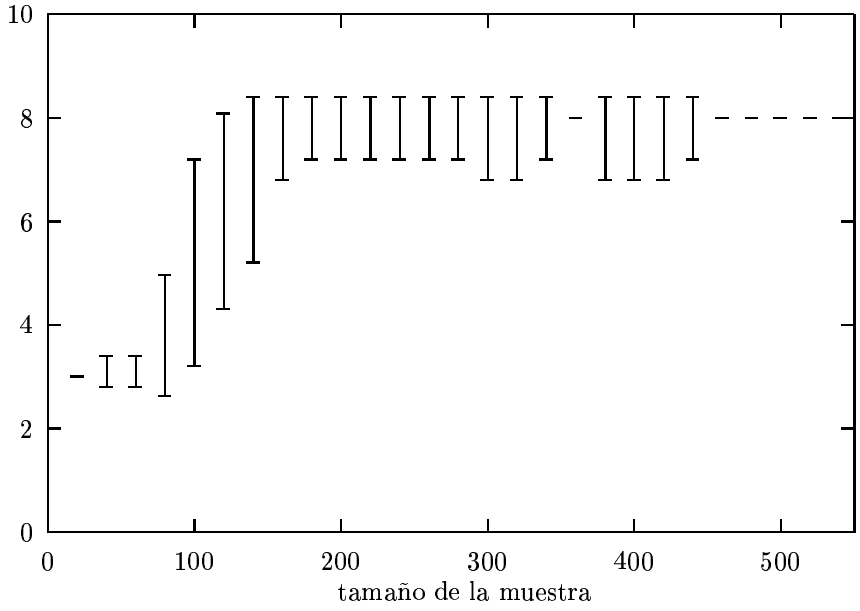

Figura 6.8: Número de reglas de la hipótesis en función del número de ejemplos generados por la gramática 6.7.

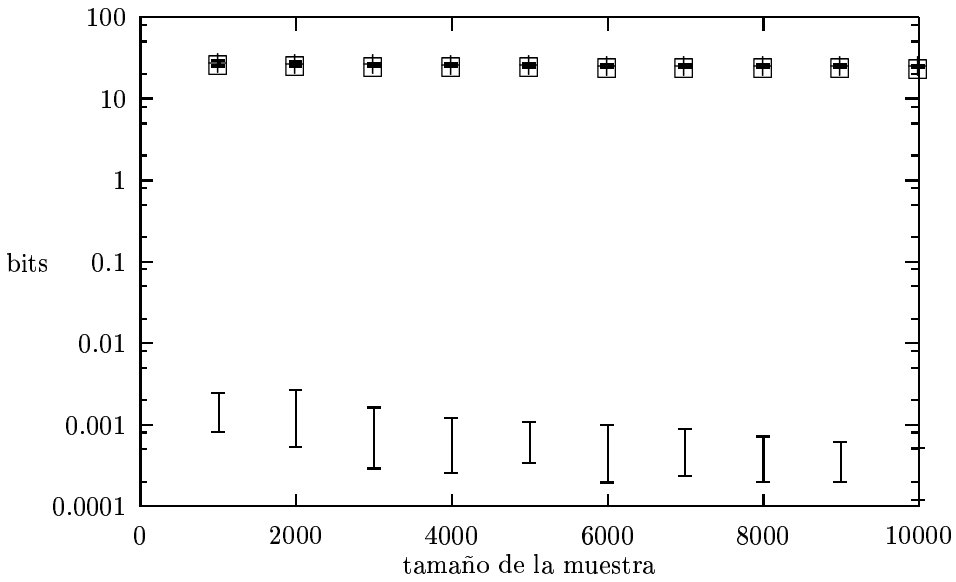

Figura 6.9: Entropía relativa entre la gramática 6.7 y la hipótesis propuesta por `tlips` en función del número de ejemplos.