Universitat d'Alacant
Universidad de Alicante

Departamento de Lenguajes
y
Sistemas Informáticos

Inferencia estocástica y aplicaciones de los lenguajes de árboles

Autor: Juan Ramón Rico-Juan

Tesis doctoral

Mayo 2001

La presente memoria constituye la tesis doctoral presentada por Juan Ramón Rico-Juan para la obtención del título de doctor en Informática y ha sido desarrollada bajo la dirección del Dr. Rafael C. Carrasco Jiménez y del Dr. Jorge Calera-Rubio, profesores del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante.

Agradecimientos: A todos aquellos que contribuyeron al desarrollo de esta tesis, bien con su dirección (Rafael C. Carrasco y Jorge Calera), sus consejos (Mikel L. Forcada) o su insustituible aportación técnica (José Oncina y Juan Antonio Pérez), a todos mis compañeros del grupo de investigación de *Reconocimiento de Formas e Inteligencia Artificial* y también al *Departamento de Lenguajes y Sistemas Informáticos*.

Índice General

Presentación

La *inferencia gramatical* es un caso particular del *aprendizaje inductivo* que se puede definir como la tarea de descubrir estructuras comunes en ejemplos que supuestamente ha generado el mismo proceso. El conjunto de ejemplos es usualmente un conjunto de cadenas definidas sobre un alfabeto específico que se describe como un *lenguaje formal*.

La inferencia *gramatical estocástica* además incorpora información estadística al modelo.

La inferencia gramatical se ha utilizado durante las últimas décadas en reconocimiento de formas y en otras tareas como la compresión de datos y el procesamiento de lenguaje natural. El reconocimiento de formas se ha mostrado como una herramienta fundamental en tareas como: reconocimiento de caracteres manuscritos, de firmas, del habla, análisis e interpretación de imágenes, etc.

Esta tesis se centra en la inferencia gramatical estocástica, concretamente en el estudio de árboles etiquetados y sus aplicaciones. El trabajo está dividido en tres bloques. El primero, dedicado a los *lenguajes de árboles k-testables,* donde se incorpora un modelo estocástico a este tipo de lenguajes (capítulo 4) y dos de sus posibles aplicaciones (capítulos 5 y 6). El segundo bloque se denomina *otros modelos de árboles* y reune dos aplicaciones sobre árboles etiquetados que no utilizan modelos probabilísticos (capítulos 7 y 8). Por último, en el tercer bloque se indican *las conclusiones y los trabajos futuros*.

El capítulo 1 es introductorio y describe diferencias entre tipos de datos estructurados, cómo éstos se pueden aprender con modelos para luego utilizarlos en tareas como la clasificación de muestras o la compresión de texto.

Si se requieren nociones previas sobre la compresión y conceptos relacionados con ella como la entropía, sistemas de codificación, etc. en el capítulo 2 se hallan definiciones y conceptos que se usarán el los capítulos posteriores; si no, se puede prescindir de su lectura.

En el capítulo 3 se revisan diversos modelos estocásticos para cadenas y su uso en la compresión y en la clasificación con algunos problemas a considerar.

La contribución original de esta tesis comienza propiamente en el capítulo 4, donde se define un modelo estocástico de árboles y su inferencia a partir de muestras.

El capítulo 5 aborda la compresión de ficheros de datos con árboles etiquetados aplicando diferentes variantes del modelo anterior, comparando los resultados con otros modelos de compresión muy difundidos. El capítulo 6 aplica diferentes variaciones de nuestro modelo a la clasificación de caracteres manuscritos (previamente segmentados y transformados en árboles etiquetados). En este capítulo, también se realiza una comparación con otro modelo de clasificación de árboles etiquetados.

El capítulo 7 describe un sistema nuevo de compresión para ficheros de puntos en tres dimensiones utilizando una distribución por cada dimensión.

En el capítulo 8 detalla un sistema de reconocimiento de palabras manuscritas continuas fuera de línea utilizando los árboles etiquetados como representación de las palabras manuscritas y la distancia de edición entre árboles para su clasificación.

Por último se exponen las conclusiones y trabajos futuros derivados de la realización de esta tesis.

Capítulo 1

Introducción

1.1 Cadenas, árboles y grafos

Los sistemas informáticos transforman los datos de entrada en una representación simbólica más adecuada para su tratamiento. Intuitivamente, cuanto más natural sea esta representación, más rendimiento se obtendrá en los resultados. Por ejemplo, el concepto matemático de grafo parece más adecuado para describir dibujos e imágenes que el concepto de cadena o sucesión de símbolos. Por ello, parece más sencillo encontrar las características comunes a un conjunto de imágenes si se codifican éstas como grafos que si se hace como cadenas. Dado que encontrar éstos rasgos comunes es el primer paso del diseño de la mayoría de algoritmos de clasificación de imágenes, es de esperar que se obtengan así mejores resultados.

Por otro lado, los sistemas de clasificación suelen interpretar los datos como agrupaciones de elementos simples o *primitivas* que se agrupan en *patrones* siguiendo ciertas reglas. Una forma muy utilizada para describir estas reglas de formación de patrones es mediante el uso de reglas gramaticales (Fu 1982), lo que se conoce como aproximación sintáctica.

En particular, las gramáticas de grafos han sido estudiadas teóricamente (Engelfriet and Heyker 1991; Fahmy and Blostein 1992; Courcelle et al. 1993)

y aplicadas en tareas de reconocimiento de patrones (Matsello 1991; Flasinski 1992; Rekers 1994).

Sin embargo, la gran capacidad de representación de este tipo de gramáticas va unido a una cierta complejidad matemática y, sobre todo, a la dificultad de diseñar algoritmos eficientes de análisis sintáctico, cálculo de distancias (similitud) o de aprendizaje automático de estas gramáticas.

El concepto matemático de árbol y las gramáticas de árboles ocupan un interesante lugar intermedio entre los métodos de cadenas y los de grafos. Por un lado su capacidad expresiva es superior a la de las cadenas y permiten describir con naturalidad estructuras en las que se dan relaciones jerárquicas entre sus componentes. Por otro lado, su manejo desde el punto de vista teórico es mucho más sencillo que el de las gramáticas de grafos y disponemos de algoritmos eficientes para muchas tareas, entre ellas el aprendizaje automático de este tipo de gramáticas (Sima'an et al. 1996; Abe and Mamitsuka 1997; Carrasco et al. 2001).

Por esto, aún a costa de perder ciertamente capacidad descriptiva, las gramáticas de árboles constituyen un campo de investigación de interés. Tradicionalmente su aplicación se ha visto limitada a ciertas áreas como la bioquímica y genética (Abe and Mamitsuka 1997) o el procesamiento del lenguaje natural (Sima'an et al. 1996). Uno de los problemas con los que nos hemos encontrado durante la realización de este trabajo ha sido la falta de bases de datos extensas cuyo contenido fuesen datos estructurados como árboles. Esta dificultad es más relevante, si cabe, para los métodos de tipo estadístico que hemos planteado, donde la necesidad de grandes corpora es patente. Por este motivo, algunos de los métodos que proponemos deberán ser ensayados más exhaustivamente más adelante. Es de esperar que el crecimiento y desarrollo de bancos lingüísticos del tipo del Penn Tree-bank (Marcus et al. 1994), la expansión de la estructuración de documentos en las bibliotecas digitales (por ejemplo, mediante XML, que aporta una estructura arbórea al texto), etc. nos permitan disponer inminentemente de más y mayores bases

de datos.

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 and 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). Habitualmente, 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 reducida 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-Kasami (Aho and Ullman 1972), el de Early (1970), o para el caso particular de gramáticas regulares Viterbi (1967).

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 (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 $S = \{x_1, x_2, \ldots\}$ 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 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.

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. La información contenida en la muestra completa S es suficiente para descartar todas las hipótesis que son demasiado generales. Sin embargo, 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 de varias formas: por ejemplo, si se dispone de información adicional acerca del orden de presentación de los elementos de la muestra, si se asume que los ejemplos han sido generados de acuerdo con una distribución probabilística preestablecida aunque desconocida o, como en el caso de esta tesis, se restringe la clase de lenguajes que se quiere inferir. En nuestro caso supondremos que los lenguajes pertenecen a la clase k-testable, un subconjunto propio de los lenguajes regulares.

La ventaja de estos modelos es, por tanto, que en caso de que los datos puedan describirse por un modelo k-testable, disponemos de algoritmos que garantizan que el modelo correcto será identificado en el límite. Sobre este modelo construiremos un modelo probabilístico basado en el criterio de máxima verosimilitud y lo aplicaremos en tareas de clasificación y compresión de datos.

Dentro de la aproximación sintáctica, los modelos estocásticos han sido

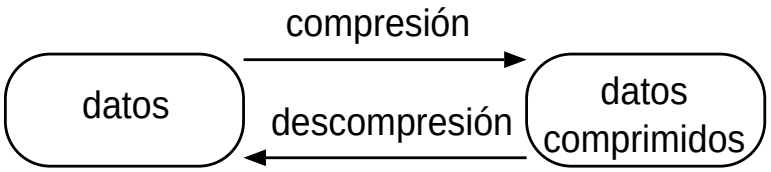


Figura 1.1: Esquema de compresión reversible

ampliamente utilizados en tareas de predicción y de clasificación. Tal y como argumenta Fu (1982), los modelos estocásticos permiten tratar de forma natural fenómenos como la superposición de patrones entre clases, esto es, patrones que pertenecen a más de una clase bien debido a errores de clasificación o a la presencia de ruido en la fuente. En algunos problemas, como en la predicción de series temporales, los modelos estocásticos constituyen la aproximación más realista (piénsese por ejemplo en la predicción bursátil). En otros, como el tratamiento del lenguaje natural (Manning and Schütze 1999), los métodos probabilísticos constituyen uno de los procedimientos más utilizados para resolver la ambigüedad léxica y sintáctica.

Para que el aprendizaje sea posible es en general necesario hacer algunas hipótesis sobre la forma en que se generan los patrones observados. En nuestro caso, supondremos que el modelo subyacente pertenece a la clase k-testable.

1.3 Compresión de texto

La compresión de datos, en general, transforma una entrada en códigos que ocupan menor espacio. En principio interesa que esta transformación sea reversible (figura 1.1). En este caso, se habla de compresión sin pérdidas ("lossless compression"). Aunque hay situaciones en las que una cierta distorsión es aceptable (por ejemplo el estándar JPG admite cierta distorsión de la imagen), en este trabajo nos centraremos en métodos de compresión sin pérdidas y, específicamente, para entradas de tipo texto estructurado con

	estáticos	adaptativos
símbolo a símbolo	Huffman	compresión aritmética
basado en diccionario	—	Ziv-Lempel

Tabla 1.1: Ejemplos de distintos tipos de compresión de texto.

Método	Nº aproximado de bits promedio por símbolo
Huffman	5
Ziv-Lempel	4
Compresión aritmética	2

Tabla 1.2: Eficiencia relativa de distintos métodos para comprimir texto en inglés. Cada símbolo sin comprimir ocupa un byte = 8 bits.

forma de árbol y nubes de puntos que describen superficies 3D

Los métodos de compresión de texto pueden dividirse en:

1. métodos *símbolo a símbolo*, que codifican un símbolo cada vez, asignando un código más corto a aquellos más probables y

2. métodos basados en *diccionario*, que codifican fragmentos de texto como referencias a entradas de un diccionario.

Además, la compresión puede ser *estática*, si el código generado no depende de la posición del fragmento comprimido en el texto, o *adaptativa*, si la compresión se realiza teniendo en cuenta la parte ya comprimida. Estos dos criterios permiten clasificar los métodos más importantes de compresión de texto tal y como se representa en la tabla 1.1.

En la práctica, la división entre métodos símbolo a símbolo y métodos basados en diccionario no está clara en todos los casos, ya que hay métodos híbridos que utilizan características de las dos técnicas. No obstante, en esta tesis nos centraremos sobre todo en los métodos *adaptativos símbolo a símbolo*: en particular se usarán modelos de tipo *PPM* (Prediction by Partial Matching) y de *contexto finito* (k-gramas) combinados con técnicas de *compresión aritmética*. El motivo para ello es que los mejores resultados para

compresión de texto publicados hasta la fecha (Witten et al. 1987; Nelson 1991; Witten et al. 1999) se obtienen combinando técnicas de codificación aritmética con modelos adaptativos (tabla 1.2).

Sin embargo, es preciso aclarar que la compresión aritmética tiene algún inconveniente: no permite que la descompresión del texto comience en un punto arbitrario (algo que sí permiten los métodos estáticos) y además resulta en general más lenta que los métodos basados en diccionario. A cambio, los métodos adaptativos no precisan transmitir el modelo que se está usando para la compresión, algo que requiere un espacio adicional.

Capítulo 2

Introducción a la codificación aritmética

Los modelos estocásticos son muy importantes para la compresión, ya que ellos son los encargados de predecir el próximo símbolo con una determinada probabilidad y de ella dependerá la eficiencia de la compresión.

Por ejemplo, si quisiéramos comprimir un texto en el que pueden aparecer únicamente las palabras {`muy`, `buenos`, `días`}, bastaría con asignar los códigos binarios {00,10,11} respectivamente, a las palabras. Se sustituye la palabra por el código correspondiente y reducirá el tamaño del texto original. Si tenemos el texto

`buenos buenos días muy buenos días`

que ocupa 34 bytes (un byte por carácter) se codificaría como

101011001011

con un tamaño total de 11 bits. Pero si sabemos que la palabra `buenos` aparece con mucha más probabilidad que las otras dos, se podrían codificar las palabras {`muy`, `buenos`, `días`} con {00, 1, 01}. Si codificamos de nuevo

el ejemplo anterior,

$$110100101$$

el mismo texto ahora ocuparía 9 bits.

Siguiendo con el ejemplo anterior nos podríamos hacer varias preguntas:

1. ¿Existe un límite (en número de bits) para la compresión?
2. Si existe este límite, ¿cuáles son los códigos óptimos que debemos utilizar?

La respuesta a la primera pregunta es que se puede establecer un límite para la compresión, llamado *límite entrópico*, de forma que no existe método de compresión capaz de comprimir un símbolo en promedio con menor número de bits.

Respecto de la segunda pregunta, existen varios métodos para asignar los códigos óptimos en la compresión. Dos de los principales son los *códigos de Huffman* y la *codificación aritmética*.

2.1 Códigos, entropía y compresión

Sea una variable aleatoria X con un rango $\mathcal{A}_X = \{a_1, \ldots, a_N\}$ y probabilidades $\mathcal{P}_X = \{p_1, \ldots, p_N\}$, es decir, $P(a_i) = P(X = a_i) = p_i$, $p_i \geq 0$ y $\sum_{x \in \mathcal{A}_X} P(x) = 1$.

Una codificación (binaria) $\mathcal{C}$ para la variable aleatoria X es una función $\mathcal{C} : \mathcal{A}_X \to \{0,1\}^+$. La codificación de un texto $\mathcal{C}^+ : \mathcal{A}_X^+ \to \{0,1\}^+$es

$$\mathcal{C}^+(x_1 x_2 \cdots x_N) = \mathcal{C}(x_1)\,\mathcal{C}(x_2) \cdots \mathcal{C}(x_N) \tag{2.1}$$

Algunas condiciones son básicas para que la codificación sea útil. La primera es que la codificación de una cadena sea unívocamente descodificable, lo que permite la reversibilidad de la codificación.

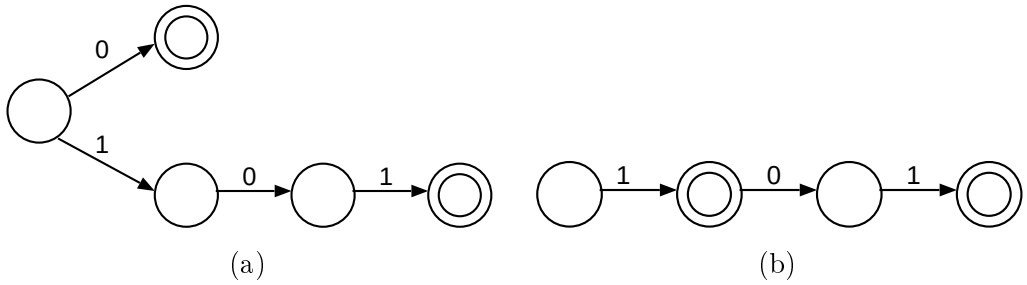


Figura 2.1: Árbol de prefijos de las codificaciones: a) $\mathcal{C}_1 = \{0, 101\}$; b) $\mathcal{C}_2 = \{1, 101\}$.

La codificación $\mathcal{C}_X$ es *unívocamente descodificable* si $\mathcal{C}^+$es inyectiva en $\mathcal{A}_X^+$, es decir,

$$\forall x, y \in \mathcal{A}_X^+,\ x \neq y \Rightarrow \mathcal{C}^+(x) \neq \mathcal{C}^+(y). \tag{2.2}$$

Por ejemplo, sea una variable aleatoria X: $\mathcal{A}_X = \{a, b, c, d\}$, $\mathcal{P}_X = \{1/2, 1/4, 1/8, 1/8\}$ y la codificación $\mathcal{C}_0 = \{1000, 0100, 0010, 0001\}$. Entonces $\mathcal{C}_0(ab) = 10000100$. La definición de $\mathcal{C}_0$ es un ejemplo de codificación unívocamente descodificable, ya que no es posible obtener el mismo resultado con dos entradas distintas.

Una segunda condición es que el código debe ser eficientemente descodificable. Esto es, que no sea preciso analizar todo el código para comenzar la descodificación. Para ello, se introduce el siguiente concepto.

La codificación $\mathcal{C}_X$ es *instantánea* si ningún código es prefijo de otro. En ese caso, el final de la codificación de un símbolo se puede detectar instantáneamente. Toda codificación instantánea es unívocamente descodificable.

Si $\mathcal{C}(\mathcal{A})$ es el conjunto de símbolos codificados y $\mathcal{C}$ es instantánea, el árbol de prefijos de $\mathcal{C}(\mathcal{A})$ sólo tiene asignaciones de códigos en las hojas (véase figura 2.1). Por ejemplo, la codificación $\mathcal{C}_1 = \{0, 101\}$ es instantánea, ya que 0 no es prefijo de 101, ni 101 es prefijo de 0. En cambio $\mathcal{C}_2 = \{1, 101\}$ no es instantánea, porque 1 es prefijo de 101 por tanto, cuando descodifiquemos

una secuencia $1\cdots$ no es suficiente con examinar el 1 y tomar una decisión, porque podría ser el código $1 \in C_2$ o el comienzo del código $101 \in C_2$. Como se aprecia en la figura 2.1 el árbol b) contiene un estado interno final.

El *número de bits esperado* con la codificación $\mathcal{C}_X$ es

$$L\left(\mathcal{C}_X, X\right) = \sum_i p(a_i) l(a_i) \tag{2.3}$$

donde $l(a_i)$ es la longitud de la codificación para el evento a_i.

Por otro lado, el *contenido de información* para codificar un suceso $a_i \in \mathcal{A}_X$ se define como $I\left(a_i\right) = \log_2\left(1/p_i\right)$, y la entropía, se define como:

$$H\left(X\right) = \sum_i p(a_i) I(a_i) \tag{2.4}$$

Se puede demostrar (Shannon 1948) que para cualquier codificación $\mathcal{C}_X$

$$H\left(X\right) \leq L\left(\mathcal{C}_X, X\right), \tag{2.5}$$

es decir, la entropía es una cota inferior de la longitud promedio de codificación por símbolo.

Cuanto más nos acerquemos a la entropía, mejor habremos asignado los códigos $\mathcal{C}_X$. Comparando las ecuaciones (2.4) y (2.3) observamos que si hacemos corresponder $l(a_i)$ con $I(a_i)$ obtendremos una codificación óptima. Obviamente, esto sólo es posible si para todos los elementos, los valores $I(a_i)$ son enteros.

Supongamos que tenemos un dado con 6 caras distintas y que todas ellas son equiprobables, por lo que nuestro alfabeto $\mathcal{A}_X$ tiene 6 símbolos, $P\left(X\right) = 1/6$ y $\mathcal{C}_X = \{000, 001, 010, 011, 100, 101, 110\}$. En ese caso tendríamos una entropía $H\left(X\right) = 2.59$ y $L\left(\mathcal{C}_X, X\right) = 3$ satisface la ecuación (2.5). Supongamos ahora un dado con probabilidades $\mathcal{P}_Y = \{1/2, 1/10, 1/10, 1/10, 1/10, 1/10\}$ y $\mathcal{C}_Y = \{0, 1000, 1001, 101, 110, 111\}$. Tendríamos una entropía $H\left(Y\right) = 2.16$ y $L\left(\mathcal{C}_Y, Y\right) = 2.2$. También en este caso se satisface la ecuación (2.5) pero

podemos comprimir más los datos usando la codificación $\mathcal{C}_Y$ que la $\mathcal{C}_X$, que seguiría usando 3 bits por símbolo. Este ejemplo ilustra la conveniencia de, en la medida de lo posible, asignar códigos más largos a los resultados más improbables. El ideal es $l(a_i) = log_2(1/p_i)$. Esto es precisamente lo que intenta la codificación de Huffman que presentamos brevemente a continuación.

Nótese que si $p_i = 0$ tenemos $I(a_i) = \infty$, lo que producirá en el mejor de los casos códigos de longitud enorme para ese símbolo. Debemos evitar que el modelo asigne una probabilidad cero a un símbolo que puede aparecer en la práctica, ya que ello deterioraría considerablemente la compresión.

2.2 Codificación de Huffman

Un método de compresión eficiente basado en probabilidades es la codificación de Huffman. En el trabajo de Huffman (1952) se describía un método para la creación de una tabla de códigos para un conjunto de símbolos dado, sabiendo sus probabilidades. La tabla de codificación de Huffman garantiza que produce la menor cantidad posible de bits para cada símbolo de entrada mediante los llamados *códigos de mínima redundancia*. Una descripción de cómo se generan los códigos de Huffman puede hallarse en Cover and Thomas (1991), Mackay (1998) y Witten et al. (1999).

La codificación de Huffman asigna un código de salida a cada símbolo s. La longitud de estos códigos se asigna estrictamente dependiendo de la probabilidad del símbolo. El número de bits óptimos usados para cada símbolo es $I(s)$.

Un problema que aparece con esta codificación es el llamado *bit extra*. Hufffman garantiza que la longitud media de sus códigos cumple

$$H(X) \leq L(\mathcal{C}_X, X) \leq H(X) + 1 \tag{2.6}$$

con lo que para cada código tenemos de 0 a 1 bit de más. Si el valor de $H(X)$ es grande no es importante esta fracción extra, pero si es pequeño, los

eventos con alta probabilidad incorporan este bit extra haciendo ineficiente la compresión. Por ejemplo, si un carácter tiene una probabilidad de $p(s) = 0.99$, el número óptimo de bits es $I(s) = 0.0145$ que es un número cercano a 0; en cambio, el esquema de Huffman le asignaría 1 bit para codificarlo.

Un segundo problema de la codificación de Huffman surge cuando queremos hacerla adaptativa. Cuando se usan métodos no adaptativos, el programa de compresión lee los datos de entrada y calcula las estadísticas. Los datos son codificados con respecto a las estadísticas que no cambian a lo largo del proceso de compresión. El codificador típicamente construye la tabla de probabilidades a partir de la entrada, por lo que el descodificador debe tener esta tabla antes de comenzar a descodificar. Esta tabla se le debe pasar como cabecera en el fichero comprimido, lo que incrementa inevitablemente el tamaño de éste. Obviamente, esta aproximación tendrá problemas serios si la estadística para el modelo consume más espacio que los datos que se comprimen.

La solución a este problema es usar *esquemas adaptativos*. En la compresión de datos adaptativa, el compresor y el descompresor comienzan con el mismo modelo. El compresor codifica un símbolo usando el modelo existente, entonces actualiza el contador del modelo con el nuevo símbolo. El descompresor descodifica un símbolo usando el modelo existente, y luego actualiza el modelo. Mientras el algoritmo que actualiza el modelo funcione exactamente igual para el compresor y el descompresor, el proceso puede funcionar perfectamente sin la necesidad de pasar estadísticas del compresor al descompresor. La compresión adaptativa de datos tiene la desventaja de comenzar a comprimir con estadísticas poco óptimas. Sin embargo, si descontamos el tamaño de la estadística del modelo de los datos comprimidos, un algoritmo adaptativo será generalmente más eficiente que un modelo estadístico fijo.

El problema que tiene combinar esquemas adaptativos con la codificación de Huffman es que hay que recalcular los códigos de Huffman cada vez que se lee un carácter y esto es un proceso muy costoso. De hecho, algoritmos

Símbolo	Probabilidad
ESPACIO	1/10
A	3/10
C	1/10
D	1/10
E	1/10
N	2/10
U	1/10

Tabla 2.1: Tabla de probabilidades correspondiente al mensaje "UNA CADENA"

eficientes que construyen códigos de Huffman adaptativos no aparecieron hasta 20 años después de que Huffman publicara su codificación (Gallager 1978; Cormack and Horspool 1984; Vitter 1989).

2.3 Codificación aritmética

La codificación aritmética es un respetable candidato para reemplazar la codificación de Huffman. En codificación aritmética no se reemplaza un símbolo por un código determinado. En cambio, se coge la entrada y se reescribe como un único número en la salida. Cuando más largo es el mensaje más bits se necesitan para codificar el número final.

La ventaja fundamental de la codificación aritmética respecto a la de Huffman es que trata el texto completo $s_1 s_2 \ldots s_N$ como el resultado de un experimento aleatorio, con lo que el efecto de pérdida de un bit es despreciable. De hecho, en la codificación aritmética se asigna un número entre 0 y 1 a cada texto posible.

Para construir el número de la salida eficientemente se debe tener un modelo que permita predecir la probabilidad del siguiente número sin tener que procesar el resto de la entrada.

Por ejemplo, vamos a codificar el mensaje "UNA CADENA" suponiendo que las probabilidades son las que muestran la tabla 2.1. Una vez que la tabla

Símbolo	Probabilidad	Rango(low,high)
ESPACIO	1/10	[0.0, 0.1)
A	3/10	[0.1, 0.4)
C	1/10	[0.4, 0.5)
D	1/10	[0.5, 0.6)
E	1/10	[0.6, 0.7)
N	2/10	[0.7, 0.9)
U	1/10	[0.9, 1.0)

Tabla 2.2: Tabla de probabilidades y rangos correspondiente al primer símbolo del texto.

Nuevo símbolo	low	high
	0.0	1.0
U	0.9	1.0
N	0.97	0.99
A	0.972	0.978
ESPACIO	0.972	0.9726
C	0.97224	0.9723
A	0.972246	0.9722568
D	0.972255	0.9722568
E	0.97225608	0.97225626
N	0.972256206	0.972256242
A	0.9722562096	0.9722562204

Tabla 2.3: Traza completa de codificación correspondiente al mensaje "`UNA CADENA`"

Algoritmo 2.1 Codificación Aritmética

```
low=0
high=1
mientras queden símbolos de entrada haz
  lee símbolo
  range = high-low
  high = low + range*high(símbolo)
  low = low + range*low(símbolo)
fin mientras
devuelve low
```

Algoritmo 2.2 Descodificación Aritmética

```
lee número codificado
repite
   busca símbolo: low(símbolo)<=número<high(símbolo)
   imprime símbolo
   range = high(símbolo)-low(símbolo)
   número = (número - low(símbolo))/ range
hasta  no hay más símbolos
devuelve low
```

de probabilidades se conoce se necesita asignar un rango distinto entre 0 y 1 para cada uno de los símbolos del alfabeto de la entrada. Esto se hace construyendo la función de distribución acumulada. Continuando con el ejemplo anterior se obtendría la tabla 2.2.

El símbolo más significativo es el primero, ya que marcará el rango principal en el que se encontrará el número flotante final. Por ejemplo, cuando codificamos el mensaje `"UNA CADENA"` el primer símbolo es una `U` cuyo rango es [0.9, 1.0) por lo que ya sabemos que el número final del mensaje estará en este rango.

Después de procesar el primer símbolo de la entrada, sabemos que el resultado está en el rango [0.9, 0.1). Para codificar el segundo símbolo, `N`, con un rango de [0.7, 0.9), debemos insertar este segundo rango dentro del primer rango con lo que el resultado después de procesar `UN` sería [0.97, 0.99). De este modo seguiría el proceso hasta llegar al final del mensaje. El algoritmo en pseudo-código se muestra en el algoritmo 2.1 y una traza completa para el ejemplo que estamos tratando se muestra en la tabla 2.3. El número final que codifica el mensaje de `"UNA CADENA"` sería cualquiera entre 0.9722562096 y 0.9722562204.

Se debe destacar que no es preciso almacenar los números completos en la memoria, lo que resultaría inviable para textos grandes. Por ejemplo, una vez que el rango es $[0.97, 0.99)$, ya sabemos que la primera cifra de la salida es un nueve, se puede enviar a la salida y almacenar sólo los límites $[7, 9)$, liberando

memoria para seguir el procesamiento. Además, se utiliza aritmética entera para evitar errores. Véase, por ejemplo, Nelson (1991).

De igual forma pero en sentido inverso el descodificador leería el número codificado e iría interpretando los símbolo según sus rangos. El algoritmo en pseudo-código se muestra en el algoritmo 2.2.

Si se utiliza la codificación aritmética para comprimir texto, es necesario que se cumplan dos condiciones para que ésta sea efectiva:

- que la entropía de la entrada sea pequeña, ya que la entropía constituye una cota inferior de cualquier sistema de compresión;
- que el modelo estocástico que utilicemos para predecir los símbolos de la secuencia de entrada lo haga de la forma más correcta posible.

La compresión no es posible si la entropía es muy elevada. Por ejemplo, con 256 símbolos y una probabilidad uniforme de 1/256 se tendría una entropía $H = 8$ y generaría un fichero de salida de igual tamaño que el de la entrada, puesto que cada símbolo usaría 8 dígitos binarios para codificarse.

Por otro lado, la necesidad de predecir exactamente la probabilidad de los símbolos de entrada es inherente a la naturaleza de la codificación aritmética. Si tuviéramos los 256 símbolos anteriores, con una distribución diferente a una uniforme, necesitaremos un modelo estocástico que prediga los símbolos de la forma más exacta posible. Lo más destacable de este tipo de codificación es que reduce el número de los dígitos binarios necesarios para codificar un carácter. Por ejemplo, si la letra `e` representa el 25% de los datos, podría codificarse con solamente 2 dígitos binarios. Si la letra `Z` representa solamente el 0.1% de los datos, podría codificarse con 10 dígitos binarios. Si el modelo no genera las probabilidades correctamente, puede tomar 10 dígitos binarios para representar `e` y 2 dígitos binarios para representar `Z`, causando la expansión de la entrada en vez de la compresión.

Una descripción más detallada de la codificación aritmética puede encontrarse en Witten et al. (1987), que también incluye una implementación

de este tipo de codificación. Una descripción más pedagógica se halla en el libro de Mackay (1998). En Nelson (1991) se presenta una implementación eficiente de la codificación aritmética usando lenguaje C.

Capítulo 3

Modelos estocásticos para la compresión y la clasificación de cadenas

Todo modelo estocástico asigna probabilidades a los distintos resultados observables y éstas pueden servir para la compresión o para la clasificación de nuevos datos.

Cuando se comprime utilizando un codificador aritmético, éste necesita un modelo probabilístico para codificar los símbolos. El descompresor usa el mismo modelo estocástico para encontrar cuál es el símbolo actual a partir de la salida del compresor (figura 3.1).

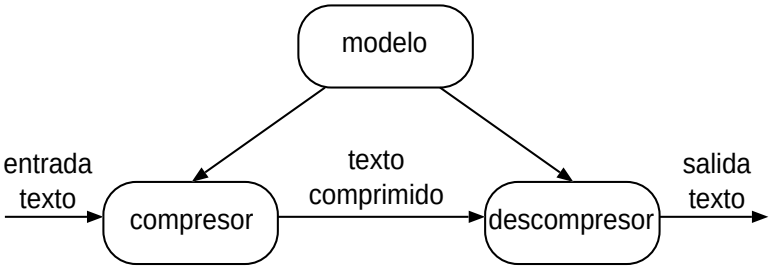


Figura 3.1: Esquema de compresión usando modelos estocásticos

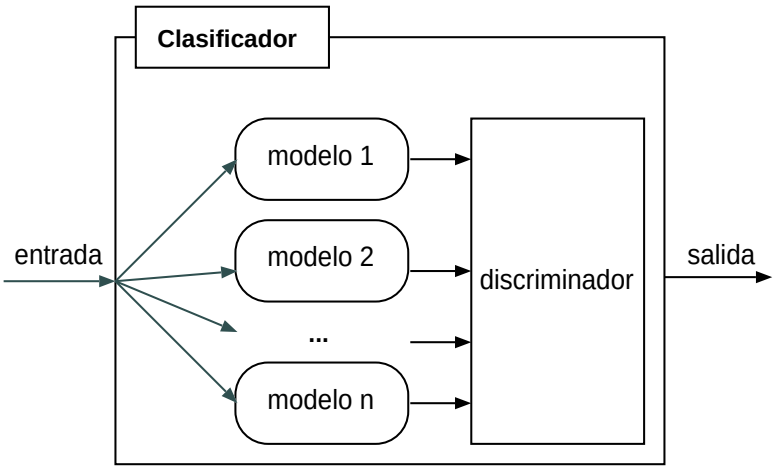


Figura 3.2: Esquema de un clasificador basado en modelos estocásticos.

Cuando se clasifica, existen tantos modelos como clases (figura 3.2). La probabilidad que suministra cada modelo para una misma entrada sirve para discriminar y clasificar la entrada como perteneciente a la clase más probable. Este método de selección se llama *criterio de máxima verosimilitud* (Duda and Hart 1973).

En este capítulo se recopilan los modelos estocásticos más utilizados para texto.

En el siguiente capítulo se desarrolla la extensión y la aplicación de estos modelos a estructuras de datos con forma de árbol.

3.1 Modelos de contexto finito (k-gramas)

Este tipo de modelo se basa en una idea muy simple: las probabilidades para cada símbolo de la entrada se calculan basándose en el contexto en el que aparece. El *orden* del modelo se refiere al número de símbolos que se examinan para calcular la probabilidad del actual.

Puede parecer que la probabilidad de un símbolo dado en una secuencia es fija, pero no es del todo cierto. Por ejemplo, al analizar el código fuente de

un programa de lenguaje C, la probabilidad de un carácter de nueva línea en el texto puede ser 1/40. Esta probabilidad podría determinarse explorando el texto entero y dividiendo el número de líneas entre el número total de caracteres. Pero si utilizamos un modelo que mire un carácter anterior, las probabilidades cambian. En ese caso, si el carácter anterior era `}`, la probabilidad de un carácter de nueva línea podría ser 1/2. Esta idea es la base de los modelos de contexto finito.

El modelo más simple de contexto finito es el modelo de orden 0. Esto significa que la probabilidad de cada símbolo es independiente de cualquier símbolo anterior. Para implementar este modelo sería necesario un solo vector con las frecuencias de cada símbolo. Para un modelo de orden 1, se necesitaría 256 (si estamos trabajando con bytes) vectores de frecuencias, puesto que se necesita guardar por separado las apariciones de cada símbolo para cada contexto posible. Asimismo, un modelo de orden 2 necesita 65.536 vectores o uso de matrices dispersas.

Parece lógico que aumentando el orden del modelo, las tasas de compresión y clasificación mejoren. Por ejemplo, la probabilidad del símbolo `u` que aparece en este capítulo puede ser del 5%, pero si el carácter anterior del contexto es `q`, la probabilidad podría ser del 95%. Cuando se predicen caracteres con alta probabilidad, disminuye el número de bits necesarios para la codificación y aumenta la probabilidad final para la clasificación.

Desafortunadamente, si el orden del modelo aumenta linealmente, el consumo de memoria lo hace exponencialmente. Otro problema asociado al aumento del orden k es que si las probabilidades se han estimado a partir de la experiencia, es necesario una muestra gigantesca para tener una estimación fiable, especialmente en los sucesos poco probables.

3.2 Modelos de descuento básicos

Como solución al problema anterior surgen los modelos de descuento. Estos modelos asignan una probabilidad distinta de cero a símbolos o sucesos que no han aparecido todavía en un contexto. Lógicamente, esto significa que se reduce, de alguna forma, la probabilidad asignada a los símbolos observados. La forma en que se estiman estas probabilidades da lugar a diferentes técnicas.

En primer lugar, abordaremos el *modelo de descuento de Katz* descrito en Katz (1987). Este modelo multiplica por un factor de descuento proporcional $\lambda(h, w)$ cada uno de los símbolos w que aparecen tras un contexto h siempre que su número de apariciones oscile entre 1 y un parámetro externo d, establecido a priori. De esta forma sólo se descuenta a un determinado número de símbolos que han aparecido pocas veces (entre 1 y d). El resto de símbolos tienen la misma probabilidad que en el modelo original. La probabilidad de los n_0 símbolos que no han aparecido todavía es la suma de los valores descontados proporcionalmente por $\lambda(h, w)$ dividido entre n_0.

En segundo lugar, describiremos el *modelo de descuento absoluto* (Ney and Essen 1993). Este modelo resta un valor constante b de descuento al número de apariciones de un símbolo w en un contexto h, por lo que quedan disminuidas todas las probabilidades del contexto. Los símbolos que todavía no han aparecido en el contexto h reciben del modelo original la probabilidad correspondiente al cociente entre la suma de todas las constantes b de los símbolos existentes y el número de dichos símbolos vistos el contexto w, todo ello dividido entre n_0 para que la probabilidad de los símbolos no vistos sea equiprobable.

Por último, expondremos el *modelo de descuento lineal* (Katz 1987; Jelinek 1990). Este modelo está basado en el modelo de Katz y descuenta un factor constante $\lambda(h, w) = \alpha$ a las probabilidades de todos los símbolos w de un contexto h, con lo que la probabilidad final de los símbolos no vistos es α/n_0.

No obstante estas tres técnicas descritas en este apartado afectan a un único modelo. En cambio, se puede obtener un aproximación mejor a la probabilidad de un evento o símbolo no visto usando varios modelos.

3.3 Modelos de descuento extendidos

En este apartado se usaran varios modelos a la vez, aprovechando características de cada uno de ellos para obtener la probabilidad de un símbolo. Por ejemplo, si tenemos un $k \leq k_{max} = 3$, es decir, cuatro modelos $M^{[k]}$ de contexto finito con un orden $k = 0, 1, 2, 3$, respectivamente, podemos usar simultáneamente la información contenida en los cuatro modelos.

La *interpolación de modelos* (Ney et al. 1997) se basa en un idea muy sencilla: se trata de calcular la probabilidad final como combinación lineal de las probabilidades obtenidas con cada uno de los modelos. A cada modelo $M^{[k]}$ se le asigna un peso α_k con la restricción de que todos los pesos α_k deben sumar 1.

El *modelo de suavizado multinivel* (Ney et al. 1997) trata de aplicar alguna técnica de descuento para cada modelo $M^{[k]}$ que tengamos y normalizar las probabilidades de descuento. De esta forma, se trata de calcular las probabilidades con el modelo de orden k mayor, pero si éste no tiene probabilidad asociada al símbolo actual, se usa $M^{[k-1]}$, y así sucesivamente hasta llegar al modelo básico $M^{[0]}$ que reconoce cualquier símbolo. Este modelo será descrito en la sección siguiente como modelo de *predicción por concordancia parcial*, ya que este es el nombre con el que se conoce en el ámbito de la compresión de texto.

3.4 Modelos de predicción por concordancia parcial

Abreviamos este tipo de modelos con las siglas PPM del nombre en inglés *Prediction by Partial Matching* (Cleary and Witten 1984). Como se ha comentado anteriormente, una mejora de los modelos de contexto finito consiste en aumentar el orden del modelo. Un modelo de orden 0 considera los símbolos independientes sin mirar su contexto. Si examinamos el contexto del símbolo actual mirando los caracteres anteriores de la entrada de datos, se predecirá mejor la probabilidad de los símbolos de la entrada.

Cuando tenemos modelos de orden 2, 3 o superiores, surge un problema que ilustra el siguiente ejemplo: se va a intentar predecir qué carácter viene después de `REQ` en una entrada de texto. Si la probabilidad de que el símbolo siguiente sea una `U` es del 90%, entonces, codificar este símbolo requiere menos de un dígito binario. Pero cuando estamos utilizando un modelo adaptativo, tenemos que leer gran cantidad de símbolos de la entrada para comenzar a predecir la `U` con una probabilidad alta.

El problema está en que un modelo de orden k asigna a cada carácter una probabilidad en $(0, 1]$. Inicialmente no se tiene ningún conocimiento sobre qué tipo de distribución tendrá el texto de entrada. Generalmente se empieza asignando a cada carácter la misma probabilidad. Esto significa que cada byte tiene una probabilidad de 1/256 de aparecer después de los símbolos de `REQ`. Ahora, de igual forma si la letra `U` aparece en este contexto 10 veces, su probabilidad aumentará hasta 11/266. El resto de los símbolos están ocupando un espacio valioso en el vector de probabilidades aún cuando nunca sean vistos.

La solución a este problema es fijar las probabilidades iniciales de todos los símbolos a 0 para un contexto dado, y tener un código especial ε (código de escape) que se produce cuando un símbolo no se ha visto previamente. Para el contexto anterior `REQ`, se fija la probabilidad del código de escape

inicialmente a 1 y se tiene la probabilidad del resto de símbolos fijados a 0. La primera vez que el carácter `U` sigue a `REQ`, se tendría que emitir un código de escape, seguido por el código de la `U` en un contexto distinto. Durante la actualización del modelo se podría incrementar el contador para `U` en el contexto de `REQ` a 1, así que ahora tendrá una probabilidad de 1/2. La próxima vez que aparezca, se codificará con 1 bit.

La pregunta obvia entonces es: ¿qué utilizamos como contexto después de emitir un código del escape? Una solución muy razonable y también adaptativa es: si el contexto es de orden 3 y se genera un código de escape, el contexto siguiente es de orden 2; si se vuelve a emitir un escape el próximo contexto sería el 1 y de continuar llegaríamos hasta el orden especial 0, con el que se puede codificar cualquier símbolo. Esto significa que la primera vez que se utiliza el contexto `REQ` y `U` necesita codificarse, entonces se emite un escape y se cae a un modelo de orden 2 que intenta codificar el carácter `U` con el contexto `EQ`. Si se continua sin poder codificar la letra `U` con el contexto `Q` llegaríamos hasta el modelo de orden 0 donde se emitiría el código de la `U` sin contexto. El contexto 0 inicializa la cuenta de todos los símbolos posibles a 1 y nunca se actualiza.

Cuando se comienza a codificar una secuencia se emitirán bastantes códigos de escape, ya los símbolos en el contexto donde aparecen lo harán por primera vez. Según avanza la compresión, su probabilidad irá, en general, disminuyendo.

En clasificación no es usual utilizar modelos adaptativos porque los modelos se entrenan con muestras en primer lugar y clasifican los nuevos patrones posteriormente.

Existen diversos métodos para estimar la probabilidad de estos códigos de escape. En lo que sigue, usaremos las siguientes variables:

- sea g el número de símbolos distintos que han aparecido detrás de un contexto dado h,
- sea n el número de apariciones del contexto h,

- sea c_i el número de apariciones del símbolo a_i tras el contexto h, tal que $\sum_{i=1}^{g} c_i = n$, y,
- sea t_j el número de símbolos i con frecuencia $c_i = j$ en un contexto h.

Los modelos de contexto finito de la sección anterior corresponde a una probabilidad para los escapes $p(\varepsilon) = 0$ y para los símbolos $p(i) = c_i/n$. A continuación exponemos brevemente distintas aproximaciones para el cálculo de $p(\varepsilon)$ en los modelos PPM:

- En el libro de Bell et al. (1990) se describe el método A (PPMA) que esencialmente consiste en dejar fijo el contador de códigos de escape, de modo que $p(\varepsilon) = 1/(n+1)$, mientras se actualiza el resto de reglas, $p(i) = c_i/(n+1)$. De esta forma, $p(\varepsilon)$ es cada vez menos probable.
- Una aproximación mejor que la anterior descrita por Moffat (1990), llamada método C (PPMC) asigna al código de escape el número de símbolos distintos que aparecen en la tabla de un contexto que se han visto hasta el momento. Esto es, $p(\varepsilon) = g/(n+g)$, y para el resto $p(i) = c_i/(n+g)$.
- Otra aproximación (Howard 1993) ligeramente mejor que el PPMC es el PPMD. Consiste en asignar al código de escape la mitad de las nuevas apariciones para un contexto dado, es decir, una probabilidad $p(\varepsilon) = g/(2n)$ y para el resto $p(i) = (2c_i - 1)/(2n)$.
- Una nueva aproximación propuesta por Witten and Bell (1991), llamada método X (PPMX), usa el número de *hapax legomena* (reglas de frecuencia uno) t_1. Se trata de equiparar la frecuencia de las reglas con una repetición a las que todavía no han aparecido, de forma que las probabilidades serían: $p(\varepsilon) = t_1/(n+t_1)$ y para el resto $p(i) = c_i/(n+t_1)$. Si fuera posible que $t_1 = 0$, entonces podríamos utilizar la aproximación: $p(\varepsilon) = (t_1+1)/(n+t_1+1)$ y $p(i) = c_i/(n+t_1+1)$. Refinamientos sobre esta estrategia se exploraron en Cleary et al. (1995).

Los resultados que se obtienen usando métodos PPM son equiparables al método de *compresión de ordenamiento por bloques* (apéndice A.1). También hay que añadir que la efectividad del método de *compresión basado en palabras* (apéndice A.2) se aproxima al PPM y, además es más rápido, porque codifica varios caracteres a la vez, pero el sistema de compresión genera gran cantidad de símbolos (palabras y separadores), por lo que se debe prestar especial atención a un almacenamiento eficiente de las estructuras de los datos para el modelo.

Por último, hay que destacar que los modelos PPM se han probado en diferentes contextos con muy buenos resultados de compresión (Witten et al. 1999).

3.5 Modelos de Markov dinámicos

Este método de compresión o clasificación está basado en un modelo de estados finitos (Cormack and Horspool 1987). Se le conoce con las siglas DMC (*Dynamic Markov Compression*). La eficiencia de este método de compresión es equiparable a la conseguida por los modelos PPM, pero con menor consumo de recursos.

El DMC trabaja con un alfabeto binario y es uno de los mejores métodos de compresión. El código necesario para implementar el DMC es mínimo y muy fácil de programar, pero tiende a ser lento si no se invierte un esfuerzo considerable en agilizar las operaciones que realiza.

Es un método adaptativo, tanto en las probabilidades del autómata finito como en la estructura del mismo. Las probabilidades de los estados se actualizan símbolo a símbolo (bit a bit) y la estructura cambia según métodos heurísticos de duplicación de estados.

Si tenemos un modelo como el de la figura 3.3 cada estado tiene dos transiciones: una para el valor 0 y otra para el valor 1. Además cada transición contiene el número de veces que se ha efectuado. Por ejemplo, si estamos en

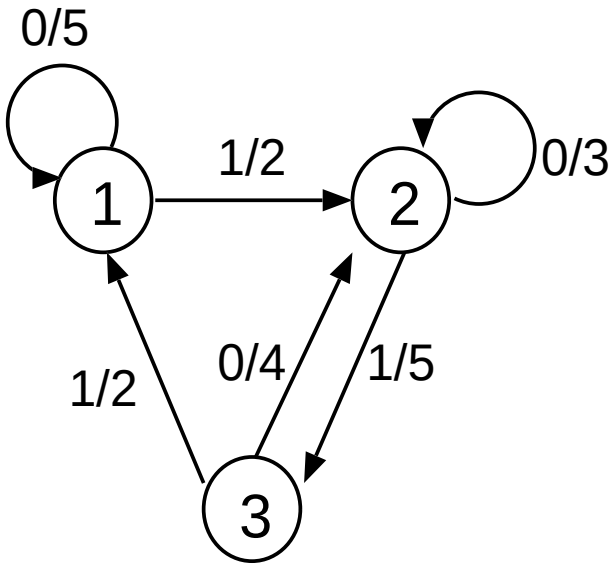


Figura 3.3: Modelo generado por un DMC.

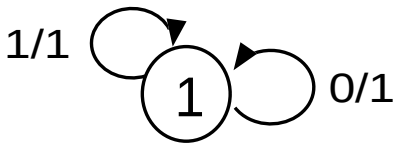


Figura 3.4: Modelo inicial de un DMC.

el estado 1, la etiqueta 0/5 significa que el valor 0 se ha repetido 5 veces, así podemos calcular la probabilidad dividiendo por la suma del número de transiciones, $5/(5+2) = 0.71$.

Los estados del modelo también se adaptan: se puede empezar con un modelo sencillo como el de la figura 3.4 y por criterios heurísticos duplicar estados cuando se considere necesario.

El modelo DMC es potencialmente mejor que el PPM. Sin embargo, el uso de modelos iniciales que se pueden estimar probabilísticamente y de heurísticas para duplicar estados, producen en la práctica modelos de contexto finito (Bell and Moffat 1989), por lo que DMC tiene un comportamiento en la práctica similar a PPM.

El DMC trabaja sobre bits, el autómata tiene sólo dos transiciones por estado, lo que lo hace muy fácil de implementar. Por otro lado hace que funcione más lento, ya que tiene que leer bit a bit. Si queremos que funcione con bytes las estructuras serán más complejas y el consumo de memoria aumentará también, con lo que las diferencias con respecto al PPM son mínimas.

Parte I

Lenguajes de árboles k-testables

Capítulo 4

Modelos de contexto finito para árboles

En este capítulo se presenta una generalización natural de los modelos de k-gramas para los lenguajes de árboles que proporciona rigor matemático a los modelos usados en esta tesis. En los capítulos siguientes usaremos sobre todo las definiciones que se hallan en las secciones 4.3 y 4.4. Esta generalización se basa en la clase k-testable, una subclase de los lenguajes reconocibles por los autómatas de árboles ascendentes. Sobre estos modelos se estiman las probabilidades de las reglas a partir de las frecuencias experimentales. Una de la ventajas que presenta esta aproximación es que el modelo se puede actualizar incrementalmente de forma fácil. Este método es una alternativa a los algoritmos de aprendizaje más costosos (como los métodos basados en el *inside-outside* como Sakakibara (1992)) o algoritmos que requieren una gran cantidad de ejemplos (como los métodos de fusión o separación de estados como Carrasco et al. (2001)). Este sistema de aprendizaje garantiza la identificación en el límite.

4.1 Introducción

Los modelos estocásticos basados en k-gramas se han usado ampliamente para el modelado del lenguaje natural (Brown et al. 1992; Ney et al. 1995), el reconocimiento del habla (Jelinek 1998) y la compresión de datos (Rubin 1976). De hecho, todo modelo estocástico puede predecir el próximo símbolo en una secuencia y, por lo tanto, puede usarse en los algoritmos de compresión aritmética de datos (Witten et al. 1987; Cover and Thomas 1991). En problemas de clasificación, la necesidad de usar modelos estocásticos surge cuando se aplica la regla de Bayes (Duda and Hart 1973) para el cálculo del error mínimo: dada una secuencia $S = s_1 s_2 \cdots$ de observaciones, el modelo estocástico M que maximiza la probabilidad condicionada $P(M \mid S)$ también maximiza $P(S \mid M) P(M)$. Por tanto, se necesita un modelo $P(S \mid M)$ para la generación de secuencias. Si el modelo estocástico estuviera basado en probabilidades condicionadas al contexto precedente, es decir,

$$P(S = s_1 s_2 \cdots s_t \mid M) = p_M(s_1)\, p_M(s_2 \mid s_1) \cdots p_M(s_t \mid s_1 s_2 \cdots s_{t-1}) \quad (4.1)$$

y, además, se asume que la probabilidad p_M depende sólo del contexto que le precede inmediatamente, en particular, los últimos $k-1$ elementos de la secuencia,

$$p_M(s_t \mid s_1 s_2 \cdots s_{t-1}) = p_M(s_t \mid s_{t-k+1} \cdots s_{t-1}) \quad (4.2)$$

el modelo de cadenas de Markov resultante (Chung 1967) recibe el nombre de modelo de k-gramas.

Desde un punto de vista teórico, los modelos de k-gramas pueden ser considerados como una extensión de los lenguajes localmente testables (García and Vidal 1990; Yokomori 1995) cuando se incorporan probabilidades al modelo. Informalmente, un lenguaje de cadenas L es localmente testable si dada una cadena ω se puede determinar si pertenece a L examinando únicamente

todas las subcadenas de ω cuya longitud es como máximo k.

Algunos trabajos previos como el de Knuutila (1993) y el de García (1993) han generalizado los algoritmos de identificación para lenguajes de cadenas localmente testables al caso de los lenguajes de árboles. Los árboles son una representación natural de los datos cuando se establecen relaciones jerárquicas entre sus componentes. En particular, las gramáticas de árboles estocásticas se han usado ampliamente para la desambiguación sintáctica (Charniak 1993; Stolcke 1995) en el procesamiento de lenguaje natural. En los modelos basados en bancos de datos como el *Penn Tree-bank* (Marcus et al. 1994) que tienen árboles de análisis sintáctico, la gramática se infiere a partir de una colección de frases analizadas manualmente y las probabilidades se estiman contando el número de expansiones de cada tipo.

Los modelos k-testables estocásticos nos servirán de base para construir modelos para compresión adaptativa de árboles y para la clasificación (capítulos 6). La especificación de la notación se introduce en la siguiente sección y la descripción del modelo se encuentra en la sección 4.3 y 4.4.

4.2 Árboles y autómatas de árboles

Dado un *alfabeto*, es decir, un conjunto finito de símbolos $\Sigma = \left\{\sigma_1, \cdots, \sigma_{|\Sigma|}\right\}$, el conjunto Σ^T, de $\Sigma-$árboles, se define mediante la gramática independiente del contexto $G = (\Sigma', \{T, F\}, T, R)$ donde el alfabeto Σ' incluye a Σ y los paréntesis izquierdo y derecho, y el conjunto de reglas R contiene:

- $T \to \sigma\,(F) \quad \forall\, \sigma \in \Sigma$
- $F \to \varepsilon \mid TF$

donde ε representa la cadena vacía. Para abreviar, a partir de ahora se escribirá σ cuando aparezca $\sigma\,(\varepsilon)$ o $\sigma\,()$. La *profundidad* de un árbol del tipo $t = \sigma$ es $\text{depth}\,(\sigma) = 0$, mientras que la profundidad del árbol $t = \sigma\,(t_1 \cdots t_m)$

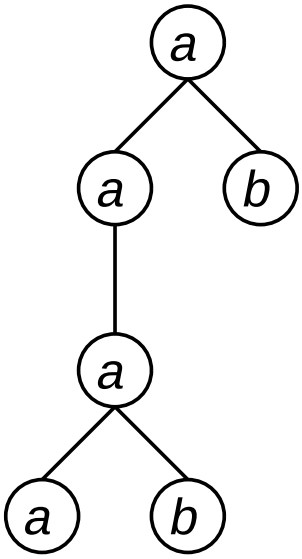


Figura 4.1: Representación gráfica del árbol $t = a(a(a(ab))b)$

es

$$\operatorname{depth}\left(\sigma\left(t_1 \cdots t_m\right)\right) = 1 + \max_{j=1}^{m}\left\{\operatorname{depth}\left(t_j\right)\right\} \tag{4.3}$$

Por ejemplo, la profundidad del árbol $a(a(a(ab))b)$ perteneciente a $\{a, b\}^T$ es 3 y su representación gráfica está dibujada en la figura 4.1.

4.2.1 Autómatas de árboles ascendentes deterministas

El concepto de autómata de árboles ascendente determinista (AAAD) generaliza el concepto de autómata finito determinista (AFD (Hopcrof and Ullman 1979)) sobre cadenas. Un AFD procesa cada cadena de entrada de izquierda a derecha y asigna un estado a cada posición de la cadena. El estado depende del símbolo en la posición actual y del estado asociado a la posición anterior. Una vez analizada la última posición se examina el estado actual: si pertenece al subconjunto de los estados de aceptación la cadena es aceptada y en

caso contrario se rechaza. En el caso de los AAAD, los árboles se procesan de abajo hacia arriba y los estados del autómata se asignan a cada nodo en el árbol. Estos estados dependen de la etiqueta del nodo y de los estados asignados previamente a sus descendientes. El estado asignado a la raíz del árbol tiene que ser un estado de aceptación para que el AAAD acepte el árbol analizado. Matemáticamente, el AAAD es una 4-tupla $A = (Q, V, \Delta, F)$ donde

- Q es un conjunto finito de *estados* $Q = \left\{q_0, q_1, \ldots, q_{|Q|}\right\}$;
- Σ es un conjunto finito de *etiquetas* $\Sigma = \left\{\sigma_0, \sigma_1, \ldots, \sigma_{|\Sigma|}\right\}$;
- $F \subset Q$ es un subconjunto de *estados de aceptación;*
- $\Delta = \{\delta_0, \delta_1, \ldots, \delta_M\}$ es el conjunto de *funciones de transición* de la forma $\delta_m : \Sigma \times Q^m \to Q$.

Comparado con el AFD de cadenas, en los AAAD no es necesario definir el estado inicial. Para cada hoja $\sigma \in \Sigma$, $\delta_0(\sigma)$ define un estado de partida. Por otra parte, si $t = \sigma(t_1, t_2, \ldots, t_m)$ es un subárbol con la etiqueta $\sigma \in \Sigma$ que expande m subárboles $t_1, t_2, \ldots, t_m$, el estado $\delta(t) \in Q$ es $\delta_m(\sigma, \delta(t_1), \delta(t_2), \ldots, \delta(t_m))$. Por tanto, $\delta(t)$ se define recursivamente para $t = \sigma(t_1, t_2, \ldots, t_m)$ como:

$$\delta(t) = \begin{cases} \delta_0(\sigma) & \text{si } m = 0 \\ \delta_m(\sigma, \delta(t_1), \delta(t_2), \ldots, \delta(t_m)) & \text{si } m > 0 \end{cases} \tag{4.4}$$

El lenguaje reconocido por el autómata A es el subconjunto de Σ^T

$$L(A) = \left\{t \in \Sigma^T \ : \ \delta(t) \in F\right\} \tag{4.5}$$

Por ejemplo, si $\Sigma = \{a, b\}$ y Δ contiene las transiciones $\delta_0(a) = q_1$, $\delta_0(b) = q_2$, $\delta_2(a, q_1, q_2) = q_2$, y $\delta_1(a, q_2) = q_1$, el resultado de procesar con A el árbol

$t = a(a(a(ab))b)$ (figura 4.1) es $\delta(t) = \delta_2(a, \delta(a(a(ab))), \delta(b))$. Recursivamente, $\delta(a(a(ab))) = q_1$ y $\delta(b) = q_2$, entonces $\delta(t) = \delta(a, q_1, q_2) = q_2$. Por convenio, supondremos que las transiciones no definidas conducen a un estado de *absorción*, esto es, a un árbol no válido. Este es el caso, por ejemplo, si el número de hijos es superior al máximo M aceptado por A.

4.2.2 Autómatas de árboles ascendentes deterministas estocásticos.

Un autómata de árboles ascendentes determinista estocástico (AAADE) incorpora una probabilidad para cada transición en el autómata y para cada posible estado raíz del árbol. La definición de una AAADE es una 5-tupla $A = (Q, \Sigma, \Delta, P, r)$ donde

- Q es un conjunto finito de *estados* $Q = \{q_0, q_1, \ldots, q_{|Q|}\}$;
- Σ es un conjunto finito de *etiquetas* $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{|\Sigma|}\}$;
- $\Delta = \{\delta_0, \delta_1, \ldots, \delta_M\}$ es un conjunto de *funciones de transición* de la forma $\delta_m : \Sigma \times Q^m \to Q$;
- $P = \{p_0, p_1, \ldots, p_M\}$ es un conjunto de distribuciones de probabilidad, llamadas *probabilidades de transición* $p_m : \Sigma \times Q^m \to [0, 1]$;
- $r : Q \to [0, 1]$ da, para cada $q \in Q$, la probabilidad $r(q)$ de que un árbol t satisfaga $\delta(t) = q$.

Debemos tener en cuenta la normalización probabilística por estados, es decir, la suma de las probabilidades de expansión desde un estado debe sumar uno,

$$\sum_{\sigma \in \Sigma} \sum_{m=0}^{M} \sum_{\substack{q_1,\ldots,q_k \in Q: \\ \delta_k(\sigma, q_1, \ldots, q_k) = q}} p_m(\sigma, q_1, ..., q_m) = 1 \tag{4.6}$$

Además,

$$\sum_{q \in Q} r(q) = 1 \tag{4.7}$$

La probabilidad que a un árbol $t = \sigma(t_1, t_2, \ldots, t_m)$ le asigna el autómata A viene dada por el producto de la probabilidades de todas las transiciones usadas cuando t es procesado por A multiplicado por la probabilidad de comenzar por el estado asociado a la raíz:

$$p(t \mid A) = r(\delta(t))\, \pi(t). \tag{4.8}$$

Donde $\pi(t)$ se define recursivamente como

$$\pi(t) = \begin{cases} p_o(\sigma) & \text{si } m = 0 \\ p_m(\sigma, \delta(t_1), \delta(t_2), \ldots, \delta(t_m))\, \pi(t_1)\, \pi(t_2) \cdots \pi(t_m) & \text{si } m > 0 \end{cases} \tag{4.9}$$

Las ecuaciones (4.8) y (4.9) definen una distribución de probabilidad $p(t \mid A)$ que es *consistente* si y sólo si

$$\sum_{t \in \Sigma^T} p(t \mid A) = 1 \tag{4.10}$$

No todos los AAADE son consistentes. Por ejemplo, tenemos $Q = \{q_0\}$, $\Sigma = \{a\}$ con las transiciones $\delta_0(a) = q_0$ y $\delta_2(a, q_0, q_0) = q_0$ cuyas probabilidades son $p_0(a) = 1/100$ y $p_2(a, q_0, q_0) = 99/100$ y la función $r(q_0) = 1$. Si se observa la definición de A se ve como la probabilidad de usar la regla $\delta_2(a, q_0, q_0) = q_0$ es 0.99 frente al 0.01 de $\delta_0(a) = q_0$. La primera regla genera dos nuevos estados de q_0 que tienden a seguir expandiéndose, mientras que

la segunda regla genera una hoja y detiene la producción. El autómata A no es consistente, ya que el valor esperado del tamaño de un árbol es infinito y no cumpliría la condición de la ecuación (4.10).

Como se demuestra en Chaudhuri et al. (1983) y en Sánchez and Benedí (1997), las gramáticas independientes del contexto cuyas probabilidades se estiman a partir de un conjunto cualquiera de ejemplos son siempre consistentes. Es fácil demostrar (Sakakibara 1992) que el lenguaje reconocido por un AAAD puede ser generado también con un gramática regular de árboles. Dado que en nuestro caso, las probabilidades de un AAAD se calculan siempre a partir de muestras aleatorias, la consistencia siempre se preserva.

4.3 Autómatas k-testables

En esta sección definiremos los lenguajes y métodos de inferencia k-testables de árboles siguiendo las ideas de Knuutila (1993) y García and Vidal (1990). Para ello definimos los conjuntos de k-forks, k-subtrees y k-root de un árbol. Estos conjuntos se generan a partir de fragmentos del árbol obtenidos cuando se observa el árbol con una ventana de tamaño k, es decir, con una profundidad máxima k. Al igual que en el caso de los modelos de k-gramas para cadenas, donde hay que especificar qué ocurre al principio y final de la cadena (donde no hay símbolos precedentes o siguientes) aquí debe tratarse de forma explícita la raíz (k-root) y las hojas (k-subtrees). Para todo $k > 0$ y para todos los árboles $t = \sigma(t_1, t_2, \ldots, t_m) \in \Sigma^T$, el k-root de t es un árbol de Σ^T definido como

$$r_k(t) = \begin{cases} \sigma(r_{k-1}(t_1), r_{k-1}(t_2), \ldots, r_{k-1}(t_m)) & \text{si } k > 1 \\ \sigma & \text{si } k = 1 \end{cases} \tag{4.11}$$

Cabe destacar que cuando $m = 0$, es decir, $t = \sigma \in \Sigma$, entonces $r_k(\sigma) = \sigma$.

Por otro lado, el conjunto $f_k(t)$ de los $k-$forks y el conjunto $s_k(t)$ de los

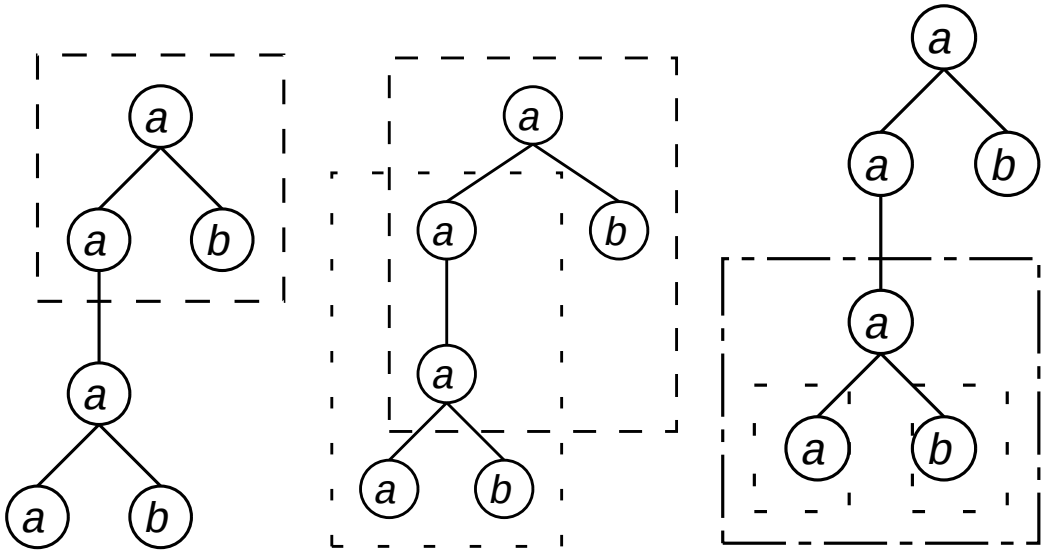


Figura 4.2: Representación gráfica de las funciones sobre el árbol $t = a(a(a(ab))b)$: (a) $r_2(t)$; (b) $f_3(t)$; (c) $s_2(t)$

k-subtrees se definen respectivamente como

$$f_k(t) = \bigcup_{j=1}^{m} f_k(t_j) \bigcup \begin{cases} r_k(t) & \text{si } \operatorname{depth}(t) \geq k-1 \\ \emptyset & \text{en caso contrario} \end{cases} \tag{4.12}$$

$$s_k(t) = \bigcup_{j=1}^{m} s_k(t_j) \bigcup \begin{cases} t & \text{si } \operatorname{depth}(t) \leq k-1 \\ \emptyset & \text{en caso contrario} \end{cases} \tag{4.13}$$

En el caso particular en el que $t = \sigma \in \Sigma$, entonces $s_k(t) = f_1(t) = \sigma$ y $f_k(t) = \emptyset$ para todo $k > 1$. Por ejemplo, si consideramos el árbol de la figura 4.1, $t = a(a(a(ab))b)$, tenemos $r_2(t) = \{a(ab)\}$, $f_3(t) = \{a(a(a)b), a(a(ab))\}$ y $s_2(t) = \{a(ab), a, b\}$ que se pueden ver gráficamente en la figura 4.2. Estas definiciones coinciden con las del artículo Knuutila (1993) excepto en el significado del parámetro k.

Un lenguaje de árboles T es un lenguaje estrictamente k-testable (con $k \geq 2$) si existen tres conjuntos finitos $\mathcal{R}, \mathcal{F}, \mathcal{S} \subseteq \Sigma^T$ tales que

$$t \in T \Leftrightarrow r_{k-1}(t) \subseteq \mathcal{R} \wedge f_k(t) \subseteq \mathcal{F} \wedge s_{k-1}(t) \subseteq \mathcal{S} \tag{4.14}$$

La ecuación (4.14) nos dice que para identificar un lenguaje k-testable de árboles T necesitamos poder identificar para cualquier árbol tres partes: la raíz, los forks de profundidad k y los subtrees con profundidad inferior o igual a $k-1$. En el ejemplo de la figura 4.2, para que t perteneciera al lenguaje T, $\mathcal{R}$ deberá contener a $a(ab)$, $\mathcal{F}$ deberá contener a $a(a(a)b)$ y $a(a(ab))$ y $\mathcal{S}$ a $a(ab)$, a y b.

En este caso es sencillo (Knuutila 1993; García 1993) construir un AAAD $A = (Q, \Sigma, \Delta, F)$ que reconoce T. Para ello es suficiente con definir:

$$\begin{aligned}
&Q = \mathcal{R} \bigcup r_{k-1}(\mathcal{F}) \bigcup \mathcal{S};\\
&F = \mathcal{R};\\
&\delta_m(\sigma, t_1, \cdots, t_m) = \sigma(t_1, \cdots, t_m) \quad \forall \sigma(t_1, \cdots, t_m) \in \mathcal{S};\\
&\delta_m(\sigma, t_1, \cdots, t_m) = r_{k-1}(\sigma(t_1, \cdots, t_m)) \quad \forall \sigma(t_1, \cdots, t_m) \in \mathcal{F};
\end{aligned} \tag{4.15}$$

Si se asume que un lenguaje de árboles L es k-testable, entonces el AAAD que reconoce L se puede aprender en el límite a partir de muestras positivas (Knuutila 1993; García 1993), es decir, a partir de ejemplos de árboles del lenguaje. Dada una muestra de ejemplos positivos S, el procedimiento para obtener el AAAD construye el autómata A usando $r_{k-1}(S)$, $f_k(S)$ y $s_{k-1}(S)$ en vez de $\mathcal{R}$, $\mathcal{F}$ y $\mathcal{S}$ respectivamente en las definiciones, ecuación (4.15), para Q, F y Δ.

Por ejemplo, dado el árbol $t = a(a(a(ab))b)$ (figura 4.1), $r_2(t)$, $f_3(t)$, $s_2(t)$ calculados anteriormente, como $r_2(f_3(t)) = \{a(ab), a(a)\}$, el AAAD

resultante sería:

$$\begin{aligned} &Q = \{a(ab), a(a), a, b\}; \\ &F = \{a(ab)\}; \\ &\delta_0(a) = a; \\ &\delta_0(b) = b; \\ &\delta_2(a, a, b) = a(ab); \\ &\delta_2(a, a(a), b) = a(ab); \\ &\delta_1(a, a(ab)) = a(a). \end{aligned} \tag{4.16}$$

Evidentemente, el AAAD así construido reconoce el árbol t, es decir, $t \in L(A)$.

A lo largo de este trabajo usaremos a menudo el nombre de regla para referirnos a los árboles que definen transiciones en el autómata. Esto es, cada árbol t de $f_k(S) \bigcup s_{k-1}(S)$ será llamado una regla por contraposición a los estados $r_{k-1}(t) \in Q$ que las generan. En el ejemplo anterior, los estados son $\{a(ab), a(a), a, b\}$ y las reglas $\{a, b, a(ab), a(a(a)b), a(a(ab))\}$ asociadas respectivamente a los estados $\{a, b, a(ab), a(ab), a(a)\}$.

4.4 Extensión estocástica de los lenguajes de árboles localmente testables

Una muestra estocástica $S = \left\{\tau_1, \tau_2, \ldots, \tau_{|S|}\right\}$ es una secuencia de árboles generados de acuerdo con una cierta distribución de probabilidad. Si nuestro modelo es un AAADE (Autómata de Árboles Ascendente Determinista Estocástico), la distribución es $p(t \mid A)$ dada por las ecuaciones (4.8) y (4.9). Recordemos que asumimos que el esquema básico de transición (es decir, los estados de Q y las diferentes funciones de transición de Δ) corresponde a un autómata k-testable, y esto nos permite inferir un AAADE mediante una muestra de una manera simple.

Para este propósito, debemos recordar que la verosimilitud de una muestra estocástica S viene dada por

$$\prod_{i=1}^{n} p\left(\tau_i \mid A\right) \tag{4.17}$$

que se maximiza (Ney et al. 1995) si el autómata A asigna para cada árbol τ en el ejemplo una probabilidad igual a la frecuencia relativa de τ en S. Dicho de otra forma, a cada transición en Δ se le asigna una probabilidad que coincide con el número relativo de veces que la regla se ha utilizado cuando se han analizado los árboles de la muestra. Por tanto, dada una muestra estocástica $S = \left\{\tau_1, \tau_2, \ldots, \tau_{|S|}\right\}$, el conjunto de estados de A es

$$Q = r_{k-1}\left(S\right) \bigcup r_{k-1}\left(f_k\left(S\right)\right) \bigcup s_{k-1}\left(S\right); \tag{4.18}$$

el subconjunto de estados de aceptación es

$$F = r_{k-1}\left(S\right); \tag{4.19}$$

las probabilidades $r\left(t\right)$ se estiman a partir de la muestra S mediante

$$r\left(t\right) = \frac{1}{|S|} C_r^{[k-1]}\left(t, S\right), \tag{4.20}$$

donde

$$C_r^{[k-1]}\left(t, S\right) = \sum_{i=1}^{|S|} C_r^{[k-1]}\left(t, \tau_i\right)$$

y

$$C_r^{[k-1]}\left(t, \tau\right) = \begin{cases} 1 & \text{si } r_{k-1}\left(\tau\right) = t \\ 0 & \text{en caso contrario} \end{cases}$$

y, finalmente, las funciones de transición de probabilidad de P se estiman mediante

$$p_m(\sigma, t_1, \ldots, t_m) = \frac{C^{[k]}(\sigma(t_1, \ldots, t_m), S)}{\sum_{i=1}^{|S|} C^{[k-1]}(r_{k-1}(\sigma(t_1, \ldots, t_m)), S)} \tag{4.21}$$

donde

$$C^{[k-1]}(t, S) = \sum_{i=1}^{|S|} C^{[k-1]}(t, \tau_i)$$

y $C^{[k-1]}(t, \tau)$ cuenta el número de k-forks o $(k-1)$-subtrees isomorfos a t en τ. Debe destacarse que τ puede contener varios k-forks isomorfos a t o varios $(k-1)$-subtrees isomorfos a t pero no ambos tipos a la vez.

Es útil guardar las probabilidades anteriormente descritas r y p como el cociente de dos términos, como se muestra en las ecuaciones (4.20) y (4.21). De esta forma, si se suministra un nuevo árbol (o subárbol) τ, se puede actualizar fácilmente el autómata A contabilizando la información adicional. Para esta actualización, es suficiente con incrementar cada término de las ecuaciones con la suma obtenida para el nuevo árbol τ.

4.5 Aproximación de un AAADE por un autómata k-testable.

Dada su definición, es obvio que todos los lenguajes k-testables estocásticos se pueden generar a partir de un AAADE. A pesar de ello, como en el caso de los lenguajes de cadenas (Stolcke and Segal 1994), la afirmación inversa no siempre es cierta. El modelo k-testable más aproximado a un autómata dado se puede obtener de la siguiente manera, basada en los resultados de (Calera-Rubio and Carrasco 1998). Suponemos que tenemos un AAADE general, no necesariamente k-testable, $A = (Q, \Sigma, \Delta, P, r)$. Para cualquier

valor de k, obtenemos un AAADE k-testable $A' = (Q', \Sigma, \Delta', P', r')$ cuyas probabilidades vienen dadas por

$$r'(j) = \sum_{i \in Q} r(i)\, \eta_{ij} \tag{4.22}$$

para todo $j \in Q'$ y

$$p'_m(\sigma, j_1, \ldots, j_m) = \frac{\sum_{i_1,\ldots,i_m \in Q} C_{\delta(\sigma,i_1,\ldots,i_m)} p_m(\sigma, i_1, \ldots, i_m)\, \eta_{i_1 j_1} \cdots \eta_{i_m j_m}}{\sum_{i \in Q} C_i} \tag{4.23}$$

donde C_i es el número esperado de nodos de tipo i en un árbol generado por A y η_{ij} representa la probabilidad de que un nodo i expanda un subárbol t tal que $r_{k-1}(t) = j$. Todos estos coeficientes pueden ser calculados (Calera-Rubio and Carrasco 1998) usando procedimientos iterativos. En particular, C_i se calcula como

$$C_i^{[n+1]} = r(i) + \sum_{j \in Q} \Lambda_{ij} C_j^{[n]} \tag{4.24}$$

con $C_i^{[0]} = 0$ y

$$\Lambda_{ij} = \sum_{m=1}^{M} \sum_{\sigma \in \Sigma} \sum_{\substack{j_1,\ldots,j_m \in Q: \\ \delta_m(\sigma,j_1,\ldots,j_m)=j}} p_m(\sigma, j_1, \ldots, j_m)\, (\delta_{ij_1} + \ldots + \delta_{ij_m}) \tag{4.25}$$

Los coeficientes $\eta_{ij}^{[n]}$ se calculan como

$$\eta_{ij}^{[n+1]} = \sum_{m=0}^{M} \sum_{\sigma \in \Sigma} \sum_{\substack{i_1,\ldots,i_m \in Q: \\ \delta_m(\sigma,i_1,\ldots,i_m)=i}} \sum_{\substack{j_1,\ldots,j_m \in Q': \\ \delta'_m(\sigma,j_1,\ldots,j_m)=j}} p_m\left(\sigma, i_1, \ldots, i_m\right) \eta_{i_1 j_1}^{[n]} \cdots \eta_{i_m j_m}^{[n]} \tag{4.26}$$

comenzando con $\eta_{ij}^{[0]} = 0$. Obsérvese que los términos con $m = 0$ no son necesariamente nulos. Obviamente, se puede calcular también la entropía relativa entre el modelo exacto A y el aproximado A' según el método descrito en (Calera-Rubio and Carrasco 1998).

4.6 Conclusiones

La extensión probabilística de los lenguajes de árboles k-testables puede considerarse como una generalización de los k-gramas para lenguajes de árboles. Estos modelos pueden actualizarse incrementalmente y permiten un grado de generalización menor que el de las gramáticas de árboles obtenidas directamente de la muestra (cuyo caso corresponde a $k = 2$). Por otro lado, este modelo permite trabajar con muestras de tamaño medio, donde los métodos de fusión de estados como el de (Carrasco et al. 2001) extraen modelos demasiado sencillos con un número insuficiente de estados.

Capítulo 5

Compresión mediante modelos k-testables adaptativos

Este capítulo presenta diferentes aproximaciones a la compresión de árboles etiquetados. En la sección 5.1 se muestra un modelo sencillo inicializado con una distribución a priori y basado en la compresión de árboles binarios. Posteriormente en las secciones 5.2 y 5.3 se describen dos modelos más elaborados, el de contexto finito y el PPM respectivamente, con los que se obtuvieron mejores resultados.

Los resultados muestran cómo un modelo estocástico adecuado para este tipo de datos incrementa la eficiencia de la compresión respecto a los métodos de propósito general. Se debe destacar que la compresión de árboles con modelos estáticos ya se ha sido objeto de estudio en Calera-Rubio et al. (1999). El problema que presenta este tipo de compresión es que hay que guardar el modelo estocástico junto a la información. Ello supone un incremento del tamaño del fichero resultante.

Como se ha explicado anteriormente, para comprimir información con métodos *símbolo a símbolo* necesitamos un modelo que sea capaz de predecir el próximo símbolo y un sistema que traduzca los símbolos en bits de acuerdo con el modelo. Para resolver la segunda parte se utiliza *codificación aritmé-*

tica siguiendo la implementación de Nelson (1991), por lo que únicamente queda definir modelos probabilísticos que predigan los símbolos de la forma más eficiente posible para luego comprimirlos.

En todos los modelos que se describen en este capítulo aparecen funciones llamadas `send` que envían un rango de probabilidad al codificador aritmético. Recuérdese que el codificador aritmético envía a la salida un número que se va construyendo encajando los rangos de probabilidad que se asignan a cada suceso. Cada suceso tiene una probabilidad asignada por el modelo estocástico, pero si ordenamos las sucesos posibles, podemos construir una función de distribución (histograma acumulado). Los valores asignados al suceso (high) y al anterior (low) son los que necesita el codificador (figura 5.1). Debe evitarse a toda costa que el modelo estocástico asigne una probabilidad cero a un suceso si este puede aparecer en la práctica. En este caso, $high = low$ y el rango $[low, high)$ estaría vacío y la salida del compresor aritmético indefinida.

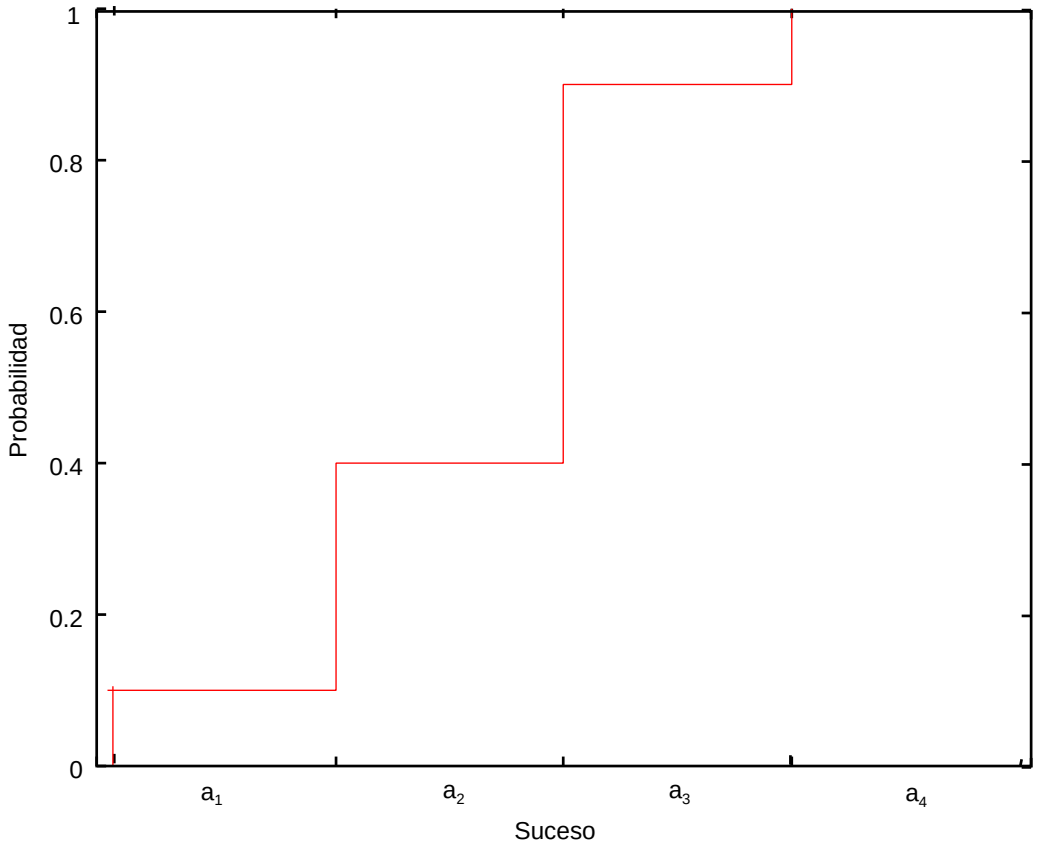
5.1 Modelo adaptativo con distribución a priori para árboles binarios

Una diferencia que existe entre las *cadenas* y los *árboles* es que cuando codificamos árboles, el modelo debe ser capaz de predecir además de un nodo etiquetado (igual que en cadenas), un número arbitrario de descendientes. Este problema se puede solucionar si restringimos el número de descendientes a 2 (árboles binarios).

En general, los árboles tienen nodos con un número variable de hijos, por lo que necesitamos definir una transformación que traduzca un árbol genérico a uno binario. Además, dicha transformación debe ser reversible, para que el descompresor pueda recuperar el árbol original. Solucionado el problema de la ariedad de los descendientes, se definirá un modelo estocástico para predecir la siguiente expansión.

suceso	p	low	high
a_1	0.1	0.0	0.1
a_2	0.3	0.1	0.4
a_3	0.5	0.4	0.9
a_4	0.1	0.9	1.0

(a)



(b)

Figura 5.1: (a) Función de probabilidad p y (b) función de distribución F para un suceso $\Omega = \{a_1, a_2, a_3, a_4\}$

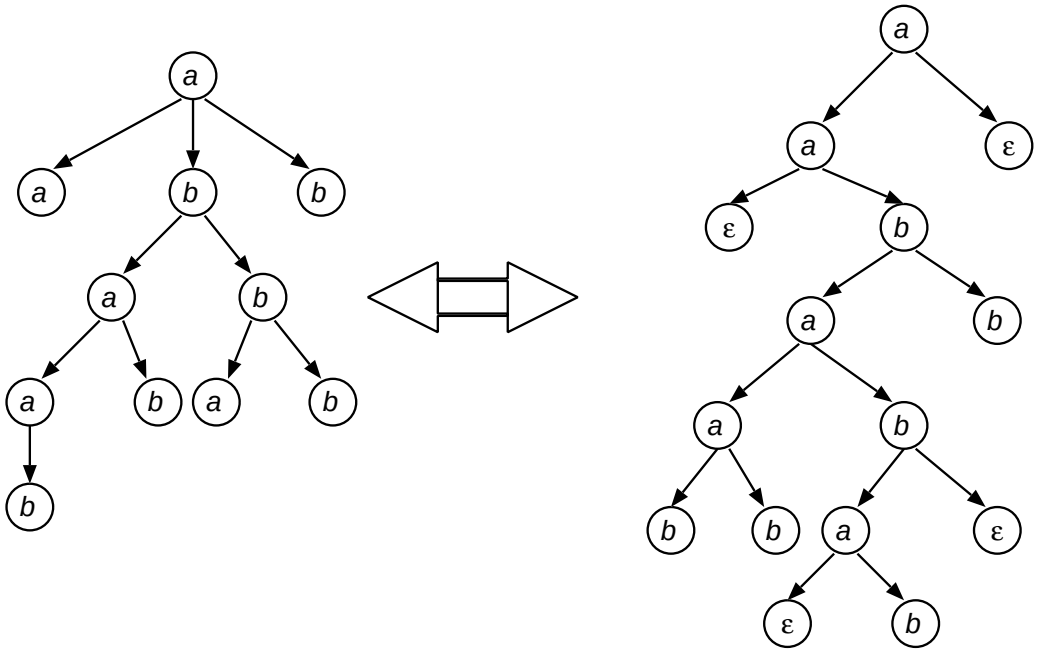


Figura 5.2: Conversión de un árbol n-ario en binario. Se ha utilizado la etiqueta ficticia ε para representar a los hijos vacíos en el árbol binario y distinguir cuándo un hijo es derecho o izquierdo.

Si se quiere transformar un árbol n-ario en otro binario sin pérdida de información, se pueden seguir los pasos descritos en el libro de Langsam et al. (1996) que propone dar sentido a los hijos del árbol binario: el hijo izquierdo de cada nodo será el primero del siguiente nivel en el árbol n-ario y, el hijo derecho será el siguiente hermano, de izquierda a derecha, en el árbol n-ario. En el caso en el que se tiene un solo hijo hay que distinguir si es el izquierdo o el derecho. Para ello se añadiría un hijo ficticio para que el proceso sea reversible. Si queremos convertir un árbol binario en n-ario se procede de manera inversa. Para clarificar el método presentamos un ejemplo en la figura 5.2.

El proceso de compresión que se sigue en este apartado se muestra en la figura 5.3. Se requiere un preproceso para convertir el fichero de árboles

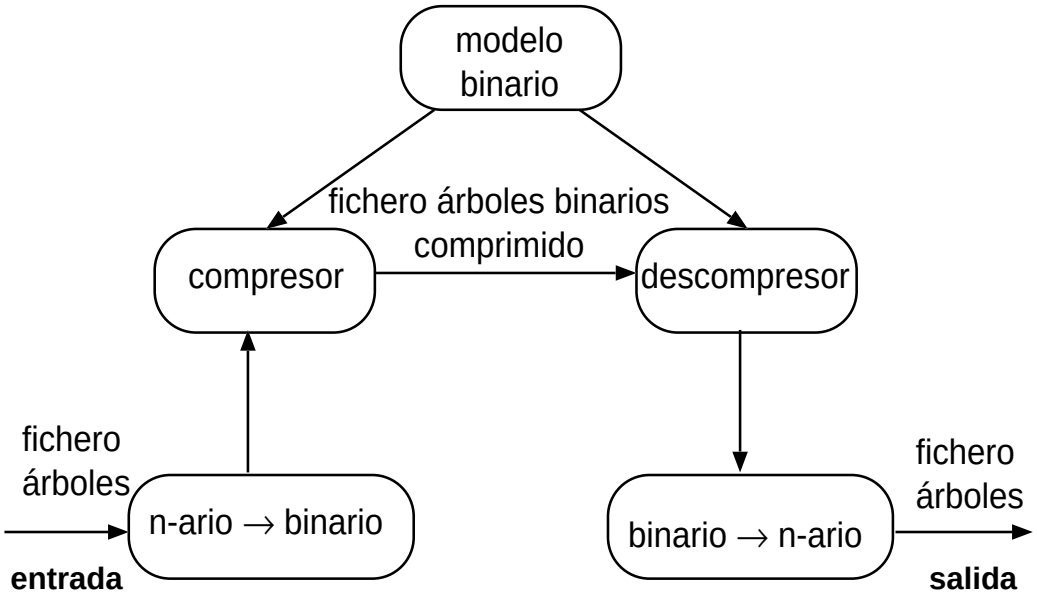


Figura 5.3: Esquema de compresión utilizando árboles binarios.

en árboles binarios y para descomprimir un postproceso que convierta los árboles binarios en árboles n-arios.

Podemos construir un modelo con todos los símbolos posibles, ya que el alfabeto es finito y la ariedad también. Por ejemplo, si tenemos un alfabeto $\Sigma = \{a, b\}$ se tendría un modelo inicial a priori como el de la tabla 5.1 donde:

- C_q es el número de apariciones de la etiqueta q y;

q	C_q	n	C_{qn}
a	21	0 2	13 8
b	21	0 2	13 8
ε	8	0	8

Tabla 5.1: Modelo de etiquetas y de ariedad para la compresión de árboles binarios con $\Sigma = \{a, b\}$ e inicializado con una distribución a priori.

- C_{qn} es el número de apariciones del número de descendientes n desde un nodo etiquetado con q.

Algoritmo 5.1 Compresión de árboles binarios con un modelo inicializado con una distribución a priori.

función compresionBinaria(τ)
  para todo ($t = \sigma(t_1, \ldots, t_m)$ subárbol de τ) **haz** [recorrido preorden]
    send(σ, m)
    update_model
  fin para
fin función

Algoritmo 5.2 Descompresión de árboles binarios con un modelo inicializado con una distribución a priori.

función descompresionBinaria(τ)
  get(σ, m)
  $\tau = (\sigma, m)$
  $n = m$
  mientras $n > 0$ **haz** [recorrido preorden]
    get(σ, m)
    append(τ, σ, m)
    $n = n + m - 1$
    update_model
  fin mientras
fin función

Con este modelo, un símbolo q del alfabeto tiene una probabilidad

$$p(q) = \frac{C_q}{\sum_{\sigma \in \Sigma} C_\sigma} \tag{5.1}$$

y el número de descendientes condicionado al símbolo q es

$$p(m \mid q) = \frac{C_{qm}}{C_q}. \tag{5.2}$$

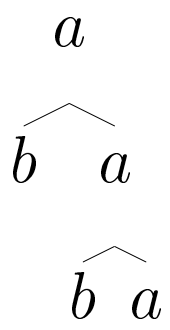


Figura 5.4: Árbol $a(ba(ba))$ usado como ejemplo.

Para poder predecir cualquier símbolo y cualquier descendiente (binario) se inicializaría el modelo con cualquier distribución de forma que el modelo:

1. sea completo, es decir, tenga todas las transiciones que pueda necesitar;

2. sea probabilísticamente consistente.

Una forma de conseguir ambos objetivos simultáneamente es inferir el modelo a partir de una muestra que contenga todos los árboles binarios de profundidad menor que el orden del modelo. En este caso, si $k = 2$, tenemos S_0=$\{a$, b, $a(\varepsilon a)$, $a(a\varepsilon)$, $a(b\varepsilon)$, $a(\varepsilon b)$, $b(\varepsilon a)$, $b(a\varepsilon)$, $b(b\varepsilon)$, $b(\varepsilon b)$, $a(aa)$, $a(ab)$, $a(ba)$, $a(bb)$, $b(aa)$, $b(ab)$, $b(ba)$, $b(bb)\}$.

Los árboles se codifican siguiendo un recorrido en preorden. Además, el modelo se actualiza cada vez que se codifica un nodo para, que de esta forma, el modelo vaya adaptándose progresivamente al contenido real del fichero que se está comprimiendo. Una versión en pseudocódigo para codificación de un árbol se muestra en el algoritmo 5.1, donde la función `send`(σ, m) envía los rangos de probabilidades correspondientes a $p(\sigma)$ y a $p(m \mid \sigma)$ al codificador aritmético. Recordemos que cuando se utiliza la codificación aritmética lo que se codifica es un número en coma flotante y que símbolo a símbolo se le suministra el rango de probabilidades de éste.

Por ejemplo, si tenemos el árbol binario de la figura 5.4, la secuencia de probabilidades de compresión emitido por la función `send`(σ, n) que se genera es

$$\texttt{send}\,(a,2)\ \texttt{send}\,(b,0)\ \texttt{send}\,(a,2)\ \texttt{send}\,(b,0)\ \texttt{send}\,(a,0)$$

La descompresión puede realizarse mediante el algoritmo 5.2. La función `get`(σ, m) recupera los símbolos del fichero comprimido mientras que quedan hijos de algún nodo por etiquetar (el número de nodos por etiquetar es n) y la función `append`(τ σ, m) reconstruye el árbol con esta información. Si quisiéramos que las probabilidades se ajustaran más a los datos tendríamos que aumentar el orden del modelo, lo que aumentaría el consumo de memoria exponencialmente. Para evitar estos problemas aplicaremos un modelo de contexto finito.

5.2 Modelo de contexto finito para árboles n-arios

El modelo que se va a detallar a continuación es de contexto finito y puede tener la ariedad arbitraria (recordemos que en la sección anterior, si el número de hijos posibles es mayor que 2 es preciso aplicar una transformación).

El modelo $M^{[k]}$ (con $k > 1$) consiste en una serie de contadores: uno, $C^{[k]}(t)$ para cada k-fork o $(k-1)$-subtree t en cualquier árbol de la parte del fichero procesada y $C_r^{[k]}(t)$ para cada $(k-1)$-root. Los contadores $C_r(t)$ se normalizan independientemente de los $C(t)$, aunque todos ellos se inicializan a cero

$$r^{[k]}\,(t) = \frac{C_r^{[k]}\,(t)}{|S|}$$

y

$$p_m\left(\sigma\, t_1, \ldots, t_m\right) = \frac{C^{[k]}\left(\sigma\left(t_1, \ldots, t_m\right)\right)}{C^{[k-1]}\left(r_{k-1}\left(\sigma\left(t_1, \ldots, t_m\right)\right)\right)}$$

Si se intentara codificar un árbol directamente con el modelo $M^{[k]}$ podría aparecer una secuencia que no sea reconocida por el modelo, por lo que tendría probabilidad 0. Para evitar esta situación se propone dividir en dos fases la codificación:

Fase 1: Se codifica en formato de texto la información necesaria para actualizar el modelo de contexto finito para árboles, tal y como se muestra en el algoritmo 5.3.
Para ello, se utiliza un modelo de codificación de texto muy eficiente basado en el método de compresión por ordenamiento de bloques (Burrows-Wheeler) descrito en el apéndice A.1. Adicionalmente, se utilizan símbolos especiales que no pertenecen al alfabeto para separar las transiciones, indicar el fin de la fase 1 y el fin de fichero (EOF). La función encargada de realizar este proceso es `sendText`(t) que envía al codificador aritmético los rangos de probabilidades de los símbolos de texto correspondientes a t. Si t es un k-root la función es `sendText_r`(t) y si es un k-fork o un $(k-1)$-subtree `sendText_p`(t). Para ello es necesario mantener un modelo para los símbolos de texto además del modelo de contexto finito que se utiliza en la fase 2.

Fase 2: Se codifica el árbol en preorden con las probabilidades del modelo de contexto finito, y además se actualiza después de examinar cada subárbol como se muestra en el algoritmo 5.4. Este algoritmo se aplicará a cada árbol τ del fichero. Las funciones `send_r`(q) y `send`(t) envían al codificador aritmético el rango de probabilidades correspondientes a la raíz q o a la regla t en el modelo $M^{[k]}$.

La función `acutaliza_modelo` conserva el contador $C^{[k]} = 1$ ó $C_r^{[k]} = 1$, si es la primera vez que se usa, ya que se ha inicializado en la fase 1. Posteriormente se incrementarán estos contadores cada vez que aparezcan sus contextos.

Algoritmo 5.3 Función para el análisis y actualización de transiciones de un árbol τ y un modelo $M^{[k]}$.

función compresion_texto_nuevas_transiciones(τ, k)
  si $C_r^{[k]}(r_{k-1}(\tau)) = 0$ **entonces**
    sendText_r($r_{k-1}(\tau)$)
    append_root_model($r_{k-1}(\tau)$)
  fin si
  para todo $(t = \sigma(t_1, \ldots, t_m)$ subárbol de $\tau)$ **haz** [recorrido preorden]
    si $C^{[k]}(r_k(t)) = 0$ **entonces** [transición no en $M^{[k]}$]
      sendText_p($r_k(t)$)
      append_rule_model($r_k(t)$)
    fin si
  fin para
fin función

Algoritmo 5.4 Llamada principal para la compresión de árboles con modelos de contexto finito.

función compresionContextoFinito(τ, k)
  compresion_texto_nuevas_transiciones(τ, k)
  send_r($r_{k-1}(t)$)
  update_model
  para todo $(t = \sigma(t_1, \ldots, t_m)$ subárbol de $\tau)$ **haz** [recorrido preorden]
    send($r_k(t)$)
    update_model
  fin para
fin función

Como ejemplo, si tomamos el árbol de la figura 5.4 con un modelo de contexto finito de $k = 2$, y enviamos el símbolo "|" como separador de transiciones y "*" como indicador de final de fase 1, la secuencia de codificación

q	$r(q)$	$\sigma\,(t_1,\ldots,t_m)$	$p_m^{[2]}\,(\sigma,t_1,\ldots,t_m)$
a	$1/1$	a	$1/1$
b	0	b	$1/2$
		$b(ab)$	$1/2$

Tabla 5.2: Modelo de contexto finito $M^{[2]}$, extraído de la fase 1, para el árbol $a(ba(ba))$.

obtenida en la fase 1 es

$$a \mid b \mid b(ba) \mid *$$

La fase 2 comienza con un modelo de contexto finito $M^{[2]}$ para árboles como el de la tabla 5.2 y generaría los códigos

$$\texttt{send\_r}(a)\,\texttt{send}\,(a(ba))\;\texttt{send}\,(b)\;\texttt{send}\,(a(ba))\;\texttt{send}\,(b)\;\texttt{send}\,(a)$$

5.3 Modelo de predicción por concordancia parcial para árboles n-arios

Este modelo de compresión es más eficiente que el anterior, ya que utiliza varios modelos de contexto finito y aplica el modelo de mayor orden k posible. Este modelo es una extensión de los modelos *PPM* para cadenas (apartado 3.4), en el que los *escapes*, se aplican a cada estado del AAADE. En este modelo, al igual que en el anterior, se predicen los próximos símbolos a comprimir usando como contexto actual el de k-forks o el de k-subtrees en el que se encuentra el nodo.

Si una regla no se puede aplicar porque no existe en un modelo $M^{[k]}$, se intenta aplicar el modelo $M^{[k-1]}$. En general, el mejor modelo, (el que más comprime) es el de mayor orden. En cambio, el más genérico pero que comprime menos es el modelo con $k = 1$ que contiene solamente información

sobre la etiqueta y el número de hijos del nodo.

Algunas consideraciones importantes:

1. Una diferencia respecto de la compresión de cadenas es que hay diferentes formas de recorrer un árbol. El recorrido por *niveles y de izquierda a derecha* ofrece la ventaja de poder actualizar el modelo antes de haber terminado de analizar el árbol completo. Este hecho mejora la compresión de los ficheros con poca cantidad de árboles.

2. Cada modelo $M^{[k]}$ (con $k > 1$) consiste en una serie de contadores: $C^{[k]}(t)$ para cada k-fork o $(k-1)$-subtree y $C_r^{[k]}(t)$ para cada k-root, encontrado en la parte de fichero procesada. Además, se añade a cada contexto un contador adicional $C_r^{[k]}(\varepsilon)$ o $C_\varepsilon^{[k]}(t)$ para cada código de escape. Todos los contadores se inicializan a cero y se normalizan con $C^{[k-1]}(r_{k-1}(t))+C_\varepsilon^{[k]}(r_{k-1}(t))$ en el primer caso y con $C_r^{[k-1]}(t)+C_r^{[k]}(\varepsilon)$ en el segundo caso, esto es, si $t = \sigma(t_1, \ldots, t_m)$:

$$r^{[k]}(t) = \frac{C_r^{[k]}(t)}{|S| + C_r^{[k]}(\varepsilon)}$$

y

$$p_m(\sigma\, t_1, \ldots, t_m) = \frac{C^{[k]}(\sigma(t_1, \ldots, t_m))}{C^{[k-1]}(r_{k-1}(\sigma(t_1, \ldots, t_m))) + C_\varepsilon^{[k]}(r_{k-1}(\sigma(t_1, \ldots, t_m)))}$$

Recuérdese que el estado asociado a una regla t de $M^{[k]}$ es $r_{k-1}(t)$

3. El modelo básico ($k = 1$) consiste en un contador $C^{[1]}(\sigma)$ para cada símbolo σ perteneciente al alfabeto finito Σ. Están inicializados a 0 y asignan una probabilidad que es independiente del contexto

$$p^{[1]}(\sigma) = \frac{1 + C^{[1]}(\sigma)}{|\Sigma| + \sum_{a \in \Sigma} C^{[1]}(a)} \tag{5.3}$$

A diferencia de las cadenas, los árboles en un nodo tienen un número de descendientes que se debe de predecir antes de codificar su etiquetas. Por ello, este modelo básico incorpora la codificación de los $m \leq M$ descendientes para un determinado nodo σ. Para prevenir desbordamientos, ya que la ariedad no está acotada, se utilizará $M+1$ contadores para cada símbolo ($C_M^{[1]}(m \mid \sigma)$ todos inicializados a 1), reservando la última posición $C_M^{[1]}(\varepsilon \mid \sigma)$ para indicar que se necesitan M contadores más.

Algoritmo 5.5 Llamada principal para la compresión PPM para árboles

función compresionPPM(τ, kmax)
  encode_r($r_{kmax-1}(\tau)$, kmax)
  para todo (t subárbol de τ) **haz** [recorrido por niveles]
    si depth $(t) \geq k-2$ **entonces**
      encode_p($(r_{kmax}(t)$, kmax)
    fin si
  fin para
fin función

Con estas consideraciones, el algoritmo llama a un procedimiento (representado en el algoritmo 5.5), para cada τ contenido en el fichero y con un máximo orden de k_{max} permitido por los modelos.

En este procedimiento se llama a la función `encode_r` que genera el subárbol raíz necesario para comenzar la compresión (algoritmo 5.6). Si el modelo k no es capaz de codificar esta raíz se ayuda del modelo $k-1$, y así sucesivamente hasta llegar al modelo $M^{[1]}$. Una vez codificada la raíz procedemos a codificar el resto del árbol con la función `encode_p` que genera los subárboles necesarios para construir el árbol completo (algoritmo 5.7). Como en la anterior codificación, si el modelo k no es capaz de codificar el subárbol actual se recurre al modelo $k-1$, y así sucesivamente hasta llegar al modelo base que es capaz de generar cualquier subárbol.

Las funciones de compresión arriba citadas se ayudan de otras auxiliares que codifican un subárbol según la probabilidad de un modelo, enviando esta

Algoritmo 5.6 Función para la codificación de la raíz.

```
función encode_r(t,k) [t = σ(t_1, ..., t_m)]
  si C_r^[k](t) > 0 entonces [raíz en M_k]
    send_r(t, k)
    update_models_root
  si no [raíz no en M^[k]]
    send_r(ε, k)
    inc(C_r^[k](ε))
    si k > 2 entonces
      encode_r(r_{k-2}(t), k − 1)
      encode_p(t, k − 1)
    si no
      send_1(σ)
      inc(C^[1](σ))
    fin si
    inc(C_r^[k](t))
  fin si
fin función
```

información al codificador aritmético para que codifique el subárbol. Estas funciones son:

- `send_r`(t, k) envía el rango de probabilidades del subárbol raíz t en el modelo $M^{[k]}$;
- `send_p`(t, k) envía el rango de probabilidades del subárbol t en el modelo $M^{[k]}$;
- `send_1`(σ) envía el rango de probabilidades de la etiqueta σ en el modelo $M^{[1]}$;
- `send_1M`$(m \mid \sigma)$ envía el rango de probabilidades de m hijos según la etiqueta σ en el modelo $M^{[1]}$;
- `send`(t, q, k) envía el rango de probabilidades correspondientes a la transición t perteneciente al estado q en el modelo $M^{[k]}$. Hay remarcar

Algoritmo 5.7 Función para la codificación del árbol.

```
función encode_p(t,k)
  si C^[k](t) > 0 entonces [subárbol en M^[k]]
    send(t, r_{k-1}(t), k)
    update_models(t, k)
  si no [subárbol no en M^[k]]
    send(ε, r_{k-1}(t), k)
    inc(C_ε^[k](r_{k-1}(t)))
    si k > 2 entonces [t = σ(t_1, ..., t_m)]
      para todo (j = 1, ..., m) haz
        si depth(t) ≥ k − 2 entonces
          encode_p(t_j, k − 1)
        fin si
      fin para
    si no [t = σ(σ_1, ..., σ_m)]
      mientras m ≥ C_M^[1](ε | σ) × mmax haz
        send_1M(ε, σ)
        inc(C_M^[1](ε | σ))
      fin mientras
      send_1M(m, σ)
      inc(C_M^[1](m | σ))
      para todo (j = 1, ..., m) haz
        send_1(σ)
        inc(C^[1](σ_j))
      fin para
    fin si
    inc(C^[k](t))
  fin si
fin función
```

que es necesario suministrar como parámetro la transición y el estado porque en ocasiones $t = \varepsilon$, y en este caso, $q \neq r_{k-1}(t)$;

También se utiliza la función `inc`$(C^{[k]}(t))$ que incrementa en uno el contador $C^{[k]}(t)$. Adicionalmente, el parámetro externo mmax indica el número de hijos que se puede codificar inicialmente en el modelo $M^{[1]}$ y será el incremento, cada vez que se supere el número de máximo de hijos para una etiqueta σ.

Algoritmo 5.8 Llamada principal para la de descompresión PPM para árboles

```
función descompresionPPM(kmax)
  τ=decode_r(kmax)
  para todo (t subárbol de τ) haz [recorrido por niveles]
    si depth(t) ≥ k − 2 entonces
      t=decode_p(t, kmax)
      replace(t, τ)
    fin si
  fin para
  devuelve τ
fin función
```

De una forma similar a la compresión se define la descompresión (algoritmo 5.8). Al igual que la compresión, esta llamada principal se basa en otras dos: `decode_r` (algoritmo 5.9) y `decode_p` (algoritmo 5.10) que son la funciones inversas de `encode_r` y `encode_r`, respectivamente. Estas funciones son capaces de reconstruir el árbol τ a partir de la codificación. La primera, `decode_r`, reconstruye la raíz y el resto del árbol lo reconstruye `decode_p`. La función `update_models` actualiza los modelos inferiores o iguales al k que se le pasa como parámetro y la función `replace`(t, τ) enraíza el k-fork t en el nodo correspondiente según el recorrido por niveles y de izquierda a derecha.

Las funciones auxiliares en las que se apoyan los algoritmos anteriores son las inversas de las funciones de compresión.

Algoritmo 5.9 Función para la descodificación de la raíz.

función decode_r(k)
  t=get_r(k)
  si $t \neq \varepsilon$ **entonces** [raíz en $M^{[k]}$]
    update_models_root(t, k)
  si no [raíz no en M_k]
    inc($C_r^{[k]}(\varepsilon)$)
    si $k > 2$ **entonces**
      t=decode_r($k-1$)
      t=decode_p(t, $k-1$)
    si no [$t = \sigma$]
      t=get_1(σ)
      inc($C^{[1]}(\sigma)$)
    fin si
    inc($C_r^{[k]}(t)$)
  fin si
  devuelve t
fin función

5.4 Resultados

Con el fin de comparar los diferentes modelos se han creado ficheros artificiales con diferente tamaño y número de árboles. Para ello, se ha utilizado el siguiente AAADE:

$$
\begin{array}{llcll}
1: & q_0 & = & \delta_7(S, q_4, q_1, q_5, q_0, q_6, q_0, q_7) & (0.2) \\
2: & q_0 & = & \delta_5(S, q_4, q_1, q_5, q_0, q_7) & (0.2) \\
3: & q_0 & = & \delta_2(S, q_8, q_1) & (0.6) \\
4: & q_1 & = & \delta_3(E, q_1, q_9, q_2) & (0.3) \\
5: & q_1 & = & \delta_1(E, q_2) & (0.7) \\
6: & q_2 & = & \delta_3(T, q_2, q_{10}, q_3) & (0.2) \\
7: & q_2 & = & \delta_1(T, q_3) & (0.8) \\
8: & q_3 & = & \delta_1(F, q_{11}) & (0.9) \\
9: & q_3 & = & \delta_1(F, q_1) & (0.1)
\end{array}
$$

$$
\begin{array}{llcll}
10: & q_4 & = & \delta_0(\texttt{i}) & (1.0) \\
11: & q_5 & = & \delta_0(\texttt{t}) & (1.0) \\
12: & q_6 & = & \delta_0(\texttt{e}) & (1.0) \\
13: & q_7 & = & \delta_0(\texttt{f}) & (1.0) \\
14: & q_8 & = & \delta_0(\texttt{p}) & (1.0) \\
15: & q_9 & = & \delta_0(+) & (1.0) \\
16: & q_{10} & = & \delta_0(*) & (1.0) \\
17: & q_{11} & = & \delta_0(\texttt{n}) & (1.0)
\end{array}
$$

Algoritmo 5.10 Función para la decodificación del árbol.

función decode_p(t,k)
  t'=get(t, k)
  si $t' \neq \varepsilon$ **entonces** [subárbol en $M^{[k]}$]
    update_models(t', k)
    $t = t'$
  si no [subárbol no en $M^{[k]}$]
    inc($C_{\varepsilon}^{[k]}(r_{k-1}(t))$)
    si $k > 2$ **entonces**
      para todo $(j = 1, \ldots, m)$ **haz**
        si $\text{depth}(t) \geq k - 2$ **entonces**
          decode_p(t_j, $k-1$)
        fin si
      fin para
      $t = \sigma(t_1, \ldots, t_m)$
    si no [$t = \sigma$]
      m=get_1M(σ)
      mientras $m = \varepsilon$ **haz**
        inc($C_M^{[1]}(\varepsilon \mid \sigma)$)
        m=get_1M(σ)
      fin mientras
      inc($C_M^{[1]}(m \mid \sigma)$)
      para todo $(j = 1, \ldots, m)$ **haz**
        get_1(σ_j)
        inc($C_M^{[1]}(\sigma_j)$)
      fin para
      $t = \sigma(\sigma_1, \ldots, \sigma_m)$
    fin si
    inc($C^{[k]}(t)$)
  fin si
  devuelve t
fin función

El primer número identifica la transición, el número entre paréntesis es la probabilidad de transición y los terminales aparecen en letra tipográfica. Además, $r(q_i) = 1$ si $i = 0$, y cero en caso contrario. Este AAADE genera frases condicionales del tipo `if...then...else...endif`, `if...then...endif` y órdenes como `print` junto con expresiones numéricas con operadores. Esta gramática también se utilizó en el artículo de Calera-Rubio et al. (1999) para generar ficheros de test.

A los ficheros generados se les han aplicado los modelos adaptativos descritos en este capítulo:

- un modelo completo para árboles binarios inicializado con una distribución a priori;
- un modelo de contexto finito para árboles *ITLAC* (Incremental Tree Languages Arithmetic Compresion) con $k = 3$;
- un modelo adaptativo de tipo PPM, *ITLAC-PPM* (Incremental Tree Languages Arithmetic Compresion with PPM modelling) con $k_{max} = 3$.

Por otra parte, se ha comparado con otro modelo estático (no adaptativo) de compresión para árboles (*TLAC*, Tree Languages Arithmetic Compresion) descrito en Calera-Rubio et al. (1999) que está basado a su vez en el algoritmo *tlips* (Carrasco et al. 1998) para la inferencia de lenguajes de árboles estocásticos. Además, se han aplicado dos modelos de compresión lineal:

- una implementación del algoritmo de Ziv and Lempel (1977), *gzip* (J. L. Gailly, 1992-1993);
- una implementación (Nelson 1991) muy eficiente para texto del modelo de k-gramas con $k = 3$.

Los resultados obtenidos se presentan en la tabla 5.3 y en las gráficas 5.5 y 5.7. En ellas se presentan los tamaños obtenidos en función de la parte del

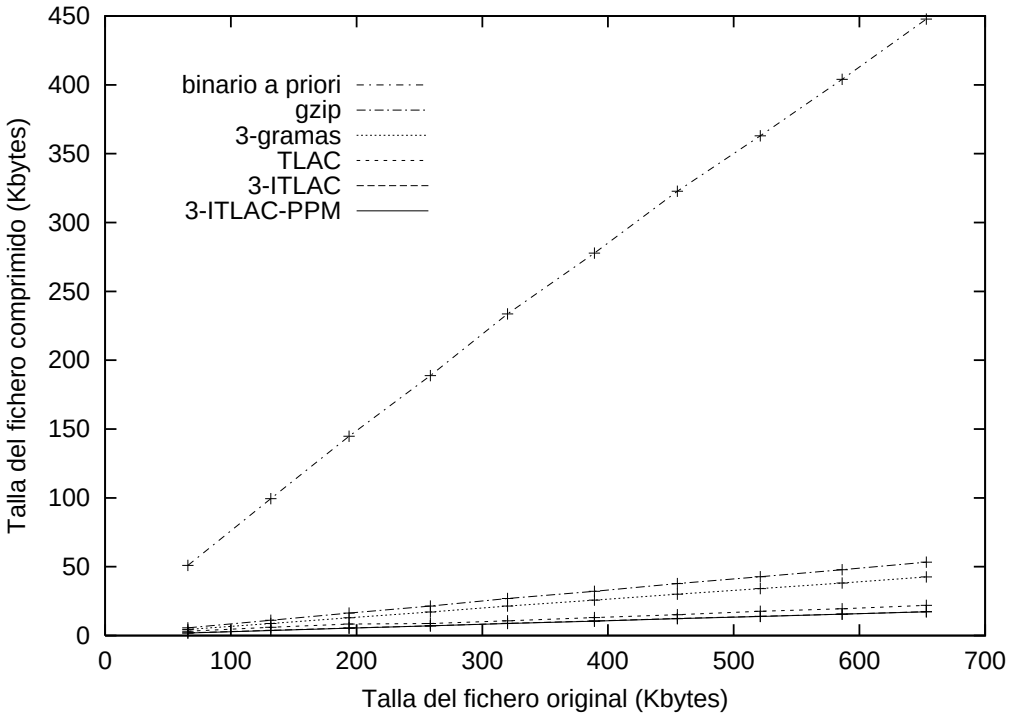


Figura 5.5: Talla de los ficheros de prueba comprimidos. Los resultados de los métodos 3-ITLAC y 3-ITLAC-PPM se superponen en esta gráfica.

fichero procesada para los algoritmos descritos en este capítulo junto a *gzip*, k-gramas y $TLAC$. Se tiene una visión relativa con las tasas de compresión y luego una visión global con la talla de los ficheros en bytes. Las gráficas 5.6 y 5.8 amplían las anteriores y permiten una mejor evaluación de los resultados al prescindir de los dos métodos más ineficientes (binario y gzip).

También se ha probado el modelo $ITLAC\text{-}PPM$ con un fichero de 6129 Mbyte perteneciente a árboles del Penn Tree-bank (Marcus et al. 1994). La tasa de compresión obtenida es de 13.94, comparada con 6.52, obtenida con el *gzip*, 10.92 (*bzip2*) o 9.08 utilizando compresión aritmética de 3-gramas. Estos resultados muestran que este nuevo modelo permite tasas de compresión mayores.

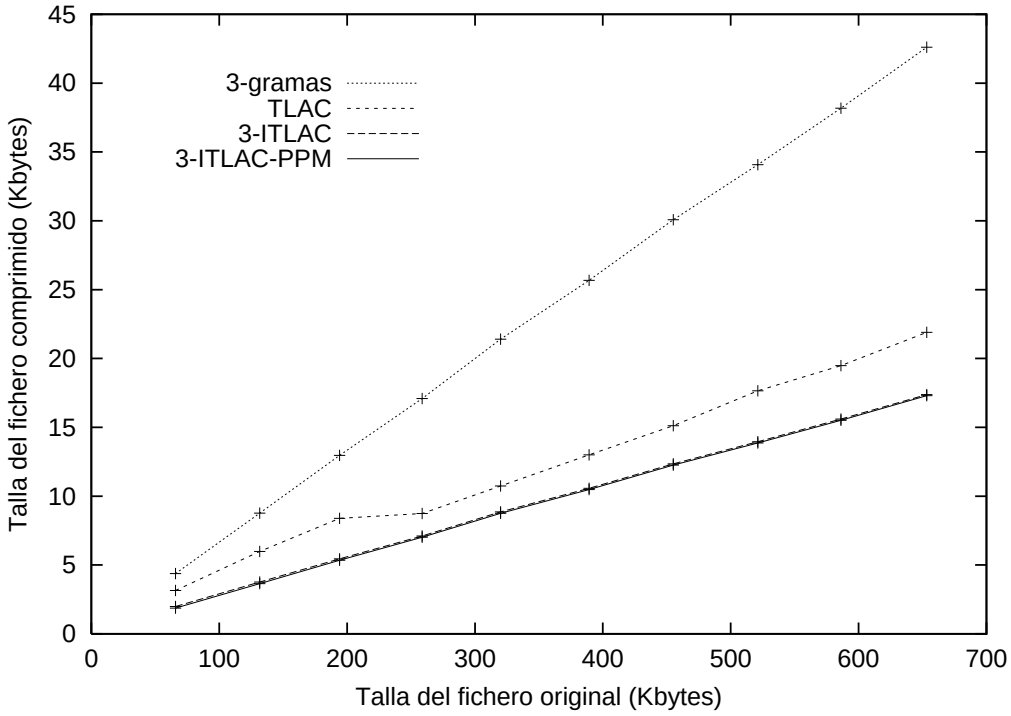


Figura 5.6: Talla de los ficheros de prueba comprimidos con los 4 mejores métodos. Los resultados de los métodos 3-ITLAC y 3-ITLAC-PPM se superponen en esta gráfica.

nº árboles	original	gzip	3-gramas	binario	TLAC	3-ITLAC	3-ITLAC-PPM
1000	66	6	4	51	3	2	2
2000	132	11	9	100	6	4	4
3000	194	16	13	145	8	5	5
4000	259	21	17	189	9	7	7
5000	320	27	21	234	11	9	9
6000	389	32	26	278	13	11	10
7000	455	38	30	323	15	12	12
8000	521	43	34	363	18	14	14
9000	586	48	38	404	19	16	16
10000	653	53	43	448	22	17	17

Tabla 5.3: Tamaño en Kbytes para 10 ficheros de prueba.

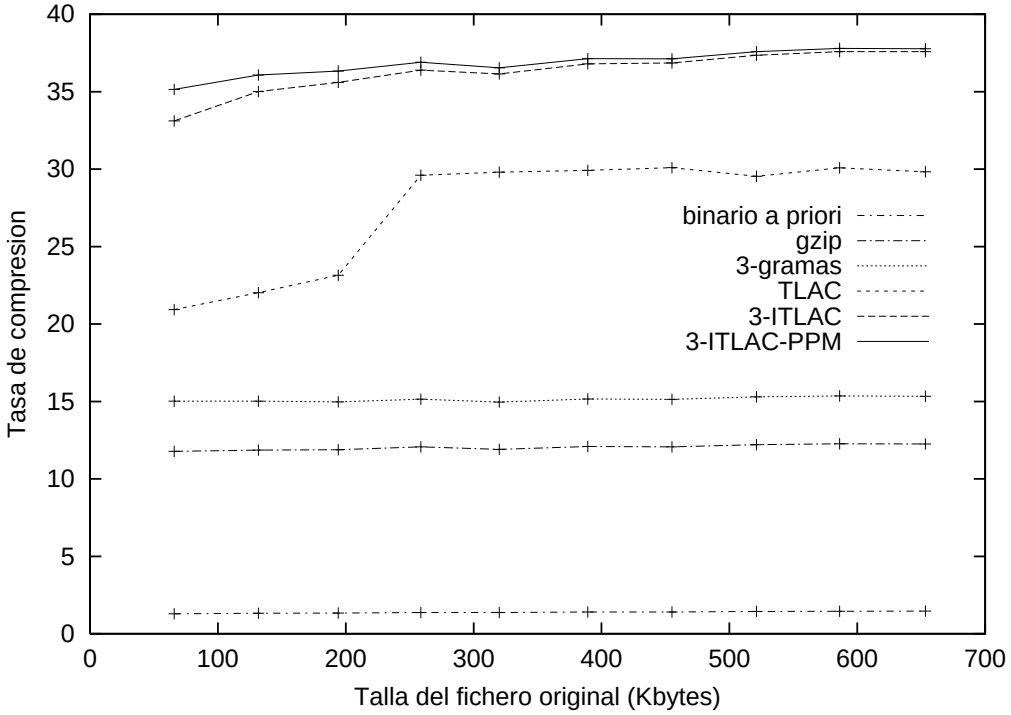


Figura 5.7: Tasas de compresión de los ficheros de prueba.

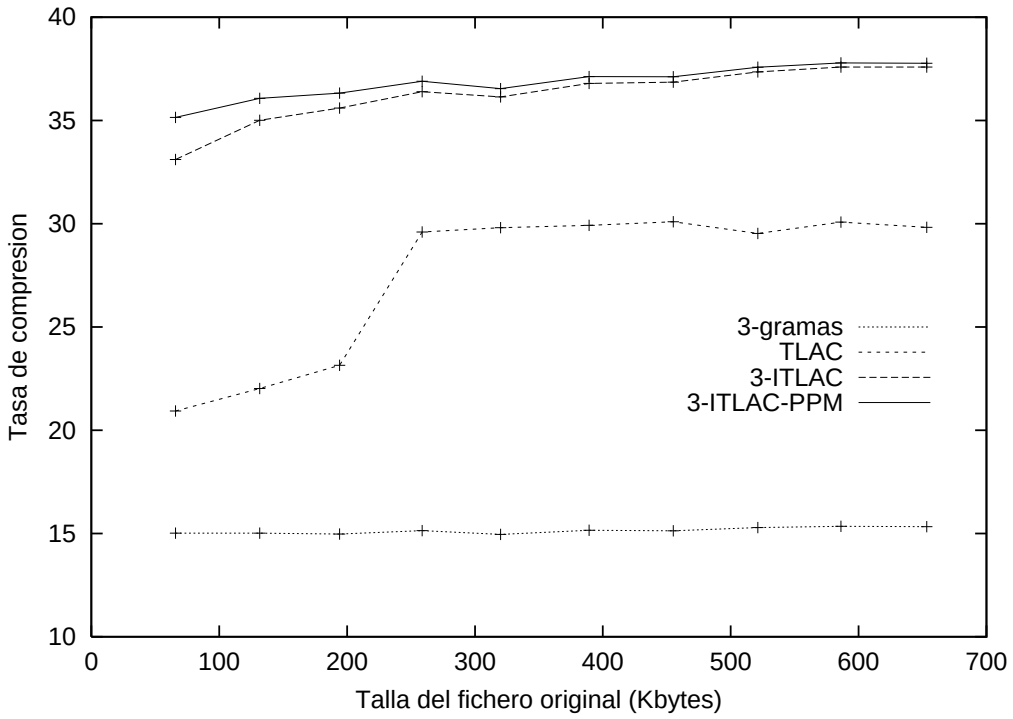


Figura 5.8: Tasas de compresión de los ficheros de prueba con los 4 mejores métodos.

5.5 Conclusiones

En este capítulo se ha presentado una comparación exhaustiva entre diferentes algoritmos de compresión de información con estructura de árbol. La primera aproximación (modelo adaptativo para árboles binarios inicializado una distribución a priori) no mejora al clásico *Ziv Lempel.* Ello se debe a la simplicidad del modelo, por lo que sus probabilidades de predicción no son eficientes. Los nuevos algoritmos *ITLAC* e *ITLAC-PPM* mejoran a un buen algoritmo de compresión de árboles como es el *TLAC* (Calera-Rubio et al. 1999) y produce un fichero 3 veces menor que los obtenidos usando un implementación extendida del Ziv-Lempel (*gzip*)

Los mejores resultados se han obtenido utilizando una extensión para árboles del suavizado multinivel (PPM).

La aplicación sobre datos artificiales permite concluir que este procedimiento garantiza la tasa de compresión máxima independientemente del tamaño del fichero que se quiera comprimir.

Aplicado sobre árboles de análisis sintáctico de una base de datos lingüística (Penn Tree-bank) este método sigue proporcionando los mejores resultados y permite tasas de compresión que superan en un 30% a las mejores técnicas de propósito general.

Capítulo 6

Clasificación mediante modelos k-testables

En la clasificación estocástica se asigna cada ejemplo o patrón a la clase que lo genera con mayor probabilidad. En este capítulo se exponen diferentes aproximaciones al cálculo de la probabilidad de un árbol mediante autómatas de árboles ascendentes deterministas estocásticos. Se presenta una aplicación directa como es la clasificación de caracteres manuscritos aplicando modelos estocásticos de árboles. Asimismo y con el propósito de establecer una comparación en otro tipo de técnicas, se presentará un método no probabilístico basado en la distancia de edición de árboles.

6.1 Introducción

El método general de clasificación de un patrón caracterizado como árbol consiste en la inferencia de un AAADE para cada clase a partir de muestras bien clasificadas; un nuevo patrón se asignará a la clase que maximice su verosimilitud (Duda and Hart 1973). Para ello, una vez inferido el AAADE, la probabilidad condicionada de un árbol se calcula multiplicando las probabilidades de las transiciones utilizadas para procesar el árbol.

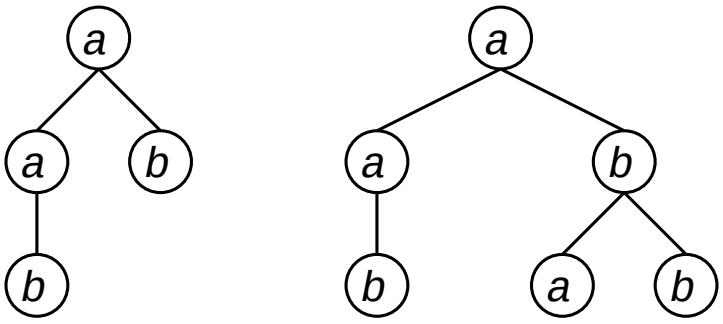


Figura 6.1: Árboles de la clase A.

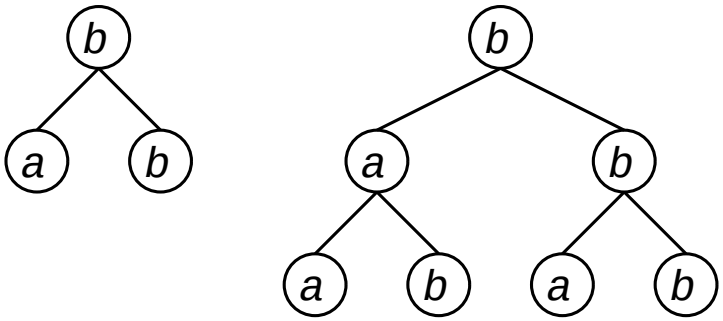


Figura 6.2: Árboles de la clase B.

q	$r(q)$	$\sigma(t_1, \ldots, t_m)$	$p_m(\sigma, t_1, \ldots, t_m)$
a	0	a	2/2
b	0	b	3/3
$a(b)$	0	$a(b)$	1/1
$a(ab)$	2/2	$a(a(b)b)$ $a(a(b)b(ab))$	1/2 1/2
$b(ab)$	0	$b(ab)$	1/1

Tabla 6.1: AAADE obtenido a partir de los ejemplos de la figura 6.1 con $k = 3$.

q	$r(q)$	$\sigma(t_1, \ldots, t_m)$	$p_m(\sigma, t_1, \ldots, t_m)$
a	0	a	3/3
b	0	b	3/3
$a(ab)$	0	$a(ab)$	1/1
$b(ab)$	2/2	$b(ab)$ $b(a(ab)b(ab))$	2/3 1/3

Tabla 6.2: AAADE correspondiente a los ejemplos de la figura 6.2 con $k = 3$.

El problema aparece cuando esta medida asociada al árbol a clasificar no proporciona suficiente información. Para ilustrarlo vamos a mostrar un ejemplo. Sean los árboles de muestra de la figura 6.1 para la clase A y 6.2 para la clase B, con los que se obtienen los AAADE de la tablas 6.1 y 6.2 respectivamente. En dichas tablas aparecen los estados q del AAADE en la primera columna, las probabilidades $r(q)$ en la segunda, las transiciones $\sigma(t_1, \ldots, t_m)$ tales que $\delta(\sigma(t_1, \ldots, t_m)) = q$ en la tercera y su correspondiente probabilidad $p_m(\sigma, t_1, \ldots, t_m)$ en la cuarta. Para calcular la probabilidad de un árbol, por ejemplo $t = a(ab)$, según estos dos AAADE distintos (A y B) debemos analizar el árbol y aplicar las reglas del AAADE correspondiente según la ecuación (4.8) y (4.9). En este caso se obtiene

$$p(t \mid A) = r^A(a(ab))\ p_2^A(a, a, b)\ p_0^A(a)\ p_0^A(b) = 1/3 \tag{6.1}$$

$$p(t \mid B) = r^B(a(ab))\ p_2^B(a, a, b)\ p_0^B(a)\ p_0^B(b) = 0 \tag{6.2}$$

De acuerdo con estos resultados, deberíamos asignar el árbol $t = a(ab)$ a la clase A, ya que $p(t \mid A) > p(t \mid B)$.

En cambio, si el árbol fuera $t = a(a(b))$, obtendríamos

$$p(t \mid A) = r^A(a(a))\ p_1^A(a, a(b))\ p_1^A(a, b)\ p_0^A(b) = 0 \tag{6.3}$$

$$p\left(t \mid B\right) = r^B\left(a\left(a\right)\right)\, p_1^B\left(a, a(b)\right)\, p_1^B\left(a, b\right)\, p_0^B\left(b\right) = 0 \tag{6.4}$$

Como se obtiene una probabilidad nula en ambos casos la asignación a una u otra clase será arbitraria y poco fiable. Para evitar estos casos es preciso suavizar los modelos.

Este problema ha sido estudiado anteriormente para los modelos de k-gramas aplicados a cadenas y hay propuestas varias soluciones:

- los modelos de descuento: modelo de Katz (Katz 1987), descuento absoluto (Ney and Essen 1993) y descuento lineal (Katz 1987; Jelinek 1990);
- la interpolación de modelos (Ney et al. 1997) y
- los modelos PPM (Cleary and Witten 1984) descritos en la sección 3.2.

A continuación, se proponen tres métodos de suavizado de los AAADE que les permiten obtener valores no nulos de probabilidad para los árboles que requieren transiciones no observadas en la muestra.

6.2 Métodos de suavizado

La notación seguida en este apartado será la siguiente:

- $M^{[k]}$ es un AAADE de orden k;
- S es el conjunto de muestras para la inferencia de los modelos $M^{[k]}$;
- $N\left(\sigma\right)$ es el número de nodos etiquetados con el símbolo σ en la muestra S;
- N es el número total de nodos en S;
- μ_σ es la media aritmética del número de descendientes de los nodos con etiqueta σ;

- μ es la media aritmética del número de descendientes de toda la clase, esto es, $\mu = 1 - |S|/N$;
- $L(m \mid \sigma)$ es el número de veces que aparece un nodo con el símbolo σ y m hijos en la muestra S;
- $\mathrm{P}(m \mid \mu)$ es la distribución de Poisson con media μ.

6.2.1 Interpolación de modelos

La técnica de inferencia descrita en la sección 4.4 permite obtener simultáneamente varios modelos estocásticos $M^{[k]}$, uno para cada valor de k variando desde 2 hasta k_{max}. Intuitivamente, es más probable que un modelo con k grande asigne una probabilidad nula a un árbol, debido a que muchos de los parámetros del modelo no han podido ser estimados a partir de la muestra. Pero incluso un modelo 2-testable puede predecir, en algún caso una probabilidad nula. Una solución habitual para resolver este problema es calcular la probabilidad como combinación lineal de todos los modelos (con unos coeficientes fijos), incluyendo un modelo base, $M^{[1]}$, que nunca asigna probabilidades nulas.

Si se calcula la probabilidad de un árbol combinando linealmente todos los modelos:

$$p(\tau \mid A) = \sum_{k=1}^{k_{max}} \alpha_k \, p\left(\tau \mid M^{[k]}\right) \tag{6.5}$$

con la restricción $\sum_{k=1}^{k_{max}} \alpha_k = 1$, se tiene la seguridad, siempre de que $\alpha_1 > 0$, de que $p(\tau \mid A) > 0$ para cualquier árbol τ.

Por tanto, para que la interpolación no asigne probabilidad nula a ningún árbol es importante disponer de un modelo base $M^{[1]}$ que nunca presente este problema. Pasamos a describir un modelo base $M^{[1]}$ que reconoce cualquier

árbol sea cual sea el número máximo de hijos. Se define este modelo mediante dos distribuciones: una, $p_L(m \mid \sigma)$, para el número de hijos asociado a la etiqueta σ y otra, $p^{[1]}(\sigma)$, para las etiquetas. Ambas distribuciones deben adaptarse lo mejor posible a las muestras de entrenamiento de cada clase, no utilizando parámetros obtenidos fuera del entrenamiento mientras no sea necesario. Con esto,

$$p_L(m \mid \sigma) = \begin{cases} \dfrac{L(m \mid \sigma)}{N(\sigma)} \left(1 - \lambda_L(m, \sigma)\right) & \text{si } L(m \mid \sigma) > 0 \\ \Lambda_L(\sigma) \dfrac{\mathrm{P}(m \mid \mu_\sigma)}{1 - \displaystyle\sum_{l : L(l \mid \sigma) > 0} \mathrm{P}(l \mid \mu_\sigma)} & \text{si } L(m \mid \sigma) = 0 \wedge N(\sigma) > 0 \\ \mathrm{P}(m \mid \mu) & \text{en caso contrario} \end{cases} \tag{6.6}$$

donde $\lambda_L(m, \sigma)$ es la parte de probabilidad que se descuenta a los casos vistos en la muestra para asignárselos a los nos vistos mediante el coeficiente $\Lambda_L(\sigma)$ que es

$$\Lambda_L(\sigma) = \sum_m \frac{L(m \mid \sigma)}{N(\sigma)} \lambda_L(m, \sigma)$$

Nótese que $\Lambda_L(\sigma) > 0$ si $N(\sigma) > 0$, ya que en ese caso, para algún valor de m, $L(m \mid \sigma) > 0$.

La distribución $p_L(m \mid \sigma)$ está basada en la frecuencia relativa de la aparición de las etiquetas del alfabeto en las muestras de entrenamiento y con una distribución de descendientes con decrecimiento exponencial. Se ha utilizado la función de distribución de Poisson

$$\mathrm{P}(m \mid \mu_\sigma) = \mathrm{e}^{-\mu_\sigma} \frac{\mu_\sigma^m}{m!} \tag{6.7}$$

por estar centrada la máxima probabilidad en el número medio de descendientes para una etiqueta dada (μ_σ).

Por otra parte,

$$p^{[1]}(\sigma) = \begin{cases} \frac{N(\sigma)}{N}(1-\lambda(\sigma)) & \text{si } N(\sigma) > 0 \\ \frac{\Lambda}{|\Sigma| - C} & \text{si } N(\sigma) = 0 \end{cases} \tag{6.8}$$

donde

$$\Lambda = \sum_{\sigma \in \Sigma} \frac{N(\sigma)}{N} \lambda(\sigma),$$

y C es el número de símbolos distintos aparecidos en las muestras de entrenamiento. Nótese que si $C = |\Sigma|$, entonces se debe tomar $\lambda(\sigma) = 0$ para cualquier símbolo σ.

Los coeficientes λ deben reasignar probabilidad de los procesos observados a los no observados. Es posible definir de muchas formas diferentes las funciones de descuento λ (Ney et al. 1997; Witten et al. 1999) pero nuestros resultados son poco sensibles a dicha elección. Por ello hemos optado por utilizar un procedimiento similar al del métodos PPMX descrito en la sección 3.4. En concreto hemos utilizado

$$\lambda(\sigma) = \frac{\frac{1}{N} \sum_{\sigma: N(\sigma)=n} N(\sigma)}{1 + \sum_{\sigma: N(\sigma)=n} 1}$$

siendo ahora $n = \min_\sigma N(\sigma)$. De forma análoga

$$\lambda_L(m, \sigma) = \frac{\frac{1}{N(\sigma)} \sum_{\sigma: L(m|\sigma)=n} L(m \mid \sigma)}{1 + \sum_{\sigma: L(m|\sigma)=n} 1}$$

siendo $n = \min_m L(m \mid \sigma)$. Estos coeficientes tienen la propiedad interesante

de que nunca se anulan.

Para seguir interpretando este modelo como un AAADE es necesario definir la función de transición $\delta(t)$ de forma distinta a la ecuación (4.4):

$$\delta_m(\sigma, \sigma_1, \ldots, \sigma_m) = \sigma \tag{6.9}$$

También es diferente a la ecuación (4.20) la forma en que se calcula $p_m^{[1]}(t)$ y se define como:

$$p_m^{[1]}(\sigma, \sigma_1, \ldots, \sigma_m) = p_L(m \mid \sigma) \prod_{i=1}^{m} p^{[1]}(\sigma_i) \tag{6.10}$$

y $r^{[1]}(\sigma) = p^{[1]}(\sigma)$.

Con esta modificaciones y para los ejemplos de las figuras 6.1 (clase A) y 6.2 (clase B), se obtienen los AAADE que se muestran en las tablas 6.3 y 6.4 respectivamente. Si se calcula la probabilidad del árbol $t = a(a(b))$ con el modelo $M_A^{[1]}$ se obtiene

$$\begin{aligned} p\left(t \mid M_A^{[1]}\right) &= p^{[1]}(a)\ \mathrm{p_L}(1 \mid a)\ p^{[1]}(a)\ p_L(1 \mid a)\ p^{[1]}(b)\ p_L(0 \mid b) \\ &= 0.0117 \end{aligned} \tag{6.11}$$

y con el modelo $M_B^{[1]}$

$$\begin{aligned} p\left(t \mid M_B^{[1]}\right) &= p^{[1]}(a)\ \mathrm{p_L}(1 \mid a)\ p^{[1]}(a)\ p_L(1 \mid a)\ p^{[1]}(b)\ p_L(0 \mid b) \\ &= 4.6 \times 10^{-4} \end{aligned} \tag{6.12}$$

Si de nuevo calculamos las probabilidades para el árbol $t = a(a(b))$ con el método de interpolación y para unos coeficientes $\alpha_1 = 0.2$, $\alpha_2 = 0.3$ y $\alpha_3 = 0.5$ se obtiene

σ	$p^{[1]}(\sigma)$	μ_σ	m	$p_L(m \mid \sigma)$
ε	$\Lambda_r^{[1]} = 0$			
a	$5/10$	1.2	ε	$\Lambda^{[1]}(a) = 0.1$
			0	$1/5\,(1-0.1)$
			1	$2/5\,(1-0.1)$
			2	$2/5\,(1-0.1)$
b	$5/10$	0.4	ε	$\Lambda^{[1]}(a) = 0.1$
			0	$4/5\,(1-0.1)$
			2	$1/5\,(1-0.1)$

(a)

q	$r^{[2]}(q)$	$\sigma(t_1,\ldots,t_m)$	$p_m^{[2]}(\sigma, t_1,\ldots,t_m)$
a	$2/2$	$a(ab)$	$2/5$
		$a(b)$	$1/5$
		a	$2/5$
b	0	b	$4/5$
		$b(ab)$	$1/5$

(b)

q	$r^{[3]}(q)$	$\sigma(t_1,\ldots,t_m)$	$p_m^{[3]}(\sigma, t_1,\ldots,t_m)$
a	0	a	$1/1$
$a(b)$	0	$a(b)$	$2/2$
$a(ab)$	$2/2$	$a(a(b)b)$	$1/2$
		$a(a(b)b(ab))$	$1/2$
b	0	b	$4/4$
$b(ab)$	0	$b(ab)$	$1/1$

(c)

Figura 6.3: Modelos $M_A^{[k]}$ correspondientes a la clase A (figura 6.1): (a) $M_A^{[1]}$ con $\mu = 0.8$; (b) $M_A^{[2]}$; (c) $M_A^{[3]}$.

σ	$p^{[1]}(\sigma)$	μ_σ	m	$p_L(m \mid \sigma)$
ε	$\Lambda_r^{[1]} = 0$			
a	$4/10$	0.5	ε	$\Lambda^{[1]}(a) = 0.125$
			0	$3/4\,(1 - 0.125)$
			2	$1/4\,(1 - 0.125)$
b	$6/10$	1	ε	$\Lambda^{[1]}(b) = 0.3333$
			0	$3/6\,(1 - 0.3333)$
			2	$3/6\,(1 - 0.3333)$

(a)

q	$r^{[2]}(q)$	$\sigma(t_1, \ldots, t_m)$	$p_m^{[2]}(\sigma, t_1, \ldots, t_m)$
a	0	$a(ab)$	$1/4$
		a	$3/4$
b	$2/2$	b	$3/6$
		$b(ab)$	$3/6$

(b)

q	$r^{[3]}(q)$	$\sigma(t_1, \ldots, t_m)$	$p_m^{[3]}(\sigma, t_1, \ldots, t_m)$
a	0	a	$3/3$
$a(ab)$	0	$a(ab)$	$1/1$
b	0	b	$3/3$
$b(ab)$	$2/2$	$b(ab)$	$2/3$
		$b(a(ab)b(ab))$	$1/3$

(c)

Figura 6.4: Modelos $M_B^{[k]}$ correspondientes a la clase B (figura 6.2); (a) $M_B^{[1]}$ con $\mu = 0.8$; (b) $M_B^{[2]}$; (c) $M_B^{[3]}$.

- para la clase A:

$$\begin{aligned} p(t \mid A) &= \alpha_1 p\left(t \mid M_A^{[1]}\right) + \alpha_2 p\left(t \mid M_A^{[2]}\right) + \alpha_3 p\left(t \mid M_A^{[3]}\right) \\ &= 2.3 \times 10^{-3} \end{aligned} \tag{6.13}$$

  Nótese que $p\left(t \mid M_A^{[2]}\right) = p\left(t \mid M_A^{[3]}\right) = 0$ porque tienen al menos la probabilidad de una transición nula;

- y para la clase B:

$$\begin{aligned} p(t \mid B) &= \alpha_1 p\left(t \mid M_B^{[1]}\right) + \alpha_2 p\left(t \mid M_B^{[2]}\right) + \alpha_3 p\left(t \mid M_B^{[3]}\right) \\ &= 9.2 \times 10^{-5} \end{aligned} \tag{6.14}$$

  De nuevo $p\left(t \mid M_B^{[2]}\right) = p\left(t \mid M_B^{[3]}\right) = 0$ porque también tienen al menos la probabilidad de una transición nula.

Como podemos comprobar nuestro procedimiento asignaría el árbol $t = a(a(b))$ a la clase A, ya que $p(t \mid A) > p(t \mid B)$.

Este método tiene el inconveniente de anular completamente la probabilidad $p\left(t \mid M^{[k]}\right)$ para $k > 1$ si una sola transición del árbol de entrada no pertenece al modelo $M^{[k]}$, perdiendo así toda información probabilística que el modelo $M^{[k]}$ proporciona sobre el resto del árbol. Este hecho perjudica la precisión de la clasificación. En las secciones siguientes estudiaremos distintas formas de aprovechar al máximo esta información.

6.2.2 Suavizado mediante distribución a priori

En el apartado anterior hemos calculado la probabilidad de un árbol utilizando interpolación de varios modelos $M^{[k]}$. De esta forma la probabilidad asignada a árboles no reconocidos por ninguno de los modelos de orden $k > 1$

es en general muy pequeña, ya que provienen únicamente del modelo base $M^{[1]}$ que debe distribuir un pequeña masa de probabilidad entre un gran número de sucesos. Otra opción para evitar probabilidades nulas es utilizar un modelo con una distribución inicial a priori que acepte todos los árboles que pueden aparecer en la entrada. Este procedimiento ha sido usado habitualmente en los modelos de k-gramas, como por ejemplo en Mackay (1998). Sin embargo, en este caso existe una dificultad: si no se fija el máximo número de hijos el número de sucesos no observados es infinito.

En cambio, si se tiene un alfabeto finito Σ, un valor de k dado y la ariedad máxima M acotada, se puede construir el autómata completo con sus contadores correspondientes a los estados $C_r^{[k]}(t)$ y a las transiciones $C^{[k]}(t)$ inicializados, por ejemplo a 1, para que proporcione siempre una probabilidad no nula. Pero esta inicialización tiene un inconveniente, y es que el autómata resultante no es consistente. Una solución para que lo sea es generar todos los árboles posibles de profundidad 0 hasta $k-1$ y con nodos de ariedad entre 0 y M. Por ejemplo, tomando $\Sigma = \{a, b\}$, $k = 2$ y $M = 2$, obtendríamos 14 árboles S_0=$\{a$, b, $a(a)$, $a(b)$,$b(a)$, $b(b)$, $a(aa)$, $a(ab)$,$a(aa)$, $a(ab)$, $b(aa)$, $b(ab)$,$b(aa)$, $b(ab)\}$. Si el AAADE se inicializa con esta muestra se consiguen varias ventajas:

1. No se producirán transiciones con probabilidad 0, ya que con la inicialización anterior se generan todos los estados con todas las transiciones posibles para cada estado.

2. El AAADE que se infiera incrementalmente después de esta inicialización será también consistente, ya que se ha inferido a partir de muestras (Chaudhuri et al. 1983).

3. Cuando un AAADE se incremente con muestras reales de una clase, las probabilidades de las transiciones se alejarán de la inicialización previa mejorando la clasificación.

Cabe destacar que aunque la generación de todos los ejemplos iniciales tiene un coste exponencial con respecto a k, no hace falta guardar todo el modelo en memoria con la cantidad de recursos que supondría. La forma de cálculo se puede optimizar para que sólo guarde en memoria los contadores de las ecuaciones (4.20) y (4.21) descritos en el capítulo 4. Los valores correspondientes a los ejemplos iniciales se aplicarán sólo a los cálculos de las probabilidades $p_m(t)$ y $r(q)$.

Tomamos de nuevo como ejemplo las figuras 6.1 para la clase A y 6.2 para la clase B, para obtener los AAADE correspondientes, mostrados en las tablas 6.3 y 6.4 respectivamente. Estas tablas representan el resultado de la inicialización con S_0 y los árboles de cada una de las clases. Se han incluido puntos suspensivos en algunas de las casillas porque el número de transiciones totales es muy elevado (422) y la tabla ocuparía varias páginas.

De nuevo calculamos las probabilidades para el árbol $t = a(a(b))$ con el modelo suavizado actual. Para la clase A se obtiene

$$\begin{aligned} p(t \mid A) &= r(a(a))\, p_1(a, a(b))\, p_1(a, b)\, p_0(b) \\ &= 2.3125 \times 10^{-4} \end{aligned} \tag{6.15}$$

Si ahora calculamos la probabilidad para la clase B, quedaría

$$\begin{aligned} p(t \mid B) &= r(a(a))\, p_1(a, a(b))\, p_1(a, b)\, p_0(b) \\ &= 2.3055 \times 10^{-4} \end{aligned} \tag{6.16}$$

Como podemos comprobar, el árbol $t = a(a(b))$ sería asignado a la clase A, ya que $p(t \mid A) > p(t \mid B)$.

Observando la forma en que se han realizado los cálculos con las ecuaciones (6.15) y (6.16) se observa que las probabilidades de $p_1(a, a(b))$ y $r(a(a))$ son distintas de cero a pesar de que estos casos no ha sido vistos en los ejemplos de entrenamiento de las clases A y B. Observamos que la única diferencia está en el cálculo de $p_1(a, b)$: en la clase A esta transición ha sido

q	$r^{[3]}(q)$	$\sigma(t_1,\ldots,t_m)$	$p_m^{[3]}(\sigma,t_1,\ldots,t_m)$
a	$1/424$	a	1
b	$1/424$	b	1
$a(a)$	$7/424$	$\vdots$	$\vdots$
$a(b)$	$7/424$	$a(b)$ $\vdots$ $a(b(bb))$	$61/67$ $\vdots$ $1/67$
$a(aa)$	$49/424$	$\vdots$	$\vdots$
$a(ab)$	$51/424$	$a(ab)$ $\vdots$ $a(a(ab)b(ab))$ $\vdots$ $a(a(bb)b(bb))$	$60/109$ $\vdots$ $2/109$ $\vdots$ $1/109$
$a(ba)$	$49/424$	$\vdots$	$\vdots$
$a(bb)$	$49/424$	$\vdots$	$\vdots$
$b(aa)$	$49/424$	$\vdots$	$\vdots$
$b(ab)$	$49/424$	$b(ab)$ $\vdots$ $b(a(bb)b(bb))$	$60/108$ $\vdots$ $1/108$
$b(ba)$	$49/424$	$\vdots$	$\vdots$
$b(bb)$	$49/424$	$\vdots$	$\vdots$

Tabla 6.3: AAADE correspondiente a los ejemplos de la figura 6.1 clase A para un modelo con $k = 3$, inicializado a priori con $M = 2$. En este caso $|Q| = 14$ y $|\Delta| = 422$. Los puntos suspensivos indican que hay transiciones y probabilidades del autómata completo no representadas.

q	$r^{[3]}(q)$	$\sigma(t_1,\ldots,t_m)$	$p_m^{[3]}(\sigma,t_1,\ldots,t_m)$
a	$1/424$	a	1
b	$1/424$	b	1
$a(a)$	$7/424$	$\vdots$	$\vdots$
$a(b)$	$7/424$	$\vdots$	$\vdots$
$a(aa)$	$49/424$	$\vdots$	$\vdots$
$a(ab)$	$49/424$	$a(ab)$ $\vdots$ $a(a(bb)b(bb))$	$60/110$ $\vdots$ $1/110$
$a(ba)$	$49/424$	$\vdots$	$\vdots$
$a(bb)$	$49/424$	$\vdots$	$\vdots$
$b(aa)$	$49/424$	$\vdots$	$\vdots$
$b(ab)$	$51/424$	$b(ab)$ $\vdots$ $b(a(ab)b(ab))$ $\vdots$ $b(a(bb)b(bb))$	$61/110$ $\vdots$ $2/110$ $\vdots$ $1/110$
$b(ba)$	$49/424$	$\vdots$	$\vdots$
$b(bb)$	$49/424$	$\vdots$	$\vdots$

Tabla 6.4: AAADE correspondiente a los ejemplos de la figura 6.2 clase B para un modelo con $k = 3$, inicializado a priori con $M = 2$. En este caso $|Q| = 14$ y $|\Delta| = 422$. Los puntos suspensivos indican que hay transiciones y probabilidades del autómata completo no representadas

actualizada 2 veces, mientras que en la clase B no ha sido actualizada, hecho que favorece probabilísticamente a la clase A.

El problema de la inicialización a priori es que no será útil si aparece un árbol en el que algún nodo tiene más de M hijos. El procedimiento que se presenta a continuación resuelve este problema.

6.2.3 Modelo de predicción por concordancia parcial

El uso de una distribución inicial para evitar los casos de probabilidad 0 presenta el inconveniente de tener la ariedad máxima acotada. En este apartado se propondrá un nuevo modelo sin esta restricción.

La idea en la que se basa es similar a la explicada en el apartado 3.4 y utilizada para la compresión: consiste en mantener varios modelos $M^{[k]}$ con $1 \leq k \leq k_{max}$. Conforme va descendiendo el valor de k, el modelo generaliza más. Lo que se pretende es calcular siempre la probabilidad de una transición $p_m(t)$ o $r(t)$ con el modelo de mayor k posible. Si éste no tuviera la transición necesaria se generará un código de *escape* e intentaríamos calcular la probabilidad de t con el modelo $k-1$. Si fuera necesario, seguiríamos así hasta llegar al modelo con $k=1$ que es capaz de reconocer cualquier árbol. En lo que sigue tomaremos $k_{max}=3$.

La notación seguida en este apartado será la siguiente:

- $C_r^{[k]}(t)$ es el número de árboles $\tau \in S$ tales que $r_{k-1}(\tau) = t$;
- $C^{[k]}(t)$ es el número de k-forks o $(k-1)$-subtrees isomorfos a t en los árboles $\tau \in S$;
- $\Lambda^{[k]}$ es la probabilidad correspondiente al código ε de *escape* en el modelo $M^{[k]}$;
- $\lambda^{[k]}(t)$ es el factor de descuento en un contexto t.

Como se va a intentar siempre buscar las probabilidades en el modelo de k máxima, la definición de $p(\tau \mid A)$ es $p(\tau \mid A) = p^{[k_{max}]}(\tau \mid A)$, con

$$p^{[k_{max}]}(\tau \mid A) = r^{[k_{max}]}(r_{k-1}(\tau))\ \pi^{[k_{max}]}(\tau) \tag{6.17}$$

Si $\tau = \sigma(\tau_1, \cdots, \tau_m)$ entonces

$$\pi^{[k_{max}]}(\tau) = \begin{cases} p_m^{[k_{max}]}(\sigma, r_{k-1}(\tau_1), \cdots, r_{k-1}(\tau_m)) \prod_{j=1}^{m} \pi^{[k_{max}]}(\tau_j) & \text{si } m > 0 \\ p_0^{[k_{max}]}(\sigma) & \text{si } m = 0 \end{cases}$$

Para asegurar que en ningún caso se producen probabilidades nulas, las probabilidades $p_m^{[k_{max}]}$ usarán, cuando sea necesario, las probabilidades $p_m^{[k_{max}-1]}$ y así recursivamente hasta llegar a $p^{[1]}$ donde siempre se asignaran probabilidades no nulas.

Definiremos a continuación las ecuaciones correspondientes a los modelos $M^{[1]}$, $M^{[2]}$ y $M^{[k]}$ para cualquier $k > 2$. Comencemos por el modelo base $M^{[1]}$. Este modelo base es análogo al introducido en el apartado 6.2.1. De nuevo, el modelo define dos distribuciones, una para el número de hijos de la etiqueta σ, $p_L(m \mid \sigma)$, y otra para la etiqueta, $p^{[1]}(\sigma)$.

La primera se define de la siguiente manera

$$p_L(m \mid \sigma) = \begin{cases} \dfrac{L(m \mid \sigma)}{N(\sigma)} \left(1 - \lambda_L^{[1]}(m, \sigma)\right) & \text{si } L(m \mid \sigma) > 0 \\ \Lambda_L^{[1]}(\sigma) \dfrac{\mathrm{P}(m \mid \mu_\sigma)}{1 - \displaystyle\sum_{l:L(l\mid\sigma)>0} \mathrm{P}(l \mid \mu_\sigma)} & \text{si } L(m \mid \sigma) = 0 \wedge N(\sigma) > 0 \\ \mathrm{P}(m \mid \mu) & \text{en caso contrario} \end{cases} \tag{6.18}$$

El coeficiente de normalización $1 - \sum_{l:L(l\mid\sigma)>0} \mathrm{P}(l \mid \mu_\sigma)$ coincide con $\sum_{l:L(l\mid\sigma)=0} \mathrm{P}(l \mid \mu_\sigma)$, es decir, el sumatorio de las probabilidades de todos los casos que no han aparecido en un contexto, pero esta segunda forma da lugar a sumatorios

infinitos, por ello en la ecuación (6.18) nos basamos en el cálculo del complementario para evitar este problema.

La probabilidad $\Lambda_L^{[1]}(\sigma)$ se define como

$$\Lambda_L^{[1]}(\sigma) = \sum_m \frac{L(m \mid \sigma)}{N(\sigma)} \lambda_L^{[1]}(m, \sigma).$$

Nótese que $\Lambda_L^{[1]}(\sigma) > 0$ siempre que $N(\sigma) > 0$, ya que al menos para un valor de m, es $L(m \mid \sigma) > 0$. La distribución para las etiquetas se define

$$p^{[1]}(\sigma) = \begin{cases} \frac{N(\sigma)}{N}\left(1 - \lambda^{[1]}(\sigma)\right) & \text{si } N(\sigma) > 0 \\ \frac{\Lambda^{[1]}}{|\Sigma| - C} & \text{si } N(\sigma) = 0 \end{cases} \tag{6.19}$$

donde

$$\Lambda^{[1]} = \sum_{\sigma \in \Sigma} \frac{N(\sigma)}{N} \lambda^{[1]}(\sigma).$$

Nótese que si la muestra no es trivial ($N > 0$), entonces $N(\sigma) > 0$ para algún σ y por tanto $\Lambda^{[1]} > 0$. De las consideraciones anteriores, se deduce que $M^{[1]}$ no asigna nunca probabilidades nulas. El significado de los símbolos es idéntico al de la sección 6.2.1.

A continuación, pasamos a definir $M^{[2]}$. Para todo 2-subtree $t = \sigma(\sigma_1, \sigma_2, \cdots, \sigma_m)$, definimos la probabilidad de la raíz

$$r^{[2]}(\sigma) = \begin{cases} \frac{C_r^{[2]}(\sigma)}{|S|}\left(1 - \lambda_r^{[2]}(\sigma)\right) & \text{si } C_r^{[2]}(\sigma) > 0 \\ \frac{\Lambda_r^{[2]}}{E_r^{[2]}} p^{[1]}(\sigma) & \text{en caso contrario} \end{cases} \tag{6.20}$$

donde

$$\Lambda_r^{[2]} = \sum_{\sigma\in\Sigma: C_r^{[2]}(\sigma)>0} \frac{C_r^{[2]}(\sigma)}{|S|}\,\lambda_r^{[2]}(\sigma)$$

y la normalización $E_r^{[2]}$ es

$$E_r^{[2]} = 1 - \sum_{\sigma\in\Sigma: C_r^{[2]}(\sigma)>0} p^{[1]}(\sigma)\,. \tag{6.21}$$

Dado que hemos justificado que $p^{[1]}(\sigma) > 0$ y por otro lado, $\Lambda_r^{[2]}$ no puede ser cero, podemos concluir que $r^{[2]}(\sigma) > 0$ para todos los símbolos.

Se define para las transiciones, siendo $t = \sigma(\sigma_1, \sigma_2, \cdots, \sigma_m)$

$$p_m^{[2]}(\sigma, \sigma_1, \sigma_2, \cdots, \sigma_m) = \begin{cases} \dfrac{C^{[2]}(t)}{C^{[1]}(\sigma)}\left(1 - \lambda^{[2]}(t)\right) & \text{si } C^{[2]}(t) > 0 \\ \dfrac{\Lambda^{[2]}(\sigma)}{E^{[2]}(\sigma)}\,p_L(m \mid \sigma)\displaystyle\prod_{j=1}^{m} p^{[1]}(\sigma_j) & \text{en caso contrario} \end{cases} \tag{6.22}$$

donde

$$\Lambda^{[2]}(\sigma) = \sum_{t: C^{[2]}(t)>0 \,\wedge\, r_1(t)=\sigma} \frac{C^{[2]}(t)}{C^{[1]}(\sigma)}\,\lambda^{[2]}(t)$$

y la normalización

$$E^{[2]}(\sigma) = 1 - \sum_{m: L(m|\sigma)>0} p_L(m \mid \sigma) \sum_{\substack{\sigma_1,\cdots,\sigma_m: \\ C^{[2]}(\sigma(\sigma_1,\cdots,\sigma_m))>0}} \prod_{i=1}^{m} p^{[1]}(\sigma_i)\,. \tag{6.23}$$

De nuevo, es sencillo justificar que $p_m^{[2]}$ es siempre mayor que cero, ya que $\Lambda^{[2]}(\sigma) > 0$.

Ahora detallamos las ecuaciones correspondientes a $M^{[k]}$ para $k > 2$. Por brevedad en la notación, definimos el conjunto

$$\Sigma_k^T = \left\{ t \in \Sigma^T \mid \operatorname{depth}(t) < k \right\}$$

de todos los árboles con profundidad menor que k

Definimos para todo $t = \sigma(t_1, \cdots, t_m) \in \Sigma_{k-1}^T$

$$r^{[k]}(t) = \begin{cases} \dfrac{C_r^{[k]}(t)}{|S|}\left(1 - \lambda_r^{[k]}(t)\right) & \text{si } C_r^{[k]}(t) > 0 \\ \dfrac{\Lambda_r^{[k]}}{E_r^{[k]}}\, r^{[k-1]}\left(r_{k-2}(t)\right) p_m^{[k-1]}(\sigma, t_1, \cdots, t_m) & \text{en caso contrario} \end{cases} \tag{6.24}$$

donde

$$\Lambda_r^{[k]} = \sum_{u \in \Sigma_k^T : C_r^{[k]}(u) > 0} \frac{C_r^{[k]}(u)}{|S|} \lambda_r^{[k]}(u) \tag{6.25}$$

y

$$E_r^{[k]} = 1 - \sum_{\sigma \in \Sigma} \sum_{m=0}^{M} \sum_{\substack{t_1, \cdots, t_m : \\ C_r^{[k]}(\sigma(t_1, \cdots, t_m)) > 0}} r^{[k-1]}\left(r_{k-2}\left(\sigma(t_1, \cdots, t_m)\right)\right) p_m^{[k-1]}(\sigma, t_1, \cdots, t_m) \tag{6.26}$$

donde M es la ariedad máxima de las muestras de entrenamiento.

Por otro lado, para todo $t = \sigma(t_1, t_2, \ldots, t_m) \in \Sigma_k^T$ definimos

$$p_m^{[k]}(\sigma, t_1, t_2, \ldots, t_m) = \begin{cases} \dfrac{C^{[k]}(t)}{C^{[k-1]}(r_{k-1}(t))}\left(1 - \lambda^{[k]}(t)\right) & \text{si } C^{[k]}(t) > 0 \\ \dfrac{\Lambda^{[k]}(r_{k-1}(t))}{E^{[k]}(r_{k-1}(t))}\displaystyle\prod_{j=1}^{m} p_{m_j}^{[k-1]}\left(\overline{t_j}\right) & \text{en caso contrario} \end{cases} \tag{6.27}$$

donde $\overline{t_j}$ representa a $(\sigma_j, t_{1j}, t_{2j}, \ldots, t_{mj})$ si $t_j = \sigma_j(t_{1j}, t_{2j}, \ldots, t_{mj})$

$$\Lambda^{[k]}(t) = \sum_{\substack{u \in \Sigma_k^T : C^{[k]}(u) > 0 \\ r_{k-1}(u) = t}} \frac{C^{[k]}(u)}{C^{[k-1]}(r_{k-1}(u))} \lambda^{[k]}(u) \tag{6.28}$$

y

$$E^{[k]}(\sigma(t_1, t_2, \ldots, t_m)) = 1 - \sum_{\substack{u_1, \ldots, u_m \in \Sigma_{k-1}^T : \\ r_{k-1}(u_i) = t \wedge C^{[k]}(\sigma(u_1, \ldots, u_m)) > 0}}^{[k-1]} \prod_{i=1}^{m} p_{m_j}^{[k-1]}\left(\overline{u_j}\right). \tag{6.29}$$

Nótese que en las fórmulas anteriores se dan dos posibilidades:

1. el contexto actual suministra la probabilidad de una determinada transición t;

2. el contexto actual no puede suministrar la probabilidad de la transición t, en cuyo caso, proporciona la probabilidad Λ del *escape* y se pasa a un contexto en $M^{[k-1]}$. Pero esta caída se debe normalizar ya que sólo se realizará para las transiciones t que pudiendo pertenecer al contexto actual su $C^{[k]}(t) = 0$. Por lo que la probabilidad resultante cuando se caiga de modelo deberá dividirse por el complementario de la reglas que el contexto $k-1$ nunca evaluará realmente, es decir, 1 menos la suma de probabilidades de las reglas cuyo $C^{[k]}(t) > 0$ en el contexto

actual, evaluadas por el modelo $M^{[k-1]}$.

En el ejemplo que se va a mostrar a continuación, se han calculado la probabilidad Λ de los *escapes,* ε, a partir de los valores menos vistos por estado, de acuerdo con trabajos previos para análisis de cadenas como el modelo PPMX (Witten and Bell 1991). La única diferencia es que en vez de usar las *hapax legomena*, reglas con una única aparición, hemos tomado las reglas que aparecen sólo n veces siendo n el mínimo en un contexto. Además se ha aplicado la forma de descuento lineal (Katz 1987; Jelinek 1990).

Debemos destacar que hemos utilizado diferentes aproximaciones al cálculo de estas probabilidades Λ como las basadas en PPMC (Moffat 1990), PPMD (Howard 1993) y el modelo de descuento del valor absoluto de (Ney and Essen 1993) sin diferenciarse significativamente en la tasa de aciertos, por lo que se ha optado por exponer en detalle una sola de las aproximaciones realizadas.

Las expresiones utilizadas en esta aproximación son de tipo siguiente:

- $C^{[k]}(t)$ es el número de apariciones de la regla t en la muestra,
- $n = \min\left\{C^{[k]}(t) \,:\, r_{k-1}(t) = q\right\}$,
- $R(q) = \left\{t : r_{k-1}(t) = q \,\wedge C^{[k]}(t) = n\right\}$,
- $N(q)$ es el número de reglas en el estado q cuyo número de apariciones es el mínimo n, esto es $N(q) = |R(q)|$

$$\lambda(q) = \frac{\sum_{t=\sigma(t_1,\ldots,t_m)\in R(q)} p_m(\sigma, t_1, \ldots, t_m)}{N(q)+1} \tag{6.30}$$

Seguiremos con el ejemplo de las figuras 6.1 para la clase A y 6.2 para la clase B. Hay que remarcar que ahora tenemos tres modelos distintos para la clase A y otros tres para la clase B, como se muestra en las figuras (6.5 y 6.6)

σ	$p^{[1]}(\sigma)$	μ_σ	m	$p_L(m \mid \sigma)$
ε	$\Lambda_r^{[1]} = 0$			
a	$5/10$	1.2	ε 0 1 2	$\Lambda_r^{[1]} = 0.1$ $1/5\,(1-0.1)$ $2/5\,(1-0.1)$ $2/5\,(1-0.1)$
b	$5/10$	0.4	ε 0 2	$\Lambda^{[1]}(b) = 0.1$ $4/5\,(1-0.1)$ $4/5\,(1-0.1)$

(a)

q	$r^{[2]}(q)$	$\sigma(t_1,\ldots,t_m)$	$p_m^{[2]}(\sigma, t_1,\ldots,t_m)$
ε	$\Lambda_r^{[2]} = 0.5$		
a	$2/2\,(1-0.5)$	ε $a(ab)$ $a(b)$ a	$\Lambda_r^{[2]} = 0.1$ $2/5\,(1-0.1)$ $1/5\,(1-0.1)$ $2/5\,(1-0.1)$
b	0	ε b $b(ab)$	$\Lambda^{[2]}(b) = 0.1$ $4/5\,(1-0.1)$ $1/5\,(1-0.1)$

(b)

q	$r^{[3]}(q)$	$\sigma(t_1,\ldots,t_m)$	$p_m^{[3]}(\sigma, t_1,\ldots,t_m)$
ε	$\Lambda_r^{[3]} = 0.5$		
a	0	a	$1/1$
$a(b)$	0	ε $a(b)$	$\Lambda^{[3]}(a(b)) = 0.5$ $2/2\,(1-0.5)$
$a(ab)$	$2/2\,(1-0.5)$	ε $a(a(b)b)$ $a(a(b)b(ab))$	$\Lambda^{[3]}(a(ab)) = 0.3333$ $1/2\,(1-0.3333)$ $1/2\,(1-0.3333)$
b	0	b	$4/4$
$b(ab)$	0	ε $b(ab)$	$\Lambda^{[3]}(b(ab)) = 0.5$ $1/1\,(1-0.5)$

(c)

Figura 6.5: AAADE correspondientes a la clase A (figura 6.1): (a) $M_A^{[1]}$ con $\mu = 0.8$; (b) $M_A^{[2]}$; (c) $M_A^{[3]}$.

σ	$p^{[1]}(\sigma)$	μ_σ	m	$p_L(m \mid \sigma)$
ε	$\Lambda_r^{[1]} = 0$			
a	$4/10$	0.5	ε	$\Lambda^{[1]}(a) = 0.125$
			0	$3/4\,(1 - 0.125)$
			2	$1/4\,(1 - 0.125)$
b	$6/10$	1	ε	$\Lambda^{[1]}(b) = 0.3333$
			0	$3/6\,(1 - 0.3333)$
			2	$3/6\,(1 - 0.3333)$

(a)

q	$r^{[2]}(q)$	$\sigma(t_1, \ldots, t_m)$	$p_m^{[2]}(\sigma, t_1, \ldots, t_m)$
ε	$\Lambda_r^{[2]} = 0.5$		0.0333
a	0	ε	$\Lambda^{[2]}(a) = 0.125$
		$a(ab)$	$1/4\,(1 - 0.125)$
		a	$3/4\,(1 - 0.125)$
b	$2/2\,(1 - 0.5)$	ε	$\Lambda^{[2]}(b) = 0.3333$
		b	$3/6\,(1 - 0.3333)$
		$b(ab)$	$3/6\,(1 - 0.3333)$

(b)

q	$r^{[3]}(q)$	$\sigma(t_1, \ldots, t_m)$	$p_m^{[3]}(\sigma, t_1, \ldots, t_m)$
ε	$\Lambda_r^{[3]} = 0.5$		
a	0	a	$3/3$
$a(ab)$	0	ε	$\Lambda^{[3]}(a(ab)) = 0.5$
		$a(ab)$	$1/1\,(1 - 0.5)$
b	0	b	$3/3$
$b(ab)$	$2/2\,(1 - 0.5)$	ε	$\Lambda^{[3]}(b)) = 0.1667$
		$b(ab)$	$2/3\,(1 - 0.1667)$
		$b(a(ab)b(ab))$	$1/3\,(1 - 0.1667)$

(c)

Figura 6.6: AAADE correspondientes a la clase B (figura 6.2): (a) $M_B^{[1]}$ con $\mu = 0.8$; (b) $M_B^{[2]}$; (c) $M_B^{[3]}$.

respectivamente, ya que la técnica que estamos aplicando es con $k_{max} = 3$. Volvamos a calcular las probabilidades para el árbol $t = a(a(b))$ con el modelo PPM para árboles descrito anteriormente.

Los cálculos para la clase A quedarían como sigue (en los cálculos que se presentan a continuación se omitirá en las funciones cualquier referencia a la clase A para no sobrecargar los subíndices o superíndices)

$$\begin{aligned} p\left(t \mid A\right) &= p^{[3]}\left(t \mid A\right) = r^{[3]}\left(a(a)\right)\, \pi^{[3]}\left(a(a(b))\right) \\ &= 4.491 \times 10^{-3} \end{aligned} \tag{6.31}$$

Ampliando los términos,

$$\begin{aligned} r^{[3]}\left(a(a)\right) &= \frac{\Lambda_r^{[3]}}{E_r^{[3]}}\, r^{[2]}\left(a\right)\, p_1^{[2]}\left(a, a\right) \\ &= 0.0499 \end{aligned} \tag{6.32}$$

y

$$\begin{aligned} \pi^{[3]}\left(a(a(b))\right) &= p_1^{[3]}\left(a, a(b)\right)\, p_1^{[3]}\left(a, b\right)\, p_0^{[3]}\left(b\right) \\ &= 0.09 \end{aligned} \tag{6.33}$$

Si ahora calculamos la probabilidad para la clase B quedaría como sigue (de nuevo se omitirá cualquier referencia a la clase B para no sobrecargar los subíndices o superíndices)

$$\begin{aligned} p\left(t \mid B\right) &= p^{[3]}\left(t \mid B\right) = r^{[3]}\left(a(a)\right)\, \pi^{[3]}\left(a(a(b))\right) \\ &= 0.01 \times 10^{-3} \end{aligned} \tag{6.34}$$

Ampliando términos

$$\begin{aligned} r^{[3]}\left(a(a)\right) &= \frac{\Lambda_r^{[3]}}{E_r^{[3]}}\, r^{[2]}\left(a\right)\, p_1^{[2]}\left(a, a\right) \\ &= 1.839 \times 10^{-3} \end{aligned} \tag{6.35}$$

y

$$\begin{aligned} \pi^{[3]}\left(a(a(b))\right) &= p_1^{[3]}\left(a, a(b)\right) p_1^{[3]}\left(a, b\right) p_0^{[3]}\left(b\right) \\ &= 8.9475 \times 10^{-3} \end{aligned} \tag{6.36}$$

Como podemos comprobar de nuevo, el árbol $t = a(a(b))$ pertenecería a la clase A, ya que, $p\left(t \mid A\right) > p\left(t \mid B\right)$.

Observando la forma en que se han realizado los cálculos, ecuaciones (6.31) y (6.34), hay que remarcar que en la segunda ecuación ha caído en más ocasiones de modelo y ha utilizado más parámetros λ, lo que justifica en gran medida que la probabilidad final ha sido menor.

6.3 Clasificación no probabilística

En este apartado se definirá una distancia entre un árbol y un AAADE que sirva para valorar cuán distinto es el árbol respecto del autómata. La medida se basa en distancia de edición entre árboles y compara las reglas que pertenecen al conjunto Δ del AAADE.

El cálculo de la distancia de edición se puede acelerar usando técnicas de programación dinámica, disminuyendo su coste temporal, (Zhang and Shasha 1989; Oommen et al. 1996) con lo que se obtiene una complejidad de

$$\mathcal{O}\left(|T_1| \times |T_2| \times \min\left(\text{depth}\left(T_1\right),\ \text{leaves}\left(T_1\right)\right) \times \min\left(\text{depth}\left(T_2\right),\ \text{leaves}\left(T_2\right)\right)\right)$$

donde T_1 y T_2 son los árboles comparados. En nuestro caso compararemos árboles cuya profundidad es como máximo k por lo que la complejidad queda como

$$\mathcal{O}\left(|T_1| \times |T_2|\right)$$

La función de distancia $d\left(t, A\right)$ entre un árbol y un AAADE la definiremos

de la siguiente forma

$$d(t,A) = \min_{\alpha \in Q: r(\alpha) > 0} \{d(\alpha, r_{k-1}(t))\} + \sum_{\beta \in R_t} \min_{\alpha \in R_A} \{d(\alpha,\beta)\} \times C(\beta,t) \quad (6.37)$$

donde:

- $A = (Q, \Sigma, \Delta, P, r)$ es el AAADE correspondiente a la clase;
- R_A es el conjunto de reglas de A, es decir, $f_k(S) \bigcup s_{k-1}(S)$;
- R_t es $f_k(t) \bigcup s_{k-1}(t)$;
- $C(\beta, t)$ es el contador de k-forks o $(k-1)$-subtrees isomorfos a β en t;
- $d(\alpha,\beta) : \alpha, \beta \in \Sigma^T$ es la distancia de edición entre dos árboles.

La función $d(t, A)$ devuelve el peso correspondiente a la suma de las mínimas distancias entre las transiciones inferidas por un AAADE con t como patrón a clasificar, y las transiciones correspondientes al autómata A. Cuanto mayor sea este valor mayor será la diferencia entre el árbol t y el autómata A.

Las muestras para inferir los AAADE se encuentran representadas en las figuras 6.1 para la clase A y 6.2 para la clase B, y los AAADE asociados a las clases anteriores se muestran en las tablas 6.1 y 6.2 respectivamente.

Calculemos ahora los valores para cada clase si se suministra como entrada el árbol $t = a(a(b))$ con un peso unitario para las operaciones básicas de la distancia de edición entre árboles (inserción $w_I = 1$, borrado $w_B = 1$ y sustitución $w_{ab} = 1$).

Los cálculos correspondientes a la ecuación (6.37) para la clase A queda-

rían como sigue

$$\begin{array}{lcll} \min_{\alpha\in Q:r(\alpha)>0}\{\alpha, d\,(a(a))\} & = & d\,(a(ab),\, a(a)) & = \; 1 \\ \min_{\alpha\in R_A}\{\alpha,\, d\,(a(a(b)))\}\times 1 & = & d\,(a(a(b)b),\, a(a(b))) & = \; 1 \\ \min_{\alpha\in R_A}\{d\,(\alpha,\, a(b))\}\times 1 & = & d\,(a(b), a(b)) & = \; 0 \\ \min_{\alpha\in R_A}\{d\,(\alpha,\, b)\}\times 1 & = & d\,(b, b) & = \; 0 \\ & & & \\ d\,(t, A) & = & 1+1+0+0 & = \; 2 \end{array} \tag{6.38}$$

Y para la clase B,

$$\begin{array}{lcll} \min_{\alpha\in Q:r(\alpha)>0}\{\alpha,\, d\,(a(a))\} & = & d\,(b(ab),\, a(a)) & = \; 2 \\ \min_{\alpha\in R_A}\{d\,(\alpha,\, a(a(b)))\}\times 1 & = & d\,(a(ab),\, a(a(b))) & = \; 2 \\ \min_{\alpha\in R_A}\{d\,(\alpha,\, a(b))\}\times 1 & = & d\,(a(b), a(b)) & = \; 0 \\ \min_{\alpha\in R_A}\{d\,(\alpha,\, b)\}\times 1 & = & d\,(b, b) & = \; 0 \\ & & & \\ d\,(t, B) & = & 2+2+0+0 & = \; 4 \end{array} \tag{6.39}$$

Como podemos comprobar de nuevo, el árbol $t = a(a(b))$ pertenecería a la clase A ya que $d\,(t, A) < d\,(t, B)$. A diferencia de los modelos anteriores que buscaban la máxima verosimilitud de pertenencia a una clase, aquí se busca la mínima diferencia entre el valor de la medida dada entre el árbol y el autómata.

A la vista de los cálculos de las ecuaciones (6.38) y (6.39), se observa que los pesos del estado raíz $a(a)$ son diferentes: en la clase A se obtiene un peso de $d\,(a(a), a(ab)) = 1$, mientras que en la clase B se obtiene un peso de $d\,(a(a), b(ab)) = 2$. Al igual que en la regla $a(a(b))$: en la clase A existe una parecida con un solo cambio $d\,(a(a(b)), a(a(b)b)) = 1$, mientras que la regla más parecida en la clase B necesita dos cambios $d\,(a(a(b)), a(ab)) = 2$, con lo que el valor final de la medida es menor para la clase A.

6.4 Resultados

En este apartado se aplican los modelos descritos en el apartado anterior a una base de datos de caracteres manuscritos aislados que ha sido ampliamente utilizada (Gómez et al. 1995; Micó and Oncina 1998; López and Piñaga 2000; Pérez-Cortés et al. 2000): la NIST SPECIAL DATABASE 3 del National Institutte of Standards and Technology.

6.4.1 Extracción de características

A todas las imágenes binarias utilizadas para el entrenamiento y test se les ha aplicado el algoritmo de Arcelli and di Baja (1985) de esqueletizado de imágenes para eliminar información redundante. Estas imágenes son el punto de partida del proceso de representación en forma de árbol. Para ello, se han seguido Para ello, se han seguido los mismos criterios que en López and Piñaga (2000):

1. Se escoge el pixel más a la izquierda de la imagen y se le asigna la raíz del árbol con una etiqueta especial "@".

2. Cada nodo del árbol tendrá tantos descendientes como vecinos tiene el pixel.

3. Cada rama se extenderá siguiendo su pixel vecino hasta que se cumpla una de las siguientes condiciones:

   (a) la rama actual alcanza la longitud del parámetro establecido *window*=8*;*

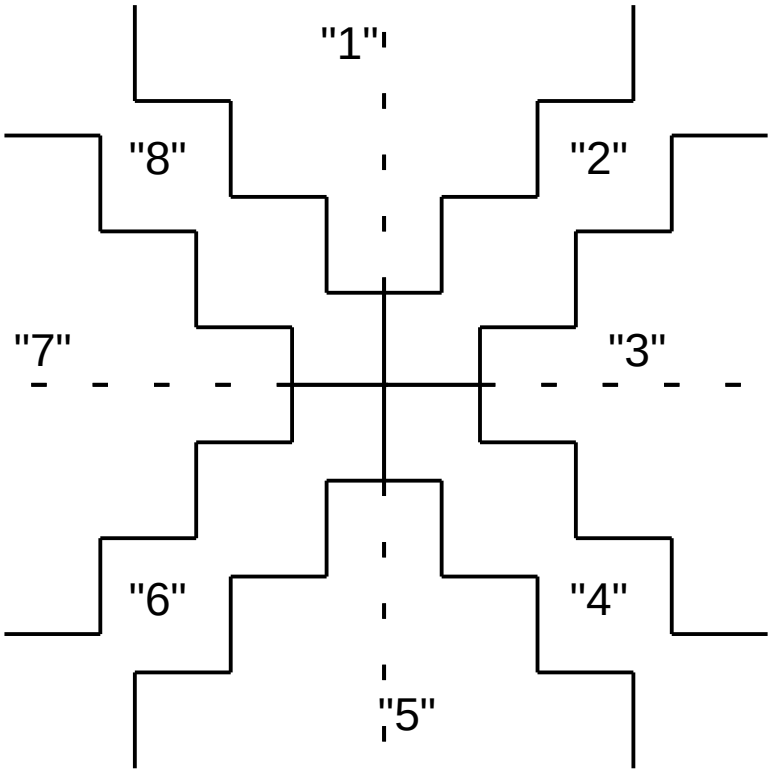
   (b) el pixel no tiene más vecinos (pixel final);

   (c) el pixel tiene más de un vecino (intersección de pixels).

4. Se asigna un nuevo nodo al pixel final del paso 3. El nodo se etiqueta según el esquema de la figura 6.7.

$$\begin{cases} \text{si } x \geq 0 & \begin{cases} y = \pm 2 \lceil x \rceil \\ y = \pm \frac{\lfloor x \rfloor}{2} \end{cases} \\ \text{en caso contrario} & \begin{cases} y = \pm 2 \lceil x - 1 \rceil \\ y = \pm \frac{\lfloor x+1 \rfloor}{2} \end{cases} \end{cases}$$

(a)



(b)

Figura 6.7: (a) ecuaciones para dividir el espacio espacio 2D en 8 regiones; el norte se etiqueta con **"1"** y siguiendo el sentido horario se etiqueta el resto de regiones consecutivamente. Cuando se asigna una etiqueta a un segmento, el pixel inicial se sitúa en el origen de coordenadas; las etiquetas asignadas dependen de la posición relativa de la situación del pixel inicial; (b) representación gráfica

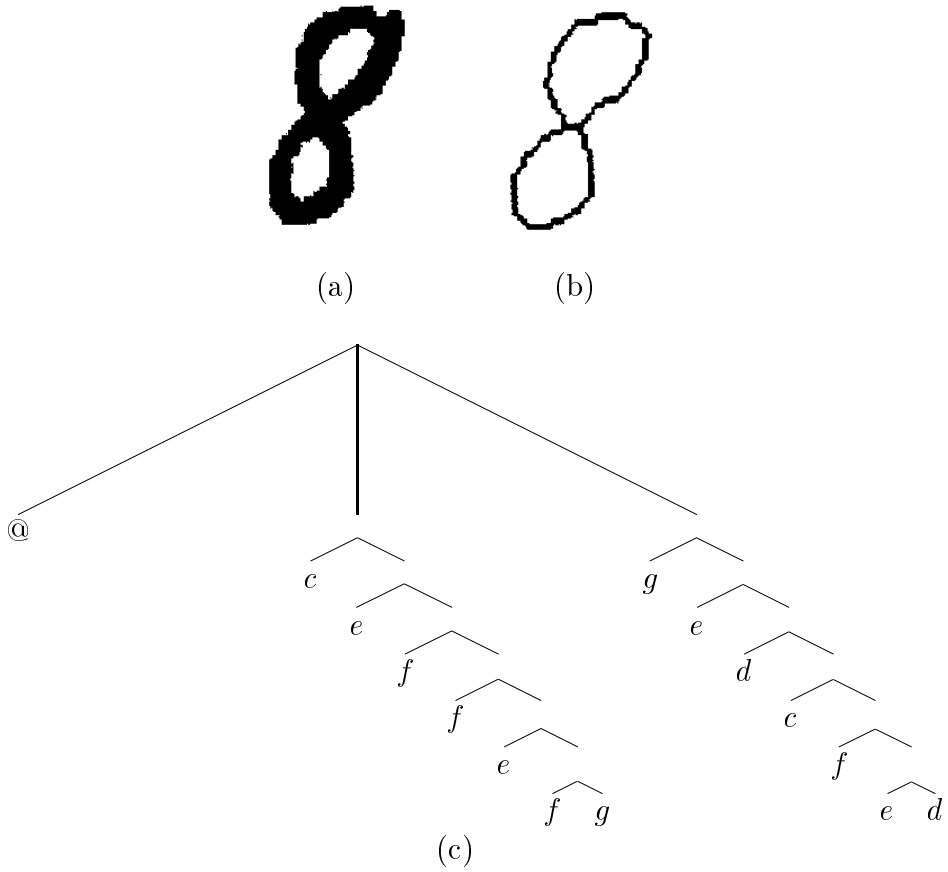


Figura 6.8: Ejemplo de un dígito: (a) Imagen original; (b) Imagen esqueletizada; (c) Árbol etiquetado resultante

5. Para cada nodo con vecinos, se vuelve al paso 2.

Los árboles que se obtienen con este procedimiento tienen etiquetados sus nodos. Para transformar estos árboles en esqueletos sin pérdida de información se aplica el operador siguiente:

$$\begin{aligned} &Sk(a) = a && \forall a \in \Sigma \\ &Sk\left(\sigma\left(t_1, \ldots, t_n\right)\right) = \alpha\left(\sigma, Sk\left(t_1\right), \ldots, Sk\left(t_n\right)\right) && \forall t_1, \ldots, t_n \in \Sigma^T, \sigma \in \Sigma \end{aligned} \tag{6.40}$$

Donde α representa un símbolo especial no contenido en Σ.

Un ejemplo de características se muestra en la figura 6.8.

6.4.2 Análisis de resultados

Los árboles etiquetados que se han usado para realizar los experimentos son los mismos con los que se realizaron las pruebas de clasificación de López and Piñaga (2000) donde se aplicaba la definición de una distancia de edición entre un árbol etiquetado y un autómata determinista de árboles (López et al. 2000) cuyos resultados fueron de un 80% de precisión.

Las pruebas que se han realizado con nuestros modelos se han estructurado en 5 subconjuntos de tamaño 1000, 2000, 3000, 4000 y 5000, utilizándose para cada uno de los subconjuntos el 75% de las muestras para entrenamiento y el 25% restante para test.

Se han realizado varias pruebas con diferentes k: 3, 5 y 7 sobre el método de clasificación no probabilístico obteniéndose los resultados mostrados en la figura 6.9. El mejor de estos resultados obtenidos, $k = 7$, será el representativo de este modelo. Los resultados obtenidos por los modelos descritos en este capítulo se comparan, adicionalmente, con el método de clasificación por distancia entre árboles detallado en apartado 8.3. Como se puede observar en la tabla 6.5, el método que consigue menor precisión es el de interpolación de modelos y los que obtienen mejor precisión son el de medida entre

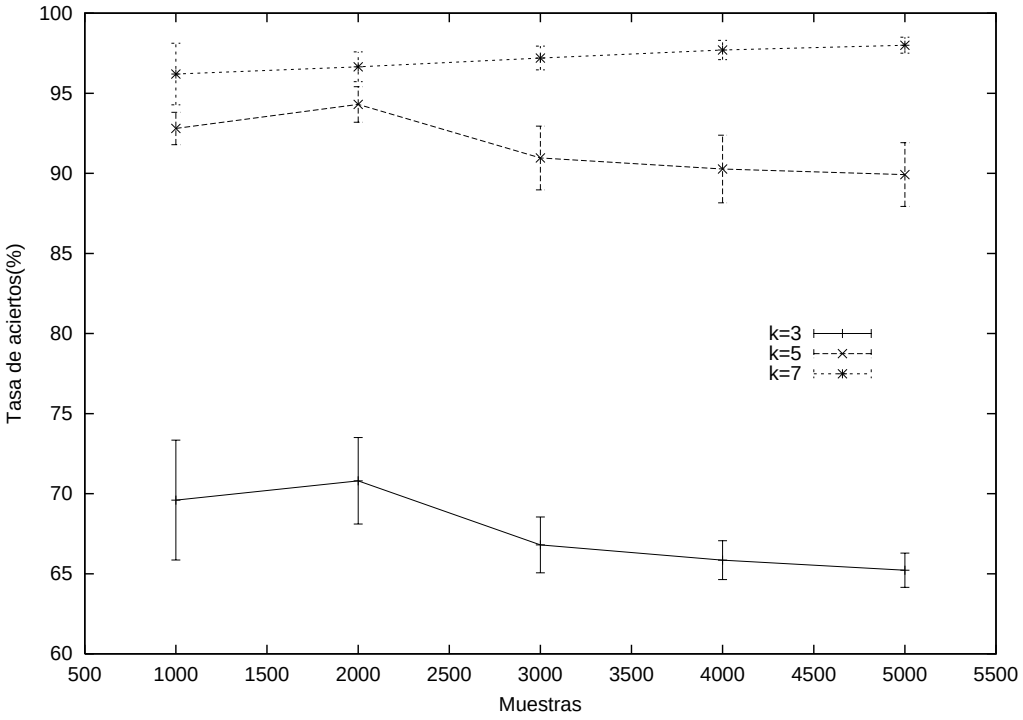


Figura 6.9: Tasa de clasificación para el modelo de clasificación no probabilística.

#	interpolación		distrib. a priori		PPM k=3		medida AA		distancia AA	
muestras	μ	D	μ	D	μ	D	μ	D	μ	D
1000	70.9	3.4	82.7	1.2	82.9	3,5	96.2	1.9	95.0	0.9
2000	77.9	1.2	85.4	1.6	88.1	1.4	96.7	0.9	97.2	0.6
3000	78.5	0.4	85.7	0.5	88.5	0.7	97.2	0.7	96.5	0.6
4000	81.4	0.3	86.0	0.6	98.3	0.5	97.7	0.6	97.3	0.5
5000	82.5	0.2	86.4	0.5	90.6	0.7	98.0	0.5	98.0	0.4

Tabla 6.5: Tasa de clasificación para los modelos de este capítulo donde μ es la media aritmética y D la desviación estándar.

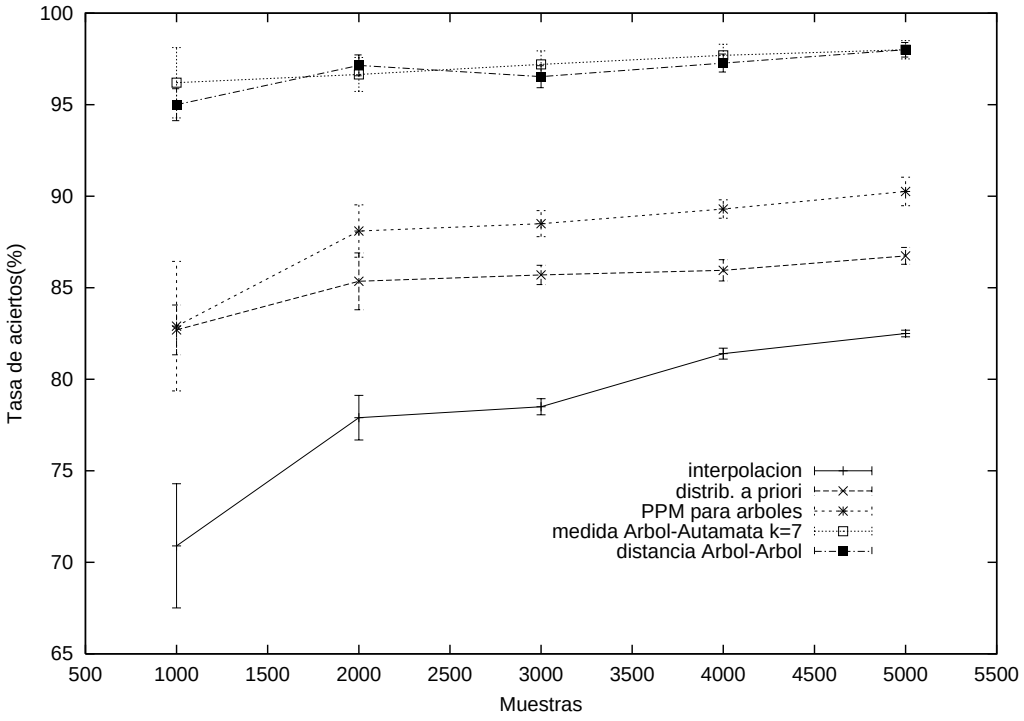


Figura 6.10: Tasa de clasificación para los modelos de este capítulo.

árbol-autómata y el de distancia de edición entre árboles.

6.5 Conclusiones

Tal y como se ha comentado en cada uno de los apartados, los modelos expuestos en este capítulo tienen diferentes características. Si queremos escoger entre los modelos probabilísticos puros, aquellos que consiguen una mayor precisión son el modelo con distribución a priori y el modelo de predicción por concordancia parcial, llegando a obtener una precisión en promedio del 86.7% y del 90.3% respectivamente. Si deseamos escoger entre los modelos no probabilísticos, el que se define en el apartado 6.3 y el basado en distancia de edición entre árboles, apartado 8.3, consiguen una precisión en promedio del 98%. Estos últimos detectan similitudes entre los árboles etiquetados que ayudan a la clasificación y que los anteriores no pueden contemplar.

Aunque lo tasa de aciertos de los métodos basados en distancias supera al de los métodos probabilísticos, los primeros resultan extremadamente lentos para su aplicación en bases de datos extensas o cuando el tiempo de respuesta del sistema es crítico.

Parte II

Otros modelos de árboles

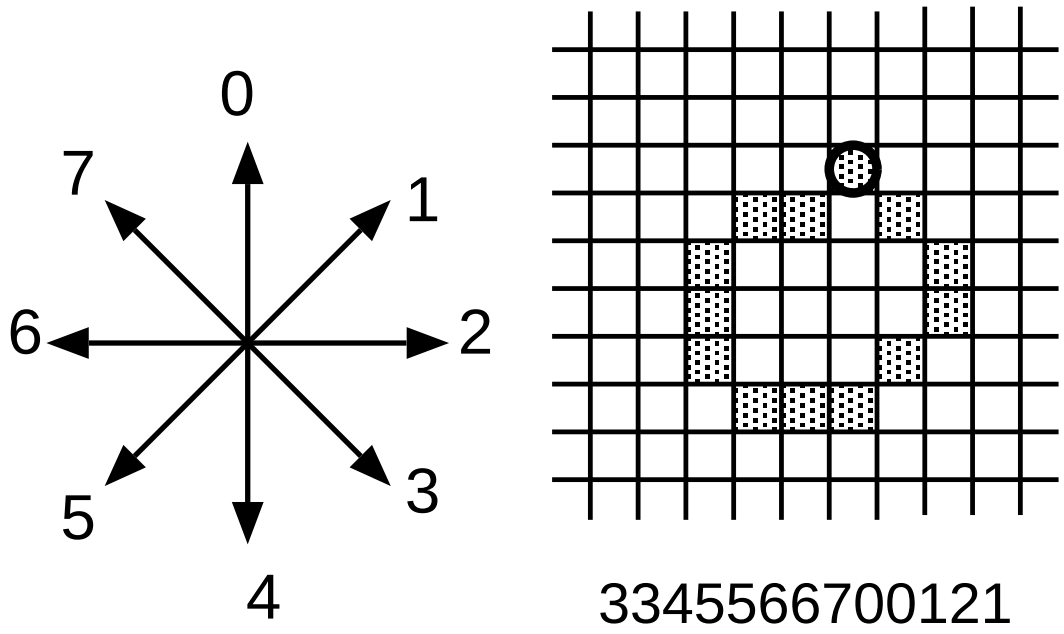
Capítulo 7

Compresión de superficies

En muchas aplicaciones, los objetos se representan como un conjunto de puntos tridimensionales desordenados que se han obtenido a partir de un "escaner". En estos casos, interesa encontrar una forma eficiente de almacenamiento de datos. Este capítulo presenta un esquema de compresión aritmética que usa una representación arbórea del conjunto de datos y permite mejorar los tasas de compresión respecto a los métodos de propósito general.

7.1 Introducción

Una gran área de investigación actual es la compresión eficiente de imágenes, especialmente de imágenes bidimensionales e imágenes de vídeo. Algunos trabajos previos tratan la compresión de imágenes en tres dimensiones (Cochran et al. 1996). En algunos casos se han estudiado aplicaciones médicas (Ihm and Park 1998) en las que toda la información, incluyendo la relativa a puntos interiores del objeto, es relevante. Sin embargo, existen casos (por ejemplo, en el diseño industrial asistido por ordenador) en los que sólo nos interesa la información exterior del objeto, esto es, la descripción de su superficie. En la mayoría de estos casos, los datos se obtienen utilizando un escaner de tres dimensiones que recorre la superficie del objeto y genera como salida



(a) (b)

Figura 7.1: Codificación de una figura en 2D: (a) códigos asignados a las direcciones en el plano; (b) Contorno de una figura y su codificación.

un conjunto de puntos tridimensionales (vectores de tres componentes en un espacio euclídeo). Algunos trabajos precedentes se han centrado en la construcción, a partir del conjunto de puntos tridimensionales, de modelos para describir geométricamente el objeto por medio de superficies (Hoppe et al. 1994), consiguiendo así una codificación eficiente de la imagen. Sin embargo, el coste de estos algoritmos es siempre alto.

En el caso más sencillo de las imágenes en dos dimensiones, el contorno de una figura simple se puede codificar eficientemente con una cadena de símbolos (Freeman 1961). Para este propósito basta con escoger el sentido en la que se codificará figura (horario o antihorario) como se muestra en la figura 7.1. Así, cada punto tiene dos vecinos (un predecesor y un sucesor) y su representación como cadena de símbolos se genera simplemente escribiendo

las posiciones relativas entre puntos consecutivos. Las figuras bidimensionales se describen habitualmente como una colección de *pixels*. Dado que cada pixel tiene 8 vecinos como máximo, un byte es suficiente para describir la posición relativa de un pixel con respecto al anterior. De esta manera, si una figura está simplemente conectada[1], una cadena contiene toda la información necesaria para reconstruir la figura, salvo por una translación global que no es relevante en la mayoría de las aplicaciones. Con este método se consigue una reducción considerable en el tamaño del fichero necesario para guardar la información (Gokmen et al. 1996).

El método descrito anteriormente permite una codificación y descodificación simple y rápida. No obstante, en el caso de imágenes tridimensionales, surgen algunas diferencias importantes:

1. El objeto está limitado por una superficie en vez de por una curva y el número de vecinos de un voxel se incrementa a 26, comparado con los 8 de un pixel.

2. En las configuraciones habituales, la precisión del escaner es muy superior a la distancia mínima entre puntos, por lo que estos dejan de ser adyacentes.

3. Además, un punto en la superficie puede tener más de 2 vecinos, por lo que se debe establecer una prioridad entre los vecinos para determinar un camino. De hecho, puede ser imposible escanear la superficie del objeto con un solo camino que pase una sola vez por cada punto.

El último punto sugiere que una forma natural para describir la superficie es una representación de tipo árbol, tal y como se describe en la sección siguiente.

[1]Una figura simplemente conectada es una figura con un solo borde.

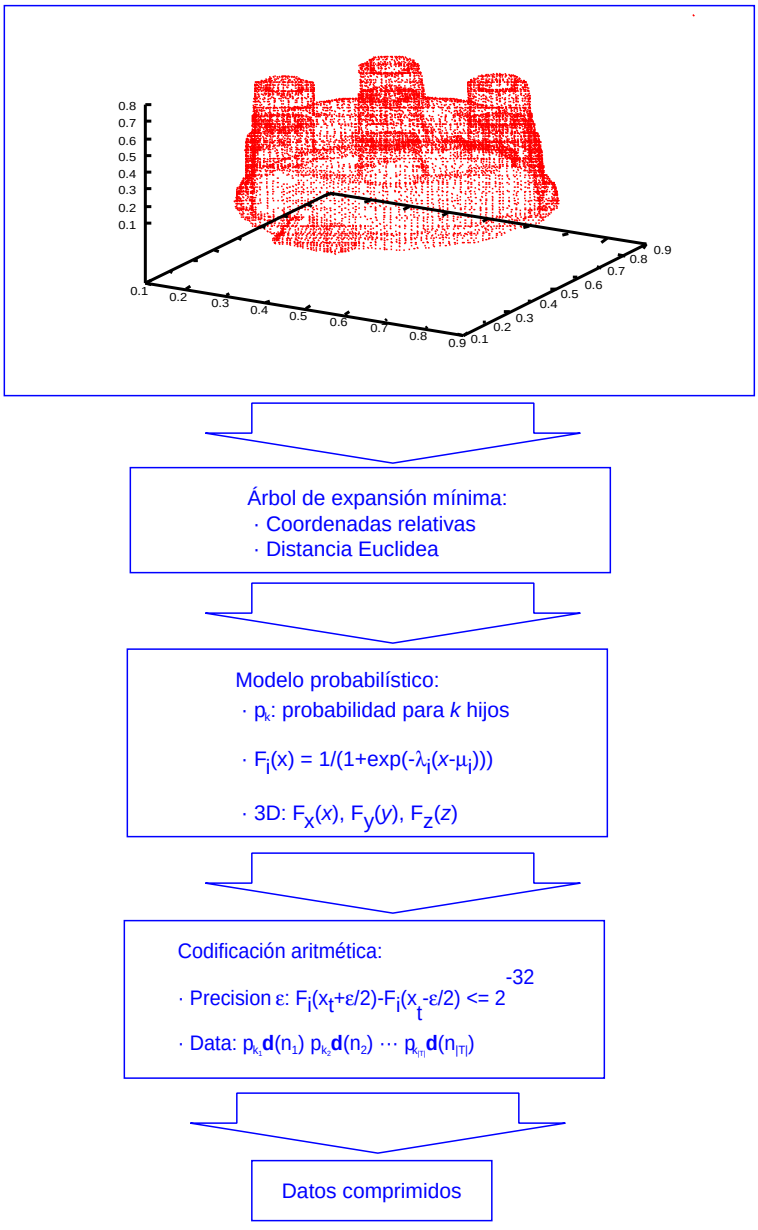


Figura 7.2: Esquema general usado para la compresión.

7.2 Representación de datos y modelado.

El esquema general seguido en este capítulo se muestra en la figura 7.2, en la que podemos ver el diagrama cuyas partes detallaremos a continuación.

Dado un conjunto de vectores tridimensionales $S = \{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_{|S|}\}$ como el de la figura 7.3a, se puede definir un grafo totalmente conectado G en el que el conjunto de nodos es S y los pesos de las aristas $(\mathbf{r}_i, \mathbf{r}_j)$ es la distancia euclídea entre $\mathbf{r}_i$ y $\mathbf{r}_j$. Entonces, se puede construir T (figura 7.3c), el árbol de expansión mínima[2] (MST) de G, usando alguno de los algoritmos estándar (consultar, por ejemplo, Cormen et al. (1990)). Debido a las propiedades[3] geométricas del espacio euclídeo, el máximo número de descendientes de un nodo es 12 (teniendo 12 una probabilidad 0). Sin embargo, en nuestro caso, los puntos están distribuidos sobre una superficie (es decir, localmente en un plano) con lo que el número de vecinos que se encuentran en la práctica es siempre inferior a 6.

Cada nodo n en el MST con padre m es etiquetado con un vector $\mathbf{d}_n$ igual a la diferencia $\mathbf{d}_n = \mathbf{r}_m - \mathbf{r}_n$. Para el nodo raíz p, tomaremos $\mathbf{d}_p = \mathbf{r}_p$. En nuestra aproximación, se comprime la información mediante la compresión aritmética (Witten et al. 1987; Cover and Thomas 1991) de la entrada. Por este motivo, necesitamos modelos estocásticos que describan la estructura del árbol y de los componentes de tipo vector contenidos en T:

1. Con la estructura del árbol se calcularán las probabilidades p_k de que un nodo se expanda en un número de hijos k (figura 7.3b). La probabilidad se estima como el número relativo de subárboles en T que tienen k hijos, donde se asume implícitamente que este valor no depende de la posición del nodo.

2. A partir de los vectores de componentes $\mathbf{d}(n)$, que contienen tres nú-

[2]MST corresponde a la siglas inglesas Minimun Spanning Tree.

[3]Esta cuestión está relacionada con el empaquetamiento de esferas con una densidad óptima.

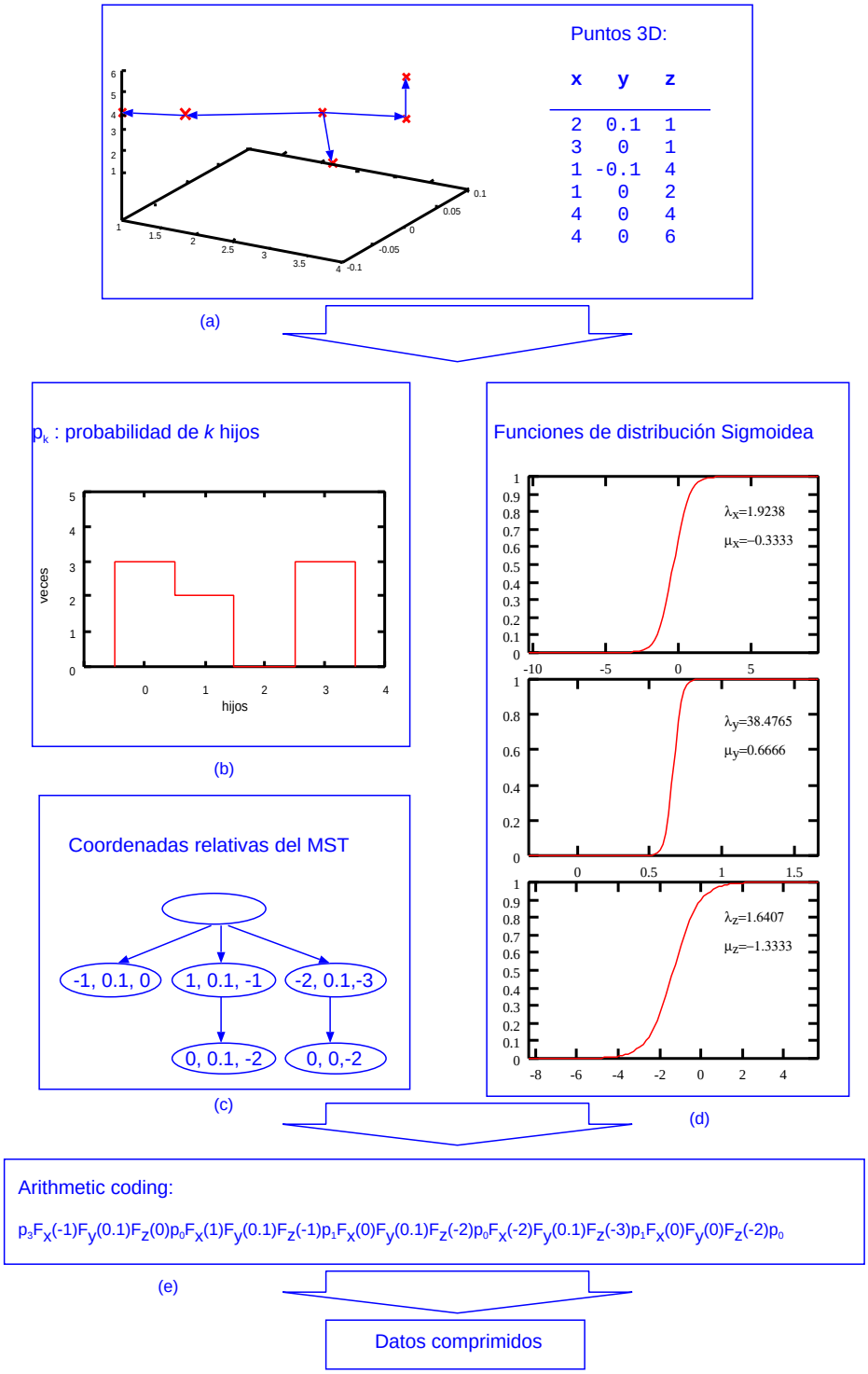


Figura 7.3: Compresión de una figura a partir de puntos tridimensionales. a) Dibujo de los puntos de la figura, el MST y tabla de datos. b) Histograma de la función p_k. c) MST en coordenadas relativas. d) Gráfica de las funciones de distribución de las tres coordenadas e) Codificación del ejemplo.

meros en formato de coma flotante, se calculará tres distribuciones de probabilidad $F_1(x)$, $F_2(y)$ y $F_3(z)$ tal y como se presenta en la figura 7.3d, una para cada componente. Una codificación eficiente requiere que estas distribuciones sean invertibles eficientemente. Por ello, en vez de usar la función de distribución normal, se ha usado la función sigmoidea

$$F_i(x) = \frac{1}{1 + \exp(-\lambda_i (x - \mu_i))} \tag{7.1}$$

cuya función de densidad es $\lambda_i F_i(x)(1 - F_i(x))$. Los parámetros que aparecen el la fórmula anterior son: μ_i (la media aritmética) y $\lambda_i = \pi\sqrt{3}/\sigma_i$, siendo σ_i es la desviación estándar[4].

7.3 Codificación aritmética de los datos

La compresión aritmética se realiza siguiendo un recorrido en preorden de T, codificando cada nodo n como el vector de coordenadas asociado $\mathbf{d}(n)$ y el número de hijos que descienden de éste. Por ejemplo, la codificación que se muestra en la figura 7.3e corresponde al árbol del la figura 7.3c. Por lo tanto, el fichero de salida contiene los datos comprimidos con el modelo y los parámetros de la compresión como cabecera (los parámetros μ_i y λ_i junto con las probabilidades p_k). Entonces, la codificación aritmética se calcula usando consecutivamente los cuatro modelos contenidos en la cabecera. Con el objeto de evitar posibles errores de redondeo durante la descodificación (debido a que se utiliza una aritmética de 32 bits), las colas de la distribución F se tratan de manera especial. Si tenemos un determinado escaner con una

[4]La fórmula de cálculo de λ_i se obtiene al maximizar la esperanza de la función sigmoidea $F_i(x)$.

precisión ε, existe $x_t > 0$ tal que

$$F\left(x_t + \frac{\varepsilon}{2}\right) - F\left(x_t - \frac{\varepsilon}{2}\right) \leq 2^{-32} \tag{7.2}$$

Todo x tal que $|x| \geq x_t$ es considerado una anomalía. En estos casos, que son muy improbables, se genera el código para la región $|x| \geq x_t$ seguido por el número x sin codificar.

7.4 Resultados y discusión

Se han comprimido con con el método descrito previamente (LCS ≡ Lossless Compression of Surfaces) nueve ficheros diferentes con un número variable de datos (entre 4 y 19 mil puntos). Podemos ver el aspecto de cada uno representado en las figuras 7.4, 7.5 y 7.6. Para comparar, se ha utilizado diferentes compresores de propósito general:

1. un compresor del tipo Ziv-Lempel (gzip);

2. un codificador de Huffman avanzado (bzip2, que utiliza la transformada de Burrows-Wheeler y codificación de Huffman) y;

3. un codificador aritmético basado en k-gramas descrito en Nelson (1991).

Los resultados se presentan en la tabla 7.1 y gráficamente se pueden ver en la figura 7.7. Las tasas de compresión se definen como el cociente entre tamaño del fichero original y el tamaño del fichero comprimido.

Como se muestra en la tabla, el método aquí descrito es mejor que otros métodos existentes de propósito general y, permite obtener unas tasas de compresión hasta 5 veces mayores.

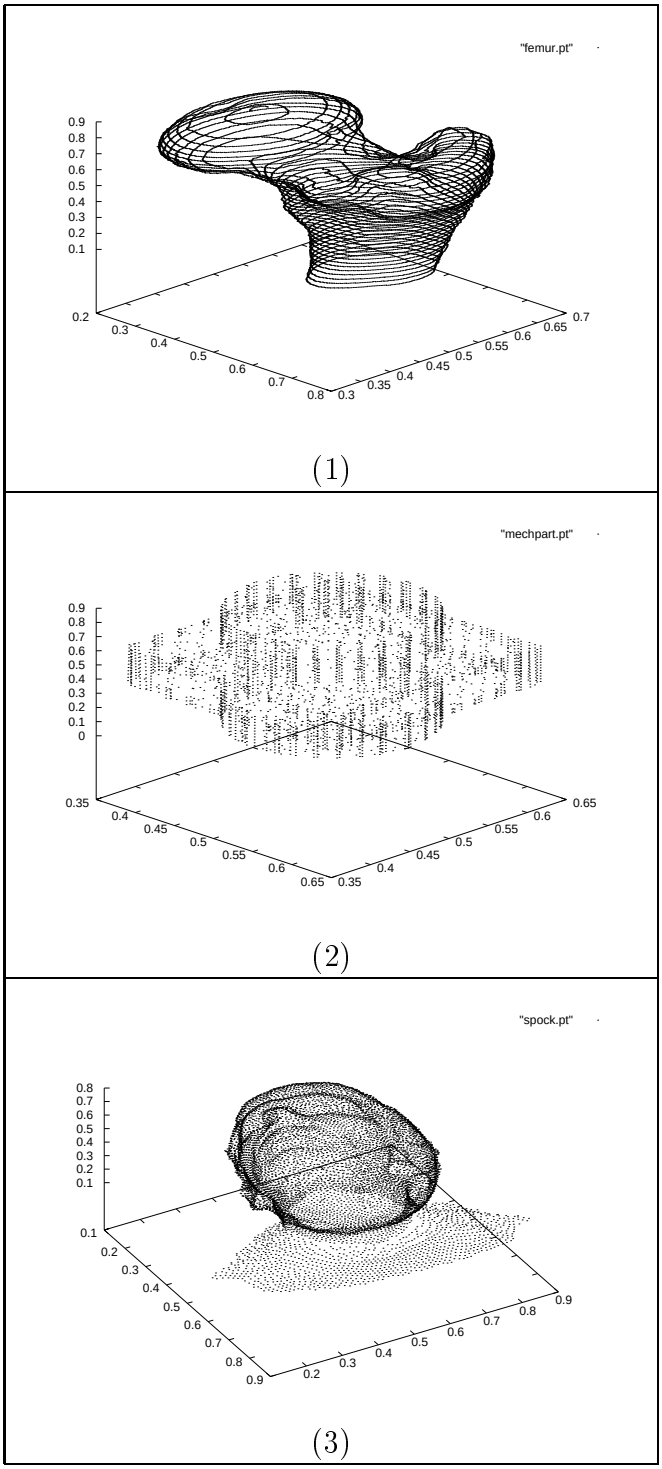
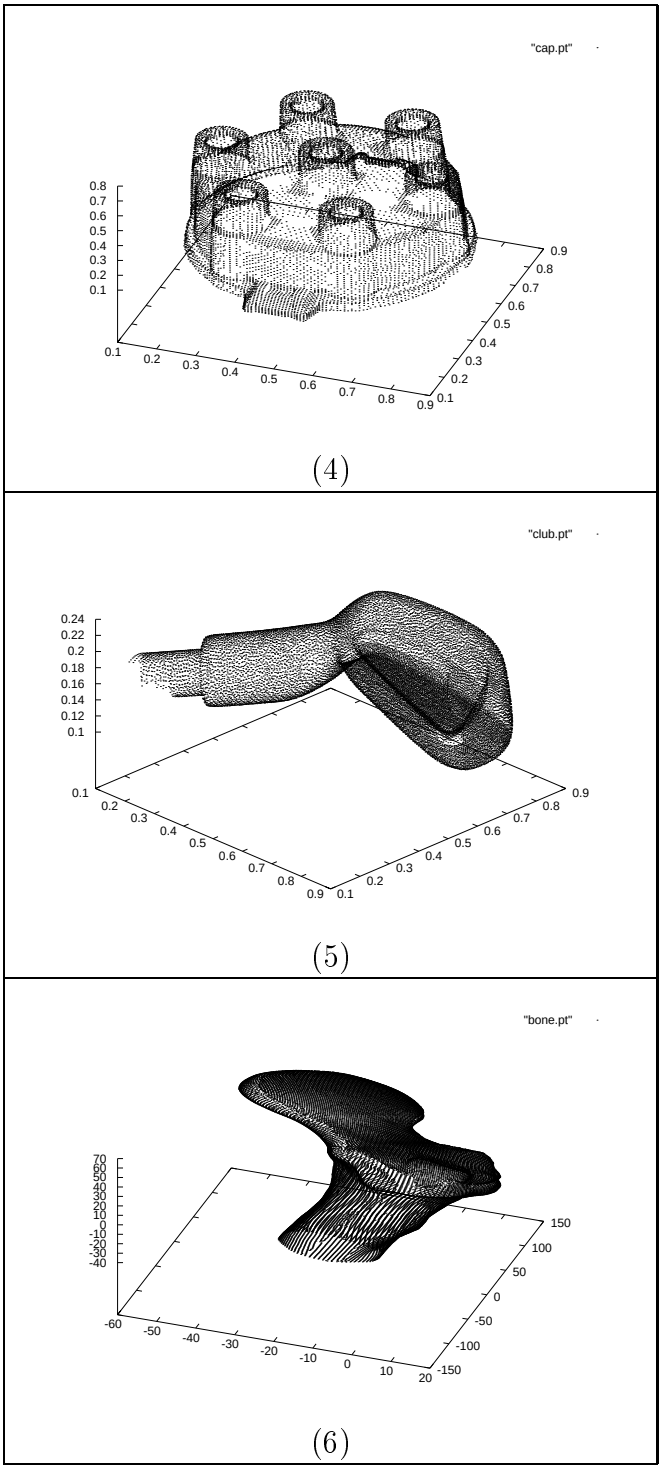


Figura 7.4: Representación gráfica de las tres primeras figuras comprimidas.



(4)

(5)

(6)

Figura 7.5: Representación gráfica de las figuras comprimidas de la 4 a la 6.

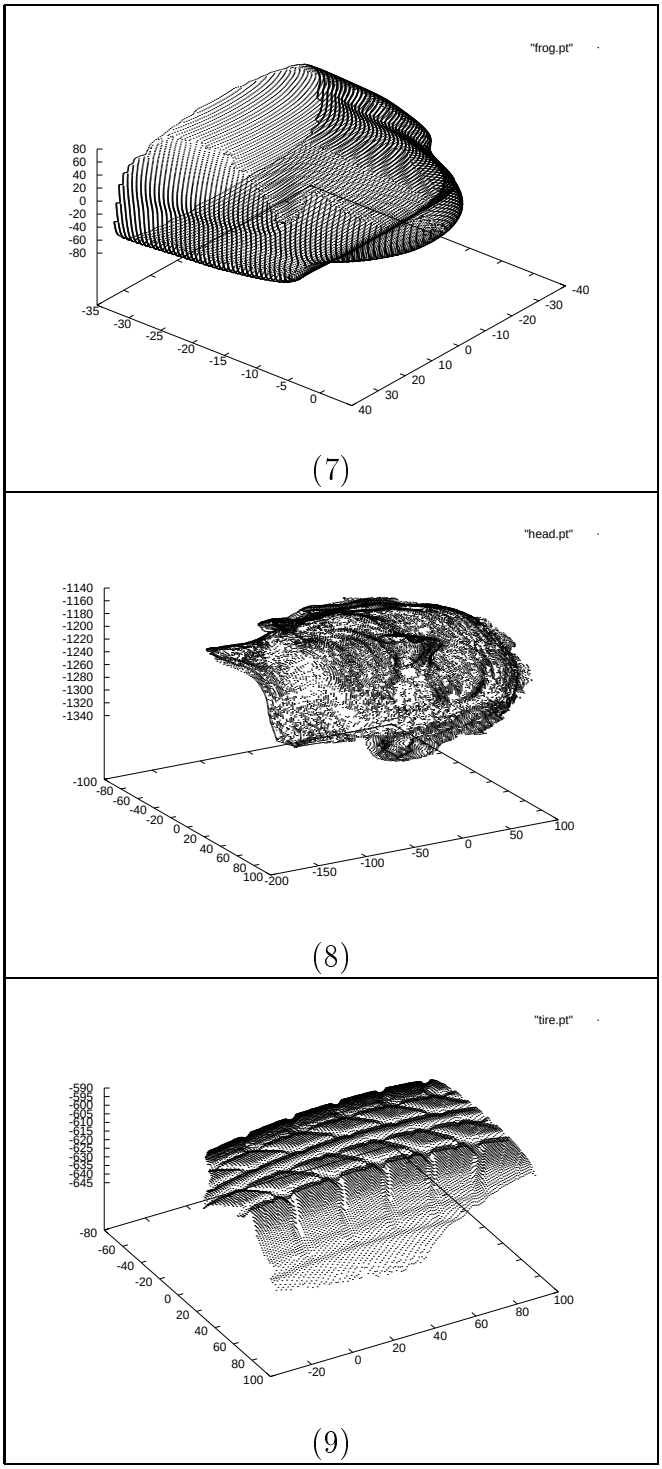


Figura 7.6: Representación gráfica de las figuras comprimidas de la 7 a la 9.

fichero #	gzip	3-gramas	bzip2	LCS	original
1	103.3	78.2	55.2	83.3	484.3
2	391.1	32.8	31.0	23.4	108.8
3	77.8	73.4	69.4	54.8	253.6
4	127.8	110.1	117.0	72.5	339.9
5	162.4	138.6	152.1	88.9	450.0
6	194.7	154.0	157.9	88.1	529.2
7	178.6	142.1	147.8	77.9	519.0
8	147.0	108.9	114.3	63.6	400.0
9	118.2	99.0	96.0	57.3	367.4

Tabla 7.1: Tamaño en Kbytes para 9 conjuntos de datos diferentes.

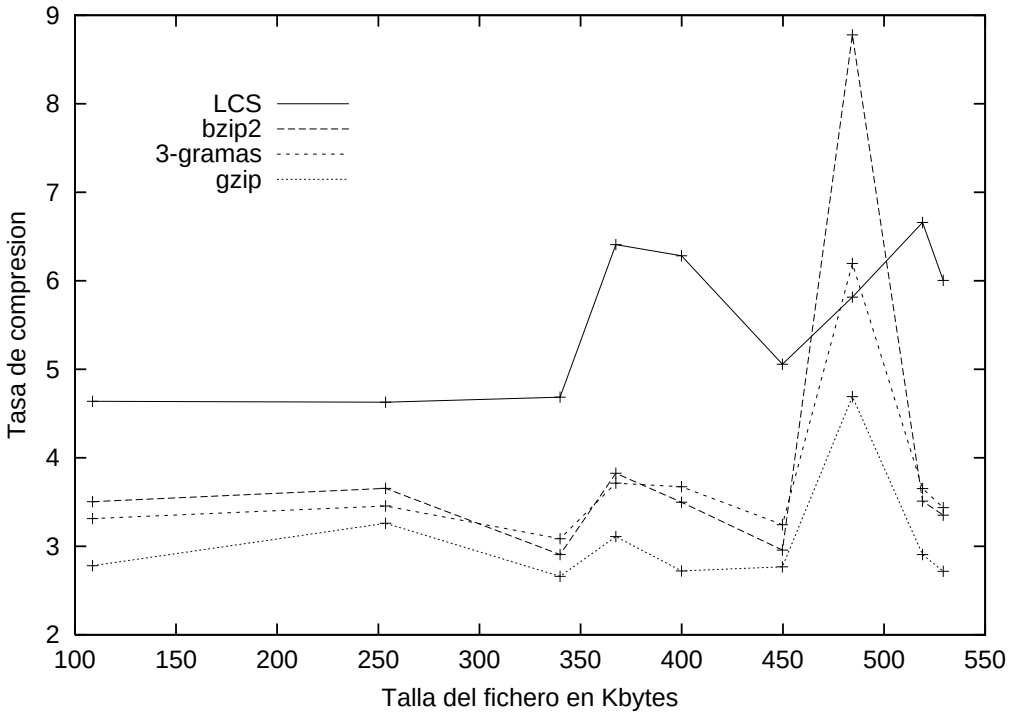


Figura 7.7: Tasas de compresión para 9 conjuntos de datos diferentes. Las líneas sólo sirven para guiar la vista.

7.5 Conclusiones

En este capítulo se ha implementado un codificador aritmético que comprime ficheros de datos que contienen puntos que describen superficies de objetos tridimensionales. El método usa una representación simple de la superficie que consiste en un árbol con las posiciones relativas entre puntos y modeliza esta estructura y sus componentes. La complejidad temporal coincide con la de la construcción estándar del árbol de expansión mínima. Las tasas de compresión son parecidas o significativamente mejores que las obtenidas con métodos generales. Además, se pueden considerar refinamientos sobre el modelo de datos para incrementar la eficiencia del modelo y las tasas de compresión: por ejemplo, usar una función de distribución tridimensional, en vez de las funciones independientes para las tres coordenadas $F_1(x)$, $F_2(y)$ y $F_3(z)$ usadas en esta implementación.

Capítulo 8

Reconocimiento de palabras manuscritas

Este capítulo describe una aproximación geométrica al problema del reconocimiento de palabras manuscritas fuera de línea[1]. Se detalla un nuevo método para extraer, a partir de la imagen de una palabra aislada, características estructuradas en forma de árbol. A partir de ellos, se construye una base de datos de árboles que posteriormente servirá para clasificar nuevas palabras. Para ello se utilizará la distancia de edición entre árboles para calcular el vecino más cercano. La distancia de edición se optimiza para conseguir unos resultados cercanos a un 95% de palabras clasificadas correctamente.

8.1 Introducción

El reconocimiento de palabras manuscritas fuera de línea es un problema interesante por sus aplicaciones prácticas. La mayoría de los sistemas se basan en escanear documentos manuscritos para entrenar un sistema con el estilo de escritura particular de cada escritor y dejar posteriormente que el sistema clasifique el resto del texto. Muchos sistemas (Suen et al. 1980;

[1]La palabra inglesa es off-line

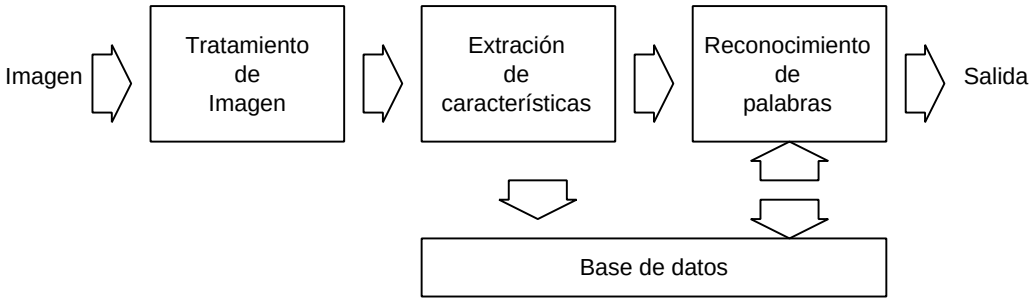


Figura 8.1: Etapas en el proceso de reconocimiento de palabras.

Elliman and Banks 1991; Bunke et al. 1995) reconocen letras aisladas y luego, en una fase posterior, usan un diccionario e información estadística para reconocer las palabras. En este capítulo se pretende clasificar directamente palabras completas. De esta forma, se evita el uso de diccionarios y se integra de una manera más natural el reconocimiento de letras enlazadas formando palabras. Supondremos que disponemos de algoritmos para aislar palabras en un documento como el de Wang et al. (1997).

La idea principal es extraer características de las palabras manuscritas de un corpus y construir una base de datos directamente de las imágenes escaneadas. Las nuevas palabras se comparan con la base de datos creada anteriormente. Debemos remarcar que este método no utiliza diccionario adicional ni algoritmos de división de palabras en letras. El algoritmo extrae las características directamente de la imagen binaria el esquema general se muestra en la figura 8.1.

Este capítulo está estructurado de la siguiente manera: la sección 8.2 describe la forma en la que la imagen de la palabra es tratada y transformada en características con estructura de árbol. La sección 8.3 explica el método de clasificación y la sección 8.4 muestra los resultados experimentales. Finalmente, la sección 8.5 expone las conclusiones.

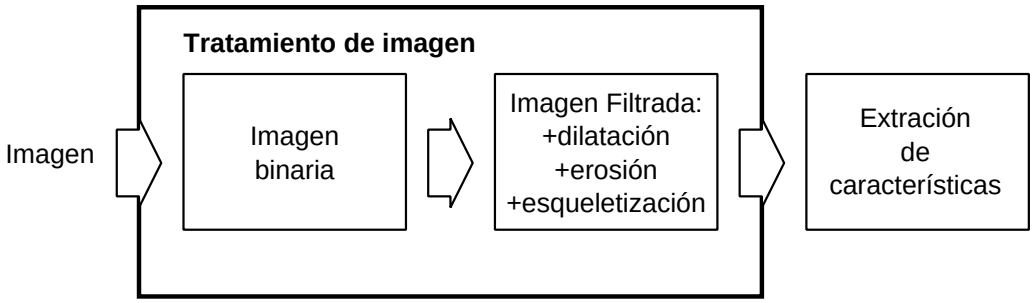


Figura 8.2: Subetapas del proceso del tratamiento de la imagen.

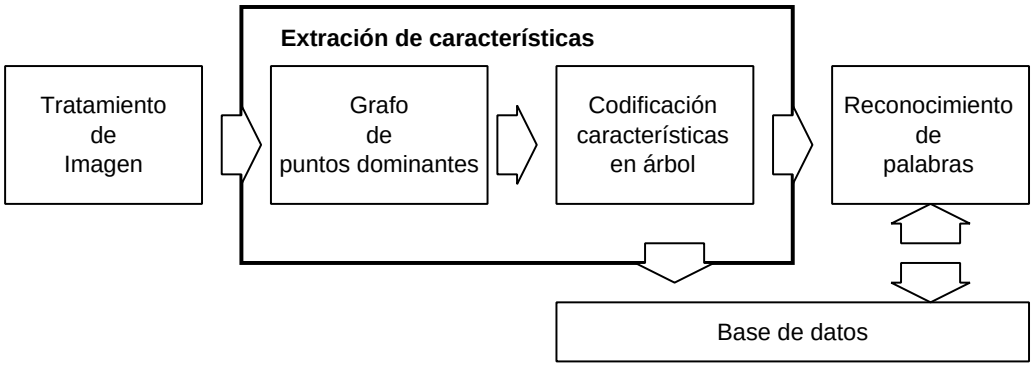


Figura 8.3: Subetapas de la extracción de características.

8.2 Imagen y características

8.2.1 Tratamiento de la imagen

Las imágenes de las que partimos son palabras manuscritas. Normalmente las imágenes originales en reconocimiento fuera de línea se preprocesan (Huang and Yan 1997) para eliminar ruido e información redundante. En nuestro caso, en la etapa de preprocesado se aplica a la imagen filtros de morfología matemática (Serra 1982) y un algoritmo de esqueletización (Carrasco and Forcada 1995) como se muestra en la figura 8.2.

Para cada documento se realizan los siguiente pasos:

1. Se escanea y se obtiene una imagen de grises en la primera fase del proceso.

2. La imagen de grises se transforma en una imagen binaria . Este proceso produce en la imagen resultante un efecto de dientes de sierra[2] en los bordes de la palabra. Por tanto, se usan algunas operaciones de morfología matemática como la dilatación y la erosión (Serra 1982) para suavizar los contornos.

3. Para eliminar los pixels redundantes se utiliza una modificación del algoritmo de esqueletización de Nagendraprasad-Wang-Gupta debida a Carrasco and Forcada (1995).

Los pasos anteriores se pueden ver gráficamente en la figura 8.4.

8.2.2 Extracción del árbol de características

Una vez el contorno de la palabra se ha esqueletizado, se aplican las subetapas de *extracción de características* (figura 8.3) a la imagen binaria. Primero, se extraen los puntos dominantes como en Li and Yeung (1997), Powalka et al.

[2]Se ha traducido de la palabra inglesa *aliasing*

(a)

(b)

(c)

Figura 8.4: Ejemplo sobre el tratamiento de imagen. a) Imagen en escala de grises. b) Imagen binaria. c) Imagen filtrada(dilatada+erosionada+esqueletizada)

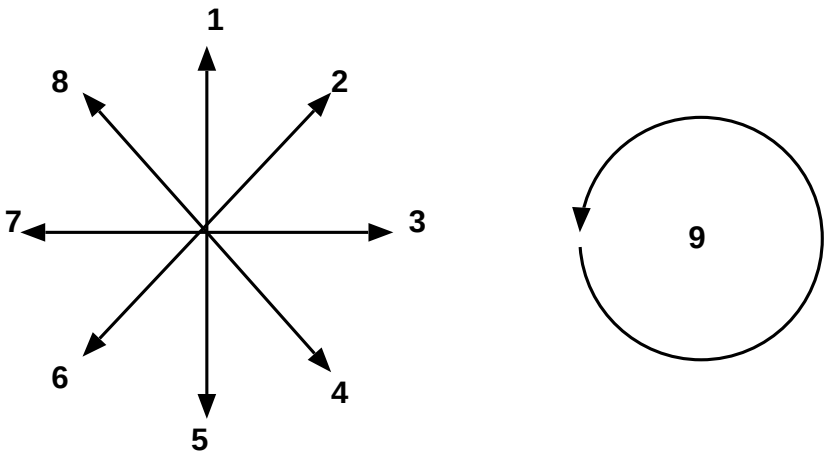
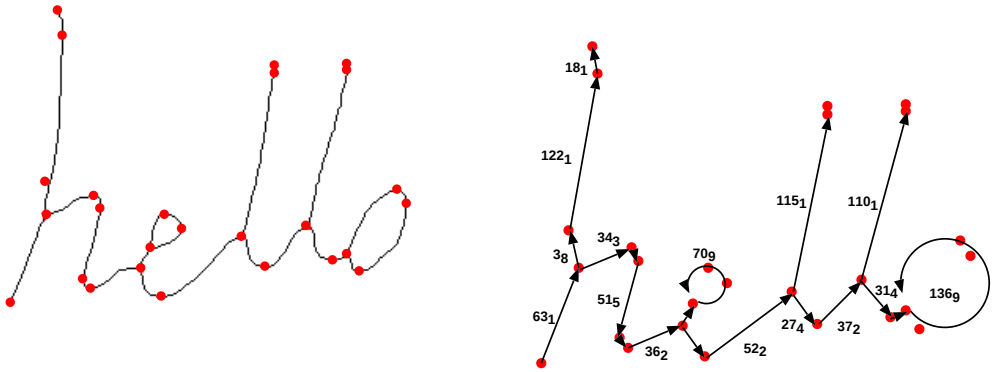


Figura 8.5: Códigos para los segmentos y bucles.



(a) (b)

hello 23 (63;1(34;3(9;4(51;5(8;4(36;2(15;2(70;9())28;4(
58;2(115;1(4;1())27;4(37;2(110;1(3;8(122;1(18;1()))))))
(c)

Figura 8.6: a) Puntos dominantes; b) Árbol de características ($\text{NORMA}_{\text{dirección}}$); c) Cadena del árbol.

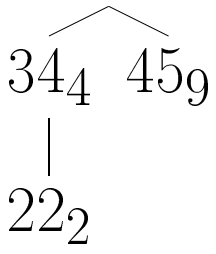


Figura 8.7: Representación gráfica de 4 (34;4(22;2())45;9())

(1997) e Iñesta et al. (1998). Estos puntos se utilizan para describir cambios significativos en los trazos de la palabra manuscrita que cumplen una de estas condiciones:

- Puntos finales (esto es, puntos simplemente conectados).
- Puntos correspondientes a extremos locales de curvatura.
- Puntos de intersección.

En esta aproximación, las características con estructura de árbol se extraen a partir de las primitivas de dirección (figura 8.5) que unen puntos dominantes. El método escoge como raíz el punto dominante más a la izquierda (en caso de empate escogería el superior) y construye el resto del árbol uniendo los puntos dominantes más próximos que están conectados en la imagen binaria. Cada nodo del árbol está etiquetado con una cadena que describe la diferencia del nodo actual con su ancestro en el árbol. La cadena se obtiene utilizando una de las nueve posibles direcciones de la figura 8.5. Se muestra un ejemplo en la figura 8.6b. El árbol de características se expresa de una manera funcional como una cadena con símbolos parentizados que sirve como entrada al algoritmo de cálculo de distancias entre árboles.

El significado de las características obtenidas de la palabra "*hello*" que muestra la figura 8.6c es el siguiente: la etiqueta 63_1 representa un vector

(a)

(b)

hello 24 (63;1(34;3(9;4(51;5(8;4(12;3(26;3(16;2(70;9())27;4(58;2(115;1(
4;1())27;4(37;2(109;1(5;1())31;4(8;3(136;9())))))))))))3;8(121;1(18;1(()))))
(c)

hell 21 (75;1(40;3(11;4(60;5(10;4(42;2(16;2(83;9())33;4(68;2(
136;1(5;1())32;4(44;2(129;1(6;1())32;4(()))))))))))3;8(143;1(22;1(()))))
(d)

Figura 8.8: a) Imagen correspondiente a la palabra "*hello*" con la letra "*h*" desconectada del resto. b) Imagen de grises correspondiente a la palabra "*hell*". c) Árbol de características de la imagen a. d) Árbol de características de la imagen b.

de longitud 63 con la dirección 1; la etiqueta 34_3 corresponde a un vector con longitud 34 y dirección 3; sin embargo, la etiqueta 70_9 representa un bucle de longitud 70. Como ejemplo sencillo se puede ver el la figura 8.7 una representación gráfica de un árbol etiquetado.

Si las imágenes correspondientes a dos palabras muestran pequeñas diferencias como, por ejemplo alguna letra desconectada (figura 8.8a comparada con 8.4a), la distancia entre ellos será aún pequeña debido a la forma en la que se ha obteniendo el árbol de características. Recuérdese que el algoritmo enlaza los puntos dominantes vecinos que no se han utilizado hasta el momento siguiendo un orden de izquierda a derecha y de arriba a abajo. Por ejemplo, el valor de la distancia entre el árbol de características de la palabra en la figura 8.6c y la de la figura 8.8c es de tan solo 78 unidades, mientras que la distancia entre los árboles de características de la figura 8.6c y la de la figura 8.8d es de 654 unidades. Los detalles sobre el cálculo de distancia se encuentran en la siguiente sección.

8.3 Clasificación

La distancia de edición se calcula usando técnicas de programación dinámica como en Zhang and Shasha (1989) y Oommen et al. (1996). Este algoritmo se puede aplicar a árboles ordenados y etiquetados, con una complejidad temporal

$$\mathcal{O}\left(|T_1| \times |T_2| \times \min\left(\text{depth}\left(T_1\right),\ \text{leaves}\left(T_1\right)\right) \times \min\left(\text{depth}\left(T_2\right),\ \text{leaves}\left(T_2\right)\right)\right)$$

donde T_1 y T_2 son los árboles comparados. Como mejora, se aplica también un algoritmo que adapta los pesos de la distancia de edición para incrementar la tasa de clasificación correcta.

La técnica Leaving-One-Out (Duda and Hart 1973) se utiliza para estimar la tasa de error. Para cada escritor, cada prototipo se compara con el resto y se clasifica. La tasa de error se obtiene aplicando este método a todos los

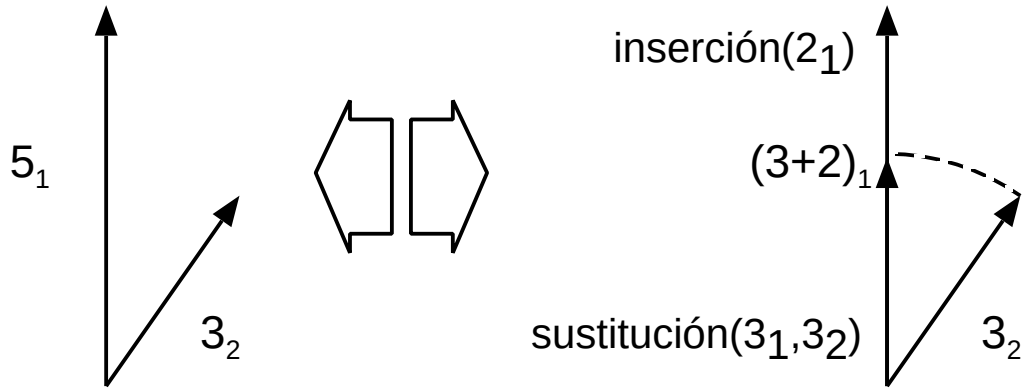


Figura 8.9: Ejemplo de la operación sustitución entre el vector 5_1 y 3_2. La equivalencia gráfica que representa es sustitución$(5_1, 3_2)$ = sustitución$(3_1, 3_2)$ + inserción(2_1)

prototipos de la base de datos.

El cálculo de la distancia de edición se basa en tres operaciones: inserción, borrado y sustitución. Cada operación tiene asociado un peso. Las operaciones elementales de inserción y borrado tienen el mismo peso $w_I = w_D$, y la sustitución de un vector unitario de tipo a por otro del tipo b tiene un coste de w_{ab}. Dados dos vectores cualesquiera $\mathbf{m}_a$ y $\mathbf{n}_b$, el borrado del vector $\mathbf{m}_a$ tiene un coste $m\, w_I$ y la sustitución de $\mathbf{m}_a$ por $\mathbf{n}_b$ tiene un coste $\min\{m, n\}\, w_{ab} + |m - n|\, w_I$. Se puede ver un ejemplo en la figura 8.9.

El método iterativo que se ha utilizado para ajustar los pesos de sustitución w_{ij} es:

$$w_{ij}^t = w_{ij}^{t-1} + \alpha \frac{n_{ij}^{'} - n_{ij}}{\max\left\{n_{ij}^{'},\, n_{ij}\right\}} \tag{8.1}$$

donde:

- α es un valor para controlar el incremento de los pesos y su valor oscila entre 0 y 1;
- w_{ij} son los pesos para las operaciones de sustitución de la dirección i

$$w_{ij}^0 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 3 & 2 & 1 & 4 \\ 1 & 0 & 1 & 2 & 3 & 4 & 3 & 2 & 4 \\ 2 & 1 & 0 & 1 & 2 & 3 & 4 & 3 & 4 \\ 3 & 2 & 1 & 0 & 1 & 2 & 3 & 4 & 4 \\ 4 & 3 & 2 & 1 & 0 & 1 & 2 & 3 & 4 \\ 3 & 4 & 3 & 2 & 1 & 0 & 1 & 2 & 4 \\ 2 & 3 & 4 & 3 & 2 & 1 & 0 & 1 & 4 \\ 1 & 2 & 3 & 4 & 3 & 2 & 1 & 0 & 4 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{bmatrix} \tag{8.2}$$

Tabla 8.1: Pesos iniciales para la operación de sustitución.

por la j, y su valor está comprendido entre 0 y 4. Estos valores se corresponden con la mínima y máxima diferencia entre los valores de las direcciones de la figura 8.5. Se garantiza que no se excede de este rango: si $w_{ij}^t < 0$ entonces se asigna 0 y si $w_{ij}^t > 4$ se asigna 4.

- n_{ij} es el número de sustituciones del tipo ij realizadas por los prototipos correctamente clasificados;
- n'_{ij} es el número de sustituciones del tipo ij realizadas por los prototipos incorrectamente clasificados.

De esta manera, los prototipos incorrectamente clasificados incrementan su distancia con respecto de su vecino más cercano, mientras que los correctamente clasificados la disminuyen. El algoritmo de adaptación se aplica un número determinado de veces (establecido a priori), y los pesos que se escogen para la clasificación son aquellos que han obtenido mayor tasa de clasificación con el conjunto de entrenamiento.

Los pesos de inserción y borrado se fijan $w_I = w_D = 2$, ya que es la mitad del valor máximo de un peso de sustitución w_{ij} que se inicializan a

$$\min\left(|i-j|, 8 + \min(i,j) - \max(i,j)\right) \tag{8.3}$$

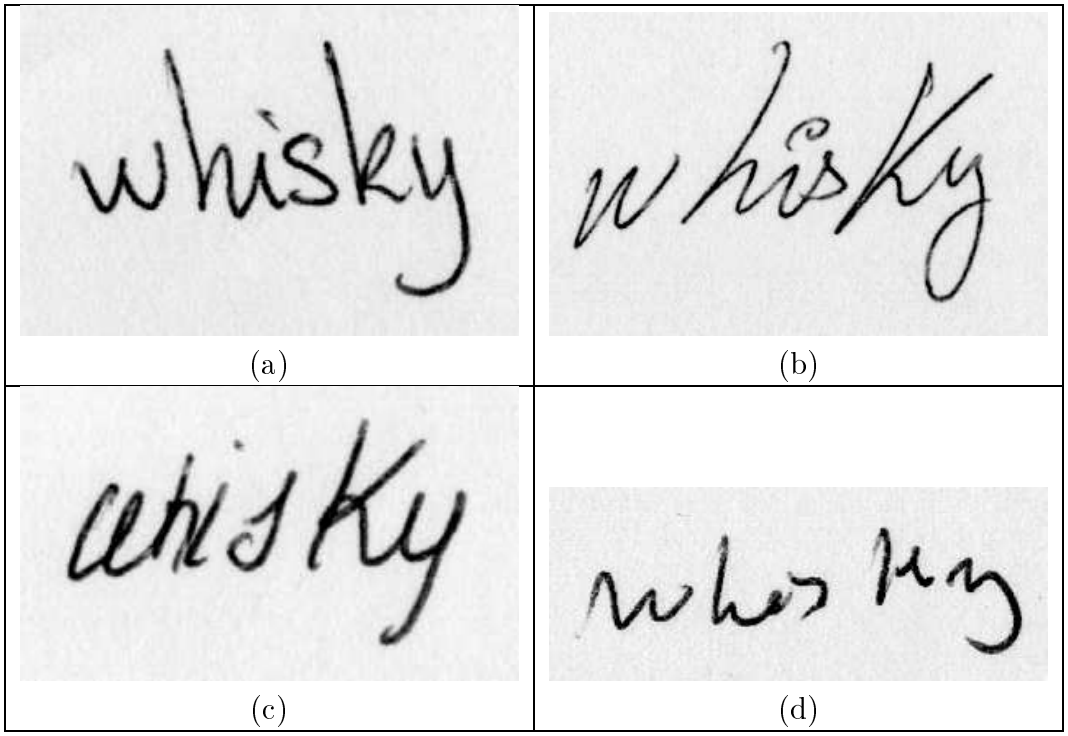


Figura 8.10: Ejemplo de la palabra whisky escrita por cuatro escritores diferentes.

que asigna menos peso aquellas direcciones cercanas y más peso a las direcciones distantes (figura 8.5).

8.4 Resultados

El método propuesto ha sido aplicado a dos bases de datos diferentes:

Base de datos #1: Se ha recogido muestras de palabras manuscritas de cuatro estilos diferentes como podemos ver en la figura 8.10. Se ha construido una base datos con un total de 2.400 muestras (600 por escritor). Cada escritor ha escrito 50 palabras distintas (tabla 8.2) y cada palabra se ha repetido 12 veces. Todas la palabras que forman la base de datos tienen de cuatro a seis letras. Los ejemplo se han obtenido

bola	burro	campo	casa	cola
creer	dedo	dentro	eres	fuera
gato	hola	jarro	kilo	lago
laxo	listo	manta	mata	mimo
mirar	mola	morro	niño	para
pelo	penal	pensar	perro	poder
queso	rata	rico	rota	salta
sedal	señal	sigo	silla	sofá
sola	sopa	suyo	turno	tutor
tuyo	vivo	whisky	zona	zumo

Tabla 8.2: Ejemplo de palabras usadas en la base de datos #1.

usando un escaner con una resolución de 300 dpi y 256 niveles de grises a partir de páginas escritas con bolígrafo negro.
Hay que destacar que nuestro interés es estudiar cuán bueno es nuestro algoritmo aplicado a un escritor, y por lo tanto, no estamos interesados es construir un gran base de datos con muchos escritores. Las palabras tienen un longitud similar para no dar la ventaja al sistema de que pueda distinguir las palabras simplemente por su tamaño.

Base de datos #2: El vocabulario de test ha sido extraído del corpus LOB (Lancaster Oslo y Bergen) que ha sido utilizado, por ejemplo, en Senior and Robinson (1998) para reconocimiento de caracteres manuscritos continuos fuera de línea utilizando redes neurales recurrentes y modelos de Markov ocultos. Se han seleccionando 20 ejemplos de cada una de las 34 palabras más frecuentes.

El número de ejemplos que se ha escogido ha sido de 150, 300, 450 y 600 prototipos para cada escritor. El algoritmo de clasificación se aplica a cada grupo para un escritor determinado y la media por grupo se muestra en las tablas 8.3 y 8.4.

Se puede ver en la tabla 8.4 y figura 8.12 como la mayor tasa medio de clasificación obtenido ha sido del 95% en promedio, mientras que para un

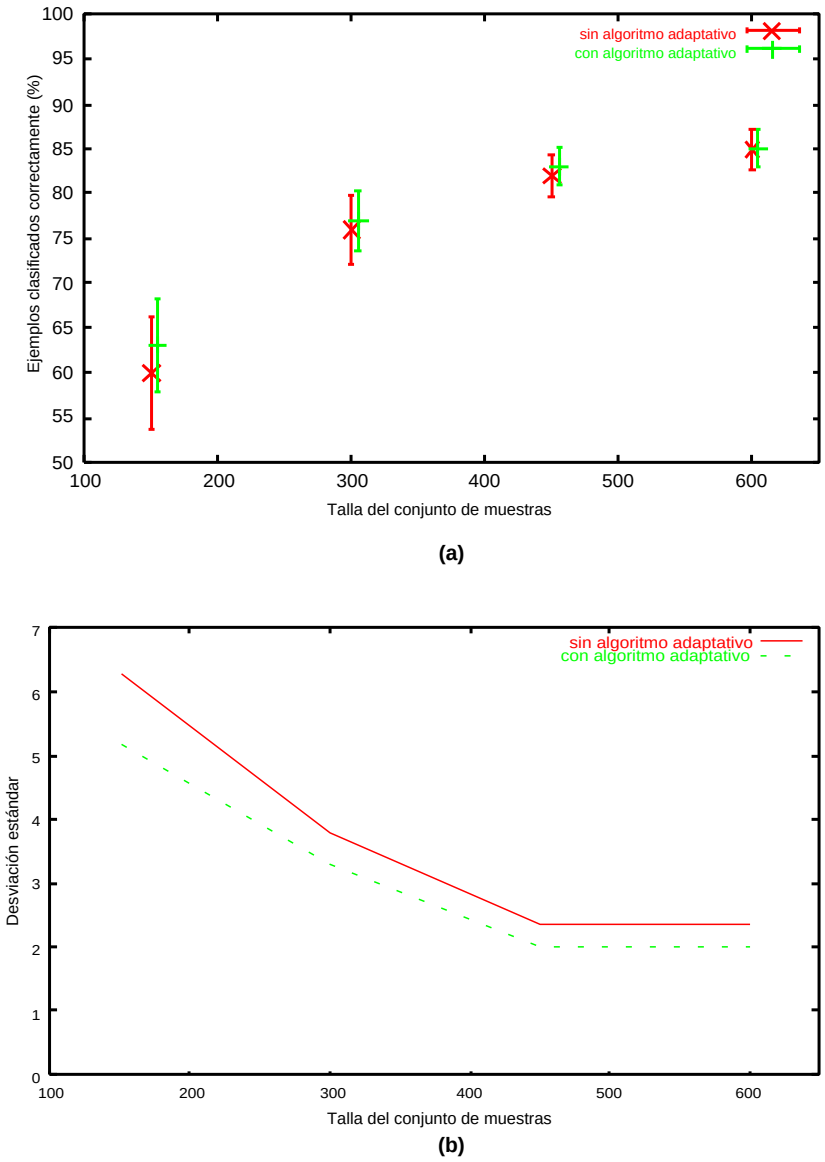
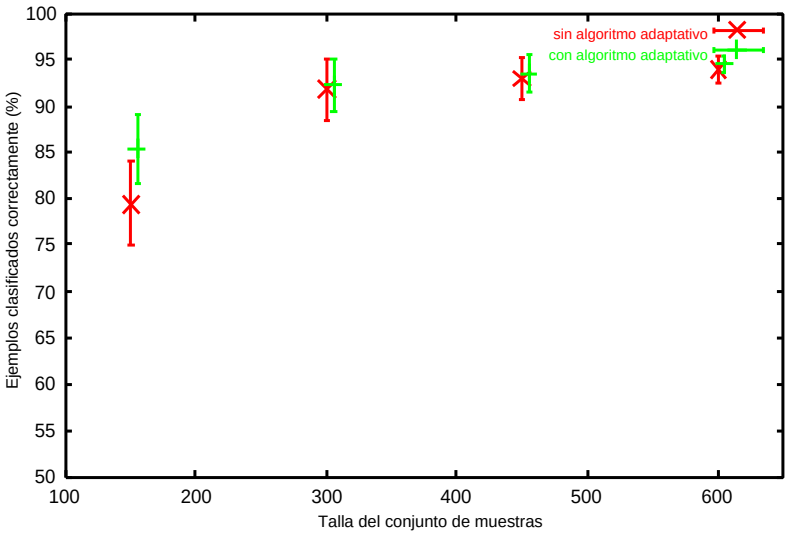
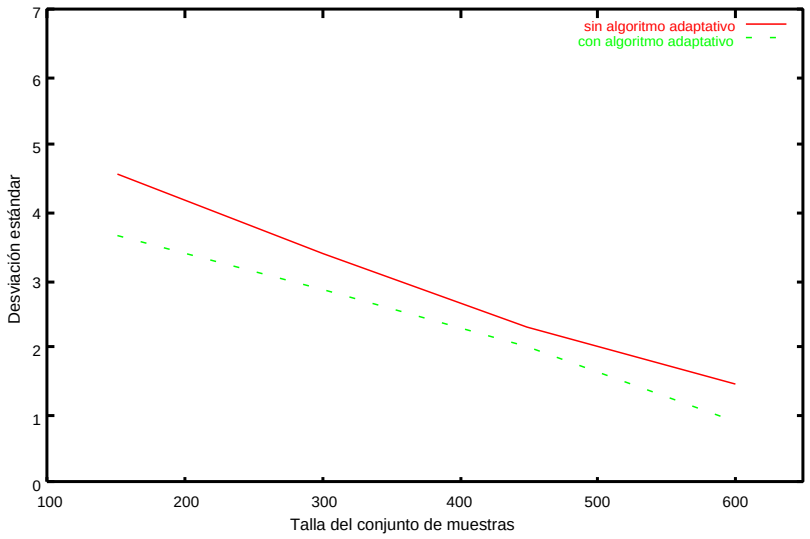


Figura 8.11: Base de datos #1: a) Tasa de clasificación de palabras media sin y con algoritmo adaptativo. b) Resultados de la desviación estándar de los experimentos sin y con el algoritmo adaptativo.



(a)



(b)

Figura 8.12: Base de datos #2: a) Tasa de clasificación de palabras media sin y con algoritmo adaptativo. b) Resultados de la desviación estándar de los experimentos sin y con el algoritmo adaptativo.

# prototipos	sin algoritmo adaptativo μ	D	con algoritmo adaptativo μ	D
150	60	6.3	63	5.2
300	76	3.8	77	3.3
450	82	2.4	83	2.1
600	85	2.3	85	2.0

Tabla 8.3: Tasa μ (porcentaje) de clasificación de palabras sin y con el algoritmo adaptativo y D la desviación estándar sobre la base de datos #1, respectivamente.

prototipos	sin algoritmo adaptativo μ	D	con algoritmo adaptativo μ	D
150	79	4.6	86	3.7
300	91	3.4	92	2.9
450	93	2.3	94	2.0
600	94	1.5	95	0.9

Tabla 8.4: Tasa μ (porcentaje) de clasificación de palabras sin y con el algoritmo adaptativo y D la desviación estándar sobre la base de datos #2 (LOB), respectivamente.

escritor dado ha llegado hasta el 95,5%.

Cuando se aplica el algoritmo más sencillo (sin algoritmo adaptativo de pesos), los resultados oscilan entre el 60% y el 85% en la base de datos #1 (tabla 8.3 y figura 8.11) y entre el 79% y el 94% en la base de datos #2 (tabla 8.4 y figura 8.12), dependiendo del número de prototipos de la muestra. Para el algoritmo de los pesos adaptativos los resultados han estado entre un 63% y un 85% (tabla 8.2) y entre un 86% y un 95% (tabla 8.4), según la base de datos.

Si comparamos los resultados entre las dos tablas se observa claramente que con la base de datos #2 se obtienen mejores resultados. Esto es debido a que la base datos #2 tiene menor número de clases y las longitudes de sus palabras no son homogéneas. Recordemos que la base de datos #1 la habíamos construido con palabras desde 4 hasta 6 letras, incrementando la dificultad para la clasificación.

También se observa que el algoritmo adaptativo es más útil cuando tenemos pocos prototipos y se observa un disminución en la desviación estándar de los resultados (figuras 8.11b y 8.12b).

8.5 Conclusiones

Hemos explorado un nuevo método para la extracción de características con estructura de árbol a partir de la imagen de una palabra. El método usa una generalización de la distancia de edición entre árboles optimizada para este problema. El método se ha aplicado al reconocimiento de palabras manuscritas fuera de línea. El algoritmo es fácil de aplicar y los resultados son muy esperanzadores.

Con la implementación actual, si incrementamos el tamaño de la base de datos con nuevos prototipos se incrementa el tiempo de clasificación linealmente. Este hecho se puede mejorar con algoritmos como el AESA (Approximating Eliminating Search Algorithm) u otros relacionados como Micó

et al. (1996) y Micó and Oncina (1998). Estos algoritmos calculan la distancia entre algunos prototipos de la base de datos en la fase de preproceso, y usan esta información preprocesada para acelerar la búsqueda del vecino más cercano.

Otros algoritmos, como por ejemplo Li and Yeung (1997), necesitan información adicional, además de la imagen de las palabras. Por ejemplo, tienen un diccionario restringido de palabras, por lo que es más difícil incrementar la base de datos con nuevas muestras, o necesitan algoritmos para separar las palabras en letras e información estadística, como los utilizados en Suen et al. (1980), Elliman and Banks (1991) y Bunke et al. (1995) .

Vías futuras que podrían seguirse basándose en el trabajo actual serían incrementar el número de escritores y de muestras o generalizar el algoritmo para reconocer una muestra de un escritor cualquiera. Una idea interesante sería aplicar este algoritmo a un sistema de reconocimiento de firmas.

Parte III

Conclusiones y trabajos futuros

Capítulo 9

Conclusiones y trabajos futuros

De forma general, los desarrollos de este trabajo se han realizado alrededor de los lenguajes de árboles k-testables y sus aplicaciones. La aportaciones de esta tesis han sido:

- el planteamiento de una extensión de los lenguajes de árboles k-testables actualizable incrementalmente que permite un grado de generalización menor que el de la gramáticas de árboles obtenidas directamente de la muestra;
- la aplicación eficiente de variaciones del algoritmo anterior a la compresión y a la clasificación de muestras con estructura de árbol etiquetado, obteniendo unos resultados favorables respecto de otros métodos relacionados con estas áreas;
- la definición de un método de compresión para imágenes 3D de superficies basado en el árbol de expansión mínima y distribuciones sobre números reales, una por dimensión, con una tasas de compresión superiores al resto de métodos de compresión;
- la implementación de un sistema de reconocimiento de palabras manuscritas continuas basada en la distancia de edición de árboles con una

eficiencia de hasta el 95%.

Parte del contenido de esta tesis ha sido publicado en los siguiente artículos:

- J. R. Rico-Juan, J. Calera-Rubio, and R.C. Carrasco. Probabilistic k-testable tree-languages. In A.L. Oliveira, editor, *Proceedings of 5th International Colloquium, ICGI 2000*, Lisbon (Portugal), volume 1891 of Lecture Notes in Computer Science, pages 221-228, Berlin, 2000. Springer.

- J. R. Rico-Juan, J. Calera-Rubio and R. C. Carrasco. Lossless Compression of Surfaces Described as Points. In F. J. Ferri, J. M. Iñesta, A. Amin and P. Pudil, editors, *Proceedings of the Joint IAPR International Workshops SSPR2000 and SPR2000*, Alicante, Spain, August/September 2000. , vol. 1876 of Lecture Notes in Computer Science, 457-461. Berlin (2000) Springer

- J. R. Rico-Juan. Off-line cursive handwitten word recognition based on tree extraction and an optimized classificaciton distance. In M. I. Torres and A. Sanfeliu, editors, *Pattern Recognition and Image Analysis: Proceedings of the VII Symposium Nacional de Reconocimiento de Formas y Análisis de Imágenes*, Volume II, pages 16-16, AERFAI, Geneve.

Entre los problemas que quedan abiertos tras este trabajo se encuentran:

- La evaluación de estos métodos con bases de datos más extensas, especialmente de tipo lingüístico y para resolver ambigüedades sintácticas.

- Generalizar los métodos de suavizado y descuento para gramáticas independientes del contexto, donde el árbol de análisis sintáctico no es necesariamente único.

- La aplicación de los métodos de compresión a imágenes bidimensionales codificadas como quadtrees u otras estructuras de tipo arbóreo.
- El desarrollo de métodos eficientes de predicción y compresión de estructuras más generales como los grafos dirigidos acíclicos.

Alicante, 15 de mayo de 2001.

Apéndice A

Otros métodos de compresión de cadenas

Este apéndice está dedicado a dos métodos de compresión cuyas directrices de funcionamiento difieren del uso de los modelos estocásticos utilizados en esta tesis. Como se han considerado importantes se les ha dedicado este apartado.

A.1 Compresión por ordenamiento de bloques

La compresión por ordenamiento de bloques (*Block-Sorting Compresión*) es una aproximación que se publicó en el año 1994 (Burrows and Wheeler 1994). Es un enfoque especial porque transforma el texto antes de comprimirlo. El descompresor aplicará la inversa de esta técnica para obtener el texto final. Es una técnica análoga a la del coseno discreto para la compresión de imágenes o la transformada de Fourier para señales. Una desventaja del ordenamiento por bloques es que la entrada está divida en bloques que se procesan de uno en uno, mientras que un método adaptativo procesaría los caracteres individualmente según le llegan.

Esta transformación de ordenamiento por bloques, también llamada trans-

I		F
m	ississipp	i
i	ssissippi	m
s	sissippim	i
s	issippimi	s
i	ssippimis	s
s	sippimiss	i
s	ippimissi	s
i	ppimissis	s
p	pimississ	i
p	imississi	p
i	mississip	p

(a)

I		F
i	mississip	p
i	ppimissis	s
i	ssippimis	s
i	ssissippi	m
m	ississipp	i
p	imississi	p
p	pimississ	i
s	ippimissi	s
s	issippimi	s
s	sippimiss	i
s	sissippim	i

(b)

I	F
i	p
i	s
i	s
i	m*
m	i
p	p
p	i
s	s
s	s
s	i
s	i

(c)

Tabla A.1: Transformada de Burrows-Wheeler de la cadena *missisippi*: (a) rotaciones de la cadena; (b) ordenación alfabética por contexto; (c) cadena ordenada (columna inicial I) y cadena permutada (columna final F)

formación de Burrows-Wheeler, presenta la peculiaridad de que es reversible. En ella:

1. Se divide el texto en bloques de tamaño n.
2. Se generan todas las n permutaciones cíclicas (rotaciones) del bloque.
3. Se ordenan alfabéticamente todas las permutaciones.
4. Se escribe el último símbolo de cada permutación siguiendo la nueva ordenación .

En la tabla A.1 se representa este procedimiento aplicado al bloque "`mississippi`" y cuyo resultado es "`pssmipissi`".

La ventaja de esta transformación es que es invertible (si trabajáramos con la cadena ordenada no se podría reconstruir la cadena original). En efecto de esta última cadena "`pssmipissi`" se podría obtener la original siguiendo estos pasos:

1. Se ordena la cadena F. Se obtiene I="`iiiimppssss`".

2. Se asigna a la cadena C la primera letra de la cadena original, esto es, "`m`".

3. Se busca la última letra de C en la columna F de la tabla.

4. Se añade a C la letra de la columna I de la fila hallada.

5. Se borra esa fila de la tabla.

6. Si quedan filas en la tabla se vuelve al paso 3.

Es fácil comprobar que tras estos pasos, se obtiene C="`mississippi`".

Previsiblemente, la nueva secuencia obtenida, F, puede ser comprimida más eficientemente, ya que está parcialmente ordenada y su entropía es menor. Nótese que el símbolo que precede a contextos idénticos (o similares) es probablemente parecido.

Para que el resultado sea más efectivo es conveniente aplicar a la cadena resultante de la transformada de Burrows-Wheeler una codificación *mtf* (move-to-front) que consiste básicamente en codificar los símbolos según sus posiciones. Supongamos que tenemos un vector con 256 entradas, una para cada símbolo y que están situados según su código ASCII. Se emite el índice, n, del símbolo que se está examinando, s. Los símbolos desde el índice 0 hasta el $n-1$ se incrementan en 1 y s se lleva al principio del vector, quedando en el índice 0. Este proceso se repite hasta agotar la entrada. Si aplicamos esta codificación a la cadena F="*pssmipissi*", `índices`(L)={112, 115, 115, 109, 105, 112, 105, 115, 115, 105, 105}; el resultado sería mtf(L)={112, 115, 0, 111, 108, 4, 1, 4, 0, 1, 0}.

Además, conviene aplicar también un *rle* (run-length-encoder). Este codificador sustituiría las secuencias consecutivas del mismo valor por una codificación compacta de esté valor y su longitud, ya que cuando se examinan

bloques enteros se produce esta repetición de valores consecutivos. Por supuesto, estos dos últimos codificadores tienen su inversa para aplicarlo en el descodificador.

A.2 Compresión basada en palabras

En este método, el texto o fichero que se va a comprimir se separa en palabras (caracteres alfanuméricos consecutivos) y separadores (espacios en blanco, signos de puntuación, etc) y se comprime cada parte por separado. Obviamente, para poder recuperar el texto original es necesario que el texto este compuesto por palabras y separadores, alternativamente.

Una forma de comprimir que da buenos resultados es utilizar un modelo de contexto finito de orden 0 para las palabras y otro de orden superior para los separadores.

Además, se utiliza un símbolo especial, *escape*, que indica que se va a codificar una palabra carácter a carácter. Esto ocurre siempre que una palabra aparece por primera vez y por tanto no se puede utilizar el modelo de orden 0.

Bibliografía

Abe, N. and Mamitsuka, H. (1997). Predicting protein secondary structure using stochastic tree grammars. *Machine Learning*, 29:275–301.

Aho, A. and Ullman, J. D. (1972). *The Theory of Parsing, Translation, and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

Arcelli, C. and di Baja, G. S. (1985). A width-independent fast thinning algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7:463–474.

Bell, T. and Moffat, A. (1989). A note on the DMC data compression scheme. *The Computer Journal*, 32(1):16–20.

Bell, T. C., Cleary, J. G., and Witten, I. H. (1990). *Text Compression*. Prentice-Hall, Englewood Cliffs, NJ.

Brown, P. F., Pietra, V. J. D., de Souza, P. V., Lai, J. C., and Mercer, R. L. (1992). Class-based n-gram models of natural language. *Computacional Linguistics*, 18(4):467–479.

Bunke, H., Roth, M., and Schukat-Talamazzini, E. G. (1995). Off-line cursive handwriting recognition using Hidden Markov Models. *Pattern Recognition*, 28(9):1399–1413.

Burrows, M. and Wheeler, D. J. (1994). A block-sorting lossless data compresion algorithm. Technical Report 124, Digital. Systems Research Center.

Calera-Rubio, J. and Carrasco, R. C. (1998). Computing the relative entropy between regular tree languages. *Information Processing Letters*, 68(2):283–289.

Calera-Rubio, J., Carrasco, R. C., and Oncina, J. (1999). Tree languages arithmetic compression. In Torres, M. I. and Sanfeliu, A., editors, *Pattern Recognition and Image Analysis: Proceedings of the VII Symposium Nacional de Reconocimiento de Formas y Análisis de Imágenes*, volume I, pages 405–411. AERFAI, Geneve.

Carrasco, R. C. and Forcada, M. L. (1995). A note on the Nagendraprasad-Wang-Gupta thinning algorithm. *Pattern Recognition Letters*, 16:539–541.

Carrasco, R. C., Oncina, J., and Calera-Rubio, J. (1998). Stochastic inference of regular tree languages. In Slutzki, V. H. . G., editor, *Grammatical Inference: 4th International Colloquium, ICGI'98*, number 1433 in Lecture Notes in Artificial Intelligence. Subseries of Lecture Notes in Computer Science, pages 187–198. Springer.

Carrasco, R. C., Oncina, J., and Calera-Rubio, J. (2001). Stochastic inference of regular tree languages. *Machine Learning*. to appear.

Charniak, E. (1993). *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts.

Chaudhuri, R., Pham, S., and García, O. N. (1983). Solution of an open problem on probabilistic grammars. *IEEE Transactions on Computers*, 32(8):758–750.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124. Also in Readings in mathematical psychology, R.D. Luce, R. Bush, and E. Galanter (eds.), New York: Wiley, pp. 105-124, 1965.

Chung, K. L. (1967). *Markov Chains with Stationary Transition Probabilities*. Springer, Berlin, 2 edition.

Cleary, J. G., Teahan, W. J., and Witten, I. H. (1995). Unbounded length contexts for PPM. In Storer, J. A. and Cohn, M., editors, *Data Compression Conference*, pages 52–61, Los Alamitos, CA. IEEE Computer Society Press.

Cleary, J. G. and Witten, I. H. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Comunications*, 32(4):396–402.

Cochran, W. O., Hart, J. C., and Flynn, P. J. (1996). Fractal volume compression. *IEEE Transactions on Visualization and Computer Graphics*, 2(4):313–322.

Cormack, G. and Horspool, N. (1987). Data compression using dynamic Markov modelling. *The Computer Journal*, 30(6):541–550.

Cormack, G. V. and Horspool, R. N. (1984). Algorithms for Adaptive Huffman Codes. *Information Processing Letters*, 18(3):159–165.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press, Cambrige, MA.

Courcelle, B., Engelfriet, J., and Rozenberg, G. (1993). Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46:218–270.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. Wiley Series in Telecomunications. John Wiley & Sons, New York, NY, USA.

Duda, R. O. and Hart, P. E. (1973). *Pattern Recognition and Scene Analysis*. John Wiley and Sons, New York.

Early, J. (1970). An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13:94–102.

Elliman, D. G. and Banks, R. N. (1991). A comparison of two neural networks for hand-printed character recognition. In *Second International Conference on Artificial Neural Networks*, volume 349, pages 224–228.

Engelfriet, J. and Heyker, L. (1991). The string generating power of context-free hypergraph grammars. *Journal of Computer and System Sciences*, 43:328–360.

Fahmy, H. and Blostein, D. (1992). A survey of graph grammars: Theory and applications. In *Procedings of the 11th International Conference on Pattern Recognition*, volume 2, pages 294–298, The Hague, Netherlands.

Flasinski, M. (1992). Towards constructing a self-learning graph grammar-based pattern recognition system. *Archives of Control Sciences*, 1:223–248.

Freeman, H. (1961). On the encoding of arbitrary geometric configurations. *IRE Transactions on Electronic Computer*, 10:260–268.

Fu, K. S. (1982). *Syntactic Pattern Recognition and Applications*. Prentice Hall, Engelwood Cliffs, NJ.

Gallager, R. G. (1978). Variations on a Theme by Huffman. *IEEE Transactions on Information Theory*, IT-24(6):668–674.

García, P. (1993). Learning k-testable tree sets from positive data. Technical Report DSIC-ii-1993-46, DSIC, Universidad Politécnica de Valencia.

García, P. and Vidal, E. (1990). Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intellgence*, 12(9):902–925.

Gokmen, M., Ersoy, I., and Jain, A. K. (1996). Compression of fingerprint images usign hybrid image model. In *Int. Conf. on Image Processing*, volume III, pages 395–398, Lausanne.

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

Gómez, E., Micó, M. L., and Oncina, J. (1995). Testing the linear approximating eliminating search algorithm in handwritten character recognition tasks. In *Actas del VI Simposium Nacional de Reconocimiento de Formas y Analisis de Imagenes*, pages 212–217, Córdoba (Spain).

Hopcrof, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages and Computation.* Addison–Weslesy, Massachusetts.

Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., and Stuetzle, W. (1994). Piecewise smooth surface reconstruction. In Glassner, A., editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 295–302. ACM SIGGRAPH, ACM Press.

Howard, P. G. (1993). The design and analysis of efficient lossless data compression systems. Technical Report CS-93-28, Brown University. Ph.D. Tesis.

Huang, K. and Yan, H. (1997). Off-line signature verification based on geometric feature extraction and neural classficacion. *Pattern Recognition*, 30:9–17.

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. In *Proc. Inst. Radio Eng.*, pages 1098–1101. Published as Proc. Inst. Radio Eng., volume 40, number 9.

Iñesta, J. M., Mateo, B., and Sarti, M. A. (1998). Reliable polygonal approximations of imaged read objects though dominant point detection. *Pattern Recognition*, 31:685–697.

Ihm, I. and Park, S. (1998). Wavelet-based 3D compression scheme for very large volume data. In Davis, K. B. W. and Fourier, A., editors, *Proceedings of the 24th Graphics Interface*, pages 107–116, San Francisco. Morgan Kaufmann.

Jelinek, F. (1990). Self-organized language modeling for speech recognition. In Waibel, A. and Lee, K.-F., editors, *Readings in Speech Recognition*, pages 450–506. Morgan Kaufmann Publishers, Inc., San Maeio, California.

Jelinek, F. (1998). *Statistical methods for Speech Recognition.* The MIT Press, Cambridge, Massachusetts.

Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP–35(3):400–401.

Knuutila, T. (1993). Inference of k-testable tree languages. In Bunke, H., editor, *Advances in Structural and Sysntactic Pattern Recogniton (Proc. Intl. Workshop on Structural and Syntactic Pattern Recogniton)*, Bern, Switzerland. World Scientific.

Laird, P. (1988). *Learning from Good and Bad Data.* Kluwer, Boston, MA.

Langsam, Y., Augenstein, M., and Tenenbaum, A. M. (1996). *Data structures using C and C++.* Prentice-Hall, Englewood Cliffs, NJ 07632, USA.

Li, X. and Yeung, D. (1997). On-line handwritten alphanumeric character recognition using dominant points in strokes. *Pattern Recognition*, 30:31–34.

López, D. and Piñaga, I. (2000). Syntactic pattern recognition by error correcting analysis on tree automata. In Ferri, F. J., Iñesta, J. M., Amin, A., and Pudil, P., editors, *Advances in Pattern Recognition*, volume 1876 of *Lecture Notes in Computer Science*, pages 133–142, Berlin. IAPR, Springer-Verlag.

López, D., Sempere, J. M., and García, P. (2000). Error correcting analysis for tree languages. *International Journal of Pattern Recognition and Artificial Intelligence*, 14(3):357–368.

Mackay, D. J. C. (1998). *Information Theory, Probability and Neural Networks.* To be published. Available at http://wol.ra.phy.cam.ac.uk/mackay/itprnn/book.html.

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processin.* MIT Pres, Cambridge, MA.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., , and Schasberger, B. (1994). *"The Penn Treebank: Annotating Predicate Argument Structure.* Morgan Kaufmann, San Francisco, CA.

Matsello, V. V. (1991). Drawings recognition using two-dimensional graph grammars. *International Journal Imaging Systems and Technology*, 3:244–288.

Micó, M. L. and Oncina, J. (1998). Comparison of fast nearest neigbour classifiers for handwritten character recognition. *Pattern Recognition Letters*, 19:351–356.

Micó, M. L., Oncina, J., and Carrasco, R. C. (1996). A fast branch and bound nerarest neighbour classifier in metric spaces. *Pattern Recognition Letters*, 17:731–739.

Moffat, A. (1990). Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921.

Nelson, M. R. (1991). Arithmetic coding and statistical modeling. *Dr. Dobb's Journal of Software Tools*, 16(2):16–18, 20, 22, 24, 26, 29, 104, 106–108.

Ney, H. and Essen, U. (1993). Estimating small probabilities by Leaving-One-Out. In *Third European Conference on Speech Communication and Technology*, pages 2239–2242, Berlin, Germany.

Ney, H., Essen, U., and Kneser, R. (1995). On the estimation of smal probabilities by Leaving-One-Out. *IEEE trans. on Pattern Analysis and Machine Intelligence*, 17(12):1202–1212.

Ney, H., Martin, S., and Wessel, F. (1997). Statistical language modeling using Leaving-One-Out. In Young, S. and Bloothoft, G., editors, *Corpus Based Methods in Language and Speech Processing*, Lecture Notes in Computer Science, pages 174–207, The Netherlands. Kluwer Academic Publishers.

Oommen, B. J., Zhang, K., and Lee, W. (1996). Numerical similarity and dissimilarity measures between two trees. *IEEE Transactions on Computers*, 45(12):1426–1434.

Pérez-Cortés, J. C., Llobet, R., and Arlandis, J. (2000). Fast and accurate handwritten character recongnition using approximate nearest neigbours search on large databases. In Ferri, F. J., Iñesta, J. M., Amin, A., and Pudil, P., editors, *Advances in Pattern Recognition*, volume 1876 of *Lecture Notes in Computer Science*, pages 767–776, Berlin. IAPR, Springer-Verlag.

Powalka, R. K., Sherkat, N., and Whitrow, R. J. (1997). Word shape analysis for a hybrid recognition system. *Pattern Recognition*, 30:421–445.

Rekers, J. (1994). On the use of Graph Grammars for defining the Syntax of Graphical Languages. In *Proceedings of the Colloquium on Graph Transformation*, Palma de Mallorca, Spain.

Rico-Juan, J. R. (1999). Off-line cursive handwritten word recognition based on tree extraction and an optimized classification distance. In Torres, M. I. and Sanfeliu, A., editors, *Pattern Recognition and Image Analysis: Proceedings of the VII Symposium Nacional de Reconocimiento de Formas y Análisis de Imágenes*, volume II, pages 15–16. AERFAI, Geneve.

Rico-Juan, J. R., Calera-Rubio, J., and Carrasco, R. C. (2000a). Lossless compression of surfaces described as points. In Ferri, F. J., Iñesta, J. M., Amin, A., and Pudil, P., editors, *Advances in Pattern Recognition*, volume 1876 of *Lecture Notes in Computer Science*, pages 457–461, Berlin. Springer-Verlag.

Rico-Juan, J. R., Calera-Rubio, J., and Carrasco, R. C. (2000b). Probabilistic k-testable tree-language. In Oliveira, A. L., editor, *Proceedings of 5th International Colloquium*, volume 1891 of *Lecture Notes in Computer Science*, pages 221–228, Lisboa (Portugal). Springer-Verlag.

Rubin, F. (1976). Experiments in text file compression. *Communications of the ACM*, 19(11):617–623.

Sakakibara, Y. (1992). Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1):23–60.

Sánchez, J. A. and Benedí, J. M. (1997). Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1052–1055.

Senior, A. W. and Robinson, A. J. (1998). An off-line cursive handwriting

recognition system. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 20:309–321.

Serra, J. (1982). *Image Analysis and mathematical morphology*. Academic Press.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell Syst. Technical Jrnl.*, 27:379–423, 623–656.

Sima'an, K., Bod, R., Krauwer, S., and Scha, R. (1996). Efficient disambiguation by means of stochastic tree substitution grammars. In Jones, D. B. and Somers, H. L., editors, *Proc. of the Int. Conf. on New Methods in Language Processing. Manchester, UK, 14–16 Sep 1994*, pages 50–58, London, UK. UCL Press.

Stolcke, A. (1995). An efficient context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.

Stolcke, A. and Segal, J. (1994). Precise n-gram probabilities from stochastic context-free grammars. Technical Report TR-94-007, International Computer Science Institute, Berkey, CA.

Suen, C. Y., Berthod, M., and Mori, S. (1980). Automatic recognition of handprinted characters – the state of art. In *IEEE*, volume 68(4), pages 469–487.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269.

Vitter, J. S. (1989). Dynamic Huffman Coding. *ACM Transactions on Mathematical Software*, 15(2):158–167.

Wang, J., Leung, M. K. H., and Hui, S. C. (1997). Cursive word reference line detection. *Pattern Recognition*, 30:503–511.

Witten, I. H. and Bell, T. C. (1991). The zero frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes*. Morgan Kaufmann, San Francisco, CA, USA, 2 edition.

Witten, I. H., Neal, R. M., and Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM*, 30(6).

Yokomori, T. (1995). On polynomial-time learnability in the limit of strictly deterministic automata. *Machine Learning*, 19(2):153–179.

Zhang, K. and Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18:1245–1262.

Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343.