BIBLIOTECA VIRTUAL
MIGUEL DE CERVANTES

Universidad de Alicante

Inferencia de lenguajes racionales estocásticos

Rafael C. Carrasco Jiménez

Tesis de Doctorado

Facultad: Escuela Politécnica Superior

Director: Dr. José Oncina Carratalá

1997

Universidad de Alicante
Departamento de Lenguajes
y Sistemas Informáticos

Inferencia de lenguajes racionales estocásticos

Rafael C. Carrasco Jiménez
Tesis doctoral

Mayo 1997

La presente memoria constituye la tesis doctoral presentada por Rafael C. Carrasco Jiménez para la obtención del título de Doctor en Informática y ha sido desarrollada bajo la dirección del Dr. Jose Oncina Carratalá, profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante

Para Carmen.

Agradecimientos: A todos aquellos que contribuyeron al desarrollo de esta tesis, bien con su dirección (Jose Oncina), sus sugerencias (Enrique Vidal, Mikel Forcada), o su insustituible aportación técnica (Emilio Corbí, Pepe Verdú). También a todos mis compañeros del Departamento de Lenguajes y Sistemas Informáticos, donde tan buena acogida he recibido.

El hombre no se libera de la acción por no emprenderla.
Baghavad Gita

Índice General

1 Introducción **11**

1.1 Aprendizaje inductivo 12

1.2 Inferencia gramatical 14

1.3 Identificación en el límite 15

1.4 Identificación de lenguajes estocásticos 18

1.5 Aprendizaje PAC 19

2 Autómatas finitos estocásticos y lenguajes **23**

2.1 Alfabetos y lenguajes 25

2.2 Autómatas finitos deterministas 27

2.3 Gramáticas regulares 29

2.4 Autómatas finitos deterministas estocásticos 31

2.5 Autómatas finitos de árboles 33

2.6 Autómatas estocásticos de árboles 35

3 Distancia entre lenguajes estocásticos **37**

3.1 Entropía de un lenguaje estocástico 38

3.2 Entropía relativa entre lenguajes 42

3.3 Entropía de un lenguaje de árboles 43

3.4 Entropía relativa entre lenguajes de árboles 47

3.5 Convergencia de una variable aleatoria 49

3.6 Aproximación contra identificación 52

4 Procedimientos clásicos de aprendizaje 59
4.1 Modelos de Markov ocultos 59
4.2 Modelos bayesianos y fusión de estados 65

5 Inferencia de lenguajes regulares 71
5.1 Antecedentes 72
5.2 Formalismo 73
5.3 Algoritmo de inferencia 79
5.4 Convergencia del algoritmo 82
5.5 Número de ejemplos necesarios para la convergencia . . 85
5.6 Resultados y discusión 86

6 Inferencia de lenguajes racionales de árboles 97
6.1 Antecedentes 97
6.2 Formalismo 99
6.3 Algoritmo de inferencia 107
6.4 Inferencia probabilística 109
6.5 Resultados 111

7 Inferencia estocástica con redes neurales 119
7.1 Antecedentes 119
7.2 Arquitectura de la red 121
7.3 Resultados y discusión 124

8 Conclusión y perspectivas 131

Presentación

En esta memoria se presentan una serie de algoritmos que permiten la identificación de forma eficiente de lenguajes racionales (esto es, lenguajes reconocidos por autómatas finitos) a partir de muestras aleatorias. El significado de la inferencia, en particular para los lenguajes estocásticos, los criterios de éxito en el aprendizaje y otros conceptos básicos son presentados en el capítulo 1. El capítulo 2 contiene una introducción a los autómatas finitos estocásticos. En el capítulo 3 se discuten distintos métodos para evaluar la calidad de los modelos obtenidos. Dichos métodos miden la distancia entre las distribuciones de probabilidad propuestas y las correctas basándose en la teoría de la información. Algunos técnicas y métodos tradicionales usados en el aprendizaje de lenguajes estocásticos son revisados en el capítulo 4. En el capítulo 5 se presenta un algoritmo nuevo para la identificación de lenguajes racionales de cadenas y en el capítulo 6, otro algoritmo para la identificación de lenguajes racionales de árboles. En el capítulo 7 se explora la posibilidad de que las redes neurales recurrentes de segundo orden identifiquen de forma robusta lenguajes racionales a partir de ejemplos estocásticos. Por último se realiza una discusión sobre los problemas abiertos y las posibles continuaciones de este trabajo.

Capítulo 1

Introducción

"Learning" is making useful changes in the working of our minds.
Marvin Minsky, The society of mind.

Según el *Diccionario de la lengua española* (Real Academia Española 1992) aprender es

> "adquirir el conocimiento de alguna cosa por medio del estudio o de la experiencia."

En esta disertación nos vamos a concentrar en este segundo aspecto de la definición: el aprendizaje a partir de la experiencia. Más concretamente, nuestro interés se centrará en el *aprendizaje computacional* a partir de ejemplos.

Durante las últimas décadas, la denominada *inteligencia artificial* ha buscado sustituir a los seres humanos por ordenadores en la realización de las tareas más pesadas o que más tiempo nos ocupan. También se ha planteado, como objetivo más ambicioso, la posibilidad de reproducir los comportamientos característicos de la inteligencia humana. Sin embargo, la idea de que el cerebro humano no es más que un complejo ordenador sigue siendo una cuestión controvertida (Penrose

1992). Dada la extrema complejidad del cerebro, que contiene del orden de 10^{10} neuronas densamente interconectadas, no disponemos hoy en día de un modelo adecuado para describir su funcionamiento. Esto hace imposible el diseño de mecanismos que simulen globalmente la actividad inteligente, y nos obliga a plantearnos objetivos más modestos como que dichos mecanismos simulen comportamientos específicos o que realicen adecuadamente tareas restringidas. Aún así, esta tarea es difícil. Una posibilidad que reduciría la cantidad de información que debe incluirse a priori en el sistema es que el ordenador aprenda a reaccionar adecuadamente tras un período de entrenamiento. Idealmente, la máquina respondería tras el entrenamiento con un comportamiento semejante al de un ser humano con las mismas experiencias. Este proceso de adquisición de conocimientos a partir de la experiencia es lo que tradicionalmente se ha denominado *aprendizaje inductivo*, cuya historia revisamos rápidamente en la sección siguiente.

1.1 Aprendizaje inductivo

El procedimiento inductivo fue sometido a una demoledora crítica por parte de David Hume (Hume 1748:67)

> "Todas las inferencias realizadas a partir de la experiencia, por tanto, son efecto de la costumbre y no del razonamiento.",

cuyas consecuencias se extienden hasta el siglo XX. De hecho, Bertrand Russell llegó a afirmar a este respecto que Hume "representa la bancarrota de la racionalidad" (Russell 1946). El filósofo Karl Popper intentó justificar los métodos inductivos desde el punto de vista de la lógica. Según Popper (Popper 1972), es posible preferir una hipótesis a otras por lo que respecta a su verdad o falsedad basándose en justificaciones empíricas: dado que los hechos experimentales pueden refutar algunas de ellas "preferimos aquella cuya falsedad no haya sido demostrada". Sin embargo, desde este punto de vista, no existe ningún

motivo racional para elegir una entre las hipótesis no rechazadas, pues el hecho de que una hipótesis concuerde con los experimentos realizados, por muy grande que sea el número de éstos, no garantiza que el acuerdo se mantenga en los experimentos futuros. Dicho de otra forma, los experimentos no sirven para verificar una hipótesis, sólo para refutar algunas de ellas.

Para solventar este problema, el mismo Popper introdujo el concepto de simplicidad en el proceso de inducción:

> "Hemos de valorar más los enunciados sencillos que los menos sencillos, porque nos dicen más, porque su contenido empírico es mayor y porque se pueden contrastar mejor."

Aunque a primera vista resulta interesante, su definición de simplicidad como *grado de contrastabilidad* no resulta práctica, ni es lo suficientemente precisa como para ser aplicada con generalidad. A este respecto, puede leerse la crítica de Carl G. Hempel en Hempel (1966:77).

En este punto, el concepto de Gold (1967) de *identificación en el límite* proporcionó un criterio riguroso para la elección de unas hipótesis sobre otras: en determinadas circunstancias, algunos procedimientos de formulación de hipótesis (pero no todos) garantizan que la hipótesis correcta será la única hipótesis propuesta después de un conjunto suficientemente grande de observaciones (si bien, no es posible especificar qué debe entenderse por "suficientemente grande"). Veremos en la sección 1.3 que dado cualquier método que permita ordenar las hipótesis, el criterio "proponer la primera hipótesis compatible con los experimentos" permite la identificación de la hipótesis correcta. Este resultado sugiere una definición rigurosa (aunque bastante flexible) del concepto de simplicidad y proporciona un indudable apoyo lógico al procedimiento inductivo.

1.2 Inferencia gramatical

El proceso de aprender la gramática correcta para un lenguaje a partir de ejemplos es conocido con el nombre de *inferencia gramatical.* La teoría de las gramáticas generativas fue desarrollada en los años cincuenta y sesenta a partir de las ideas del filólogo americano Noam Chomsky (Chomsky 1956). Su pretensión de encontrar un formalismo matemático para describir los lenguajes naturales resultó menos exitosa de lo esperado en cuanto a su objetivo de permitir el diseño de programas que pudieran interpretar o traducir textos. En cambio, la aplicación de estas ideas ha resultado especialmente provechosa en el ámbito de la informática, sobre todo en el desarrollo de lenguajes de programación y compiladores (Aho & Ullman 1972) y en la teoría de la computación, especialmente en el aprendizaje computacional (Laird 1988) y en los métodos sintácticos de reconocimiento de patrones (Fu 1982). Dado que, utilizando una codificación adecuada, cualquier conjunto de ejemplos puede describirse como cadenas de símbolos, el problema del aprendizaje del conjunto se reduce al de aprender las reglas de generación de estas cadenas. De una forma más general, los ejemplos se descomponen en estructuras elementales llamadas *primitivas* que aparecen formando *patrones* según ciertas reglas. Estas reglas de generación de patrones constituyen la gramática del lenguaje. La gran ventaja de la formulación gramatical o sintáctica es que un número reducido de reglas es capaz de describir un conjunto virtualmente infinito de patrones. Para ello, es suficiente con que la gramática incluya recursividad en sus reglas.

Una vez conocida la gramática mediante el proceso de inferencia, cualquier tarea de clasificación queda reducido a un problema de análisis sintáctico, esto es: se trata únicamente de decidir si el patrón por clasificar pertenece al lenguaje definido mediante la gramática. Para esta tarea existen diversos algoritmos eficientes de análisis, como el de Cocke, Younger (1967) y Kasami o el de Earley (1970), por lo que en esta memoria nos concentraremos en el problema de inferir la

gramática correcta.

1.3 Identificación en el límite

En este contexto, entenderemos el aprendizaje como la adquisición de la capacidad para realizar con éxito una tarea. Una definición rigurosa de esta idea intuitiva fue formulada por Gold en 1967 (Gold 1967), mediante el criterio de *identificación en el límite*. Según este criterio, se puede aprender un concepto si existe un procedimiento que garantiza que durante el proceso de aprendizaje sólo se producirá un número finito de errores.

De forma más precisa, dado un dominio Ω, llamaremos *lenguaje* o *concepto* a cualquier subconjunto $L \subset \Omega$, *ejemplo* de L a cualquier elemento $x \in L$ y *muestra positiva* a una secuencia infinita $S = \{x_1, x_2, ...\}$ de ejemplos de L, no necesariamente distintos. La *muestra finita* S_n está formada por los n primeros elementos de S. Diremos que el lenguaje L es identificable en el límite si existe un procedimiento $A(S, n)$ que, dada una muestra S del lenguaje L:

- para cada número natural $n \in \mathbb{N}$, propone como hipótesis para L un lenguaje $h_n = A(S, n)$;
- además, h_n converge a L en el sentido de que $h_n = L$ excepto para un número finito de valores de n, es decir, existe un valor n_0 tal que $h_n = L$ siempre que $n \geq n_0$.

Una clase de lenguajes F es identificable en el límite si todos los lenguajes de la clase son identificables en el límite mediante un mismo procedimiento A.

Gold también demostró que muchas clases importantes de lenguajes (entre ellos los lenguajes racionales que estudiaremos más adelante) no pueden ser identificados en el límite a partir de ejemplos. En muchos casos, una muestra positiva no es suficiente para identificar el lenguaje, esencialmente debido al problema de la generalización excesiva. La *generalización excesiva* se produce cuando se formula una

hipótesis que es más general que la correcta. Por ejemplo, h_2 generaliza a h_1 si $h_1 \subset h_2$. Si sólo disponemos de ejemplos, no siempre es posible elegir un procedimiento que identifique en cualquier caso la hipótesis correcta. Por ejemplo, si la clase F contiene el lenguaje Ω y además todos los lenguajes finitos $L \subset \Omega$, no existe un criterio para elegir entre Ω y L_n (siendo L_n el lenguaje finito formado exactamente por los ejemplos de S_n) que garantice la identificación en el límite.

Sin embargo, existen vías alternativas que evitan estas dificultades. Una de ellas es la utilización de muestras completas. Una *muestra completa* incluye no sólo los ejemplos del lenguaje sino también los contraejemplos, de forma que cada elemento aparece clasificado como perteneciente o no al concepto. De esta forma, $S = \{(x_1, \mu_1), (x_2, \mu_2)...\}$ donde $\mu_i = 1$ indica que x_i es un *ejemplo* ($x_i \in L$) y $\mu_i = 0$ indica que es un *contraejemplo* ($x_i \notin L$). Análogamente, $S_n = (x_1, \mu_1), ..., (x_n, \mu_n)$. La información contenida en la muestra completa S es suficiente para descartar todas las hipótesis que son demasiado generales.

Vamos a ver como un método enumerativo es suficiente para identificar un lenguaje que pertenece a un clase F recursivamente numerable de lenguajes[1] utilizando, para ello, muestras completas. Supongamos que el conjunto de hipótesis es $\{h_0, h_1, h_2, ...\}$ y que $h_r = L$ es la hipótesis correcta. Diremos que h_k es compatible con S_n si h_k contiene a todos los ejemplos de S_n y ninguno de sus contraejemplos o ejemplos negativos. El procedimiento enumerativo para identificar la hipótesis correcta consiste en elegir siempre como hipótesis aquella h_k con subíndice menor que es compatible con S_n.

Es evidente que de esta forma nunca se propondrá como hipótesis h_j si $j > r$. De hecho, no es posible que h_r sea rechazada, dado que nunca puede aparecer un ejemplo o contraejemplo incompatible con h_r. Por otro lado una hipótesis rechazada por S_n también lo será por todas las muestras finitas posteriores S_m tales que $m > n$. Dado que

[1] A nuestros efectos, C es un conjunto recursivamente numerable significa que existe un procedimiento algorítmico para asignar a cada $n \in \mathbb{N}$ uno de los elementos de C, de forma que dicho procedimiento es exhaustivo (suprayectivo) en C.

sólo existe un número finito de hipótesis h_i tales que $i < r$, resulta entonces evidente que no puede cambiarse de hipótesis más de r veces. Sólo resta, por tanto, justificar que todas las hipótesis h_i tales que $i < r$ acaban siendo descartadas para que la identificación en el límite siempre se alcance.

En efecto, si h_r no es compatible con h_i es porque existe al menos algún elemento en la diferencia simétrica de los dos conjuntos $h_r \oplus h_i$. Es decir, existe $x \in h_r$ tal que $x \notin h_i$ o bien, $x \in h_i$ tal que $x \notin h_r$. Si asumimos que x aparece en S, es decir que existe al menos un $m \in \mathbb{N}$ tal que $x = x_m$, entonces la hipótesis h_i es descartada por todas las S_n tales que $n \geq m$. En consecuencia, dada la muestra S, excepto para un número finito de valores de n, h_r es la primera hipótesis compatible con S_n. Nótese que, aunque queda garantizada la identificación en el límite, no disponemos de un criterio que nos permita afirmar cuándo se ha producido ésta. Además, ha sido necesario asumir que cualquier muestra contiene todos los ejemplos y contraejemplos necesarios para la identificación.

En la práctica los contraejemplos no son fáciles de conseguir, al menos en la cantidad deseada. Por ejemplo, en una tarea de reconocimiento de caracteres manuscritos podemos recoger grandes muestras de dígitos 0 y suponer que todas las formas de ceros acabarán siendo recogidas en la muestra. Sin embargo, aunque los ejemplos de otros dígitos pueden ser tomados como contraejemplos de ceros, es evidente que estos no son representativos de la clase complementaria (no todos los grafos que no representan un cero son otro dígito), por lo que el aprendizaje mostrará sistemáticamente una tendencia a la generalización excesiva.

La utilización de muestras completas puede ser evitada si se dispone de información adicional acerca del orden de presentación de los elementos de la muestra. Por ejemplo, si Ω está formado por cadenas de símbolos, es suficiente con que los elementos de la muestra positiva S aparezcan ordenados según su longitud, de forma que las cadenas que siguen a una dada sean todas de longitud mayor o igual que la

de ésta. Evidentemente, esto equivale a disponer de todos los contraejemplos. En efecto, dada $w \in S_n$, toda cadena $x \in \Omega$ de longitud menor que la de w tal que $x \notin S_n$ es necesariamente un contraejemplo del lenguaje. Otra forma de afrontar el problema de la generalización excesiva es asumir que los ejemplos han sido generados de acuerdo con una distribución probabilística preestablecida aunque desconocida. Esta opción parece corresponderse mejor con las situaciones reales. De hecho, en muchas aplicaciones como reconocimiento del habla, modelización del lenguaje natural etc, el proceso de aprendizaje involucra ejemplos ruidosos o aleatorios. La identificación de este tipo de lenguajes es discutida en la sección siguiente.

1.4 Identificación de lenguajes estocásticos

Angluin (1988) ha demostrado que es posible identificar en el límite un lenguaje a partir de muestras generadas aleatoriamente mediante distribuciones de probabilidad bastante generales, aunque sin llegar a proponer un método eficiente para ello. En cierto sentido, la regularidad estadística que presentan las muestras aleatorias es capaz de compensar la falta de datos negativos. Sin embargo, la suposición de que existe una fuente aleatoria de ejemplos no incrementa el conjunto de clases de lenguajes identificables.

Una *muestra aleatoria* del lenguaje L es una secuencia de ejemplos, $S = \{x_1, x_2, ...\}$, en la que los elementos de L aparecen repetidamente siguiendo una distribución de probabilidad $p(x)$ cuyo *soporte* coincide con L, es decir: $L = \{x \in \Omega : p(x) > 0\}$. Nótese que la muestra aleatoria no contiene información sobre los contraejemplos del lenguaje. La adición de una estructura probabilística al lenguaje no añade potencia a los métodos de identificación en el límite. Es decir, si una clase de lenguajes F es identificable en el límite utilizando muestras aleatorias, F es también identificable en el límite utilizando muestras completas (Angluin 1988).

Por otra parte, una clase recursivamente numerable de distribu-

ciones uniformemente aproximadamente computables es identificable en el límite (Angluin 1988). La distribución $p(x)$ es *uniformemente aproximadamente computable* si existe una función racional computable[2] $f : \Omega \times \mathbb{R} \to \mathbb{Q}$ que proporciona un número racional $f(x, \epsilon) \in \mathbb{Q}$ cuyo valor está como mucho a una distancia ϵ de $p(x)$. En particular, las distribuciones de probabilidad computables y racionales (es decir, aquellas que sólo toman valores en $\mathbb{Q}$), son identificables en el límite. Aunque en general los valores de las probabilidades en una distribución arbitraria pueden ser números reales, la distinción entre números reales y racionales es sólo relevante desde el punto de vista teórico, dado que en un ordenador todos ellos son representados mediante un número finito de bits. La cuestión de la identificación en el límite de probabilidades racionales será discutida con más detalle en la sección 3.6

1.5 Aprendizaje PAC

El criterio de aprendizaje formulado por Valiant (1984) es distinto del de Gold (1967), y se conoce con el nombre de *aprendizaje probablemente aproximadamente correcto* o, de forma abreviada, *aprendizaje PAC*. En este criterio, existe una distribución de probabilidad desconocida para los ejemplos y el algoritmo de aprendizaje extrae ejemplos al azar y trata de construir una hipótesis que no sea demasiado diferente del lenguaje correcto con probabilidad elevada.

En principio, una medida de la semejanza entre el lenguaje L y la hipótesis h podría definirse introduciendo una tasa de error, como el cociente del tamaño de la diferencia simétrica $L \oplus h$ y y el tamaño de L. Sin embargo, esta medida no es útil si L es un lenguaje infinito. Si disponemos de una distribución de probabilidad $p(x)$ para los elementos del dominio Ω, la probabilidad de las cadenas mal clasificadas

[2] La función f es computable si existe un algoritmo que proporciona el resultado de la función para cualquier entrada de los argumentos. Para una definición más rigurosa puede consultarse, por ejemplo, el libro de Hopcroft y Ullman (1979).

$p(h \oplus L)$ es una medida de la semejanza entre la hipótesis h y el lenguaje L, que denotaremos como $\mathrm{err}_L(h) = p(h \oplus L)$. Además, $p(x)$ nos permite generar muestras aleatorias finitas S_n de tamaño n arbitrario.

Todo lo anterior nos permite definir de forma más rigurosa un nuevo criterio de aprendizaje. Dado un dominio Ω y una clase F de lenguajes sobre Ω, F es aprendible (PAC) si existe un procedimiento A tal que para cualquier distribución de probabilidad $p(x)$ para los elementos de Ω y cualquier $L \in F$:

- toma $\epsilon > 0$ y $1 \geq \alpha > 0$ como entrada;
- puede llamar a un procedimiento $B(n)$ para generar S_n, una muestra aleatoria de L de tamaño n;
- proporciona un concepto $h = A(\epsilon, \alpha, S)$ tal que $\mathrm{err}_L(h) \leq \epsilon$ con probabilidad mayor que $1 - \alpha$.

Si el procedimiento A requiere un tiempo polinómico en términos de α y ϵ, F es aprendible *eficientemente*. Obviamente, para que esto se produzca A debe elegir también un valor para n polinómico en función de α y ϵ.

Este criterio es muy estricto, en el sentido de que no se realiza ninguna suposición sobre la distribución de probabilidad $p(x)$, ni tan siquiera que ésta sea computable. En particular, la posibilidad de aprender (PAC) una familia de lenguajes F viene dada por la dimensión de Vapnik-Chervonenkis de F, que es el tamaño del mayor conjunto $C \subset \Omega$ desmenuzado ("shattered") por F. Un conjunto C es *desmenuzado* por F si para cualquier subconjunto $X \subset C$ existe un lenguaje $L \in F$ tal que $F \cap C = X$. Es decir, existen suficientes lenguajes en F como para elegir uno que coincida (en el dominio propio de C) con cualquier subconjunto de C. De alguna forma, la dimensión de Vapnik-Chervonenkis mide la complejidad, en el sentido de capacidad expresiva, de la familia F.

Por ejemplo, si para algún elemento $x \in \Omega$ existe un lenguaje de F que contiene a x y otro que no lo contiene, la dimensión de Vapnik-

Chervonenkis de F es mayor o igual que uno. Para que la dimensión de F sea mayor o igual que 2, debe existir un par de elementos $x_1, x_2 \in \Omega$ y cuatro lenguajes $L_{00}, L_{01}, L_{10}, L_{11} \in F$ tales que $x_i \in L_{j_1 j_2}$ si y sólo si $j_i = 1$. De forma análoga se puede razonar para las condiciones con dimensión n.

Como veremos más adelante, los lenguajes racionales o regulares incluyen a todos los lenguajes finitos, y por tanto, su dimensión de Vapnik-Chervonenkis es mayor que cualquier valor n que se elija, o lo que es lo mismo, es infinita. Resulta que sólo las familias con dimensión de Vapnik-Chervonenkis finita son aprendibles (PAC) polinómicamente (Anthony & Biggs 1992), por lo que en adelante, utilizaremos el criterio de Gold para caracterizar el éxito en el aprendizaje.

Capítulo 2

Autómatas finitos estocásticos y lenguajes

Si el universo consta de un número infinito de términos, es rigurosamente capaz de un número infinito de combinaciones –y la necesidad de un Regreso queda vencida. Queda su mera posibilidad, computable en cero.
Jorge Luis Borges, Historia de la eternidad.

Las máquinas de estados finitos son uno de los mecanismos más sencillos que permiten realizar una tarea y se caracterizan por que su funcionamiento requiere tan sólo una capacidad de almacenamiento que es acotable *a priori*. Desafortunadamente, no todos los problemas pueden ser resueltos utilizando autómatas finitos. Vamos a considerar como ejemplo dos tareas del ámbito de la programación de ordenadores, aparentemente de dificultad similar:

1. Un programa que calcula el cociente de una cadena numérica

por 3. Este programa puede implementarse de forma sencilla sin que se necesite almacenar toda la cadena de entrada. Basta con ir procesando ésta símbolo a símbolo, calculando el cociente y guardando el resto para sumárselo (multiplicado por tres) a la siguiente lectura. Por tanto, un programa que disponga de un sistema de almacenamiento para guardar dos bits es suficiente para realizar la tarea. Por muy larga que sea la cadena de entrada, bastará con esperar el tiempo suficiente para obtener la salida, pero jamás se producirá el desbordamiento del programa. Los estados de la memoria utilizada pueden entenderse como los estados de un autómata finito y los cambios que se producen con la entrada pueden interpretarse como la llamada función de transición (véase la figura 2.1).

2. Un programa que invierte el sentido de la cadena. En este caso es necesario optar entre:

   (a) leer la cadena completa y después escribir el resultado, con lo que es de esperar que para una cadena lo suficientemente larga se agoten las reserva de memoria, o

   (b) utilizar una función recursiva.

   En esta segunda opción, el desbordamiento se producirá cuando se agote la pila de almacenamiento de llamadas recursivas, lo cual es inevitable si la cadena es lo suficientemente larga.

La tarea 2 pertenece a un grupo de problemas más difíciles de resolver que los del tipo 1. Mientras que éstos son resolubles mediante una máquina de estados finitos, los primeros requieren una pila de almacenamiento virtualmente ilimitada.

Precisamente por encontrarse los autómatas finitos entre los sistemas más sencillos que se pueden utilizar en la teoría de lenguajes, éstos han sido aplicados con éxito en numerosas tareas de aprendizaje y de reconocimiento de patrones. Toda la información que nos

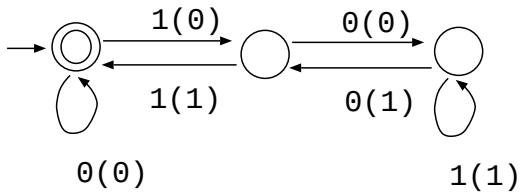

Figura 2.1: Máquina de estados finitos capaz de realizar el cociente de una cadena binaria por tres. El estado inicial aparece marcado por una flecha y la salida se forma a partir de los números entre paréntesis.

puede proporcionar un autómata finito es el estado en el que se encuentra después de haber procesado la entrada. Por ello, los estados se clasificarán en estados de aceptación y de no aceptación y servirán para clasificar las entradas en correctas e incorrectas. Los autómatas estocásticos aparecen cuando los cambios de estado del autómata se producen de forma aleatoria. En lo que sigue se introducen más detalladamente estos conceptos.

2.1 Alfabetos y lenguajes

Un *alfabeto* $\mathcal{A}$ es un conjunto finito y ordenado de símbolos. Por ejemplo: $\mathcal{A} = \{a, b, c, ..., z\}$ o el alfabeto binario $\mathcal{A} = \{0, 1\}$. El orden de sus elementos se denomina *ordenación alfabética.* Una secuencia de un número arbitrario (finito) de símbolos del alfabeto forma una *cadena, palabra* o *frase.* Por ejemplo, $w = aca$ significa que la cadena w está formada por una secuencia de tres símbolos del alfabeto: a, c y a. Dos secuencias compuestas de los mismos símbolos pero en diferente orden son distintas: $aca \neq caa$. El número de símbolos que componen la cadena es su longitud, y se representa habitualmente por $|w|$.

Al conjunto de todas las cadenas generables a partir del alfabeto $\mathcal{A}$ lo llamaremos *lenguaje universal* $\mathcal{A}^*$ y a la operación por la que se genera una cadena a partir de los símbolos que la componen, *con-*

catenación. Es posible formar cadenas de longitud arbitraria, y en particular, cabe la posibilidad de que la cadena contenga 0 símbolos. En ese caso, se trata de una *cadena vacía*, que se representará de forma especial mediante el símbolo λ y que satisface $\forall w \in \mathcal{A}^*$:

$$w\lambda = \lambda w = w. \tag{2.1}$$

Con la incorporación de λ, cuyas características son las de un elemento neutro, la concatenación dota de estructura de *monoide* a $\mathcal{A}^*$, es decir, se trata de una operación interna asociativa, con elemento neutro pero no conmutativa.

Dado que la concatenación es no conmutativa, se han de definir dos operaciones inversas, una por la izquierda y otra por la derecha. Así, si $w = xy$, siendo x, y, w palabras de $\mathcal{A}^*$, diremos que x es un *prefijo* de w mientras que y es un *sufijo* de w, y escribiremos $x = wy^{-1}$ o, alternativamente, $y = x^{-1}w$.

Por último, un *lenguaje* es un subconjunto cualquiera de cadenas de $\mathcal{A}^*$. Por ejemplo, las frases del castellano son un subconjunto de todas las cadenas que se pueden formar a partir del alfabeto habitual, y los signos de puntuación. También llamaremos, por analogía, lenguajes a conjuntos como el de las cadenas binarias que expresan un número par, o aquél que no contiene ninguna cadena o *conjunto vacío*, representado como $\emptyset$.

Los lenguajes que se pueden generar a partir de alfabetos finitos sólo pueden ser finitos o de cardinal infinito *numerable*. En efecto, si un lenguaje es infinito, siempre es posible establecer una ordenación efectiva en él, de forma que a cada cadena se le asigne un número de orden siguiendo la llamada *ordenación lexicográfica* o *canónica*. Ésta consiste en recorrer las palabras del lenguaje de menor a mayor longitud de las cadenas y dentro de cada longitud por el orden alfabético definido para el alfabeto $\mathcal{A}$. Por ejemplo, en el caso del alfabeto binario seguiría la sucesión $\mathcal{A}^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$.

Por último, la concatenación de palabras puede generalizarse para englobar también la de lenguajes: la *concatenación de dos lenguajes* L

y L' es un nuevo lenguaje formado por las cadenas obtenidas mediante concatenación de una cadena de L con otra de L', es decir,

$$LL' = \{w = xy : x \in L \wedge y \in L'\} \tag{2.2}$$

2.2 Autómatas finitos deterministas

Un autómata finito puede servir para discriminar palabras, observando el estado final en el cuál se encuentra. Por ejemplo, el autómata de la figura 2.1 permite clasificar las cadenas binarias de entrada que son múltiplo de tres y las que no, simplemente mirando cuál es el estado final después de la lectura. Los estados válidos son llamados *estados de aceptación* y aparecen marcados con doble círculo. El estado inicial (aquel en el que se encuentra el autómata antes de analizar la cadena) se señala con una flecha. Esta función como clasificadores de palabras permite que los autómatas finitos sean utilizados con frecuencia para el análisis léxico de programas o textos, es decir, para determinar qué palabras están correctamente escritas y pertenecen al lenguaje en cuestión, o para buscar determinados patrones en un texto.

Los elementos de un autómata finito determinista (DFA) son $A = (Q, \mathcal{A}, \delta, q_I, F)$, siendo:

- $Q = \{q_1, q_2, ..., q_N\}$ el conjunto finito de estados posibles del autómata;
- $\mathcal{A} = \{a_1, a_2, ..., a_L\}$ su alfabeto finito de entrada;
- $q_I \in Q$ su estado inicial;
- $F \subset Q$ el subconjunto de estados de aceptación;
- δ una *función de transición* entre estados de la forma $\delta : Q \times \mathcal{A} \rightarrow Q$ que dicta el comportamiento del autómata.

Un autómata finito opera ejecutando secuencias de movimientos o pasos según va leyendo los símbolos de la cadena de entrada. Cada

paso viene determinado por el estado actual del autómata y por el símbolo que se lee de la entrada. El movimiento consiste en leer ese símbolo (de manera que el siguiente símbolo de la cadena será leído en el siguiente movimiento) y cambiar al estado determinado por la función de transición δ.

Para describir formalmente el comportamiento del autómata sobre una cadena, debemos extender la función de transición $\delta : Q \times \mathcal{A} \to Q$ para que pueda actuar sobre un estado y una cadena y no sólo sobre un estado y un símbolo. Así, la nueva función es $\delta : Q \times \mathcal{A}^* \to Q$, de manera que $\delta(q, w)$ es el estado en el que se encontrará el autómata después de haber leído la cadena w empezando en el estado q y puede ser definida:

- $\delta(q, \lambda) = q$ (si no se lee ninguna cadena, es decir, se lee la cadena vacía, el autómata permanece en el mismo estado); y
- $\forall w \in \mathcal{A}^*, \forall a \in \mathcal{A}$ se cumple $\delta(q, wa) = \delta(\delta(q, w), a)$ (es decir, el estado al que llega el autómata al leer la cadena no vacía wa puede obtenerse por pasos: primero se calcula el estado $p = \delta(w, q)$ al que se llega con w desde q, y después se obtiene el estado al que se llega con el símbolo a desde p: $\delta(p, a)$).

Los autómatas definidos de esta forma son *deterministas*, en el sentido de que siempre está unívocamente definido cuál es el estado en que se encuentra el autómata después de haber leído una parte de la cadena de entrada. Si en algún caso no aparece definida la imagen de una transición entenderemos que la cadena no es válida o, alternativamente, que se dirige a un *estado de absorción* que recoge todas las transiciones no definidas y que no es un estado de aceptación.

Podemos definir el *lenguaje aceptado* por un DFA como:

$$L(A) = \{w \in \mathcal{A}^* : \delta(q_I, w) \in F\} \tag{2.3}$$

Por tanto, el DFA nos permite determinar si una cierta cadena w pertenece o no a $L(A)$ simplemente observando si el estado final al que se

llega mediante w partiendo de q_I es uno de los estados del subconjunto F. Todos los lenguajes reconocibles mediante autómatas finitos deterministas reciben el nombre de *lenguajes regulares* o también *lenguajes racionales*. Por tanto, dado un lenguaje L, si existe un DFA A tal que $L = L(A)$, entonces L es un lenguaje regular.

2.3 Gramáticas regulares

Una forma diferente de introducir el concepto de lenguaje es a través del uso de gramáticas generativas, siguiendo las ideas de Chomsky. Admitamos que cualquier lenguaje está constituido por unidades de información completas que reciben el nombre de oraciones o frases[1], lo cual es una aproximación en el caso de los lenguajes naturales, debido a la gran importancia del contexto. Por ejemplo, la mayoría de las *oraciones* en castellano constan de un *sujeto* y de un *predicado*. A estos términos abstractos como *oración*, *sujeto* o *predicado* les llamaremos *variables* del lenguaje y nunca aparecen en una frase construida como tales. La abstracción de las características de estos términos o variables es lo que se llama una *regla de derivación*, *regla de producción*, o simplemente *producción*. Por ejemplo, el hecho de que una *oración* consta de *sujeto* y *predicado* es una regla de producción y lo escribiremos:

$$\textit{oración} \rightarrow \textit{sujeto} \quad \textit{predicado}$$

A la variable *oración* se le denomina *símbolo inicial* o *axioma*, ya que debe estar en el origen de cualquier generación de una frase. Todas estas variables son abstracciones, y al final, deben ser sustituidas por las palabras concretas del lenguaje. A estos símbolos o palabras que pueden aparecer en una frase terminada les llamamos *terminales*, y al proceso por el cual se llega desde el concepto abstracto de *oración* hasta la frase particular se le denomina *derivación*. Por ejemplo:
oración $\Rightarrow$ *sujeto predicado* $\Rightarrow$ *modificador núcleo predicado* $\Rightarrow$ *modificador núcleo verbo complemento* $\Rightarrow$ el *núcleo verbo complemento* $\Rightarrow$

[1] que en la notación formal usada aquí, serán cadenas de símbolos.

el ordenador *verbo complemento* $\Rightarrow$ el ordenador está *complemento* $\Rightarrow$ el ordenador está averiado
es una derivación de la frase "el ordenador está averiado" donde se ha aplicado un subconjunto de reglas de producción (entre las muchas posibles del castellano):

- *oración* $\rightarrow$ *sujeto predicado*
- *sujeto* $\rightarrow$ *modificador núcleo*
- *predicado* $\rightarrow$ *verbo complemento*
- *modificador* $\rightarrow$ el
- *núcleo* $\rightarrow$ ordenador
- *verbo* $\rightarrow$ está
- *complemento* $\rightarrow$ averiado

Como se puede apreciar, se ha hecho la suposición de que las variables se sustituyen por otros términos independientemente del lugar donde aparezcan. Por ello, a este tipo de mecanismos de generación se les denomina gramáticas independientes del contexto. Una *gramática independiente del contexto* está definida por $G = (V_N, V_T, P, S)$, siendo $V_N = \{A, B, C, ...\}$ un conjunto finito de símbolos variables (no terminales), $V_T = \mathcal{A} = \{a, b, ...\}$ un conjunto finito de símbolos terminales o alfabeto, $S \in V_N$ el símbolo o variable inicial y P un conjunto finito de reglas de derivación. Al conjunto de símbolos, tanto variables como terminales, lo representaremos como $V = V_N \cup V_T$. Las reglas de derivación $r \in R$ indican cómo sustituir variables y se pueden expresar como aplicaciones $r : V_N \rightarrow V^*$, siendo V^* todas las cadenas formadas por símbolos de V.

Las cadenas generadas por una gramática son aquellas que se pueden obtener partiendo de S mediante la aplicación sucesiva de reglas de derivación de R. A este conjunto de cadenas derivables desde S se le llama *lenguaje generado por la gramática* G:

$$L(G) = \{w \in \mathcal{A}^* : S \overset{\star}{\Rightarrow} w\} \tag{2.4}$$

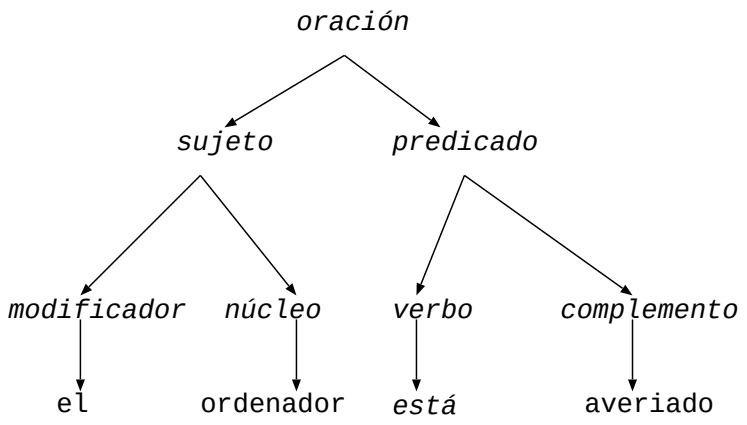

Figura 2.2: Árbol de derivación.

donde el asterisco indica una derivación en un número cualquiera de pasos. A la representación gráfica de este proceso mediante un árbol, se le llama *árbol de derivación*. La figura 2.2 representa el árbol de derivación correspondiente al ejemplo anterior.

Llamaremos *gramáticas regulares* a un subconjunto de las gramáticas independientes del contexto cuyas reglas de derivación pueden ser escritas usando únicamente los prototipos siguientes:

$$\begin{array}{lll} r: & V_N & \rightarrow V_T V_N \\ r: & V_N & \rightarrow \lambda \end{array} \tag{2.5}$$

Por ejemplo, $A \longrightarrow a$, o también $A \longrightarrow aB$. Como su propio nombre indica, todo lenguaje generable mediante una de estas gramáticas es regular y recíprocamente, todos los lenguajes regulares pueden generarse mediante una gramática regular (véase, por ejemplo, Hopcroft & Ullman 1979).

2.4 Autómatas finitos deterministas estocásticos

Un autómata finito determinista estocástico A asocia a cada cadena de $\mathcal{A}^*$ una cierta probabilidad de aparición. Para ello, se incorpora a la definición de A una función que asigna una probabilidad a cada una de las transiciones. Formalmente $A = (Q, \mathcal{A}, \delta, q_I, p)$, donde p es una función probabilística $p : Q \times \mathcal{A} \to [0, 1]$. Por ejemplo, en el autómata de la figura 2.3 las probabilidades de transición aparecen representadas sobre el arco correspondiente por los valores entre paréntesis. Debe notarse que a diferencia del caso de los autómatas no estocásticos, no se incluye en la definición una lista de estados de aceptación. La probabilidad de generar una cadena $w \in \mathcal{A}^*$ viene dada por el producto de las probabilidades de las transiciones que ejerce el camino, incluida la probabilidad de terminación en el último nodo, que es el resto hasta uno de la suma de las probabilidades de transición. En el ejemplo de la figura, el autómata asigna una probabilidad 0.075 a la cadena 11 y una probabilidad 0 a la cadena 101.

Para calcular las probabilidades anteriores es necesario conocer la probabilidad $p(q_i, \lambda)$ de terminación en cada nodo q_i. Esta probabilidad puede obtenerse a partir de la función $p(q_i, a)$ definida anteriormente. En efecto, si el autómata se encuentra en un estado q_i, el siguiente paso se determina aleatoriamente según las probabilidades $p(q_i, a)$ y $p(q_i, \lambda)$. Como las probabilidades de los sucesos que pueden ocurrir a continuación han de sumar uno, podemos escribir:

$$p(q_i, \lambda) + \sum_{a \in \mathcal{A}} p(q_i, a) = 1 \tag{2.6}$$

Por tanto, los valores de $p(q_i, \lambda)$ se pueden obtener a partir de la función p mediante la relación anterior. En el ejemplo de la figura 2.3, $p(q_1, \lambda) = 0.5$ y $p(q_2, \lambda) = p(q_3, \lambda) = 0$. Con estos valores se obtiene, por ejemplo, que la probabilidad de la cadena 11 es $p(11|A) = p(q_1, 1)\ p(q_2, 1)\ p(q_1, \lambda) = 0.075$.

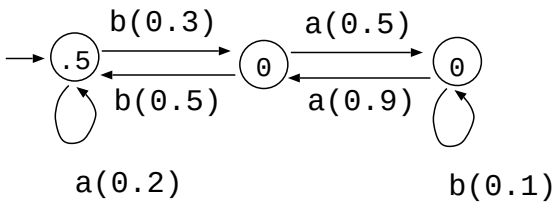

Figura 2.3: Autómata estocástico.

Si el autómata A no contiene nodos inútiles (estados desde los cuales no es posible llegar con probabilidad no nula a algún estado tal que $p(q_i, \lambda) > 0$), el autómata define una distribución de probabilidad $p(w|A)$ para las cadenas de $\mathcal{A}^*$, de forma que:

$$\sum_{w \in \mathcal{A}^*} p(w|A) = 1. \tag{2.7}$$

2.5 Autómatas finitos de árboles

Los árboles constituyen una estructura de orden superior a las cadenas, en el sentido de que se establecen relaciones más complejas entre sus componentes elementales. En una cadena, una vez definido un sentido de lectura privilegiado (habitualmente de izquierda a derecha) cada símbolo puede relacionarse con un antecesor único o con un sucesor también único. En el caso de los árboles, cada nodo puede tener un sólo *antecesor*, pero varios *descendientes*. El único nodo sin antecesores es la *raíz* del árbol. A los nodos que no tienen descendientes se les denomina *hojas* del árbol. Al igual que cada posición de la cadena lleva asociado un símbolo, los nodos del árbol pueden contener etiquetas. La figura 2.4 representa un árbol con 6 nodos. El nodo superior etiquetado con a es la raíz del árbol, y otros tres nodos son del tipo hoja (uno de los etiquetados con b y los dos con c).

Una forma compacta de codificar estos árboles etiquetados es mediante la llamada *notación funcional*. En ella, cada nodo se representa

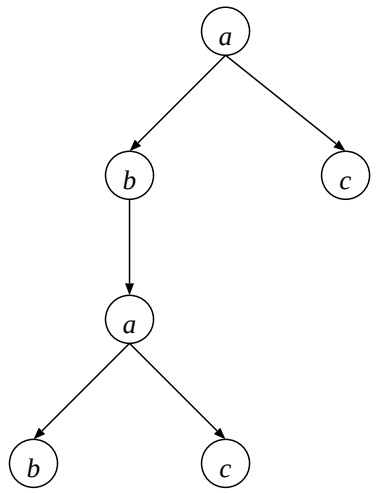

Figura 2.4: Árbol etiquetado.

como una función, cuyo nombre es la etiqueta que contiene y cuyos argumentos son los subárboles que genera cada descendiente. Por ejemplo, el árbol 2.4 se representaría en la notación funcional como $a(b(a(bc))c)$.

En un autómata de cadenas como los definidos anteriormente, se procesa la cadena de entrada de izquierda a derecha y para cada lectura y cada estado en que se encuentra el autómata se produce una transición a un nuevo estado. Cuando se ha agotado la cadena de entrada, el estado final en el que se encuentra el autómata determina la validez o no de la cadena.

En el caso de los árboles, comenzaremos a analizar el árbol por sus hojas, es decir, estableceremos un orden de lectura ascendente. En función de la etiqueta del nodo y de los estados de los descendientes se producirá una transición a un nuevo estado. Al llegar a la raíz del árbol, el estado en que se encuentra el autómata determina si el árbol es aceptado o no.

Por tanto, nos encontramos ante un mecanismo muy similar al de los autómatas finitos deterministas para cadenas al que llamare-

mos *autómata finito determinista de árboles.* Cada autómata $A = (Q, V, \delta, F)$ de este tipo constará de:

- un conjunto finito de estados $Q = \{q_1, q_2, ..., q_N\}$;
- un alfabeto finito de etiquetas para los nodos de los árboles $V = \{a_1, a_2, ..., a_L\}$;
- un subconjunto de estados de aceptación $F \subset Q$;
- un conjunto de funciones de transición $\delta = \{\delta_0, \delta_1, ...\delta_n\}$.

Dado que el número de descendientes de un nodo es variable, para describir cómo se producen las transiciones es necesario proporcionar no una sino una colección de funciones de transición $\delta = \{\delta_0, \delta_1, ...\delta_n\}$, donde n es el máximo número de descendientes admitido en el lenguaje. Cada función δ_k toma como argumentos un símbolo de V y k estados (uno por cada descendiente del nodo), es decir, $\delta_k : V \times Q^k \to Q$.

Por ejemplo, si t es una subárbol hoja con etiqueta a, entonces $k = 0$ y el estado asociado a t es $\delta(t) = \delta_0(a)$. Sin embargo, si t es un subárbol etiquetado f y con tres descendientes que generan respectivamente los subárboles t_1, t_2 y t_3, entonces $t = f(t_1, t_2, t_3)$ y el estado asociado a la raíz del subárbol es $\delta(t) = \delta_3(f, t_1, t_2, t_3)$. Como en el caso de los lenguajes de cadenas, las transiciones no contenidas en la definición conducen a árboles no aceptados. El lenguaje reconocido por el autómata A es entonces:

$$T(A) = \{t \in V^T : \delta(t) \in F\} \tag{2.8}$$

donde V^T representa todos los árboles que se pueden construir con nodos etiquetados con V.

De forma análoga al caso de las cadenas, llamamos *lenguajes racionales de árboles* a todos aquellos para los que existe un autómata finito determinista de árboles A que reconoce el lenguaje, esto es, T es racional si existe A tal que $T = T(A)$.

2.6 Autómatas estocásticos de árboles

Al igual que los autómatas finitos deterministas de cadenas pueden incorporar una función de probabilidad para generar lenguajes estocásticos, también es posible asignar una probabilidad a cada transición de un autómata finito determinista de árboles utilizando para ello un conjunto de funciones de probabilidad $p_k : V \times Q^k \to [0,1]$. En este caso, la condición de normalización es que las probabilidades de las transiciones que conducen a un mismo estado q han de sumar 1. Si denotamos, por simplicidad los estados del autómata mediante su ordinal, $Q = \{1, 2, ..., N\}$, entonces, para todo $q \in Q$

$$\sum_{k=0}^{n} \sum_{f \in V} \sum_{\substack{q_1, q_2, ..., q_k \in Q \\ q = \delta(f, q_1, q_2, ..., q_k)}} p_k(f, q_1, ..., q_k) = 1. \tag{2.9}$$

Dadas las funciones p_k, la probabilidad de generar un árbol t se obtiene efectuando el producto de las probabilidades de transición utilizadas en el análisis de t. Sin embargo, con esta definición no está garantizado que la suma de todas las probabilidades para todos los árboles sumen 1. Para que esta propiedad se satisfaga, es necesario además asignar una probabilidad $r(q)$ a cada nodo de que éste aparezca como raíz. De esta forma la probabilidad global del árbol t aparece además multiplicada por $r(q)$ si $q = \delta(t)$ es el estado asociado a la raíz, es decir, $p(t|A) = r(\delta(t))\ p(t|\delta(t))$, donde $p(t|\delta(t))$ representa la probabilidad de generar t a partir del estado $q = \delta(t)$, que debe calcularse recursivamente. Por ejemplo, para un árbol $t = f(t_1, t_2)$, obtendríamos

$$p(t|\delta(t)) = p_2(f, \delta(t_1), \delta(t_2))\ p(t_1|\delta(t_1))\ p(t_2|\delta(t_2)). \tag{2.10}$$

Siempre que

$$\sum_{q \in Q} r(q) = 1 \tag{2.11}$$

el autómata estocástico de árboles $A = (Q, V, \delta, p, r)$ establece un distribución de probabilidad sobre el lenguaje V^T, es decir,

$$\sum_{t \in V^T} p(t|A) = 1. \tag{2.12}$$

Capítulo 3

Distancia entre lenguajes estocásticos

Aunque no hubiera azar en este mundo, nuestra ignorancia de la causa real de un suceso tendría la misma influencia sobre el entendimiento.
David Hume, Enquiry concerning the human Understanding.

En este capítulo se introducen algunas herramientas que utilizaremos en el estudio de los métodos de identificación de lenguajes estocásticos. En particular, para evaluar la calidad de los modelos obtenidos encontraremos métodos eficientes para calcular la distancia de Kullback–Leibler entre lenguajes racionales, una medida de la similitud entre distribuciones probabilísticas que se basa en la teoría de la información. El cálculo de la distancia entre modelos de Markov ha sido estudiado anteriormente (Ziv & Merhav 1993, Kesidis & Walrand 1993). En lo que sigue generalizaremos estos procedimientos al caso de los autómatas finitos estocásticos, tanto de cadenas (en la sección 3.2) como de árboles (en la sección 3.4). Además, presentaremos distin-

tos criterios de contraste estadístico para variables aleatorias (3.5) y discutiremos las ventajas e inconvenientes de identificar o estimar una probabilidad (3.6).

3.1 Entropía de un lenguaje estocástico

Dado un lenguaje estocástico L con distribución de probabilidad $p(x|L)$ sobre $\mathcal{A}^*$, se define la entropía del lenguaje como el valor esperado (cambiado de signo) del logaritmo de la función de probabilidad

$$H(L) = -\sum_{x \in \mathcal{A}^*} p(x|L) \log p(x|L) \tag{3.1}$$

donde se toma por convención $0 \log 0 = 0$. Si el logaritmo se toma en base dos, el resultado se expresa en bits. La entropía es siempre un número positivo y constituye una medida del desorden del lenguaje. Por ejemplo, para un lenguaje finito formado por N cadenas equiprobables, la entropía es $\log N$ bits. Este número está relacionado con la longitud media de las cadenas si se utiliza una codificación óptima (recuérdese que el número de mensajes codificables con cadenas binarias de longitud l es 2 elevado a l). También puede interpretarse como el número esperado de preguntas (con respuesta si/no) necesarias para identificar el resultado de un experimento aleatorio realizado a partir de la distribución $p(x|L)$, suponiendo, de nuevo, que la estrategia seguida es óptima.

Es sencillo probar que si un lenguaje consta de N cadenas no equiprobables $\{x_1, x_2, ..., x_N\}$ la entropía es siempre menor que $\log N$. En el caso extremo de que todas las probabilidades son nulas excepto una que es uno, la entropía es trivialmente nula. Teniendo en cuenta que la suma de las probabilidades de todas las cadenas es uno, se trata pues de busca el máximo de la función H sobre la variedad lineal $\sum_{k=1}^{N} p_k = 1$. Introduciendo el correspondiente multiplicador de Lagrange

$$\lambda \sum_{k=1}^{N} \Delta p_k = 0, \tag{3.2}$$

donde λ es un número real arbitrario y Δ representa un diferencial, y derivando H se obtiene que la condición de máximo es

$$\Delta H = \sum_{k=1}^{N} (\Delta p_k \ \log p_k + \Delta p_k - \lambda \Delta p_k) = 0, \tag{3.3}$$

por lo que

$$\sum_{k=1}^{N} \Delta p_k \ (\log p_k + 1 - \lambda) = 0 \tag{3.4}$$

La introducción del multiplicador garantiza que todos los diferenciales Δp_k son independientes y, por tanto, $p_k = \lambda - 1$ para todas las cadenas x_k. Por tanto, el máximo se alcanza en el caso de que todas las cadenas son equiprobables. Esta disminución de la entropía puede ser interpretada en términos de longitud media de la codificación como sigue: la codificación puede ser optimizada si a las cadenas más probables se les asignan los códigos más cortos y a las menos probables los más largos. También el número de preguntas en el problema de la predicción de un resultado puede disminuirse en la misma proporción siguiendo una estrategia de agrupamiento, que divida en cada paso las cadenas en dos clases aproximadamente equiprobables.

Dos lenguajes con la misma entropía no son, en general, idénticos. Sin embargo la magnitud:

$$H(L_1, L_2) = \sum_{x \in \mathcal{A}^*} p(x|L_1) \log \frac{p(x|L_1)}{p(x|L_2)} \tag{3.5}$$

presenta la propiedad de que $H(L_1, L_2) = 0$ si y sólo si $p(x|L_1) = p(x|L_2)$ para todas las cadenas $x \in \mathcal{A}^*$. Esta magnitud es conocida con el nombre de entropía relativa o distancia de Kullback–Leibler, si bien no se trata de una auténtica distancia pues ni es simétrica ni satisface la desigualdad triangular. La entropía relativa indica la penalización (en bits) que se sufre por utilizar una distribución errónea en lugar de la correcta para diseñar la estrategia óptima en los problemas de codificación o de predicción de un resultado.

En lo que sigue, denotaremos mediante $p_L(a|x)$ la probabilidad de observación del símbolo a del alfabeto después del prefijo x, es decir, la probabilidad de a condicionada a la aparición previa de x:

$$p_L(a|x) = \frac{p(xa\mathcal{A}^*|L)}{p(x\mathcal{A}^*|L)}. \tag{3.6}$$

De forma análoga, y consistentemente con la ecuación (2.6), $p_L(\lambda|x)$ representará la probabilidad de que se observe un final de cadena después del prefijo x:

$$p_L(\lambda|x) = \frac{p(x|L)}{p(x\mathcal{A}^*|L)} \tag{3.7}$$

Con estos convenios, la probabilidad, por ejemplo, $p(ab|L)$ para la cadena ab en el lenguaje L satisface:

$$p(ab|L) = p_L(a|\lambda)\; p_L(b|a)\; p_L(\lambda|ab) \tag{3.8}$$

y su logaritmo correspondiente es, por tanto,

$$\log p(ab|L) = \log p_L(a|\lambda) + \log p_L(b|a) + \log p_L(\lambda|ab) \tag{3.9}$$

Es decir, en el cálculo de la entropía (3.1) encontraremos el término $\log p_L(b|a)$ en todos los sumandos asociados a cadenas que empiezan por ab. En general, el factor $\log p_L(a|x)$, con $a \in \mathcal{A}$, aparece para todas las cadenas que empiezan por xa, que denotaremos como $xa\mathcal{A}^*$, mientras que el factor $\log p_L(\lambda|x)$ sólo multiplica a $p(x|L)$ y podemos escribir:

$$H(L) = -\sum_{x\in\mathcal{A}^*}\sum_{a\in\mathcal{A}} p(xa\mathcal{A}^*|L)\log p_L(a|x) - \sum_{x\in\mathcal{A}^*} p(x|L)\log p_L(\lambda|x). \tag{3.10}$$

Usando (3.6) y (3.7), podemos reescribir la ecuación anterior en una forma más sencilla:

$$H(L) = -\sum_{x\in\mathcal{A}^*}\sum_{a\in\mathcal{A}^\dagger} p(x\mathcal{A}^*|L)p_L(a|x)\log p_L(a|x) \tag{3.11}$$

donde $\mathcal{A}^\dagger = \mathcal{A} \cup \{\lambda\}$.

Si L es generado por una gramática regular estocástica, existe un autómata asociado $A = (Q, \mathcal{A}, \delta, q_I, p)$ que genera L y $p_L(a|x)$ puede tomar sólo un número finito de valores distintos. De hecho, para todas las cadenas x que satisfacen que $\delta(q_I, x) = q_i$ y para todos los símbolos $a \in \mathcal{A}^\dagger$ se obtiene $p_L(a|x) = p(q_i, a)$. Los diferentes subconjuntos $L_i = \{x \in \mathcal{A}^* : \delta(q_I, x) = q_i\}$ definen una partición en L y, si definimos

$$c_i = \sum_{x \in L_i} p(x\mathcal{A}^*|L), \tag{3.12}$$

obtenemos que la entropía es

$$H(L) = -\sum_{q_i \in Q} \sum_{a \in \mathcal{A}^\dagger} c_i \, p(q_i, a) \log p(q_i, a) \ , \tag{3.13}$$

expresión que puede calcularse fácilmente si se conocen los coeficientes c_i.

Nótese que $\lambda \in L_I$ y además $p(\mathcal{A}^*|L) = 1$. Esto nos permite tratar separadamente el caso especial $x = \lambda$ y escribir

$$c_i = \delta_{iI} + \sum_{x \in \mathcal{A}^*} \sum_{\substack{a \in \mathcal{A}: \\ xa \in L_i}} p(xa\mathcal{A}^*|L) \tag{3.14}$$

donde I es el índice del estado inicial q_I y δ_{ij} es la delta de Kronecker:

$$\delta_{ij} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{en caso contrario} \end{cases} \tag{3.15}$$

Como $L = \bigcup_j L_j$ y para cada $x \in L_j$,

$$p(xa\mathcal{A}^*|L) = p(x\mathcal{A}^*|L) p(q_j, a) \ , \tag{3.16}$$

se obtiene

$$c_i = \delta_{iI} + \sum_{j=1}^{|Q|} \sum_{x \in L_j} \sum_{\substack{a \in \mathcal{A}: \\ \delta(q_j, a) = q_i}} p(x\mathcal{A}^*|L) \, p(q_j, a) \tag{3.17}$$

Por tanto, los coeficientes c_i pueden ser calculados resolviendo el sistema de ecuaciones

$$c_i = \sum_{j=1}^{|Q|} \Lambda_{ij}\, c_j + \delta_{iI} \tag{3.18}$$

donde

$$\Lambda_{ij} = \sum_{\substack{a \in \mathcal{A}\,: \\ \delta(q_j,\, a) = q_i}} p(q_j, a). \tag{3.19}$$

y $c_i^{[0]} = 0$. La inversión de la matriz Λ_{ij} suele ser costosa, por lo que resulta más eficiente un cálculo iterativo de los coeficientes:

$$c_i^{[t+1]} = \sum_{j=1}^{|Q|} \Lambda_{ij}\, c_j^{[t]} + \delta_{iI} \tag{3.20}$$

Es sencillo comprobar por inducción en t que $c_i^{[t+1]} \geq c_i^{[t]}$ y que al mismo tiempo $c_i^{[t]} \leq c_i$, por lo que el cálculo iterativo converge rápidamente al valor correcto.

3.2 Entropía relativa entre lenguajes

Es posible aplicar un procedimiento semejante al de la sección anterior para calcular la entropía relativa entre dos lenguajes estocásticos L y L', generados por A y A' respectivamente. En este caso,

$$H(L, L') = \sum_{q_i \in Q} \sum_{q'_j \in Q'} \sum_{a \in \mathcal{A}^\dagger} c_{ij}\, p(q_i, a) \log \frac{p(q_i, a)}{p'(q'_j, a)} \tag{3.21}$$

con los coeficientes

$$c_{ij} = \sum_{x \in L_{ij}} p(x\mathcal{A}^* | L), \tag{3.22}$$

donde

$$L_{ij} = \{x \in \mathcal{A}^* : \delta(q_I, x) = q_i \wedge \delta'(q'_{I'}, x) = q'_j\}. \tag{3.23}$$

Los coeficientes c_{ij} pueden ser calculados mediante la relación:

$$c_{ij}^{[t+1]} = \delta_{iI}\delta_{I'j} + \sum_{k=1}^{|Q|} \sum_{l=1}^{|Q'|} \Lambda_{ijkl}\, c_{kl}^{[t]} \tag{3.24}$$

donde

$$\Lambda_{ijkl} = \sum_{\substack{a \in \mathcal{A}: \\ \delta(q_k, a) = q_i \\ \delta'(q'_l, a) = q'_j}} p(q_k, a). \tag{3.25}$$

La expresión anterior para los coeficientes $c_{ij}^{[t+1]}$ puede probarse de forma sencilla siguiendo los mismos pasos de la sección anterior si se observa que $\lambda \in L_{II'}$ y que los antiguos coeficientes c_i satisfacen

$$c_i = \sum_{j=1}^{|Q'|} c_{ij}. \tag{3.26}$$

La figura 3.1 representa la entropía relativa entre dos lenguajes generados por dos gramáticas elegidas al azar, cada una con 10 estados y 30 reglas, ambas con el mismo alfabeto de trabajo $\mathcal{A} = \{0, 1\}$. La línea sólida representa el resultado del cálculo algorítmico mientras que los puntos son los resultados y desviaciones obtenidos cuando la entropía relativa se calcula mediante conjuntos de prueba de distintos tamaños. Se observa que, incluso para gramáticas sencillas como éstas, la convergencia al valor correcto es bastante lenta y que se requieren muestras enormes para que la estimación de la distancia entre los lenguajes sea fiable. Por ello, el procedimiento descrito en esta sección puede ser utilizado para conseguir una comprobación más precisa de los modelos obtenidos mediante inferencia gramatical.

3.3 Entropía de un lenguaje de árboles

En la sección 3.1 vimos cómo la entropía de un lenguaje regular estocástico L puede ser calculada de forma eficiente si se conoce la colección de coeficientes $c_i = \sum_{x \in L_i} p(x\mathcal{A}^*|L)$. Dichos coeficientes pueden interpretarse como el valor esperado del número de nodos de tipo q_i que se utilizan en el análisis de una cadena w elegida al azar según la distribución $p(w|L)$. Es posible realizar un razonamiento análogo para el caso de un lenguaje estocástico racional de árboles T generado

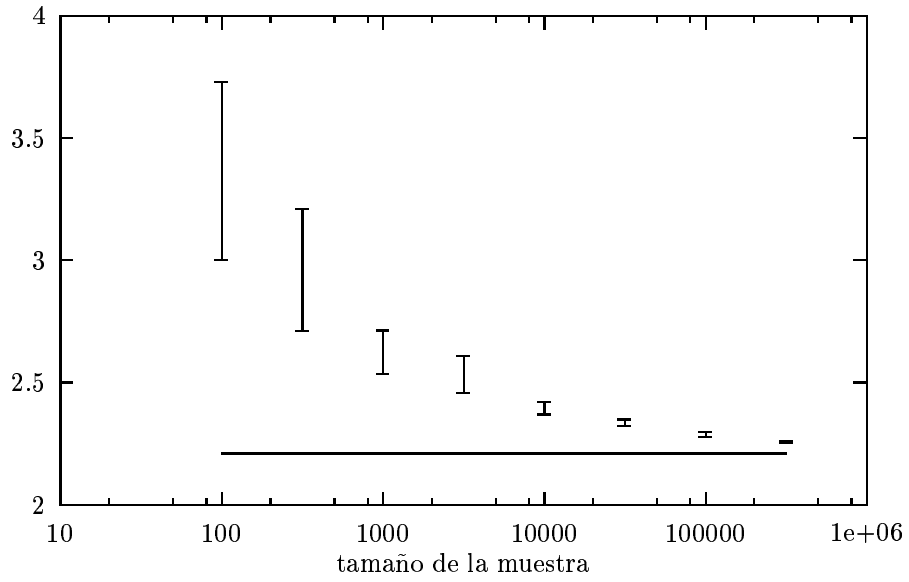

Figura 3.1: Entropía relativa en bits entre dos gramáticas de tamaño 10 generadas al azar. Línea continua: cálculo exacto. Puntos: estimación mediante muestras.

mediante el autómata $A = (Q, V, \delta, p, r)$. En este caso, la probabilidad de generación de un árbol t se compone de dos factores multiplicativos:

- Por un lado, la probabilidad $r(q)$ de que la raíz del árbol sea del tipo $q = \delta(t)$.
- Por otro, la probabilidad $p(t|q)$ de que se genere el árbol t a partir de un nodo del tipo $q = \delta(t)$; esta probabilidad ha de evaluarse recursivamente.

Recordemos que, por ejemplo, si $t = f(t_1, t_2)$, entonces

$$p(t|A) = r(\delta(t))\ p_2(f, \delta(t_1), \delta(t_2))\ p(t_1|\delta(t_1))\ p(t_2|\delta(t_2))\ . \tag{3.27}$$

La condición de normalización (2.9) garantiza que $p(t|q) = 0$ para todo $q \neq \delta(t)$.

Para simplificar la notación, en lo que sigue denotaremos los estados del autómata indicando únicamente el ordinal correspondiente: $Q = \{1, 2, ..., N\}$, de forma que $q \in Q$ es un natural entre 1 y N. Por otra parte, n denota el máximo número de descendientes que puede tener un nodo en el lenguaje, valor que viene determinado por el conjunto δ de funciones de transición.

La entropía de un lenguaje de árboles T generado por el autómata A es

$$H(T) = -\sum_{t \in V^T} p(t|A) \log p(t|A). \tag{3.28}$$

En el cálculo de la expresión anterior, el factor $\log r(q)$ multiplica a la probabilidad de todos los árboles t tales que $\delta(t) = q$, por lo que $\log r(q)$ aparecerá multiplicando a

$$\sum_{\substack{t \in V^T \\ \delta(t) = q}} p(t|A) = r(q), \tag{3.29}$$

es decir, los términos asociados a las probabilidades $r(q)$ contribuyen a la entropía con un sumando

$$H_r(T) = -\sum_{q \in Q} r(q) \log r(q) \tag{3.30}$$

Por otro lado, si tomamos por ejemplo $p_2(f, q_1, q_2)$, siendo $f \in V$ y $q_1, q_2 \in Q$, el factor $\log p_2(f, q_1, q_2)$ aparecerá multiplicando a la probabilidad de aquellos árboles que contienen un nodo del tipo $q = \delta_2(f, q_1, q_2)$ etiquetado f y que genera dos descendientes del tipo q_1 y q_2 respectivamente. Además, si hubiese m nodos con tales características en el mismo árbol t, $p(t|A)$ aparecería m veces. Es decir, $\log p_2(f, q_1, q_2)$ aparecerá multiplicado por $c_q p_2(f, q_1, q_2)$, siendo c_q el número esperado de nodos de tipo q en un árbol elegido al azar según $p(t|A)$.

Resumiendo, podemos descomponer la entropía del lenguaje T en dos términos

$$H(T) = H_r(T) + H_p(T) \tag{3.31}$$

donde $H_r(T)$ viene dado por la expresión (3.30) y $H_p(T)$ es

$$\begin{aligned} H_p(T) \quad = \quad & \sum_{k=0}^{n} \sum_{f \in V} \sum_{q_1, q_2, \ldots, q_k \in Q} c_{\delta_k(f, q_1, q_2, \ldots q_k)} \\ & p_k(f, q_1, q_2, \ldots, q_k) \log p_k(f, q_1, q_2, \ldots, q_k) \ . \end{aligned} \tag{3.32}$$

El cálculo de (3.30) y (3.32) es inmediato si se conocen los coeficientes c_q, que pueden ser evaluados de forma iterativa mediante la relación siguiente:

$$c_i^{[t+1]} = r(i) + \sum_{j \in Q} \Lambda_{ij} c_j^{[t]} \tag{3.33}$$

donde

$$\Lambda_{ij} = \sum_{k=0}^{n} \sum_{f \in V} \sum_{\substack{q_1, q_2, \ldots, q_k \in Q : \\ \delta(f, q_1, \ldots, q_k) = j}} p_k(f, q_1, q_2, \ldots, q_k)(\delta_{iq_1} + \delta_{iq_2} + \cdots + \delta_{iq_k}) \tag{3.34}$$

y para $t = 0$, se elige $c_i^{[0]} = 0$.

3.4 Entropía relativa entre lenguajes de árboles

La entropía relativa entre dos lenguajes de árboles T y T', generados por los autómatas $A = (Q, V, \delta, p, r)$ y $A' = (Q', V, \delta', p', r')$ respectivamente, viene dada por

$$H(T, T') = \sum_{t \in V^T} p(t|A) \log \frac{p(t|A)}{p(t|A')} \,. \tag{3.35}$$

Para el cálculo de esta entropía, definimos la probabilidad $\eta_{qq'}$ de que el nodo $q \in Q$ genere un subárbol t tal que $\delta'(t) = q'$.

Usando los coeficientes η_{ij}, podemos escribir la contribución a la entropía relativa de los términos del tipo $\log r'(q')$ de la siguiente forma:

$$-\sum_{t \in V^T} p(t|A) \ \log r'(\delta'(t)) = \sum_{i \in Q} \sum_{j \in Q'} r(i) \ \eta_{ij} \ \log r'(j) \tag{3.36}$$

De esta forma, obtenemos que la entropía relativa es

$$H(T, T') = H(T) + H_r(T, T') + H_p(T, T') \tag{3.37}$$

donde el término $H_r(T, T')$ es

$$H_r(T, T') = -\sum_{i \in Q} \sum_{j \in Q'} \eta_{ij} \ r(i) \log r'(j) \tag{3.38}$$

y el término $H_p(T, T')$ es

$$\begin{aligned} H_p(T, T') \ = \ & -\sum_{k=0}^{n} \sum_{f \in V} \sum_{i_1, i_2, \ldots, i_k \in Q} \sum_{j_1, j_2, \ldots, j_k \in Q'} c_{\delta_k(f, i_1, i_2, \ldots, i_k)} \\ & p_k(f, i_1, i_2, .., i_k) \log p'_k(f, j_1, j_2, ..., j_k) \ \eta_{i_1 j_1} \eta_{i_2 j_j} \cdots \eta_{i_k j_k} \end{aligned} \tag{3.39}$$

En esta última expresión hemos usado los mismos coeficientes c_i definidos en la sección anterior.

Los coeficientes η_{ij} pueden ser calculados de forma sencilla mediante un procedimiento iterativo:

$$\eta_{ij}^{[t+1]} = \sum_{k=0}^{n} \sum_{f \in V} \sum_{\substack{i_1, i_2, \ldots i_k \in Q : \\ \delta(f, i_1, \ldots, i_k) = i}} \sum_{\substack{j_1, j_2, \ldots, j_k \in Q' : \\ \delta'(f, j_1, \ldots, j_k) = j}} p_k(f, i_1, i_2, \ldots, i_k)\ \eta_{i_1 j_1}^{[t]} \eta_{i_2 j_2}^{[t]} \cdots \eta_{i_k j_k}^{[t]} \tag{3.40}$$

donde para $t = 0$ se toma $\eta_{ij}^{[0]} = 0$.

En la figura 3.2 se observa cómo la entropía relativa entre lenguajes de árboles converge muy lentamente al valor correcto cuando éste se estima a partir de muestras. Dicha gráfica representa la entropía relativa de dos gramáticas sencillas (con seis reglas cada una) que generan expresiones aritméticas:

expresión $\rightarrow$ *expresión* $+$ *término*

expresión $\rightarrow$ *término*

término $\rightarrow$ *término factor*

término $\rightarrow$ *factor*

factor $\rightarrow$ **número**

factor $\rightarrow$ (*expresión*)

donde las variables aparecen en cursiva y los terminales en negrita. Los resultados muestran que incluso con muestras enormes, el valor obtenido para la entropía está lejos del correcto. La convergencia es mucho peor que en el caso de los lenguajes de cadenas. Esto puede explicarse si se observa el hecho de que el número de elementos en un lenguaje de árboles es enorme comparado con el caso de las cadenas. Por ejemplo, mientras el número de cadenas binarias de longitud máxima L es $2^{L+1} - 1$, el de árboles binarios (es decir, con dos o cero descendientes por nodo) etiquetados con $V = \{0, 1\}$ y de profundidad

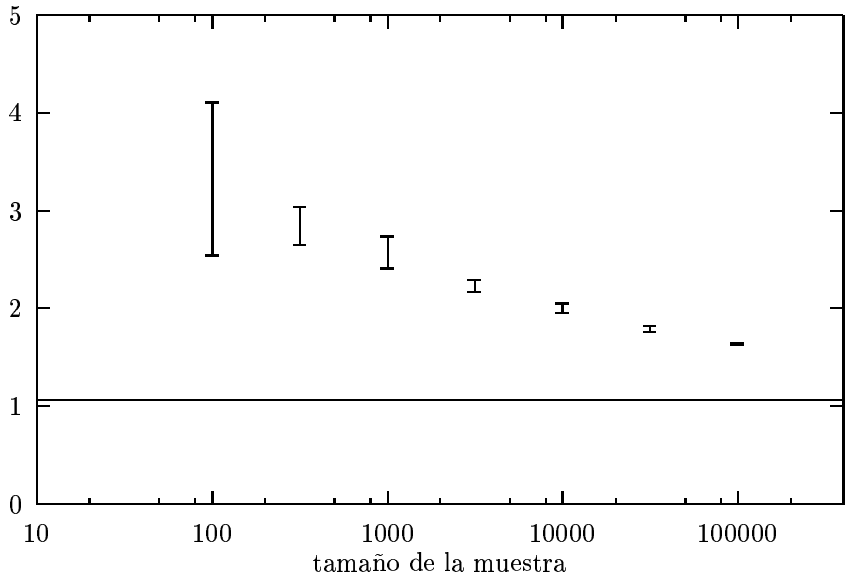

Figura 3.2: Entropía relativa en bits entre dos gramáticas de árboles con seis reglas cada una. Línea continua: cálculo exacto. Puntos: estimación mediante muestras.

máxima d crece más rápido que 2 elevado a 2^d, es decir, mientras el primero crece exponencialmente el segundo lo hace de forma exponencialmente exponencial.

3.5 Convergencia de una variable aleatoria

Los procesos aleatorios presentan una marcada tendencia a la regularidad (la llamada regularidad estadística), sobre todo cuando el número de experimentos es grande. De hecho, es posible establecer distintas cotas superiores para la diferencia entre los valores de una variable aleatoria x obtenidos a partir de una muestra y su valor esperado $\mu = \mathrm{E}(x)$. Una de estas cotas es la expresada por la desigualdad de Chebychev (Feller 1950). Recordemos que, dada un distribución de

probabilidad $f(x)$, el valor esperado de la variable aleatoria x es:

$$\mathrm{E}(x) = \int dx\ f(x)\ x \tag{3.41}$$

En particular, $\mathrm{E}(x-\mu) = 0$, por lo que normalmente se utiliza $(x-\mu)^2$ para medir la dispersión de la variable x. Al valor esperado de $(x-\mu)^2$ se le denomina *varianza* de x:

$$\mathrm{Var}(x) = \int dx\ f(x)(x-\mu)^2 \tag{3.42}$$

Dado que el integrando en(3.42) es siempre positivo, si excluimos de la integral un entorno del valor esperado μ se obtiene la siguiente desigualdad:

$$\begin{aligned} \mathrm{Var}(x) &\geq \int_{|x-\mu|\geq\epsilon} dx\ f(x)(x-\mu)^2 \geq \\ &\geq \int_{|x-\mu|\geq\epsilon} dx\ f(x)\ \epsilon^2 = \epsilon^2\, p\left(|x-\mu| \geq \epsilon\right) \end{aligned} \tag{3.43}$$

De donde se deduce que la probabilidad α de que la variable difiera de la media en un valor mayor que ϵ esta acotada:

$$\alpha = p\left(|x-\mu| \geq \epsilon\right) \ \leq\ \frac{\mathrm{Var}(x)}{\epsilon^2} \tag{3.44}$$

Por tanto, $\mathrm{Var}(x)$ es una medida de la variabilidad de x. A la probabilidad α se le llama *nivel de significación* y al valor $1-\alpha$ se le denomina *nivel de confianza*. En general interesa que α sea pequeño (por ejemplo, menor que 0.1). Esto es posible si la varianza es pequeña, o bien si se toma un intervalo suficientemente amplio (es decir, ϵ suficientemente grande).

Con frecuencia, se fija el valor de α y se busca el intervalo más pequeño que satisface (3.44). En ese caso, una forma más conveniente de escribir (3.44) es:

$$p\left(|x-\mu| < \sqrt{\frac{\mathrm{Var}(x)}{\alpha}}\right) > 1-\alpha \tag{3.45}$$

Es decir, el resultado de un experimento de la variable x se encuentra con probabilidad mayor que $1-\alpha$ a una distancia menor que $\epsilon = \sqrt{\frac{1}{\alpha}\mathrm{Var}(x)}$ del valor esperado μ.

Es sencillo comprobar que $\overline{x}$, la media aritmética obtenida después de n experimentos aleatorios de la variable x, es una nueva variable aleatoria con valor esperado $\mathrm{E}(\overline{x}) = \mathrm{E}(x)$ y varianza $\mathrm{Var}(\overline{x}) = \frac{1}{n}\mathrm{Var}(x)$. Por tanto, reescribiendo la ecuación anterior para el caso particular de $\overline{x}$ obtenemos:

$$p\left(|\overline{x}-\mu| < \sqrt{\frac{\mathrm{Var}(x)}{n\alpha}}\right) > 1-\alpha \tag{3.46}$$

Otra cota para esta diferencia viene dada por el límite de Hoeffding (Hoeffding 1963), que es válido para variables de Bernoulli, es decir, aquellas que sólo pueden presentar dos resultados: éxito o fracaso. Sea x una variable de Bernoulli con probabilidad de éxito p y sea

$$\epsilon_\alpha(n) = \sqrt{\frac{1}{2n}\log\frac{2}{\alpha}}\,. \tag{3.47}$$

Entonces,

$$p(|\bar{x}-p| < \epsilon_\alpha(n)) > 1-\alpha \tag{3.48}$$

donde $\bar{x} = f_n(x)/n$ y $f_n(x)$ representa el número de éxitos después de n experimentos.

La distribución asociada a una variable de Bernoulli es la distribución binomial que, para grandes valores de n, es aproximadamente una distribución gaussiana o normal. La *distribución normal* para una variable de media cero y varianza uno es

$$p(x>a) = \frac{1}{\sqrt{2\pi}}\int_a^\infty dx\ \exp(-\frac{x^2}{2}) \tag{3.49}$$

y por tanto

$$\begin{aligned} p(x>a) &= \frac{1}{\sqrt{2\pi}}\int_0^\infty dt\ \exp(-\frac{(t+a)^2}{2}) = \\ &= \frac{1}{\sqrt{2\pi}}\int_0^\infty dt\ \exp(-\frac{t^2}{2})\exp(-at)\exp(-\frac{a^2}{2}) \leq \end{aligned}$$

$$\begin{aligned} &\leq \frac{1}{\sqrt{2\pi}} \int_0^\infty dt \, \exp(-\frac{t^2}{2}) \exp(-\frac{a^2}{2}) = \\ &= \exp(-\frac{a^2}{2}) \end{aligned} \tag{3.50}$$

Si definimos $\alpha = p(|x| > \epsilon)$, entonces

$$\epsilon = \sqrt{2 \log \frac{2}{\alpha}} \tag{3.51}$$

es la anchura del intervalo que garantiza que $x \in [-\epsilon, \epsilon]$ con probabilidad mayor que $1 - \alpha$.

Por otro lado, para una variable de Bernoulli x con probabilidad p, la varianza de $\bar{x}$ es $p(1-p)/n$ que es siempre menor o igual que $1/4n$. Por tanto, es sencillo comprobar que el rango asociado a $\bar{x}$ es:

$$\epsilon = \sqrt{\frac{1}{2n} \log \frac{2}{\alpha}} \tag{3.52}$$

que coincide con el valor de la cota (3.47). Importa destacar que la cota de Hoeffding no es únicamente válida cuando n es grande, sino que es válida para cualquier valor de n (Hoeffding 1963).

Otro límite de interés es la regla del logaritmo iterado (Feller 1950), que permite escribir que con probabilidad 1

$$|\bar{x} - p| < \sqrt{\frac{\log \log n}{n}} \tag{3.53}$$

excepto para un número finito de valores de n.

3.6 Aproximación contra identificación

Es llamativo el hecho de que puede definirse un procedimiento para determinar (en el límite) si un número es racional o irracional (Cover 1973). Sin embargo, en un ordenador los números reales son representados mediante conjuntos finitos de bits. Esto hace que, a efectos prácticos, podamos tratar las probabilidades como números racionales. Por otro lado, calcular una probabilidad con un método

que garantice su identificación en el límite no siempre es preferible a simplemente estimar dicha probabilidad. En efecto, si aproximamos la probabilidad $p(x)$ mediante el cociente $f_n(x)/n$, donde $f_n(x)$ es el número de veces que se observa el suceso $x \in \Omega$ en una serie de n experimentos, obtenemos con frecuencia un valor que se acerca al valor real $p(x)$ mucho más rápidamente que el proporcionado por el método que identifica el valor correcto en el límite. Si bien, para un número suficiente de experimentos, la identificación en el límite proporciona el resultado exacto (siempre y cuando la probabilidad sea un número racional), el número de experimentos necesarios para que esto ocurra puede ser muy grande. No debemos olvidar que, en la práctica, sólo disponemos de muestras de tamaño finito y a menudo no demasiado grandes.

Por ejemplo, aunque el algoritmo de la figura 3.3 garantiza la identificación en el límite, de la probabilidad buscada, ésta se produce en general demasiado tarde como para que resulte interesante su utilización en experimentos reales. La comparación de las gráficas 3.6 y 3.7 muestra que, salvo para valores racionales muy sencillos para $p(x)$, la identificación necesita un número de experimentos demasiado grande como para que resulte práctica. La gráfica 3.8, sugiere que en el caso de que la probabilidad sea irracional, puede ser siempre preferible la estimación del valor de $p(x)$.

Esta conclusión se puede generalizar inmediatamente al caso en el que el número de probabilidades a considerar es finito. Sin embargo, en los problemas que estudiaremos más adelante encontraremos que el número de probabilidades a determinar es a menudo infinito. En consecuencia, investigaremos la forma de reducir este número de forma que sea finito.

Para terminar esta sección, justificaremos que el algoritmo 3.3 garantiza la identificación en el límite de la probabilidad $p(x)$ si su valor es un número racional.

En el algoritmo 3.3, la función `next`(q) proporciona el siguiente número racional a uno dado q en el intervalo [0,1], siguiendo una or-

denación inspirada en el método de Cantor que recorre todo $\mathbb{Q} \cup [0,1]$. La función $\texttt{diff}(f, n, q)$ utiliza la prueba del logaritmo iterado (3.53) que garantiza que si $q = p(x)$ es el racional correcto el resultado es incorrecto (devuelve FALSO) para un número finito de valores de n.

Por otro lado, si $q = p(x)$, $q' \neq p(x)$ y

$$|q - q'| > 2\sqrt{\frac{\log\log n}{n}} \tag{3.54}$$

entonces, aplicando la desigualdad triangular a la regla del logaritmo iterado (3.53) se obtiene que

$$\left|\frac{f}{n} - q'\right| > \sqrt{\frac{\log\log n}{n}} \tag{3.55}$$

excepto para un número finito de valores de n. Como el cociente $\log\log n/n$ tiende a cero, la condición (3.54) se satisface siempre que n sea lo suficientemente grande. Por consiguiente, cualquier valor $q' \neq p(x)$ es rechazado para un n suficientemente grande.

Dada una ordenación $\{q_1, q_2, ...\}$ de los racionales en [0,1], sólo existe un número finito de racionales anteriores al valor buscado $q_k = q$, y por ello, sólo hay que seleccionar una cota inferior n_0 para n entre un conjunto finito de valores. Para cualquier $n > n_0$, todos los valores $q' \neq p(x)$ son rechazados y $q = p(x)$ es aceptado, es decir, se alcanza la identificación en el límite de $p(x)$.

```
algorithm identifica_p
input: f (número de observaciones de x)
       n (número de experimentos)
output: q ∈ ℕ × ℕ ≃ ℚ

begin algorithm
  q = (0, 1)
  do ( mientras diff(q, f, n) )
     q = next(q)
  end do
  return q
end algorithm
```

Figura 3.3: Algoritmo que identifica $p(x) \in \mathbb{Q}$.

```
algorithm diff
input: f ∈ ℕ
       n ∈ ℕ
       q = (x, y) ∈ ℕ × ℕ
output: boolean

begin algorithm
  return |f/n − x/y| > √((log log n)/n)
end algorithm
```

Figura 3.4: Algoritmo que compara q y f/n.

```
algorithm next
input :q = (x, y) ∈ ℕ × ℕ
output :q' ∈ ℕ × ℕ

begin algorithm
```

$$q' = \begin{cases} (x+1, y-1) & \text{si } x+1 \leq y-1 \\ (1, x+y) & \text{en caso contrario} \end{cases}$$

```
  return q'
end algorithm
```

Figura 3.5: Siguiente racional a q en $[0, 1]$.

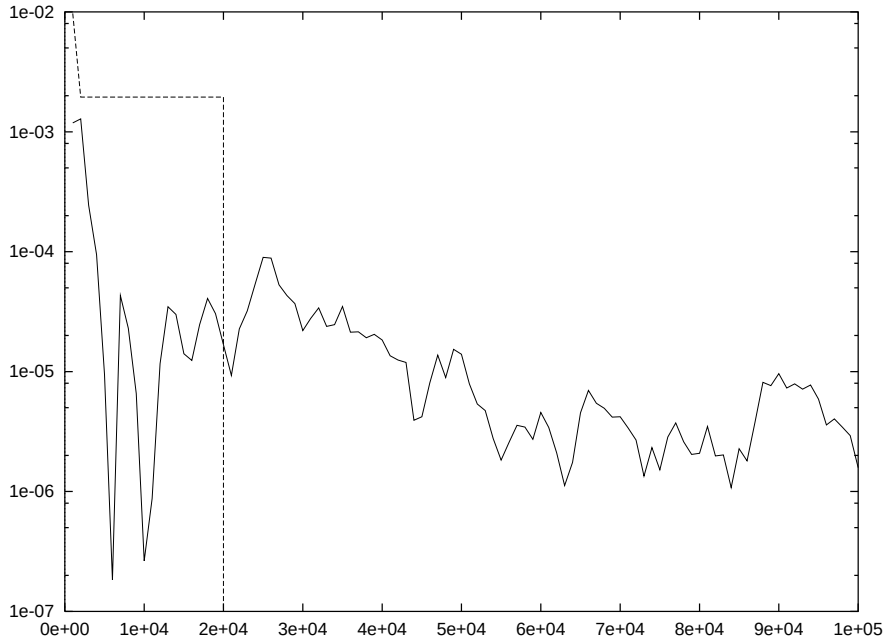

Figura 3.6: Entropía relativa entre $p(x) = 0.875 = \frac{7}{8}$ y la probabilidad experimental en función del número de experimentos. Línea continua: resultado de la estimación numérica. Línea de puntos: algoritmo de identificación. Esta última baja hasta cero a partir de 20000 experimentos.

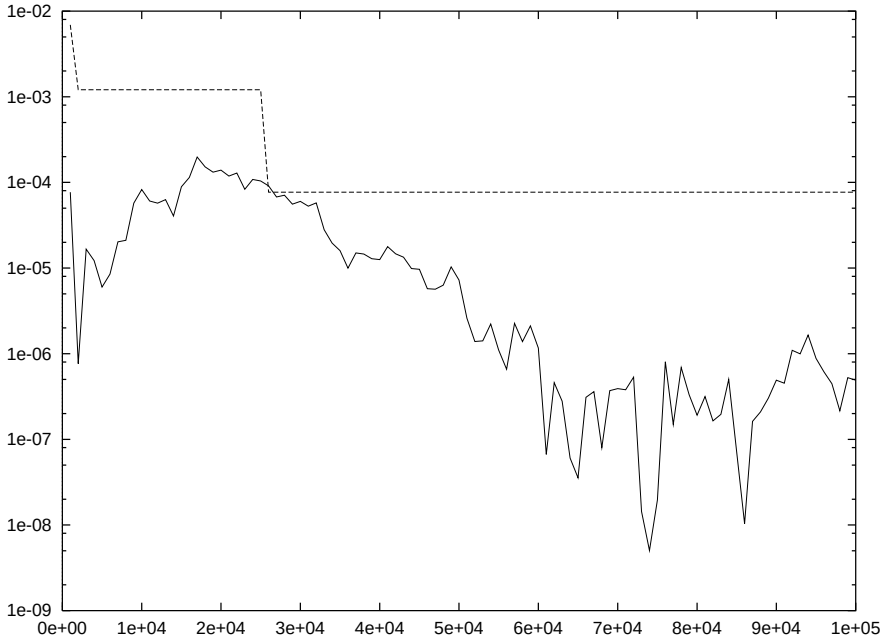

Figura 3.7: Entropía relativa entre $p(x) = 0.62$ y la probabilidad experimental en función del número de experimentos. Línea continua: estimación numérica. Línea de puntos: algoritmo de identificación.

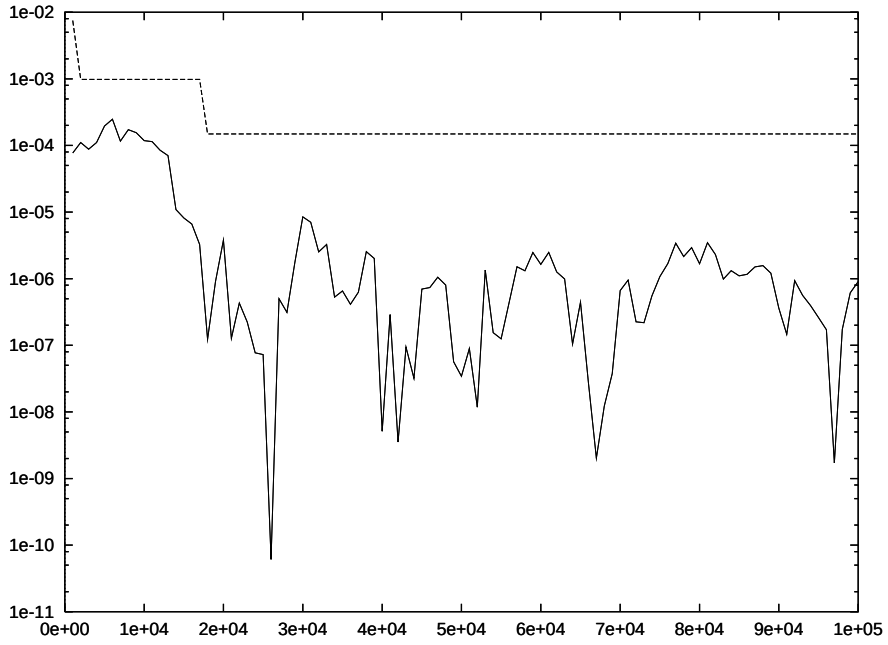

Figura 3.8: Entropía relativa entre $p(x) = \Phi = (-1 + 5^{\frac{1}{2}})/2$ y la probabilidad experimental en función del número de experimentos. Línea continua: estimación numérica. Línea de puntos: algoritmo de identificación.

Capítulo 4

Procedimientos clásicos de aprendizaje

Surely it is enough that the likes of you and I at least try to make our small contribution count for something true and worthy.
Kazuo Ishiguro, The remains of the day.

Existe una larga tradición de aplicación de los métodos estocásticos al aprendizaje. En este capítulo revisaremos aquellos que presentan una relación más estrecha con los algoritmos que vamos a desarrollar. En particular, los modelos de Markov ocultos, entrenados frecuentemente con el método de Baum-Welch, y los métodos bayesianos que determinan automáticamente el número de estados del modelo mediante mecanismos de fusión.

4.1 Modelos de Markov ocultos

Los modelos de Markov constituyen modelos probabilísticos para sucesos que ocurren secuencialmente en el tiempo. Por este motivo, han

sido utilizados con frecuencia en problemas como los de reconocimiento del habla (Rabiner 1989) y los de modelización del lenguaje natural (Brown *et al.* 1992, Suen 1979, Stolcke & Segal 1994).

Una *cadena de Markov* es un suceso aleatorio x cuya probabilidad sólo depende de un número predeterminado n de sucesos anteriores (Chung 1967), es decir:

$$\begin{aligned} p(x_t = o_t | x_{t-1} = o_{t-1}, ..., x_1 = o_1) &= \\ p(x_t = o_t | x_{t-1} = o_{t-1}, ..., x_{t-n} = o_{t-n}) & \end{aligned} \tag{4.1}$$

Este tipo de modelos ha sido utilizado con frecuencia en el tratamiento de lenguaje natural, donde son denominados modelos de n-gramas. En ellos la probabilidad de aparición de una palabra w_k está determinada por las $n-1$ palabras anteriores. Habitualmente, se eligen los modelos basados en bigramas ($n = 2$), trigramas ($n = 3$) o combinaciones de ambos (Charniak 1994). Por ejemplo, en el caso de los trigramas, la probabilidad $p_3(w_k)$ de la palabra w_k viene dada por:

$$p_3(w_k) = p(w_k | w_{k-2} w_{k-1}) \tag{4.2}$$

Para que las probabilidades de w_1 y w_2 puedan ser determinadas mediante la fórmula anterior, es necesario suponer que el texto viene precedido de unas pseudo-palabras w_{-1} y w_0. No es este el único caso en el que la falta de datos puede originar problemas: normalmente los datos están muy dispersos y hasta el 25% de los trigramas en un texto pueden no haber aparecido durante la etapa de entrenamiento (Charniak 94). Por todos estos motivos, frecuentemente se toma una combinación de modelos del tipo:

$$\hat{p}(w_k) = \lambda_1 p_1(w_k) + \lambda_2 p_2(w_k) + \lambda_3 p_3(w_k) \tag{4.3}$$

donde los parámetros λ_n deben ser elegidos automáticamente y pueden depender de la cantidad de cada tipo de n-gramas observados en la fase de entrenamiento. La interpolación entre los modelos es una cuestión aún controvertida (Kneser & Ney 1995).

Una forma de soslayar este tipo de dificultades es utilizar modelos de Markov ocultos, una generalización de las cadenas de Markov. En ellos, dada una observación previa, más de un estado (en el sentido de conjuntos distintos de probabilidades para las palabras siguientes) es posible y de ahí el nombre de modelos ocultos. Formalmente, un *modelo de Markov oculto* es una cuádrupla $M = (Q, I, \mathcal{A}, T)$ compuesta por:

- un conjunto finito de estados que denotaremos mediante su número de orden $Q = \{1, 2, ..., N\}$;
- un estado $I \in Q$ que es marcado como estado inicial;
- un conjunto finito $\mathcal{A}$ de símbolos de salida;
- un conjunto T de transiciones.

Una transición $t = (i, j, a, p_{ij}(a)) \in T$ contiene un estado i de partida, un estado j de llegada, un símbolo $a \in \mathcal{A}$ y una probabilidad $p_{ij}(a)$ de que a partir del estado i se efectúe una transición al estado j y se genere un símbolo a (si utilizamos el modelo para asignar una probabilidad a cada cadena, diremos más bien que se procesa el símbolo a de la cadena de entrada). Si bien no se admiten transiciones redundantes (es decir, transiciones con el mismo valor para los estados y el símbolo), no hay, en cambio, ninguna restricción sobre el número de transiciones que con el mismo símbolo a parten de un estado dado i. Con frecuencia, se parte de la suposición de que existen transiciones desde cada estado y con cada símbolo a todos los estados, aunque la probabilidad de algunas de ellas puede ser cero posteriormente. En este sentido, los modelos ocultos de Markov son más generales que los autómatas finitos deterministas estocásticos definidos anteriormente, y se corresponden más bien con los autómatas indeterministas estocásticos[1] una clase de

[1] Esto es, autómatas finitos indeterministas (Hopcroft & Ullman 1979) que incorporan una función de probabilidad de transición al estilo de la definida en la sección 2.4.

autómatas que no se puede reducir a la de los autómatas deterministas estocásticos, aunque la incluye propiamente.

La probabilidad de generación de una cadena $w = a_0 a_2 ... a_{L-1}$ de longitud L por el modelo M es

$$p(w|M) = \sum_{c(I,L)} \prod_{k=0}^{L-1} p_{i_k i_{k+1}}(a_k) \tag{4.4}$$

donde $c(I, L)$ representa todas las secuencias $i_0 i_2 ... i_L$ que describen caminos de longitud L entre los estados de Q que comienzan en $i_0 = I$. Estas probabilidades, así como el camino más probable para una cadena dada, pueden ser computadas de forma eficiente utilizando el algoritmo de Viterbi (1967).

Los modelos de Markov ocultos pueden ser entrenados utilizando el algoritmo de Baum y Welch (Baum 1972). Este procedimiento recalcula iterativamente las probabilidades de transición que efectúa el modelo M al analizar una cadena $w = a_0 a_2 ... a_{L-1}$. Para describir este algoritmo necesitamos definir previamente las siguientes magnitudes:

1. La *probabilidad hacia adelante* $\alpha_i(t)$ es la probabilidad de que M genere la subcadena $a_0 a_1 a_2 ... a_{t-1}$ terminando en el estado i, y está definida para $t = 0, 1, ..., L$. Las probabilidades $\alpha_i(t)$ pueden ser calculadas en un tiempo lineal en función de L de la siguiente forma. Para $t = 0$:

$$\alpha_i(t = 0) = \delta_{iI}, \tag{4.5}$$

y procediendo iterativamente, para $t = 1, ...L$:

$$\alpha_i(t) = \sum_j \alpha_j(t-1) p_{ji}(a_{t-1}). \tag{4.6}$$

Por ejemplo, $\alpha_i(t = 1) = \sum_j \delta_{jI} p_{ji}(a_0) = p_{Ii}(a_0)$, tal y como cabe esperar.

2. La *probabilidad hacia atrás* $\beta_i(t)$ se define como la probabilidad de que M genere la cadena $a_t a_{t+1} ... a_{L-1}$ partiendo del estado

i. Estas probabilidades pueden calcularse también en tiempo lineal, pues para $t = L$

$$\beta_i(t = L) = 1, \tag{4.7}$$

y para $t = 0, 1, ..., L - 1$

$$\beta_i(t) = \sum_j \beta_j(t+1) p_{ij}(a_t). \tag{4.8}$$

Por ejemplo, $\beta_i(L-1) = \sum_j p_{ij}(a_{L-1})$.

3. La probabilidad de que M genere $w = a_0 a_1 ... a_{L-1}$ es:

$$p(w|M) = \beta_I(0), \tag{4.9}$$

y también

$$p(w|M) = \sum_i \alpha_i(L). \tag{4.10}$$

Las dos ecuaciones anteriores no son más que un caso particular de la expresión general:

$$p(w|M) = \sum_i \alpha_i(t)\beta_i(t). \tag{4.11}$$

Si en la expresión anterior sustituimos $\beta_i(t)$ siguiendo (4.8), obtenemos

$$p(w|M) = \sum_{ij} \alpha_i(t) p_{ij}(a_t) \beta_j(t+1), \tag{4.12}$$

expresión que nos resultará útil más adelante.

4. La probabilidad $\gamma_i(t)$ de que, al generar w, el t-ésimo estado en la secuencia de $L+1$ estados sea i es proporcional a $\alpha_i(t)\beta_i(t)$. Teniendo en cuenta (4.11) para la normalización, obtenemos:

$$\gamma_i(t) = \frac{1}{p(w|M)} \alpha_i(t)\beta_i(t). \tag{4.13}$$

5. La probabilidad $\xi_{ij}(t)$ de que, al generar w, la transición causada por a_t ocurra entre los estados i y j es proporcional a $\alpha_i(t)p_{ij}(a_t)\beta_j(t+1)$. De nuevo, teniendo en cuenta (4.12) a efectos de normalización, obtenemos:

$$\xi_{ij}(t) = \frac{1}{p(w|M)}\alpha_i(t)p_{ij}(a_t)\beta_j(t+1) \tag{4.14}$$

Con estas definiciones, resulta sencillo describir el algoritmo de entrenamiento de Baum y Welch. En este procedimiento se recalculan iterativamente las probabilidades de transición de la siguiente forma:

$$p'_{ij}(a) = \frac{\sum_{t=0}^{L-1} \xi_{ij}(t)\delta_{a_t a}}{\sum_{t=0}^{L-1} \gamma_i(t)} \tag{4.15}$$

hasta que se llegue a las proximidades de un punto fijo. En el caso de que se disponga de una muestra $S = \{w_1, w_2, ..., w_n\}$ para el entrenamiento compuesta de numerosas cadenas, los sumatorios, tanto en el numerador como en el denominador de la ecuación (4.15), deben sumar también sobre las diferentes cadenas w_k de la muestra.

Este procedimiento garantiza que las nuevas probabilidades mejoran la verosimilitud de la muestra, $p(S|M)$, con lo que finalmente debe alcanzarse un mínimo. No existe garantía sin embargo, de que este mínimo sea global y no local. Esto constituye la mayor dificultad del procedimiento, pues los experimentos prueban que su buen rendimiento requiere una estimación del tamaño del modelo que se debe inferir (al menos, de su tamaño máximo) y alguna información sobre la estructura de este modelo para evitar que los valores iniciales de las probabilidades $p_{ij}(a)$ sean elegidos meramente al azar. Nótese que, si existe más de una representación posible del problema, el método de Baum-Welch no tiene ninguna razón para optar por la representación más sencilla. Otra dificultad importante en el aprendizaje de lenguajes estocásticos ha sido puesta de manifiesto por Abe y Warmuth (1992): tanto los autómatas probabilísticos como los modelos ocultos de Markov no pueden ser entrenados en un tiempo polinómico en función del tamaño del alfabeto $\mathcal{A}$.

Por último, merece la pena destacar que en los modelos ocultos, a diferencia de las cadenas de Markov, la probabilidad de un suceso puede depender de símbolos anteriores muy alejados en el tiempo. Por ejemplo, una cadena de Markov no puede describir un comportamiento tan sencillo como el siguiente:

$$p(w_k) = \begin{cases} p_1 & \text{si } k \text{ es par} \\ p_2 & \text{en caso contrario} \end{cases} \tag{4.16}$$

Este tipo de comportamientos, sin embargo, se puede describir adecuadamente utilizando los modelos de Markov ocultos o modelos basados en autómatas finitos estocásticos como los descritos en el capítulo 2.

4.2 Modelos bayesianos y fusión de estados

Supongamos que existe $\{M_0, M_1, M_2, ...\}$, un espacio de hipótesis o modelos mutuamente excluyentes (como mucho uno es correcto) y exhaustivas (al menos uno es correcto). Dado un conjunto de datos observado X, existen, al menos, dos criterios basados en la teoría de probabilidades para elegir la mejor hipótesis:

- verosimilitud máxima (VM), esto es, elegir el modelo M tal que la probabilidad $p(X|M)$ es máxima;
- probabilidad *a posteriori* máxima (PAM), es decir, elegir el modelo M que hace máxima la probabilidad $p(M|X)$.

Para estudiar la relación entre ambos criterios, supongamos que existe alguna forma de asignar una probabilidad independiente de los datos experimentales a cada modelo M. Esta distribución $p(M)$ es lo que se denomina la probabilidad *a priori*. Entonces, la probabilidad de que M sea el modelo correcto, dado que como resultado de un experimento hemos obtenido la muestra finita X, es

$$p(M|X) = \frac{p(X|M)p(M)}{p(X)} \tag{4.17}$$

expresión que se conoce con el nombre de *teorema de Bayes* y que es consecuencia directa de la definición de probabilidad condicionada

$$p(X|Y) = \frac{p(X \cap Y)}{p(Y)} \ . \tag{4.18}$$

Dado que el denominador $p(X)$ no depende del modelo elegido, el criterio bayesiano de inferencia (PAM) es una generalización del criterio de verosimilitud máxima (VM). En particular, éste último se corresponde con el caso en que la probabilidad $p(M)$ es constante, es decir, todos los modelos M son *a priori* equiprobables.

Las probabilidades *a priori* $p(M)$ permiten introducir de forma natural en el proceso de aprendizaje el grado de creencia asociado a los diferentes modelos. De esta manera se puede incorporar, por ejemplo, la preferencia por los modelos más sencillos. En efecto, tomando logaritmos en la expresión (4.17), se observa que maximizar $p(M|X)$ es equivalente a minimizar

$$-\log p(M|X) = -\log p(M) - \log p(X|M) \tag{4.19}$$

procedimiento que puede interpretarse como la minimización de la longitud de la descripción de los datos X conjuntamente con la del modelo de codificación M. De hecho $-\log p(M)$ es la longitud óptima de codificación del modelo dada la distribución *a priori* mientras que $-\log p(X|M)$ se corresponde con la longitud del código óptimo para los datos X usando M como modelo probabilístico. A la inversa, cualquier esquema de codificación que asigna a M un código de longitud $\lambda(M)$ puede ser utilizado para generar una distribución *a priori* de los modelos:

$$p(M) \propto \exp^{-\lambda(M)} \tag{4.20}$$

Por ejemplo, la forma más natural de codificar un modelo de Markov es simplemente enumerar sus transiciones, por lo que podemos tomar $\lambda(M)$ como el número de transiciones del modelo.

En el caso de los autómatas finitos estocásticos (o modelos de Markov ocultos), la verosimilitud máxima se consigue cuando el modelo

predice como probabilidad para cada cadena la frecuencia relativa observada en el experimento. En efecto, si el número de cadenas de tipo w en X es n_w, y la probabilidad asignada por M a w es p_w, hay que minimizar

$$-\log p(X|M) \propto -\sum_w n_w \log p_w \tag{4.21}$$

sometido a la restricción $\sum_w p_w = 1$, lo que corresponde a una minimización sobre una variedad lineal. Utilizando un multiplicador de Lagrange $\lambda \sum_w \Delta p_w = 0$, la variación de $p(X|M)$ cerca del mínimo es

$$\Delta \log p(X|M) = \sum_w (\frac{n_w}{p_w} - \lambda)\Delta p_w = 0 \tag{4.22}$$

donde, gracias al multiplicador de Lagrange, todas las variaciones Δp_w son independientes, lo que implica que cada término se anula, y por lo tanto

$$p_w = \frac{n_w}{\lambda} \ . \tag{4.23}$$

Es decir, cada probabilidad es proporcional al número de observaciones de esa cadena, y λ es el número total de cadenas en la muestra para que la normalización de la probabilidad sea la correcta.

Por tanto, la verosimilitud máxima se consigue tomando como modelo un lenguaje finito que puede ser generado, por ejemplo, mediante un autómata con estructura de árbol en el que cada nodo corresponde a uno de los prefijos observados (véase la figura 4.1) y las probabilidades de transición se eligen iguales a las frecuencias relativas en la muestra de los símbolos que siguen a cada prefijo. Modelos más pequeños (es decir, autómatas con un número menor de estados) conducen necesariamente a valores menores de la verosimilitud. A pesar de ello, esta disminución puede resultar compensada por la predisposición hacia los modelos más sencillos contenida en las probabilidades *a priori*. De esta forma es posible optar por modelos más sencillos aunque sean de verosimilitud menor.

Un algoritmo para la inferencia de modelos de Markov ocultos basado en la fusión de estados y la utilización de probabilidades *a priori*

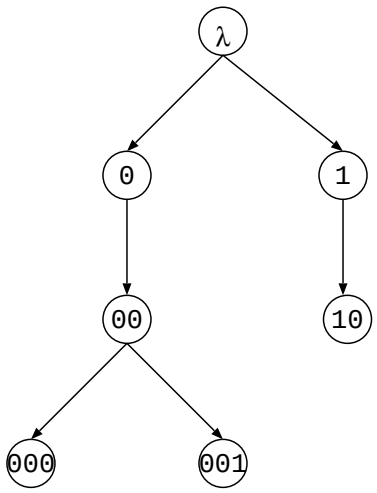

Figura 4.1: Árbol de prefijos para el conjunto $\{0, 10, 000, 001\}$

ha sido propuesto por Stolcke y Omohundro (1993). Su algoritmo puede describirse de la siguiente manera:

1. construir el modelo M_0 de verosimilitud máxima para el conjunto de datos X;

2. reiterar desde $i = 0$:

   (a) determinar el conjunto K de $|M_i|(|M_i|-1)$ posibles fusiones de estados en M_i;

   (b) para cada $k \in K$ calcular el nuevo modelo $k(M)$ y su probabilidad a posteriori $p(k(M_i)|X)$;

   (c) elegir la fusión k^* que maximiza $p(k(M_i)|X)$ y tomar como $M_{i+1} = k^*(M_i)$.

   (d) terminar si $p(M_i + 1|X) < p(M_i|X)$. En caso contrario continuar con $i = i + 1$.

Con este algoritmo, la disminución de verosimilitud provocada por la fusión de estados del modelo queda compensada por la preferencia expresada por la probabilidad *a priori* por los modelos más sencillos.

El algoritmo introduce también una probabilidad *a priori* para los parámetros del modelo (las probabilidades de transición) en forma de una distribución de Dirichlet. Esto es equivalente a admitir un conjunto de ejemplos virtuales para cada transición que se añaden a los reales a la hora de calcular los parámetros más probables. Los resultados descritos en Stolcke & Omohundro (1994) demuestran que el procedimiento proporciona buenos resultados para modelos sencillos y muestras pequeñas, comparables o mejores a los obtenidos con un entrenamiento del tipo Baum-Welch. El procedimiento no ha sido estudiado con muestras grandes y autómatas de tamaño grande debido a su elevado coste: nótese que cada iteración requiere $|M_i|^2$ evaluaciones de la verosimilitud y que $|M_i|$ puede ser lineal con el tamaño de X. Una diferencia fundamental entre este método y el que proponemos en el capítulo siguiente es que mientras el prime método no garantiza la identificación en el límite del lenguaje correcto (es decir, del conjunto de cadenas con probabilidad no nula), el nuevo método sí que presenta esta propiedad.

Capítulo 5

Inferencia de lenguajes regulares

La probabilidad a priori *de que se produzca un acontecimiento particular entre todos los posibles en el universo está próxima a cero... Nuestro número salió en el juego de Montecarlo.*
Jaques Monod, El azar y la necesidad

En este capítulo, estudiaremos la identificación de lenguajes regulares estocásticos de cadenas. Como resultado se propondrá un conjunto de algoritmos que comparan de forma sistemática los estados del árbol de prefijos de la muestra. Esta clase de algoritmos permite identificar en el límite el conjunto de cadenas que forman el lenguaje y además calcular con gran precisión sus probabilidades de aparición en el lenguaje. El tiempo necesario para que el algoritmo proporcione una hipótesis aumenta sólo linealmente con el tamaño de la muestra y, experimentalmente, la implementación desarrollada se revela como extraordinariamente rápida con vistas a su posible aplicación en tareas de reconocimiento.

5.1 Antecedentes

La identificación de lenguajes estocásticos de cadenas ha sido tratada en numerosa ocasiones anteriormente. Por ejemplo, Maryanski y Booth (1977) usaron un test del tipo χ^2 para filtrar gramáticas regulares que eran generadas mediante procedimientos heurísticos. Con este método se obtenían gramáticas razonables, pero sin garantizar la identificación en el límite de la gramática correcta. Cook, Rosenfel y Aronson (1976) definieron una función de similitud entre gramáticas estocásticas basada en la teoría de la información para buscar gramáticas sencillas que minimicen la distancia a la muestra.

El método de van der Mude y Walker (1978) fusiona variables de una gramática estocástica regular y utiliza criterios bayesianos para permitir estas fusiones. Los autores no demuestran que su algoritmo converja a la gramática correcta y además su implementación es extremadamente lenta como para poder ser aplicada en la práctica.

Recientemente, se han utilizado modelos con redes neurales para identificar lenguajes regulares (Smith & Zipser 1989, Pollack 1991, Giles 1992, Watrous & Kuhn 1992) y en algunos casos se han aplicado al caso de muestras estocásticas (Castaño, Casacuberta & Vidal 1993). Sin embargo, estos métodos comparten la desventaja de que requieren largos tiempos de cálculo y con frecuencia muestras muy grandes.

Los modelos ocultos de Markov, esencialmente equivalentes a los autómatas finitos estocásticos, han sido utilizados también con frecuencia en el tratamiento de lenguajes estocásticos. Su mayor desventaja reside en que es necesario prever de antemano el número de estados del modelo e introducir alguna información adicional sobre su estructura para que el entrenamiento no conduzca a un mínimo local. Un sistema para determinar automáticamente el número de estados del modelo de Markov ha sido desarrollado por Stolcke y Omohundro (1993). Su método intenta maximizar la probabilidad de la muestra utilizando a la vez una probabilidad *a priori* que penaliza los tamaños grandes del autómata. Tampoco estos procedimientos garantizan la

convergencia al modelo correcto.

Recientemente, Oncina y García (1992) han propuesto un algoritmo que permite la identificación correcta en el límite de cualquier lenguaje regular siempre y cuando se disponga de muestras completas. Además, el algoritmo funciona en tiempo polinómico con el tamaño de la muestra. En la práctica, el tiempo que requiere para producir su hipótesis crece sólo linealmente con el número de ejemplos en los experimentos. En este capítulo, se siguen las mismas líneas y se presenta un algoritmo (`rlips`) que construye el árbol de prefijos de la muestra y evalúa en cada nodo las probabilidades de transición de los arcos que salen de ese nodo. A continuación, se comparan parejas de nodos siguiendo un orden bien establecido (básicamente el de los niveles de profundidad en el árbol). Se acepta que dos nodos son equivalentes cuando generan —dentro de la incertidumbre estadística— el mismo sublenguaje estocástico. El proceso continúa hasta que no es posible establecer más equivalencias. Las definiciones necesarias para este capítulo se encuentran en la sección 5.2. El algoritmo es descrito en detalle en la sección 5.3 y se demuestra que es correcto en la sección 5.4. Finalmente, los resultados y su discusión son el contenido de la última sección de este capítulo.

5.2 Formalismo

Sea $\mathcal{A}$ un alfabeto finito, $\mathcal{A}^*$ el monoide libre de cadenas generadas a partir de $\mathcal{A}$ y la operación de concatenación y λ el elemento neutro o cadena vacía. Denotaremos la longitud de la cadena $w \in \mathcal{A}^*$ como $|w|$.

Dadas $x, y \in \mathcal{A}^*$, si $w = xy$ entonces escribiremos también $y = x^{-1}w$. A su vez, la expresión $x\mathcal{A}^*$ denotará al conjunto de cadenas que contienen a x como prefijo.

Diremos que x precede a y en *orden lexicográfico*, y lo denotaremos como $x < y$, en cualquiera de los siguientes casos:

1. $|x| < |y|$,

2. $|x| = |y|$ y x precede a y en orden alfabético.

Un *lenguaje estocástico* L se define mediante una distribución de probabilidad $p(w|L)$ sobre todas las cadenas w de $\mathcal{A}^*$. La probabilidad de un subconjunto de cadenas $X \subset \mathcal{A}^*$ viene dada por la suma de las probabilidades de sus cadenas:

$$p(X|L) = \sum_{x \in X} p(x|L). \tag{5.1}$$

La *identidad* de dos lenguajes estocásticos debe ser entendida de la forma siguiente:

$$L_1 = L_2 \Leftrightarrow p(w|L_1) = p(w|L_2) \; \forall w \in \mathcal{A}^*. \tag{5.2}$$

Es decir, para que dos lenguajes estocásticos sean idénticos, deben ser idénticas las probabilidades asignadas a todas y cada una de las cadenas.

A continuación vamos a presentar dos métodos para definir lenguajes estocásticos que, aunque distintas, son equivalentes: las gramáticas regulares estocásticas y los autómatas finitos estocásticos.

Una *gramática regular estocástica* (SRG), $G = (\mathcal{A}, V, S, R, p)$, consta de un alfabeto finito $\mathcal{A}$, un conjunto finito de variables V —una de las cuales, S, es el llamado símbolo inicial de la gramática—, un conjunto finito R de reglas de derivación cuya estructura es una de las siguientes:

$$\begin{array}{lcl} X & \rightarrow & aY \\ X & \rightarrow & \lambda \end{array} \tag{5.3}$$

(donde $a \in \mathcal{A}$, $X, Y \in V$), y una función real $p : R \rightarrow [0, 1]$ que representa la probabilidad de aplicación de cada regla. Obviamente, la suma de las probabilidades de las reglas asociadas a una misma variable X debe ser igual a 1. La definición que hemos tomado en (5.3), aunque aparentemente distinta a la habitualmente usada, como la de Fu (1982), es totalmente equivalente a ella. Una gramática estocástica G es *determinista* si por cada $X \in V$ y por cada $a \in \mathcal{A}$ existe como máximo una variable $Y \in V$ tal que $p(X \rightarrow aY) \neq 0$.

Toda gramática estocástica regular determinista G define un *lenguaje estocástico determinista regular* (SDRL) por medio de las probabilidades $p(w|L_G) = p(S \Rightarrow w)$. La probabilidad $p(S \Rightarrow w)$ de que la gramática G produzca la cadena $w \in \mathcal{A}^*$ se calcula de forma recursiva

$$\begin{aligned} p(X \Rightarrow \lambda) &= p(X \to \lambda) \\ p(X \Rightarrow aw) &= p(X \to aY)p(Y \Rightarrow w) \end{aligned} \tag{5.4}$$

donde Y es la única variable que satisface $p(X \to aY) \neq 0$. Si no existe tal variable, entonces $p(X \Rightarrow aw) = 0$.

Un *autómata finito determinista estocástico* (SDFA), se define como $A = (Q^A, \mathcal{A}, \delta^A, q_I^A, p^A)$, y consta de:

- un alfabeto finito $\mathcal{A}$;
- un conjunto finito de estados $Q^A = \{q_1, q_2, \ldots q_n\}$;
- un estado inicial $q_I^A \in Q^A$;
- una función de transición $\delta^A : Q^A \times \mathcal{A} \to Q^A$;
- una función de probabilidad $p^A : Q^A \times \mathcal{A} \to [0, 1]$.

La función $p^A(q_i, a)$ representa la probabilidad de que se genere a partir del estado q_i un símbolo a, produciéndose una transición posterior a $\delta^A(q_i, a)$. Se define además la función $p^A(q_i, \lambda)$ como

$$p^A(q_i, \lambda) = 1 - \sum_{a \in \mathcal{A}} p^A(q_i, a) \tag{5.5}$$

que representa la probabilidad de que la cadena termine en el nodo q_i. La restricción de que $p^A(q_i, \lambda) \geq 0$ debe cumplirse para todos los autómatas que han sido definidos correctamente.

Cada SDFA define un lenguaje regular determinista estocástico por medio de las probabilidades $p(w|L_A) = \pi(q_I^A, w)$, que a su vez se definen recursivamente como

$$\begin{aligned} \pi^A(q_i, \lambda) &= p^A(q_i, \lambda) \\ \pi^A(q_i, aw) &= p^A(q_i, a)\pi^A(\delta^A(q_i, a), w) \end{aligned} \tag{5.6}$$

Si $\delta^A(q_i, a)$ no está definida, entonces $\pi^A(\delta^A(q_i, a), w) = 0$.

Si comparamos las ecuaciones (5.4) y (5.6), observaremos la equivalencia formal entre los autómatas finitos estocásticos y las gramáticas regulares estocásticas. En caso de que la SDRG no contenga símbolos inútiles (Hopcroft & Ullman 1979), las probabilidades de todas las cadenas del lenguaje suman uno:

$$p(\mathcal{A}^*|L_G) = \sum_{w \in \mathcal{A}^*} p(w|L_G) = 1 \tag{5.7}$$

Un concepto que utilizaremos con frecuencia en lo que sigue es de lenguaje *cociente* $x^{-1}L$, que es un lenguaje estocástico cuyas probabilidades vienen dadas por:

$$p(w|x^{-1}L) = \frac{p(xw|L)}{p(x\mathcal{A}^*|L)} \tag{5.8}$$

expresión que puede entenderse como las probabilidades relativas de las subcadenas que admiten a x como prefijo. Por convención, cuando $p(x\mathcal{A}^*|L) = 0$, escribiremos que $x^{-1}L = \emptyset$ y entonces, $p(w|x^{-1}L) = 0$. Nótese que $\lambda^{-1}L = L$.

Si L es un lenguaje regular estocástico determinista, se define el *generador canónico* $M = (Q^M, \mathcal{A}, \delta^M, q_I^M, p^M)$ como:

$$\begin{aligned} Q^M &= \{x^{-1}L \neq \emptyset : x \in \mathcal{A}^*\} \\ \delta^M(x^{-1}L, a) &= (xa)^{-1}L \\ q_I^M &= \lambda^{-1}L \\ p^M(x^{-1}L, a) &= p(a\mathcal{A}^*|x^{-1}L) \end{aligned} \tag{5.9}$$

El autómata M es el SDFA mínimo que genera L y su construcción esta justificada por los siguientes hechos, que permiten generalizar el teorema de Myhill y Nerode (Hopcroft & Ullman 1979) al caso de los autómatas estocásticos:

1. El autómata M es finito y además es más pequeño que cualquier otro autómata $A = (Q^A, \mathcal{A}, \delta^A, q_I^A, p^A)$ que también genere L. Si

escribimos $q_x = \delta^A(q_I, x)$, y aplicamos repetidamente (5.6) a la definición (5.8), se llega a que

$$p(w|x^{-1}L) = \frac{\pi^A(q_I, xw)}{\pi^A(q_I, x\mathcal{A}^*)} = \frac{\pi^A(q_x, w)}{\pi^A(q_x, \mathcal{A}^*)} = \pi^A(q_x, w). \quad (5.10)$$

Como el número de valores diferentes de q_x está limitado por $|Q^A|$, esto es, por el tamaño del autómata A, la expresión anterior demuestra que también está acotado el número de lenguajes distintos $x^{-1}L$ y, en consecuencia, $|Q^M| \leq |Q^A|$.

2. La función de transición δ^M está bien definida, es decir,

$$x^{-1}L = y^{-1}L \Rightarrow \delta^M(x^{-1}L, a) = \delta^M(y^{-1}L, a). \quad (5.11)$$

De hecho, para cualquier cadena $w \in \mathcal{A}^*$ y cualquier lenguaje L

$$p(w|a^{-1}x^{-1}L) = \frac{p(aw|x^{-1}L)}{p(a\mathcal{A}^*|x^{-1}L)} = \frac{p(xaw|L)}{p(xa\mathcal{A}^*|L)} = p(w|(xa)^{-1}L). \quad (5.12)$$

y por tanto $(xa)^{-1}L = a^{-1}(x^{-1}L)$. Con esto, la relación (5.11) es inmediata. Además, la expresión (5.12) junto a la definición (5.9) nos permiten escribir $\delta^M(q_I, w) = w^{-1}L$.

3. El autómata M genera $L_M = L$. En realidad, es más sencillo probar que

$$\pi^M(x^{-1}L, w\mathcal{A}^*) = p(w\mathcal{A}^*|x^{-1}L) \quad (5.13)$$

para todas las cadenas $x, w \in \mathcal{A}^*$, propiedad que incluye el estado inicial $x = \lambda$ como caso particular: $\pi^M(q_I^M, w\mathcal{A}^*) = p(w\mathcal{A}^*|L)$, por lo que entonces $L_A = L$. La ecuación (5.13) se satisface trivialmente para cualquier x si $w = \lambda$. Siguiendo (5.6), se obtiene

$$\pi^M(x^{-1}L, aw\mathcal{A}^*) = p^M(x^{-1}L, a)\pi^M((xa)^{-1}L, w\mathcal{A}^*) \quad (5.14)$$

Finalmente, mediante un proceso de inducción en w utilizando (5.9), llegamos a que

$$\pi^M(x^{-1}L, aw\mathcal{A}^*) = p(a\mathcal{A}^*|x^{-1}L)p(w\mathcal{A}^*|(xa)^{-1}L) = p(aw\mathcal{A}^*|L). \quad (5.15)$$

Para identificar el generador canónico utilizaremos algunos conjuntos no estocásticos, como el *conjunto de prefijos* del lenguaje L que se define como

$$\mathrm{Pr}(L) = \{x \in \mathcal{A}^* : x^{-1}L \neq \emptyset\} \tag{5.16}$$

y el *conjunto de prefijos cortos* de L, que se define como:

$$\mathrm{Sp}(L) = \{x \in \mathrm{Pr}(L) : x^{-1}L = y^{-1}L \Rightarrow x \leq y\} \tag{5.17}$$

Debe observarse que $x^{-1}L \neq y^{-1}L$ para cualquier par de cadenas $x, y \in \mathrm{Sp}(L)$ distintas ($x \neq y$), y por tanto, cada cadena de $\mathrm{Sp}(L)$ puede entenderse como un representante de uno de los estados del generador canónico M. Por ello, utilizaremos estos elementos para construir el generador canónico M, añadiendo las transiciones adecuadas. Estas transiciones serán del tipo $\delta(x, a) = xa$, siempre y cuando xa sea un prefijo corto. En caso contrario, necesitaremos definir la transición de otro modo. Por este motivo, definimos el *kernel* de L

$$K(L) = \{\lambda\} \cup \{xa \in \mathrm{Pr}(L) : x \in \mathrm{Sp}(L) \wedge a \in \mathcal{A}\} \tag{5.18}$$

que está formado por los prefijos cortos más los prefijos que se obtienen de éstos por adición de un símbolo. Además, definimos la *frontera* de L como los elementos del kernel que no son prefijos cortos:

$$F(L) = K(L) - \mathrm{Sp}(L) \tag{5.19}$$

Nótese que el kernel $K(L)$ contiene como máximo $1 + |M||\mathcal{A}|$ cadenas e incluye a $\mathrm{Sp}(L)$ como parte propia.

Pretendemos identificar el generador canónico a partir de muestras aleatorias. Por ello, definimos una *muestra estocástica* S del lenguaje L como una secuencia infinita de cadenas generadas según la distribución de probabilidad $p(w|L)$. Representamos mediante S_n la subsecuencia de S formada por sus n primeras cadenas, que en general, no serán todas diferentes. El número de veces que aparece la cadena x en S_n se escribirá $c_n(x)$, y para subconjuntos de cadenas $X \subset \mathcal{A}^*$,

$$c_n(X) = \sum_{x \in X} c_n(x). \tag{5.20}$$

La subsecuencia S_n define un lenguaje estocástico L_n cuyas probabilidades son

$$p(x|L_n) = \frac{1}{n} c_n(x). \tag{5.21}$$

Por último, el *'arbol generador de prefijos* de S_n es un SDFA que genera L_n, es decir, asigna a cada cadena la probabilidad observada experimentalmente de la siguiente forma: $T_n = (Q^T, \mathcal{A}, \delta^T, q_I^T, p^T)$, con

$$\begin{aligned} Q^T &= \mathrm{Pr}(L_n) \\ \delta^T(x,a) &= \begin{cases} xa & \text{si } xa \in \mathrm{Pr}(L_n) \\ \emptyset & \text{en caso contrario} \end{cases} \\ q_I^T &= \lambda \\ p^T(x,a) &= \frac{c_n(xa\mathcal{A}^*)}{c_n(x\mathcal{A}^*)} \end{aligned} \tag{5.22}$$

Las probabilidades del tipo $p^T(x,\lambda)$ pueden calcularse usando la ecuación (5.5).

5.3 Algoritmo de inferencia

En este punto, es conveniente destacar algunas diferencias entre el proceso de identificación de lenguajes estocásticos y el de no estocásticos. Las muestras de lenguajes estocásticos contienen ejemplos repetidos que aparecen siguiendo una distribución de probabilidad $p(w|L)$, y la regularidad estadística es capaz de compensar la falta de datos negativos. Tal y como demostró Angluin (1988), muchas clases de distribuciones —y en particular los lenguajes regulares estocásticos— pueden ser identificados en el límite, en el sentido de que todas las cadenas de probabilidad no nula son aprendidas, con probabilidad uno. Para ello, es suficiente con seguir un procedimiento enumerativo como el descrito en la sección 1.3. Sin embargo, los métodos enumerativos son irrealizables en la práctica y es necesario buscar algoritmos que funcionen en tiempo polinómico.

Es importante recordar que no se trata tan sólo de aproximarse a la distribución de probabilidad correcta. Esto se puede conseguir sencillamente utilizando para las probabilidades $p(w|L)$ los valores $c_n(w)/n$ observados de la frecuencia relativa de aparición de la cadena. En la sección 3.6, señalamos que la aproximación de las probabilidades puede conseguir rápidamente valores pequeños de la entropía relativa con respecto al modelo correcto. De hecho, es sencillo comprobar que de esta forma la distancia entre la distribución correcta y la propuesta disminuye a medida que aumenta n. Sin embargo, por muy grande que sea n, si el lenguaje contiene un número infinito de cadenas con probabilidad no nula de aparición, siempre existirán cadenas que no están en S_n, lo que provoca un valor grande de la entropía relativa entre la distribución $c_n(w)/n$ y la correcta.

Por ello, lo que buscamos es un procedimiento que permita reducir el número de probabilidades a estimar a un número finito: si identificamos la estructura del generador canónico M, sólo es necesario evaluar las probabilidades de transición entre sus estados. Como el número de transiciones es finito (como mucho $|M||\mathcal{A}|$), podremos entonces estimar estas probabilidades a partir de la muestra y obtener mejores resultados. Esta estimación puede realizarse mediante el procedimiento descrito en Wetherell (1980) y justificado en Chaudury & Rao (1986). Un algoritmo que permite realizar esta tarea de identificación de la estructura probabilística es presentado en esta sección.

Para ello, vamos a definir la función lógica $\texttt{equiv}_L : K(L) \times K(L) \to \{\text{TRUE}, \text{FALSE}\}$ de la siguiente forma:

$$\texttt{equiv}_L(x, y) = \text{TRUE} \Leftrightarrow x^{-1}L = y^{-1}L. \tag{5.23}$$

La función $\texttt{equiv}_L$ sólo está definida en $K(L)$ y nos permite enunciar el siguiente lema.

Lema 1 *Dados* $\mathrm{Sp}(L)$, $F(L)$ *y* $\texttt{equiv}_L$, *la estructura del generador*

canónico es isomorfa a:

$$\begin{aligned} Q &= \mathrm{Sp}(L) \\ q_I &= \lambda \\ \delta(x,a) &= y \end{aligned} \tag{5.24}$$

donde para cada $(x,a) \in \mathrm{Sp}(L) \times \mathcal{A}$*, y es la única cadena en* $Sp(L)$ *tal que* $\mathtt{equiv}_L(xa,y)$.

Demostraci'on. Sea $\Phi : Q \to Q^M$ la función definida como $\Phi(x) = x^{-1}L$. La función Φ es un isomorfismo si $\delta^M(\Phi(x),a) = \Phi(\delta(x,a))$, es decir, si $(xa)^{-1}L = \delta(x,a)^{-1}L$. Por tanto, Φ es un isomorfismo si y sólo si $\delta(x,a)$ es una cadena $y \in \mathrm{Sp}(L)$ que satisface $\mathtt{equiv}_L(xa,y)$. Además, de acuerdo con la definición (5.17), y es única. Nótese además, que $x \in \mathrm{Sp}(L) \Rightarrow xa \in K(L)$, por lo que el resultado de $\mathtt{equiv}_L(xa,y)$ está bien definido. ■

El siguiente lema demuestra que el problema de la inferencia se puede reducir al de aprender $\mathrm{Pr}(L)$ junto a la función $\mathtt{equiv}_L$.

Lema 2 *La estructura del generador canónico de L puede ser obtenida a partir de* $\mathtt{equiv}_L$ *y de cualquier conjunto* $A \subset \mathrm{Pr}(L)$ *tal que* $K(L) \subset A$ *con el algoritmo de la figura 5.1, que además proporciona como resultado adicional los conjuntos* $\mathrm{Sp}(L)$ *y* $F(L)$.

Demostraci'on. Por inducción en el número i de iteraciones, se obtiene que $\mathrm{Sp}^{[i]} \subset \mathrm{Sp}(L)$, $F^{[i]} \subset F(L)$ y que $W^{[i]} \subset K(L)$, donde el superíndice denota el resultado después de i iteraciones. Además, si xa pertenece a $K(L)$, se puede comprobar por inducción en la longitud de la cadena que xa siempre entra a formar parte de $W^{[i]}$ para algún valor de i. Siguiendo el lema 1, para cada cadena $x \in \mathrm{Sp}(L)$, si xa está también en $\mathrm{Sp}(L)$, entonces $\delta(x,a) = xa$. Por contra, si $xa \notin \mathrm{Sp}(L)$, es decir, $xa \in F(L)$, entonces existe una única $y \in \mathrm{Sp}(L)$ tal que $\mathtt{equiv}_L(xa,y)$ y $\delta(x,a) = y$. ■

En el algoritmo anterior, podemos reemplazar A por $\mathrm{Pr}(L_n) \subset \mathrm{Pr}(L)$, dado que este conjunto contiene a $K(L)$ con probabilidad creciente al aumentar n. Por otro lado, conviene destacar que $\mathtt{equiv}_L$

permanece siempre bien definida, porque todas las llamadas a esta función se producen dentro de su dominio $K(L)$. Además, y como consecuencia de ello, el algoritmo realiza como mucho $|K(L)| \times |\mathrm{Sp}(L)|$ llamadas a $\mathtt{equiv}_L$. Por tanto, la complejidad global del algoritmo será $\mathcal{O}(|\mathcal{A}||M|^2|)$ multiplicado por la complejidad de $\mathtt{equiv}_L$.

5.4 Convergencia del algoritmo

Para evaluar la relación de equivalencia $x^{-1}L = y^{-1}L$, es posible utilizar una variación de (6.12) que mejora la convergencia:

$$L_1 = L_2 \Leftrightarrow p(a\mathcal{A}^*|z^{-1}L_1) = p(a\mathcal{A}^*|z^{-1}L_2)\ \forall a \in \mathcal{A}, z \in \mathcal{A}^* \tag{5.25}$$

Teniendo en cuenta (5.9), la relación anterior significa que para todas las cadenas $z \in \mathcal{A}^*$ y para todos los símbolos $a \in \mathcal{A} \cup \{\lambda\}$ se satisface

$$p^M((xz)^{-1}L, a) = p^M((yz)^{-1}L, a) \tag{5.26}$$

En la práctica, no se conoce el lenguaje L y la función $\mathtt{equiv}_L(x, y)$, definida como $x^{-1}L = y^{-1}L$, debe ser reemplazada por una función experimental $\mathtt{comp_n}(x, y)$, que comprueba si $x^{-1}L_n$ coincide con $y^{-1}L_n$. Esto es, utilizamos los valores p^T del 'arbol generador de prefijos en vez de los desconocidos p^M en (5.26). La figura 5.2 muestra una implementación de la función $\mathtt{comp_n}(x, y)$.

Como S_n es un muestra aleatoria, es preciso dar un margen de confianza a la diferencia entre las probabilidades de las cadenas en $x^{-1}L_n$ y $y^{-1}L_n$, tal y como lo hace la función `different` en $\mathtt{comp_n}$, que permite una cierta holgura en los resultados. Existen numerosos criterios estadísticos (Hoeffding 1963; Feller 1950; Anthony & Biggs 1992) que conducen a diferentes variaciones del algoritmo básico. En este trabajo, hemos elegido la cota de Hoeffding (1963) descrita en la sección 3.5. Recordemos, que para una variable de Bernoulli con probabilidad p y frecuencia observada f/n, dado $\alpha > 0$ y

$$\epsilon_\alpha(m) = \sqrt{\frac{1}{2m}\log\frac{2}{\alpha}} \tag{5.27}$$

entonces, con probabilidad mayor que $1-\alpha$,

$$\left| p - \frac{f}{m} \right| < \epsilon_\alpha(m) \tag{5.28}$$

De esta relación se deduce inmediatamente que, dadas dos variables de Bernoulli con probabilidades p y p' respectivamente, con probabilidad mayor que $(1-\alpha)^2$,

$$\begin{aligned} \left| \tfrac{f}{m} - \tfrac{f'}{m'} \right| &< \epsilon_\alpha(m) + \epsilon_\alpha(m') \quad \text{si} \qquad p = p' \\ \left| \tfrac{f}{m} - \tfrac{f'}{m'} \right| &> \epsilon_\alpha(m) + \epsilon_\alpha(m') \quad \text{si} \quad |p - p'| > 2(\epsilon_\alpha(m) + \epsilon_\alpha(m')) \end{aligned} \tag{5.29}$$

Debido a que $\lim_{m\to\infty} \epsilon_\alpha(m) = 0$, sólo una de las dos condiciones anteriores será cierta cuando m y m' son lo suficientemente grandes.

Esta comprobación ha sido implementada a través de la función `different` (representada en la figura 5.3), que comprueba si $p = p'$ con un nivel de confianza $(1-\alpha)^2$ para valores suficientemente grandes de m y m'.

Debido a que el número de llamadas a `different` crece si el tamaño t del 'arbol generador de prefijos aumenta, permitiremos que el parámetro α dependa de t. Incluso en ese caso, ϵ_α presenta el límite correcto para valores grandes de n, pues ϵ_α muestra una dependencia logarítmica respecto a α, y el comportamiento final depende moderadamente del valor exacto de α.

De acuerdo con (5.26), la compatibilidad de dos estados x e y de Q^T debe ser rechazada si existe alguna cadena $z \in \mathcal{A}^*$ tal que las probabilidades de transición desde xz e yz que han sido estimadas a partir de la muestra son diferentes según `different`. Demostraremos a continuación que $\texttt{comp}_\texttt{n}(x, y)$, representado en la figura 5.2, devuelve en el límite de $n \to \infty$ el mismo valor que $\texttt{equiv}_L(x, y)$ para todos los pares de cadenas x, y de $K(L)$. Por consiguiente, de acuerdo con el lema 2, la estructura correcta del generador canónico es inferida en el límite y las probabilidades de transición $p^M(x, a)$ definidas en (5.9) pueden

ser evaluadas a partir de S_n por medio de los valores experimentales $p^T(x,a)$, definidos por medio de (5.22).

Teorema 3 *Sea $t = |T_n|$ el tamaño del 'arbol generador de prefijos para S_n y $\alpha(t)$ un nivel de significación tal que $\lim_{t\to\infty}(1-\alpha(t))^t \to 1$. Entonces, con probabilidad uno, ambas funciones $\texttt{equiv}_L(x,y)$ y $\texttt{comp}_\texttt{n}(x,y)$ devuelven, excepto para un número finito de valores de n, el mismo valor para todos las cadenas $x, y \in K(L)$.*

Demostraci'on. Siguiendo (5.26), el bucle sobre z en la función $\texttt{comp}_\texttt{n}$ comprueba, para todos los subárboles con raíz en x e y, si las probabilidades de transición $p^T(xz,a)$ y $p^T(yz,a)$ son semejantes. También compara $p^T(xz,\lambda)$ con $p^T(xz,\lambda)$ en cada nodo. Hay como mucho $t-1$ arcos más t nodos en un subárbol y, por tanto, el número máximo de llamadas a `different` desde $\texttt{comp}_\texttt{n}$ es $2t$. Como `different` trabaja con un nivel de confianza por encima de $(1-\alpha)^2$, $\texttt{comp}_\texttt{n}(x,y)$ devolverá el mismo resultado que $\texttt{equiv}_L(x,y)$ con probabilidad mayor que $(1-\alpha(t))^{4t}$. ■

Por ejemplo, la condición $(1-\alpha(t))^t \to 1$ se satisface si $\alpha(t)$ decrece más rápidamente que $1/t$. Una consecuencia inmediata de la demostración anterior es que la complejidad de $\texttt{comp}_\texttt{n}$ está acotada por $|T_n|$ y por tanto, de acuerdo con la discusión del final de la sección anterior, el algoritmo `rlips` trabaja con complejidad temporal $\mathcal{O}(|T_n||\mathcal{A}||M|^2)$. Como $|T_n|$ no puede crecer más rápidamente que n, el algoritmo es, en el límite de muestras grandes, lineal con respecto al tamaño de la muestra.

Por otro lado, dado que `rlips` solamente necesita que $\texttt{comp}_\texttt{n}(x,y)$ sea correcta dentro del conjunto finito $K(L) \times \mathrm{Sp}(L)$, existe un natural n_0 tal que si $n > n_0$:

- $\mathrm{Pr}(L_n) \subset K(L)$;
- todas las llamadas a $\texttt{comp}_\texttt{n}$ devuelven el valor correcto.

En ese punto, `rlips` encuentra siempre la estructura correcta del generador canónico.

5.5 Número de ejemplos necesarios para la convergencia

Una cuestión interesante es el número de ejemplos que el algoritmo necesita para inferir correctamente el SDFA. Este número depende extraordinariamente de la estructura detallada del autómata. Sin embargo, es posible establecer una cota válida para cualquier algoritmo de la clase descrita en este trabajo.

Para cada par de cadenas $x_1, x_2 \in \mathrm{Sp}(L)$ tales que $x_1 \neq x_2$, el número aproximado de ejemplos requeridos para distinguir $x_1^{-1}L$ y $x_2^{-1}L$ será una función $\gamma(x_1, x_2)$. Dado que $x_1^{-1}L \neq x_2^{-1}L$, existe una cadena $z \in \mathcal{A}^*$ y un símbolo $a \in \mathcal{A}$, tales que, si denotamos $x_1' = x_1 z$ y $x_2' = x_2 z$, entonces:

$$|p^M(x_1', a) - p^M(x_2', a)| \neq 0. \tag{5.30}$$

No cabe esperar que la convergencia se alcance antes de que el error estadístico devenga menor que la diferencia anterior. Una estimación del error cometido, que es independiente del algoritmo particular utilizado, puede obtenerse tomando la suma $\sigma_1 + \sigma_2$, siendo,

$$\sigma_i \simeq \frac{1}{\sqrt{4np(x_i'\mathcal{A}^*|L)}} \tag{5.31}$$

donde n es el número de ejemplos en la muestra S_n.

Por tanto, sólo podemos esperar que la comparación de x_1 y x_2 sea correcta una vez que $n > N(x_1, x_2, z, a)$, siendo

$$\begin{aligned} N(x_1, x_2, z, a) &= \\ &= \left(\frac{1}{p^M(x_1',a) - p^M(x_2',a)}\right)^2 \left(\frac{1}{2\sqrt{p(x_1'\mathcal{A}^*|L)}} + \frac{1}{2\sqrt{p(x_2'\mathcal{A}^*|L)}}\right)^2 \end{aligned} \tag{5.32}$$

Podemos tomar $\gamma(x_1, x_2) = \min_{(z,a)}\{N(x_1, x_2, z, a)\}$, dado que cualquier cadena z y cualquier símbolo a son válidas para establecer que x_1 y x_2 son incompatibles. La comparación más difícil establece una cota inferior para la dificultad de identificar el generador canónico:

$$\Gamma = \max_{x_1, x_2 \in \mathrm{Sp}(L)} \{\gamma(x_1, x_2) : x_1 < x_2\} \tag{5.33}$$

La misma cota Γ se obtiene si $x_1 \in \mathrm{Sp}(L)$ y $x_2 \in F(L)$, pero en este caso es suficiente con comparar x_2 con todos los prefijos x_1 que preceden a z_2, es decir $x_1 < z_2$, siendo z_2 la única cadena en $\mathrm{Sp}(L)$ equivalente a x_2. Como ejemplo, para la gramática de Reber de la figura 5.4, se obtiene como cota inferior $\Gamma \simeq 330$ (correspondiente al caso $x = BT, y = BTX$ y $z = \lambda$), un valor acorde con el comportamiento observado en la figura 5.5.

5.6 Resultados y discusión

Se ha estudiado el comportamiento del algoritmo para una serie de gramáticas distintas. Para cada gramática, se generaron diferentes muestras usando el autómata canónico y éstas fueron utilizadas posteriormente como entrada para `rlips`. Por ejemplo, se utilizó la gramática de Reber (1967) de la figura 5.4 con el objetivo de comparar los resultados con otros previos en los que redes neurales eran entrenadas con este lenguaje (Castaño, Casacuberta & Vidal 1993).

En la figura 5.5 se representa el promedio después de 10 experimentos del número de nodos del autómata propuesto por `rlips` en función del tamaño del conjunto de muestra generado por la gramática de Reber. Tal y como se aprecia en la figura, el número de estados converge al valor correcto cuando la muestra es lo suficientemente grande. Para comprobar que no sólo el número de estados, sino también la estructura había sido inferida correctamente, se calculó la entropía relativa $H(L, A)$ entre el lenguaje correcto L y la hipótesis generada A. Este resultado aparece representado en la figura 5.6. Como referencia, también se ha dibujado $H(L, L_n)$, la entropía relativa de la muestra o, lo que es equivalente, del 'arbol generador de prefijos con respecto a la gramática objetivo. Esta última converge mucho más lentamente, remarcando la importancia de haber identificado la estructura del generador canónico a la hora de estimar las probabilidades de las cadenas del lenguaje. La diferencia no estriba sólo en que el 'arbol generador de prefijos asigna una probabilidad cero a muchas cadenas

con probabilidad de aparición no nula. Además, una vez identificada la estructura, el número de probabilidades a estimar se reduce significativamente al pasar de infinito a finito ($|M||\mathcal{A}|$).

Tal y como sugiere la figura (5.5), cuando el número de ejemplos es pequeño, el algoritmo tiende a proponer hipótesis demasiado simples (que, de alguna forma, generalizan excesivamente). Sin embargo, en cuanto el número de ejemplos disponibles es suficiente, el algoritmo identifica la estructura correcta del generador canónico. El número de ejemplos que requiere esta identificación es relativamente pequeño (aproximadamente quinientos) y consistente con la cota que se obtuvo en la sección anterior. Comparativamente, este resultado es mucho mejor que el rendimiento obtenido utilizando redes neurales (Castaño, Casacuberta & Vidal 1993) que no garantizan la convergencia en las pruebas realizadas con esta gramática, incluso utilizando decenas de miles de ejemplos aleatorios.

En la figura 5.7, se representa el tiempo medio que consume el algoritmo en función del número de ejemplos de la muestra (las dispersiones observadas son inapreciables en la figura). Aquí se comprueba que la complejidad temporal es lineal y que además el algoritmo es muy rápido incluso para muestras enormes.

El funcionamiento de `rlips` para la gramática de Reber fue comparado con el uso de modelos de Markov ocultos entrenados por el procedimiento de Baum-Welch. Cuando el número de estados del modelo es similar al de la gramática, el procedimiento BW encuentra el modelo correcto. Sin embargo los tiempos de ejecución son enormes comparados con los de `rlips`. Por ejemplo, usando 8 estados y 400 iteraciones (número que se encontró idóneo experimentalmente para la identificación) en el método BW, el algoritmo `rlips` resultó del orden de 1000 veces más rápido. Nótese que la complejidad de `rlips` depende sólo de la dificultad intrínseca del problema (tamaño de la muestra y dificultad del autómata que se debe identificar), mientras que en el método de Baum-Welch depende del conocimiento *a priori* que se tenga del problema, por ejemplo, el número máximo de estados

en el modelo. Por otro lado, si el número de estados en el modelo de Markov es mucho más grande que el del generador canónico, sus predicciones tienden a ser las de L_n, pues el entrenamiento BW no contiene ninguna preferencia por los modelos más sencillos. Como en la práctica resulta prohibitivo utilizar un gran número de estados, la situación es más bien la contraria: con frecuencia el modelo de Markov no tendrá el número de estados suficiente como para identificar el modelo correcto. En cambio, en el algoritmo `rlips` el número de estados se elige automáticamente.

La gramática de Reber 5.4 no revela toda la riqueza del algoritmo, pues los estados de la gramática pueden ser identificados directamente por sus probabilidades de transición. En cambio, en un caso como el de la figura 5.8, la separación de los estados ha de realizarse de forma indirecta, pues algunos de ellos presentan las mismas probabilidades de transición. Se trata, por tanto, de un caso en el que la identificación de la estructura es más difícil. El número de transiciones obtenido por `rlips` para esta gramática aparece representado en la figura 5.9, donde se observa que el comportamiento del algoritmo es el adecuado. También se ha representado la entropía relativa del modelo a la hipótesis y a la muestra aleatoria respectivamente en la figura 5.10. De nuevo, el algoritmo construye una hipótesis que converge rápidamente al lenguaje correcto.

Finalmente, en la figura 5.11 se representa el número de ejemplos con los que se alcanza experimentalmente la convergencia, en función de la cota Γ obtenida en la sección 5.5 Para ello se generaron gramáticas al azar de hasta ocho estados. Tal y como se observa en la figura, el número de ejemplos resulta superior en todos los casos a la cota Γ.

```
algorithm rlips
input: A ⊂ Pr(L) tal que K(L) ⊂ A
output: Q^M = Sp (prefijos cortos)
        F (frontera)
        δ^M (función de transición)
begin algorithm
  F = ∅
  Sp = {λ}
  W = 𝒜* ∩ A
  do ( while W ≠ ∅ )
     x = min W
     W = W − {x}
     wa = x : w ∈ 𝒜* ∧ a ∈ 𝒜
     if ∃y ∈ Sp : equiv_L(x, y) then
        F = F ∪ {x}
        δ^M(w, a) = y
     else
        Sp = Sp ∪ {x}
        W = W ∪ {xa ∈ A : a ∈ 𝒜}
        δ^M(w, a) = x
     endif
  end do
end algorithm
```

Figura 5.1: Algoritmo `rlips`.

```
algorithm compatible
input: x, y (cadenas)
       T_n ( 'arbol generador de prefijos )
output: boolean
begin algorithm
  do ( ∀z ∈ A*: xz ∈ Pr(L_n) ∨ yz ∈ Pr(L_n ) )
     if different (c_n(xz), c_n(xzA*), c_n(yz), c_n(yzA*), α)  then
        return FALSE
     endif
     do ( ∀a ∈ A )
        if different (c_n(xzaA*), c_n(xzA*), c_n(yzaA*), c_n(yzA*), α)  then
           return FALSE
        endif
     end do
  end do
  return TRUE
end algorithm
```

Figura 5.2: Algoritmo `compatible`

```
algorithm different
input: f, n, f', n', α
output: boolean
begin algorithm
  if n = 0 or n' = 0 then
     return FALSE
  endif
  return |f/n − f'/n'| > ε_α(n) + ε_α(n')
end algorithm
```

Figura 5.3: Algoritmo `different`

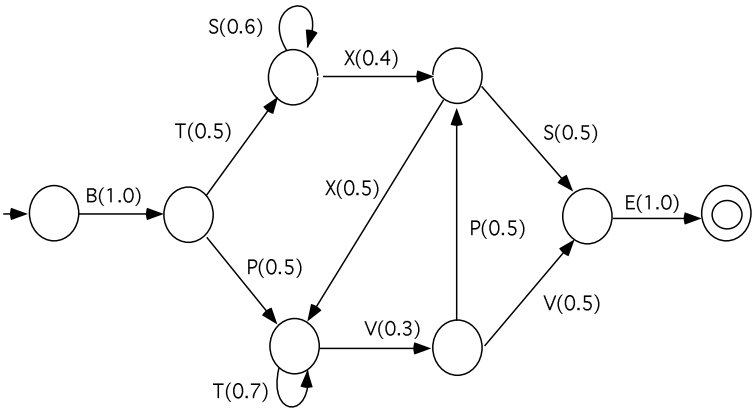

Figura 5.4: Autómata correspondiente a la gramática de Reber

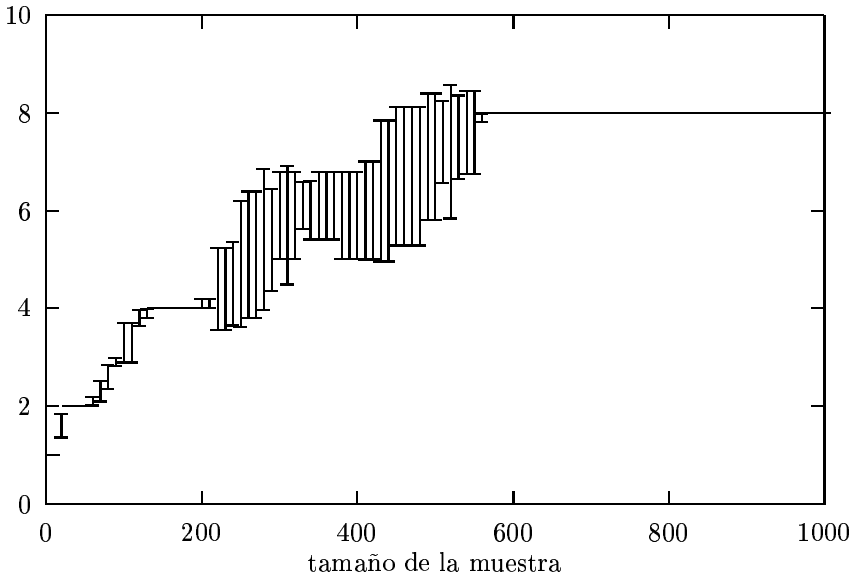

Figura 5.5: Número de nodos en la hipótesis para la gramática de Reber en función del tamaño de la muestra.

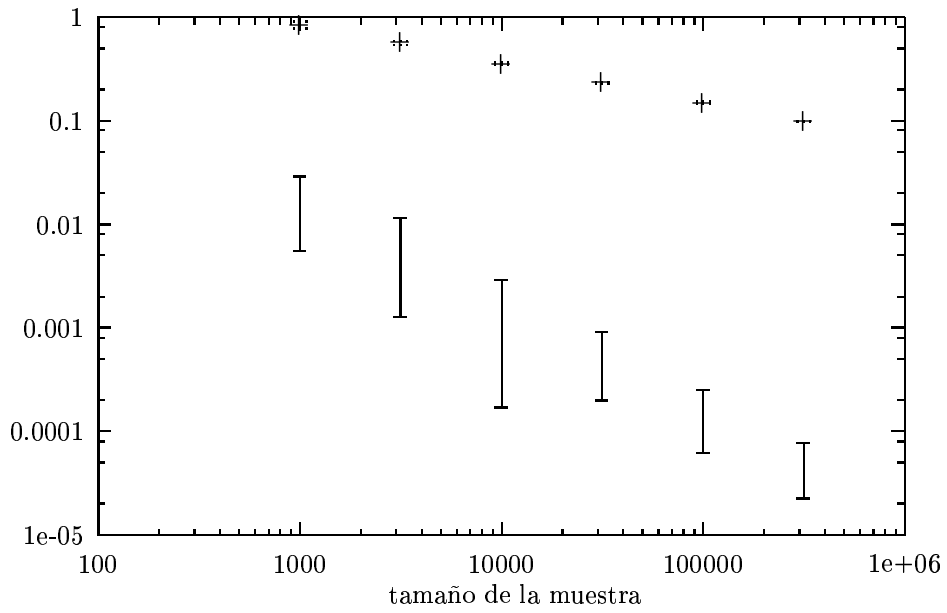

Figura 5.6: Entropía relativa (en bits) entre la gramática de Reber y: 1) la hipótesis propuesta por `rlips` (curva inferior); 2) el conjunto de muestra (curva superior).

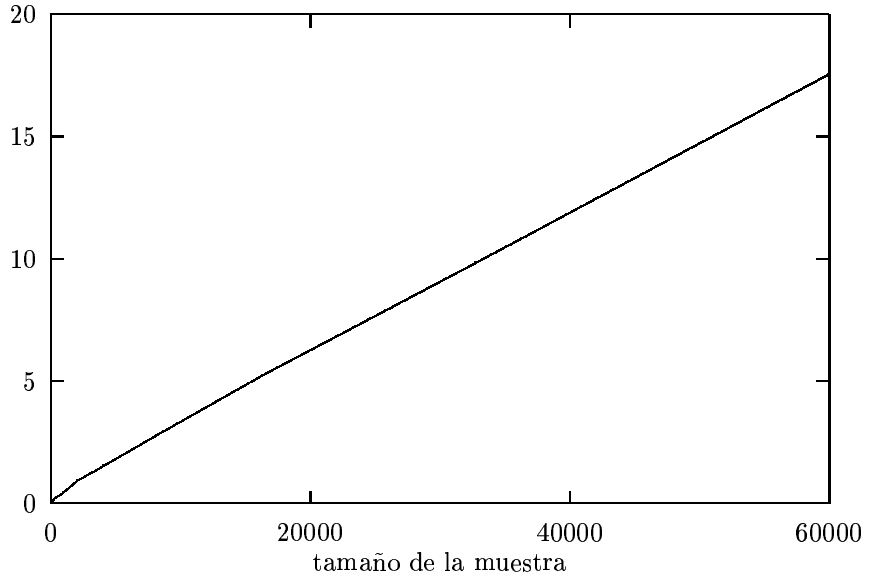

Figura 5.7: Tiempo (en segundos) que requiere la implementación del algoritmo al ser ejecutado en un Hewlett-Packard 715 (40 MIPS) en función del tamaño de la muestra.

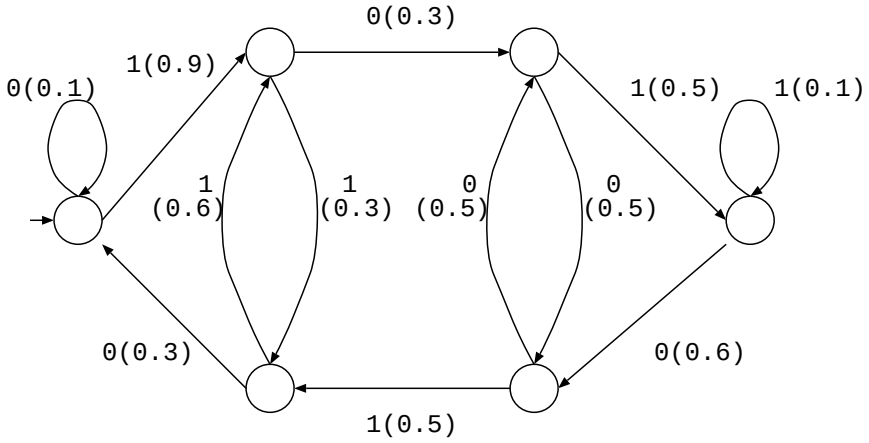

Figura 5.8: Autómata estocástico con estados cuyas probabilidades de transición son idénticas.

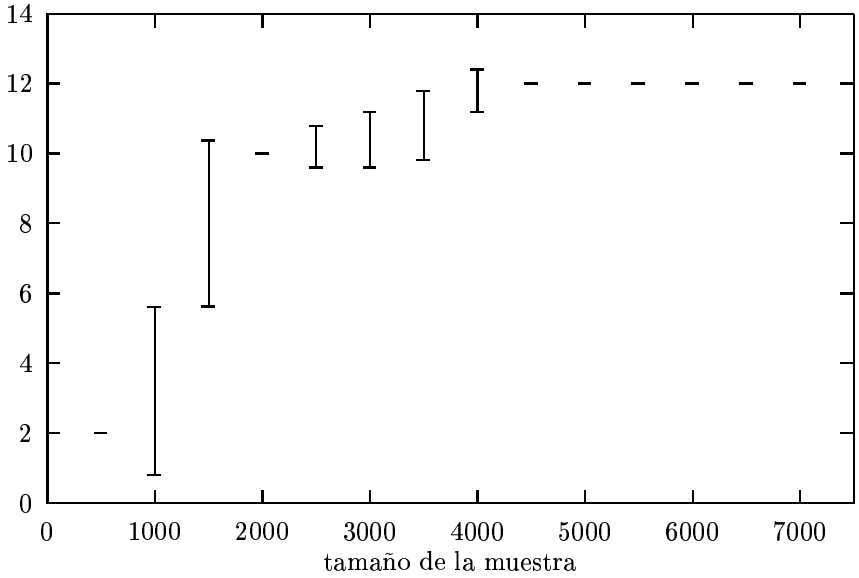

Figura 5.9: Número de nodos obtenido por `rlips` para la segunda gramática de prueba.

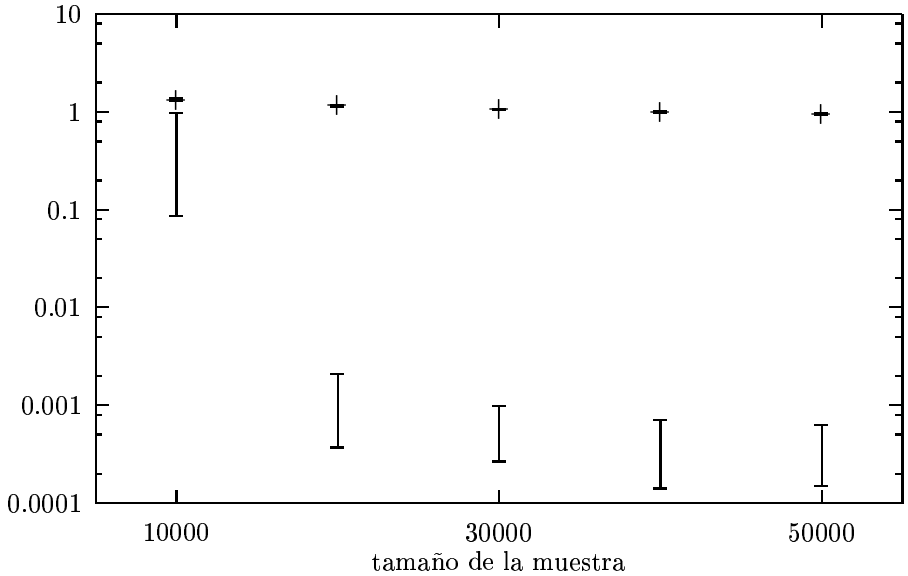

Figura 5.10: Entropía relativa entre la segunda gramática y la hipótesis propuesta por `rlips` comparada con la entropía relativa entre la gramática y el conjunto de muestra.

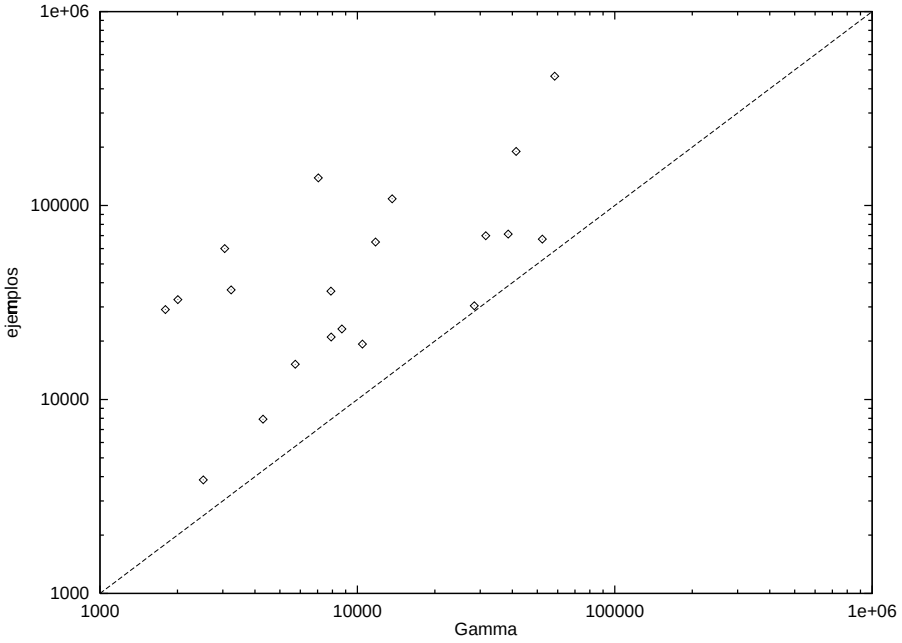

Figura 5.11: Número de ejemplos para los que se alcanza la convergencia en función de la cota teórica establecida.

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:

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

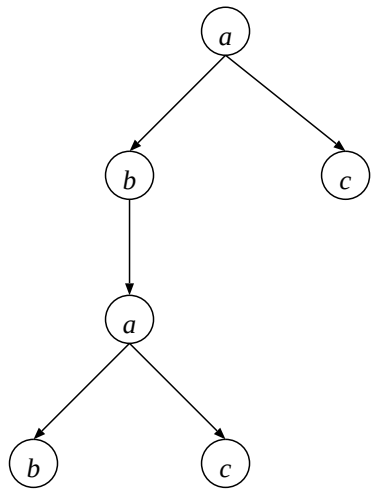

Figura 6.1: Un ejemplo de árbol

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 $i, j \in \mathbb{N}$

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

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:

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

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 $\mathrm{sk}(D_S(G))$ 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 $\mathtt{equiv_T}(t, f(t_1, ..., t_k))$. 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* $\mathtt{equiv_T}$ *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 $\mathtt{equiv_T}(t, s)$ 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 $\mathtt{equiv_T}$ 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 $\mathtt{equiv_T}$.

6.4 Inferencia probabilística

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

zado por una función probabilística $\mathtt{comp_n}(x, y)$ 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 $\mathtt{comp_n}$ tienda a $\mathtt{equiv_T}$ en el límite de n grande.

De acuerdo con (6.28), $\mathtt{equiv_T}(x, y) = \text{TRUE}$ 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), $\mathtt{comp_n}$ 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 $\mathtt{comp_n}$, 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 $\mathtt{comp_n}(x,y) = \mathtt{equiv_T}(x,y)$ en el límite de n grande.

Por último, la complejidad de $\mathtt{comp_n}$ 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 $\mathtt{comp_n}$.

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.

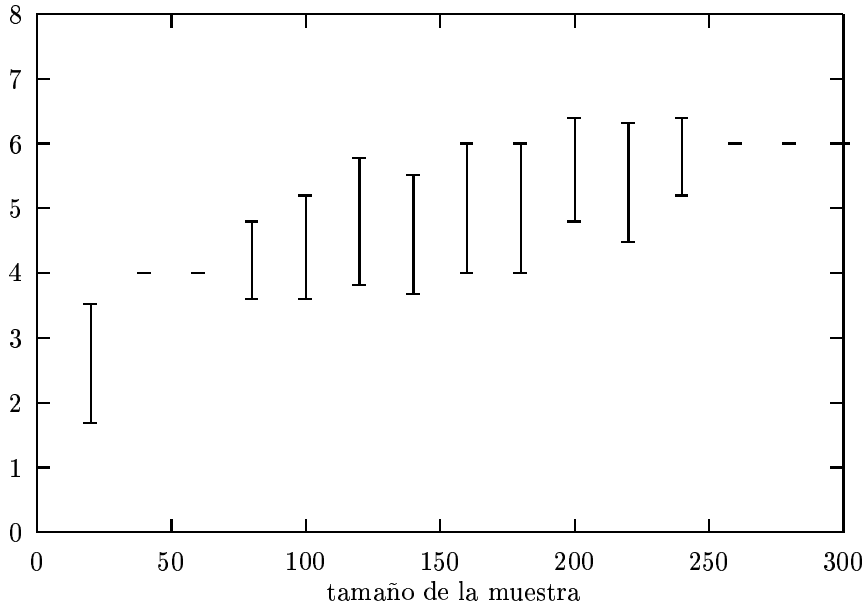

Figura 6.5: Número de reglas de la hipótesis en función del número de ejemplos. La gramática correcta 6.4 contiene 6 reglas.

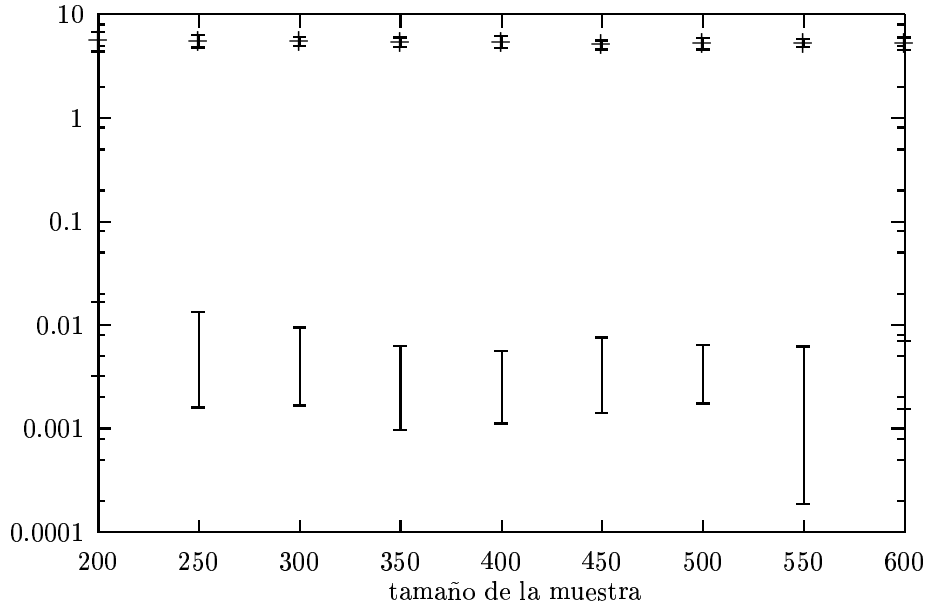

Figura 6.6: Línea inferior: entropía relativa (en bits) entre la gramática 6.4 y el modelo propuesto por `tlips` en función del número de ejemplos. Línea superior: entropía relativa entre la gramática y la muestra.

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.

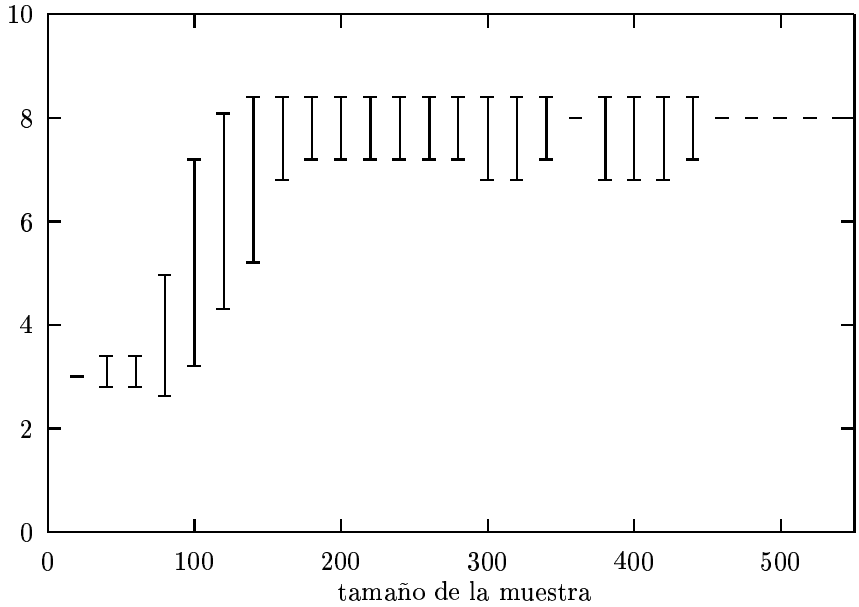

Figura 6.8: Número de reglas de la hipótesis en función del número de ejemplos generados por la gramática 6.7.

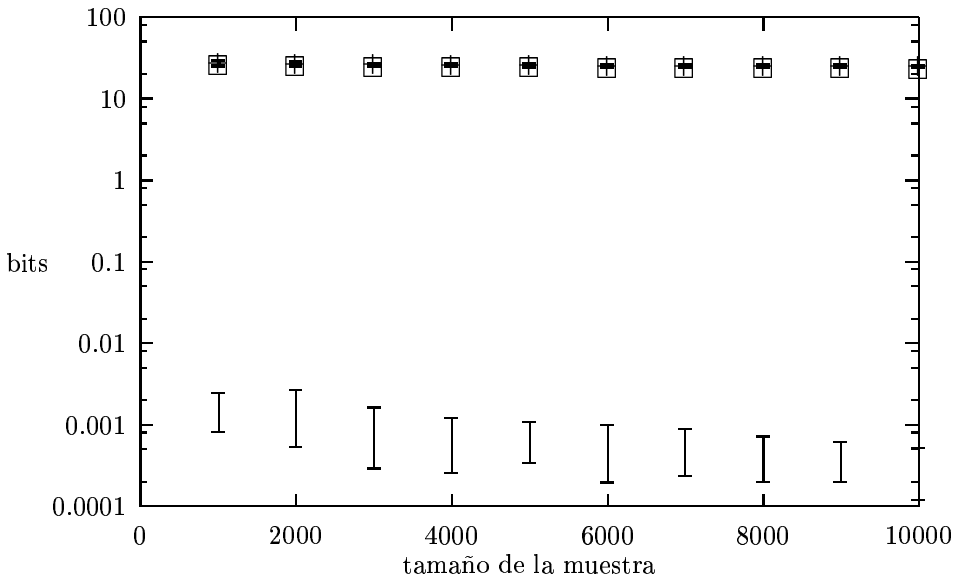

Figura 6.9: Entropía relativa entre la gramática 6.7 y la hipótesis propuesta por `tlips` en función del número de ejemplos.

Capítulo 7

Inferencia estocástica con redes neurales

Trabajos recientes han demostrado que la extracción de reglas simbólicas mejora la potencia de generalización de las redes entrenadas con muestras completas de lenguajes regulares. En este capítulo, se explora la posibilidad de aprender reglas cuando la red se entrena con datos estocásticos. Para este fin, se utiliza una red con dos capas. Si en vez de utilizar la red misma, se extrae un autómata después del entrenamiento y sus probabilidades de transición se calculan a partir de la muestra, la distancia con respecto a la distribución correcta disminuye.

7.1 Antecedentes

Trabajos recientes como los de Giles *et al.* (1992a,b) y Watrous & Kuhn (1992a,b) han explorado la capacidad de las redes neurales de segundo orden para aprender gramáticas regulares sencillas a partir de muestras de entrenamiento completas. La potencia de generalización del autómata finito extraído a partir de la red entrenada es mejor que la de la misma red (Giles *et al.* 1992a,b; Omlin & Giles 1996). En la mayoría de los casos se observa que una vez que la red ha aprendido a

clasificar todas las palabras del conjunto de entrenamiento, los estados de las neuronas ocultas visitan durante el proceso de reconocimiento sólo algunas regiones o "clusters" del espacio de configuración accesible. Estos clusters pueden identificarse con los estados del autómata (DFA) inferido y las transiciones que entre ellos ocurren cuando se lee un símbolo en la entrada determinan la función de transición del DFA. Aunque la extracción de autómatas resulta aún controvertida (Kolen 1994), un trabajo reciente de Casey (1995) ha demostrado que una red recurrente de segundo orden puede modelizar robustamente un DFA si el espacio de configuración se divide en un varias zonas disjuntas asociadas a cada estado del autómata. Un algoritmo de extracción aparece descrito en Omlin & Giles (1996). La extracción se basa en dividir el hipercubo de configuración $[0.0, 1.0]^N$ en hipercubos más pequeños, de forma que cada uno contiene como mucho uno de los clusters. Un algoritmo mejorado que no restringe la forma de los clusters es el propuesto por Blair y Pollack (1996), pero su coste temporal es muy superior al anterior.

Debido a que las muestras completas son difíciles de obtener, es interesante estudiar si el mismo comportamiento puede ser observado cuando el entrenamiento se realiza a partir de muestras aleatorias. En ese caso, la extracción de una gramática subyacente debe mejorar la evaluación de las probabilidades, sobre todo para las cadenas no contenidas en la muestra.

Por ello, en este capítulo se explora la posibilidad de extraer reglas simbólicas a partir de la red entrenada con ejemplos generados aleatoriamente. La arquitectura de la red se describe en la sección siguiente y es análoga a una red de Elman (1990) con una función siguiente-estado de segundo orden, e idéntica a la usada por Blair y Pollack (1996). Una aproximación algorítmica al mismo problema es la presentada por Carrasco y Oncina (1994).

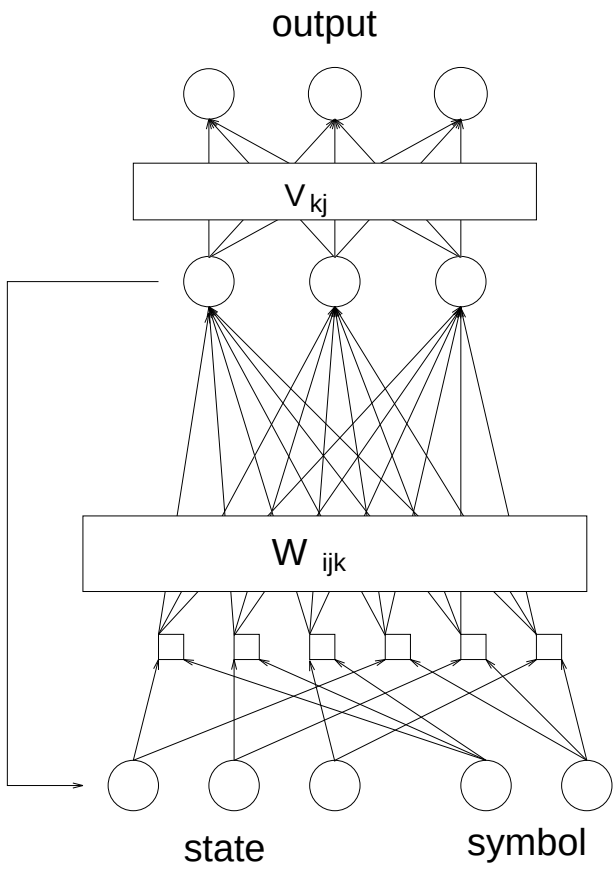

Figura 7.1: Red recurrente de segundo orden con $N = 3$ y $L = 2$

7.2 Arquitectura de la red

Supongamos que el alfabeto de entrada tiene L símbolos. Entonces, la red de segundo orden consta de N neuronas ocultas y L neuronas de entrada más $L + 1$ neuronas de salida tal y como aparece en la figura 7.1). Sus estados respectivos se etiquetan como S_k, I_k y O_k.

La red sólo lee un carácter por ciclo (tomamos $t = 0$ para el primer ciclo) y el vector de entrada es $I_k = \delta_{kl}$ cuando se procesa a_l, el l-ésimo

símbolo del alfabeto[1]. La salida $O_k^{[t+1]}$ representa la probabilidad de que a la cadena le siga el símbolo a_k para $k = 1, ...L$ y $O_{L+1}^{[t+1]}$ representa la probabilidad de que la cadena termine en ese punto. La dinámica de la red está regida por las ecuaciones

$$O_k^{[t]} = g(\sum_{j=1}^{N} V_{kj}\ S_j^{[t]}) \tag{7.1}$$

$$S_i^{[t]} = g(\sum_{j=1}^{N}\sum_{k=1}^{L} W_{ijk}\ S_j^{[t-1]}\ I_k^{[t-1]}) \tag{7.2}$$

donde $g(x) = 1/(1 + \exp(-x))$.

El entrenamiento de la red se realiza con una versión del método de Williams y Zipser (1989) conocido como "Real Time Recurrent Learning" (RTRL). La contribución de cada cadena w de la muestra a la energía (o función de error) viene dada por

$$E_w = \sum_{t=0}^{|w|} E_w^{[t]} \tag{7.3}$$

donde

$$E_w^{[t]} = \frac{1}{2}\sum_{k=1}^{L+1}\left(O_k^{[t]} - I_k^{[t]}\right)^2 \tag{7.4}$$

y $I_k^{[|w|]} = \delta_{k,L+1}$. Es sencillo probar que esta energía presenta un mínimo cuando $O_k^{[t]}$ es la probabilidad condicionada de obtener el símbolo a_k después de leer los primeros t símbolos de w. Dado que una probabilidad igual a cero (o uno) es difícil de aprender para la red, se usará una función contractiva de la entrada $f(x) = \epsilon + (1 - 2\epsilon)x$, con ϵ un pequeño número real positivo (o equivalentemente, una expansión de la salida). Esta contracción no afecta esencialmente al formalismo que aquí se presenta.

Normalmente, los pesos W_{ijk} son inicializados con valores pequeños y después son optimizados, pero los valores iniciales $S_i^{[0]}$ de las neuronas ocultas no cambian durante el entrenamiento. Tal y como se

[1] La delta de Kronecker δ_{kl} es 1 si $k = l$ y 0 en caso contrario.

detalla en Forcada & Carrasco (1995), no hay ninguna razón para forzar a estos valores iniciales a permanecer fijos, y el comportamiento de la red mejora si también se permite su aprendizaje. Este detalle resulta aquí de gran importancia, pues la probabilidad predicha por la red $O_k^{[t]}$ depende de los estados $S_i^{[t]}$, mientras que en una tarea de clasificación es suficiente con que los estados de aceptación se encuentren en una zona (la zona de aceptación) y los demás en otra zona (la de no aceptación). Con el fin de que los valores de $S_i^{[t]}$ permanezcan dentro del rango permitido $[0.0, 1.0]$ durante el descenso por gradiente, es mejor definir las entradas de la red $\sigma_i^{[t]}$ tales que $S_i^{[t]} = g(\sigma_i^{[t]})$ y tomar $\sigma_i^{[0]}$ como los parámetros que hay que aprender, ya que estos pueden tomar cualquier valor.

Por tanto, todos los parámetros V_{kj}, W_{ijk} y $\sigma_i^{[0]}$ son inicializados con pequeños valores aleatorios antes del entrenamiento, pero después se aprenden simultáneamente. Cada parámetro P se actualiza de acuerdo con el descenso por gradiente

$$\Delta P = -\alpha \frac{\partial E}{\partial P} + \eta \Delta' P \tag{7.5}$$

con una tasa de aprendizaje α y un término de momento η. El valor $\Delta' P$ representa la variación del parámetro P en la iteración anterior.

La contribución de cada símbolo de w a la derivada de la energía es:

$$\frac{\partial E_w^{[t]}}{\partial P} = \sum_{b=1}^{L+1} (O_b^{[t]} - I_b^{[t]}) O_b^{[t]} (1 - O_b^{[t]}) \sum_{a=1}^{N} (\frac{\partial V_{ba}}{\partial P} S_a^{[t]} + V_{ba} S_a^{[t]} (1 - S_a^{[t]}) \frac{\partial \sigma_a^{[t]}}{\partial P}) \tag{7.6}$$

donde se ha utilizado la siguiente propiedad de la derivada de la función sigmoidea: $g'(x) = g(x)(1 - g(x))$.

Las derivadas con respecto a V_{kj} pueden ser calculadas inmediatamente:

$$\frac{\partial E_w^{[t]}}{\partial V_{kj}} = (O_k^{[t]} - I_k^{[t]}) O_k^{[t]} (1 - O_k^{[t]}) S_j^{[t]} \tag{7.7}$$

Sin embargo, las derivadas con respecto a $\sigma_i^{[0]}$ y W_{ijk} deben ser calcu-

ladas de forma recurrente, usando las ecuaciones siguientes:

$$\frac{\partial \sigma_m^{[0]}}{\partial W_{ijk}} = 0 \tag{7.8}$$

$$\frac{\partial \sigma_i^{[0]}}{\partial \sigma_j^{[0]}} = \delta_{ij} \tag{7.9}$$

$$\frac{\partial \sigma_i^{[t]}}{\partial \sigma_j^{[0]}} = \sum_{a=1}^{N} S_a^{[t-1]}(1 - S_a^{[t-1]}) \sum_{b=1}^{L} W_{iab} \frac{\partial \sigma_a^{[t-1]}}{\partial \sigma_j^{[0]}} I_b^{[t-1]} \tag{7.10}$$

$$\frac{\partial \sigma_m^{[t]}}{\partial W_{ijk}} = \delta_{im} S_j^{[t-1]} I_k^{[t-1]} + \sum_{a=1}^{N} S_a^{[t-1]}(1 - S_a^{[t-1]}) \sum_{b=1}^{L} W_{mab} \frac{\partial \sigma_a^{[t-1]}}{\partial W_{ijk}} I_b^{[t-1]} \tag{7.11}$$

7.3 Resultados y discusión

El comportamiento de la red fue estudiado usando la siguiente gramática regular estocástica:

$$\begin{array}{lcll} S & \rightarrow & 0S & (0.2) \\ S & \rightarrow & 1A & (0.8) \\ A & \rightarrow & 0B & (0.7) \\ A & \rightarrow & 1S & (0.3) \\ B & \rightarrow & 0A & (0.4) \\ B & \rightarrow & 1B & (0.1) \end{array} \tag{7.12}$$

donde los números entre paréntesis indican la probabilidad de las reglas. La probabilidad de final de cadena para una variable puede calcularse como la diferencia entre 1 y la suma de las probabilidades de todas las reglas que contienen a dicha variable en la parte izquierda. Se usaron dos neuronas de entrada ($L = 2$) y tres de salida (asociadas a 0, 1 y final de cadena respectivamente), mientras que se tomó $N = 4$ neuronas ocultas. La red fue entrenada con 800 ejemplos aleatorios[2] y

[2]La muestra contenía 200 cadenas diferentes en promedio.

todas ellas fueron usadas en todas las iteraciones. La tasa de aprendizaje se eligió $\alpha = 0.004$ y el término de momento $\eta = 0.2$. El proceso de entrenamiento para cada muestra duró 1000 iteraciones. La figura 7.2

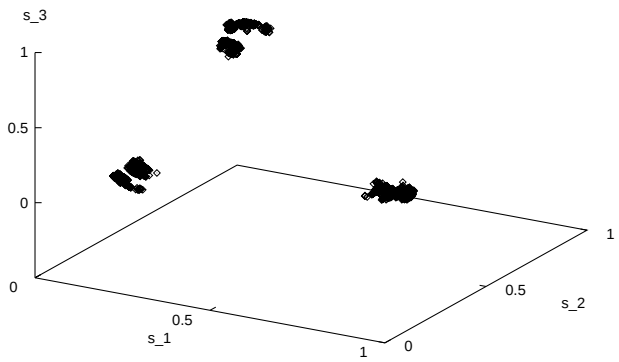

Figura 7.2: Espacio de configuración después del entrenamiento con 800 ejemplos.

muestra un a sección típica del espacio de configuración $[0.0, 1.0]^N$ de una red entrenada. Cada punto representa un vector $S^{[t]}$, generado para todas las palabras de longitud menor o igual que 10. Tal y como se ve en la figura, sólo algunas regiones del espacio son visitadas y se observan tres clusters asociados a las tres variables de la gramática. La aparición de clusters se produjo en todos los experimentos.

En la figura 7.3, se representa la proyección al espacio de salida (O_1, O_2, O_3). Los clusters se proyectan sobre tres regiones centradas en los puntos (0.2,0.8,0.0), (0.7,0.3,0.0) y (0.4,0.1,0.5), de acuerdo con las probabilidades correctas. Debe notarse que, a pesar de no ser una restricción impuesta a la red, las probabilidades suman uno aproximadamente. Debido a la contracción utilizada (se tomó $\epsilon = 0.1$), algunos valores aparecen ligeramente por debajo de cero y deben ser tratados

como imprecisiones o errores de redondeo. La figura demuestra que la red es capaz de predecir la probabilidad del siguiente símbolo con gran precisión.

Una medida de la similitud entre dos distribuciones de probabilidad, en nuestro caso la distribución $q(w)$ calculada por el modelo y las probabilidades $p(w)$ correctas es la llamada distancia de Kullback-Leibler o entropía relativa (Cover & Thomas 1991):

$$d(p,q) = \sum_w p(w) \log_2 \frac{p(w)}{q(w)} \tag{7.13}$$

cuyo valor promedio en los experimentos realizados fue de 0.02 bits. En la tabla siguiente se muestran los resultados en diez experimentos para la distancia $d(A,S)$ entre el autómata correcto A y el conjunto de muestra S, la distancia entre A y la distribución evaluada mediante la red recurrente entrenada N y, finalmente, la distancia entre el autómata correcto A y el extraído de la red A'.

exp.	$d(A,S)$	$d(A,N)$	$d(A,A')$
1	1.255	0.0337587	0.00612741
2	1.196	0.0124381	0.00203993
3	1.198	0.0374824	0.00370043
4	1.309	0.0445123	0.00140762
5	1.276	0.0230212	0.00153648
6	1.196	0.0016789	0.00142349
7	1.306	0.0187031	0.0138799
8	1.405	0.0063514	0.00716541
9	1.288	0.0134842	0.00203797
10	1.495	0.0126591	0.00415552

Se puede observar, que si el autómata era extraído de la red y se calculaban las probabilidades de transición a partir de la muestra, la distribución $q(w)$ generada por este autómata se encontraba a una distancia $d(p,q)$ promedio de 0.004 bits, mucho menor que el valor 0.02 obtenido a partir de la red directamente. Dado que en todos los experimentos aparecieron los tres cúmulos asociados a los tres estados

del autómata, este resultado es igual al que se obtiene con el algoritmo `rlips`, que también en todos los casos identificó la estructura correcta del autómata.

Si tenemos en cuenta, que la distancia entre la muestra y el autómata correcto era en promedio 1.2 bits, es posible afirmar que el autómata extraído a partir de la red mejora significativamente el rendimiento de la red en la predicción de las probabilidades de las cadenas no observadas, aunque sus resultados no mejoren el rendimiento de `rlips`. De hecho, experimentos realizados con otras gramáticas demostraron que no siempre es tan sencilla la formación de cúmulos en el espacio de configuración. Por ejemplo, la gramática

$$\begin{array}{lcll} S & \rightarrow & 0S & (0.5) \\ S & \rightarrow & 1A & (0.5) \\ A & \rightarrow & 0A & (0.1) \\ A & \rightarrow & 1B & (0.4) \\ B & \rightarrow & 0B & (0.3) \\ B & \rightarrow & 1S & (0.3) \end{array} \tag{7.14}$$

requiere la utilización de una red con cinco neuronas. Con muestras de 1000 ejemplos, se produjo la identificación en tres de diez experimentos, con una entropía relativa en estos casos de 0.38 bits (análoga a la de `rlips`). En el resto de los casos no aparecieron cúmulos, lo que produjo una entropía relativa infinita, en contraste con el buen funcionamiento de `rlips` en todos los casos. Merece la pena destacar que a veces se observa la aparición de atractores no puntuales, como el representado en la figura 7.4. Este atractor toroidal demuestra por un lado la existencia de una dinámica compleja en la red cuyo estudio puede resultar de interés. Por otro lado, es un posible indicación de la insuficiencia de los métodos de retro-propagación para obtener con la exactitud requerida el mínimo de la función de error. Una línea de investigación que abordaremos próximamente es el desarrollo o aplicación de otros métodos de entrenamiento para este tipo de redes, como por ejemplo el `ALOPEX` de Unnikrishnan y Venugopal (1994).

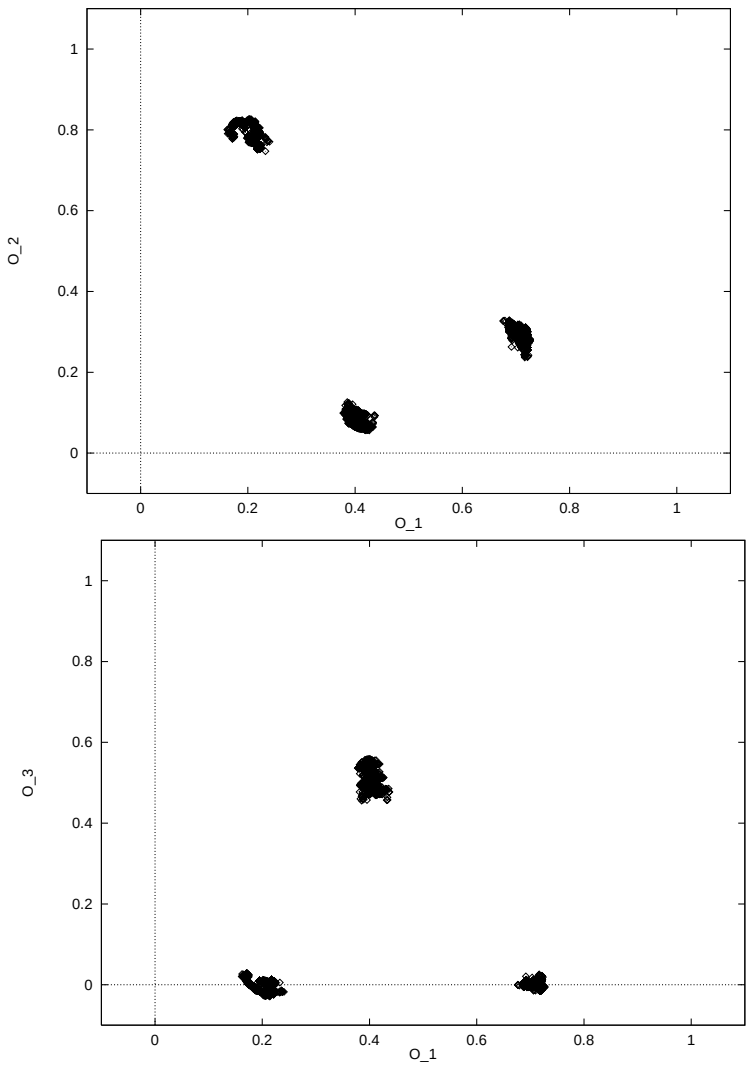

Figura 7.3: Probabilidades de salida después del entrenamiento

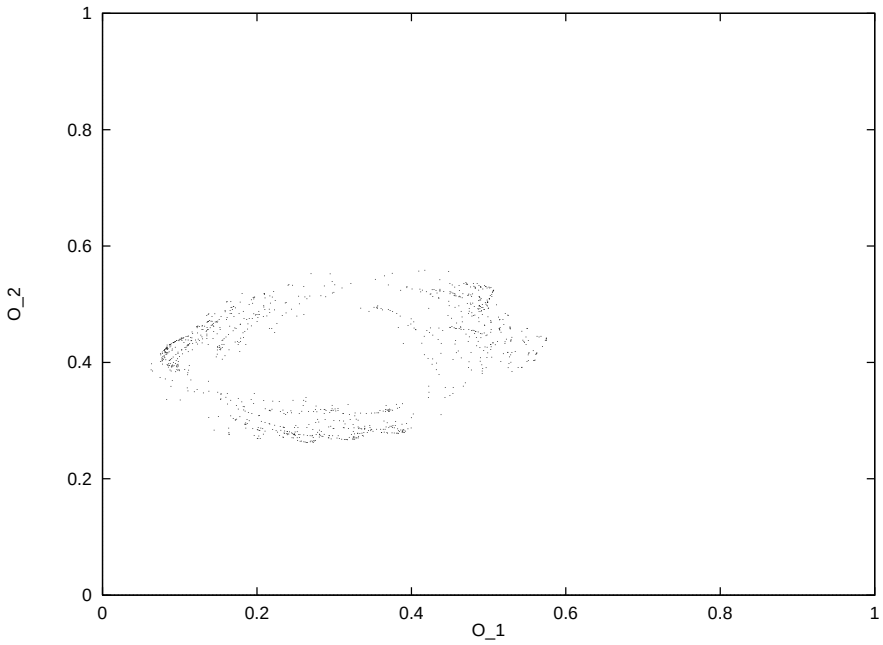

Figura 7.4: Atractor de tipo toroidal obtenido durante el entrenamiento de una red con $N = 5$ para la gramática 7.14

Capítulo 8

Conclusión y perspectivas

Ya sabes
lo que hay que hacer en este mundo:
andar,
como un arado, andar entre la tierra.
Blas de Otero, Que trata de España.

El objetivo del trabajo presentado en esta memoria ha sido el estudio de la inferencia gramatical, esto es, el aprendizaje de las reglas sintácticas que rigen la construcción de las frases de un lenguaje, realizado a partir de fuentes aleatorias de ejemplos. El interés de este problema está motivado porque numerosos problemas prácticos de clasificación de patrones, tales como el reconocimiento de voz o el procesamiento de lenguaje natural, sugieren que la descripción más adecuada ha de realizarse en términos de ejemplos ruidosos o aleatorios.

Aquí nos hemos limitado a estudiar modelos sencillos en los que la generación de los ejemplos se puede realizar mediante un sistema de memoria acotada, lo que normalmente se denomina un autómata finito. En cuanto a los lenguajes estudiados, estos se componían de estructuras lineales (cadenas) o arbóreas.

En el capítulo 3 se han desarrollado algunos métodos para medir la calidad de una hipótesis con respecto al modelo probabilístico real basándose en la teoría de la información. También se ha discutido la importancia de identificar en el límite la estructura del modelo.

En el capítulo 5 se ha propuesto un algoritmo que identifica cualquier lenguaje regular determinista de cadenas construido sobre un alfabeto arbitrario $\mathcal{A}$. La identificación se consigue a partir de ejemplos aleatorios de las cadenas del lenguaje, sin que sean necesarios los contraejemplos. Además las probabilidades del modelo se aproximan rápidamente a las correctas, consiguiendo una rápida reducción de la distancia entre ambas distribuciones expresada por la entropía relativa. Este comportamiento se consigue gracias a la identificación en el límite de la estructura del autómata finito mínimo M que genera el lenguaje. Esta identificación reduce el número de probabilidades que se deben determinar, que pasa de infinito a finito, en concreto a un número $|M||\mathcal{A}|$. Experimentalmente, el algoritmo requiere muy poco tiempo y muestras comparativamente pequeñas para identificar el lenguaje. Para muestras muy grandes, consume un tiempo lineal en función del número de ejemplos (aproximadamente 4 minutos por millón de ejemplos en un computador Hewlett-Packard 715 con 40 MIPS). El algoritmo es, por tanto, adecuado para tareas de reconocimiento en las que aparezcan ejemplos con ruido o fuentes aleatorias.

Recientemente (Ron, Singer & Tishby 1995) han propuesto algoritmos que identifican una subclase de autómatas estocásticos en el sentido PAC. Dichos algoritmos admiten sólo fusiones de estados entre un mismo nivel del árbol de prefijos y conducen a grafos acíclicos. Los autores reconocen que no poden demostrar la corrección del algoritmo cuando se admiten fusiones entre distintos niveles, lo que indica la dificultad de demostrar la convergencia PAC de algoritmos del tipo de los aquí propuestos. Sin embargo, ésta sería una línea de investigación futura de cierto intere's, así como la comparación de los rendimientos de ambos tipos de algoritmos en problemas prácticos.

En el capítulo 6 se ha presentado un algoritmo que es capaz de

aprender gramáticas independientes del contexto a partir de muestras aleatorias que contienen esqueletos de árboles de derivación de las cadenas del lenguaje. El resultado es en el límite, idéntico estructuralmente a la gramática objetivo, es decir, ambas generan el mismo lenguaje estocástico de esqueletos. El algoritmo encuentra su hipótesis en un tiempo que crece sólo linealmente con el tamaño de la muestra. Experimentalmente, la identificación se alcanza con número de ejemplos moderado y la velocidad del algoritmo resulta adecuada para futuras aplicaciones a tareas de reconocimiento.

Una línea de trabajo para el futuro que podría dar lugar a resultados interesantes es la aplicación de estos algoritmo a la inferencia de gramáticas de grafos para su utilización en problemas de reconocimiento de formas. Una extensión más inmediata podría ser la generalización del algoritmo de identificación de lenguajes de árboles para la identificación de lenguajes de grafos dirigidos acíclicos, cuyo interés se justifica por la naturalidad de la descripción de figuras planas (tales como caracteres manuscritos) en términos de este tipo de grafos.

En el capítulo 7 se ha estudiado el poder de generalización de las redes neurales de segundo orden entrenadas con muestras estocásticas de lenguajes regulares. Se ha visto que en algunos casos aparecen cúmulos en el espacio interno de representación asociados a los estados del autómata finito que genera el lenguaje. Si se calculan las probabilidades con el autómata extraído de la red, la distancia entre la distribución verdadera y el modelo se reduce de forma sustancial. Si bien, en general, el comportamiento de los algoritmos simbólicos es muy superior al de los algoritmos basados en redes neurales, las redes neurales de segundo orden se presentan como candidatas para el modelado de procesos estocásticos donde no se dispone todavía de algoritmos gramaticales. Una línea de investigación por desarrollar es la comparación de ambos tipos de métodos cuando la información de las muestras es parcialmente incorrecta o está afectada por ruido. En esa situación, las redes neurales son más robustas frente a defectos en la información y las desventajas implícitas en su entre-

namiento pueden ser total o parcialmente compensadas. Una de las dificultades de la extracción de estados en las redes entrenadas con muestras estocásticas es la necesidad de definir un método de minimización compatible con las incertidumbres estocásticas. Un intento en este sentido se encuentra en la tesis de S.C. Kremer (1996), lo que resulta en una acercamiento a los algoritmos de fusión de estados del tipo `rlips`. Otro problema por resolver es la aparición frecuente de inestabilidades durante el entrenamiento.

Finalmente, una línea de interés en la actualidad es la predicción de series temporales. Esta tarea involucra muestras probabilísticas (por lo que puede ser interesante extender los métodos simbólicos aquí discutidos a este tipo de aplicaciones), pero también fuertes componentes de ruido, no estacionariedad etc, lo que ha originado una exhaustiva utilización en el pasado de modelos neurales. El diseño de un sistema que integre las ventajas de ambos métodos puede resultar una tarea de interés en el futuro próximo.

Alicante, 13 de diciembre de 1999

Bibliografía

- Abe, N and Warmuth, M.K. (1992): "On the Computational Complexity of Approximating Distributions by Probabilistic Automata." *Machine Learning* **9** (205-260).

- Aho, A.V. and Ullman, J.D. (1972): *The theory of parsing, translation and compiling. Volume I: Parsing.* Prentice-Hall, Englewood Cliffs, NJ.

- Angluin, D. (1982): "Inference of reversible languages." *Journal of the Association for Computing Machines* **29** (741–765).

- Angluin, D. (1988): "Identifying languages from stochastic examples." Yale University Technical Report: YALEU/DCS/RR–614.

- Anthony, M. and Biggs, N. (1992): *Computational Learning Theory.* Cambridge University Press, Cambridge.

- Baum, L.E. (1972): "An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process." *Inequalities* **3** (1–8).

- Blair, A.D. and Pollack, J.B. (1996): "Precise Analysis of Dynamical Recognizers." Tech. Rep. CS-95-181, C.S. Department, Brandeis University.

- Brown, P. F., Della Pietra, V.J., deSouza, P.V., Lai, J.C. and Mercer, R.L. (1992): "Class-Based n-gram Models of Natural Language." *Computational Linguistics* **18** (467–479).

- Carrasco, R.C. and Oncina, J. (1994): "Learning stochastic regular grammars by means of a state merging method" in *Grammatical Inference and Applications,* (R.C. Carrasco and J. Oncina, Eds.). Lecture Notes in Artificial Intelligence **862**, Springer-Verlag, Berlin.

- Casey, M. "The Dynamics of Discrete-Time Computation with Application to Recurrent Neural Networks and Finite State Machine Extraction." *Neural Computation* **8** (1135–1178).

- Castaño, M.A., Casacuberta, F. and Vidal, E. (1993): "Simulation of Stochastic Regular Grammars through Simple Recurrent Networks", in *New Trends in Neural Computation* (Eds. J. Mira, J. Cabestany and A. Prieto). Springer Verlag, Lecture Notes in Computer Science **686**, 210–215.

- Chaudhury, R. and Rao, N.V. (1986): "Approximating grammar probabilities: solution of a conjecture." *Journal of the ACM* **33** (702–705).

- Charniak, E. (1994): *Statistical language learning.* MIT Press, Cambridge MA.

- Chomsky, N. (1956): "Three models for the description of language." *IRE Trans. on Information Theory* **2** (113–124).

- Chung, K.L. (1967): *Markov chains with stationary transition probabilities*, Springer-Verlag, Berlin.

- Cook, C.M., Rosenfeld, A., Alan R. Aronson, A.R. (1976): "Grammatical Inference by Hill Climbing." *Informational Science* **10** (59-80).

- Cover, T.M and Thomas, J.A. (1991): *Elements of Information Theory.* John Wiley and Sons, New York.

- Cover, T.M. (1973): "On determining the irrationality of the mean of a random variable." *Annals of Statistics* **1** (862–871).

- Earley, J. (1970): "An efficient context-free parsing algorithm." *Communications of the ACM* **13** (94–102).

- Elman, J. (1990): "Finding Structure in Time." *Cognitive Science* **14** (179–211).

- Feller, W. (1950): *An introduction to probability theory and its applications.* John Wiley and Sons, New York.

- Forcada, M.L. and Carrasco, R.C. (1995): "Learning the Initial State of a Second-Order Recurrent Neural Network during Regular-Language Inference." *Neural Computation* **7** (923–930).

- Fu, K.S. (1982): *Syntactic Pattern Recognition and Applications.* Prentice Hall, Englewood Cliffs, N.J.

- Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z. and Lee, Y.C. (1992): "Learning and Extracting Finite State Automata with Second Order Recurrent Neural Networks." *Neural Computation* **4** (393–405).

- Giles, C.L., Miller, C.B, Chen, D., Sun, G.Z., Chen, H.H., and Lee, Y.C. (1992b) "Extracting and Learning an Unknown Grammar with Recurrent Neural Networks", in *Advances in Neural Information Processing Systems* **4** (J. Moody et al., Eds), Morgan-Kaufmann, San Mateo CA, 317–324.

- Gold, E.M. (1967): "Language identification in the limit." *Information and Control* **10** (447–474).

- Hempel, C.G. (1966): *Filosofía de la ciencia natural.* Alianza Editorial, Madrid, 1973.

- Hoeffding, W. (1963): "Probability inequalities for sums of bounded random variables." *American Statistical Assoc. Jour.* **58** (13–30).

- Hopcroft, J.E. and Ullman, J.D. (1979): *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, Reading, Massachusetts.

- Hume, D. (1748): *Investigación sobre el conocimiento humano.* Alianza Editorial, Madrid 1980.

- Kesidis, G., and Walrand, J. (1993): "Relative entropy between Markov transition rate matrices." *IEEE Trans. on Information Theory* **39** (1056–1057).

- Kneser, R. and Ney, H. (1995): "Improved backing-off for m-gram language modeling." *Int. Conf. on Accoustics, Speech and Signal Process.* **1995** ().

- Kolen, J.F. (1994): "Fool's Gold: Extracting Finite-State Automata from Recurrent Network Dynamics", in *Advances in Neural Information Processing Systems* **6** (C.L. Giles, S.J. Hanson, J.D. Cowan Eds.), Morgan-Kaufmann, San Mateo CA, 501–508.

- Kremer, S.C. (1996): "A theory of grammatical induction in the connectionist paradigm" Ph.D. Thesis, University of Edmonton, Alberta.

- Laird, P.D. (1988): *Learning from good and bad data.* Kluwer Academic Publishers, Norwell, MA.

- Maryanski, F. J. and Booth, T.L. (1997): "Inference of Finite-State Probabilistic Grammars." *IEEE Transactions on Computers* **C26** (521–536).

- Omlin, C.W. and Giles, C.L. (1996): "Extraction of Rules from Discrete-Time Recurrent Neural Networks." *Neural networks* **9** (41–52).

- Oncina, J. and García, P. (1992): "Inferring Regular Languages in Polynomial Time", in *Pattern Recognition and Image Analysis*

(N. Pérez de la Blanca, A. Sanfeliu and E. Vidal Eds.), World Scientific.

- Oncina, J. and García, P. (1994): "Inference of rational tree sets". Universidad Politécnica de Valencia, Internal Report DSIC-ii-1994-23.

- Penrose, R. (1989): *The emperor's new mind* Oxford University Press.

- Pollack, J.B. (1991): "The Induction of Dynamical Recognizers." *Machine Learning* **7** (227–252).

- Popper, K. (1972): *Conocimiento objetivo.* Tecnos, Madrid, 1982.

- Rabiner, L.R., (1989): "A tutorial on hidden Markov models and selected applications in speech recognition." *Proceedings of the IEEE* **77** (257–286).

- Real Academia Española (1992): *Diccionario de la lengua española.* Espasa Calpe, Madrid.

- Reber, A.S. (1967): Implicit Learning of Artificial Grammars. *Journal of Verbal Learning and Verbal Behaviour* **6**, 855–863.

- Ron, D., Singer, Y., Tishby, N. (1995): "On the Learnability and Usage of Acyclic Probabilistic Finite Automata" in *Proceedings of the 8th Annual Conference on Computational Learning Theory (COLT'95)*, 31–40. ACM Press, New York, 1995.

- Russell, B. (1946): *A history of western philosophy.* Londres.

- Sakakibara, Y. (1992): "Efficient learning of context-free grammars from positive structural examples." *Information and Computation* **97** (23–60).

- Smith, A.W. and D. Zipser, Z. (1989): "Learning Sequential Structure with the Real-Time Recurrent Learning Algorithm." *International Journal of Neural Systems* **1** (125–131).
- Soule, S.(1974): "Entropies of probabilistic grammars." *Information and control* **25** (57–74).
- Stolcke A. and Omohundro,S. (1993): "Hidden Markov Model Induction by Bayesian Model Merging", in *Advances in Neural Information Processing Systems* **5** (C.L. Giles, S.J. Hanson and J.D. Cowan Eds.), Morgan Kaufman, Menlo Park, California.
- Stolcke, A. and Segal, J. (1994): "Precise n-gram probabilities from stochastic context-free grammars." International Computer Science Institute, TR-94-007, Berkeley, CA.
- Suen, C. Y. (1979): "n-gram statistics for natural language understanding and text processing." *IEEE Transactions on pattern analysis and machine intelligence* **2** (164–172).
- K.P. Unnikrishnan, K.P, Venugopal, K.P (1994): "Alopex: a correlation-based algorithm for feedforward and recurrent neural networks." *Neural Computation* **6** (469–490).
- Valiant, L.G. (1984): "A theory of the learnable." *Communications of the ACM* **27** (1134–1142).
- van der Mude, A. and Walker, A. (1978): "On the Inference of Stochastic Regular Grammars." *Information and Control* **38** (310–329).
- Viterbi, A.J. (1967): "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm." *IEEE Transactions on Information Theory* **13** (260–269).
- Williams, R.J. and Zipser, D. (1989): "A learning algorithm for continually running fully recurrent neural networks." *Neural Computation* **1** (270–280).

- Watrous R.L. and Kuhn, G.M. (1992): "Induction of Finite-state Languages Using Second-Order Recurrent Networks." *Neural Computation* **4** (406–414).

- Watrous, R.L. and Kuhn, G.M. (1992b): "Induction of Finite-State Automata Using Second-Order Recurrent Networks", in *Advances in Neural Information Processing Systems 4* (J. Moody et al., Eds), Morgan-Kaufmann, San Mateo CA, 306–316.

- Wetherell, C.S. (1980): "Probabilistic Languages: A Review and Some Open Questions." *ACM Computing Survey* **12** (361–379).

- Younger, D.H. (1967): "Recognition and parsing of context-free languages in time n^3." *Information and Control* **10** (189–208).

- Ziv, J. and Merhav, N.: "A measure of relative entropy between individual sequences with application to Universal classification." *IEEE Trans. on Information Theory* **39** (1270–1279).