UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Estudio de técnicas de bootstrapping en el aprendizaje de lenguajes formales

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Jardón Chornet, Rogelio
Tutor: López Rodríguez, Damián

Curso 2018-2019

Resum

En aquest treball es va a estudiar la influència de l'ús de tècniques de *bootstrapping* a l'hora d'implementar mètodes per a l'aprenentatge de llenguatges formals. Aquestes tècniques s'utilitzaran per a seleccionar y reduir la dimensió del conjunt de entrenament, estudiant si aquesta reducció té efecte al percentatge de reconeixement obtingut per l'algorisme considerat. Posteriorment, es comprovarà l'eficiència d'aquest mètode en comparació amb altres estudis.

Paraules clau: Aprenentatge DFA, inferència gramatical, bootstrapping, estratègia Red-Blue

Resumen

En el presente trabajo se va a estudiar la influencia del uso de técnicas de *bootstrapping* a la hora de implementar métodos para el aprendizaje de lenguajes formales. Estas técnicas se usarán para seleccionar y reducir el conjunto de entrenamiento, estudiando si esta reducción tiene efecto en el porcentaje de reconocimiento obtenido por el algoritmo considerado. Posteriormente, se comprobará la eficiencia del método aplicado en comparación con otros estudios.

Palabras clave: Aprendizaje DFA, inferencia gramatical, bootstrapping, estrategia Red-Blue

Abstract

In the present project we will study the influence of using bootstrapping techniques when methods are implemented in order to learn formal languages. This techniques will be used to select and reduce the training set, studying if this reduction has any efect in the recognition rate of the considered algorithm. Then, we will check the efficiency of this method compared to other studies.

Key words: DFA learning, grammatical inference, bootstrapping, Red-Blue strategy

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

CAPÍTULO 1
Introducción

La programación dirigida por eventos es un paradigma de programación que consiste en definir un orden en la ejecución de procesos mediante una serie de sucesos. Estas secuencias de eventos se guían en un gran número de ocasiones por procesos automáticos, que son fácilmente codificables mediante palabras.

Una palabra x es, según la teoría de lenguajes formales, una secuencia (o cadena) de símbolos pertenecientes a un alfabeto Σ. Al conjunto de todas las palabras que se pueden formar a partir del alfabeto Σ se le llama Σ^*, y un lenguaje L sobre Σ es un subconjunto de Σ^* formado por palabras que, en ocasiones, poseen algunas características comunes. En tareas aplicadas, puede abordarse la tarea de aprender la gramática G o el autómata A que genera o acepta, respectivamente, un lenguaje L.

El proceso por el que se aprenden o infieren estas características comunes de un lenguaje se estudia en lo que se conoce como inferencia gramatical. Este campo de la informática se centra en estudiar la forma más eficiente posible de aprender cómo representar un lenguaje objetivo, y esta representación se hace mediante un autómata o una gramática. La disciplina de la inferencia gramatical en la que nos centramos en este estudio es la de la inferencia inductiva, y su objetivo es descifrar la regla o función que define qué tienen en común un conjunto determinado de elementos, y en nuestro caso ese conjunto estará formado por palabras (secuencias de símbolos).

Este conjunto de palabras, al que nos referiremos a partir de ahora como conjunto de entrenamiento, puede seguir un modo de presentación concreto. Las diversas maneras de presentar la información se pueden caracterizar, entre otras cosas, por la pertenencia al lenguaje objetivo de las palabras dadas, o por la capacidad de realizar preguntas sobre el lenguaje objetivo durante el proceso de aprendizaje. Algunas de las formas de presentación de la información son las siguientes:

- Presentación positiva, consiste en proporcionar únicamente al proceso de inferencia algunas palabras pertenecientes al lenguaje que se desea aprender (el conjunto D^+).
- Presentación completa, que proporciona palabras que pertenecen al lenguaje objetivo y otras que no (el conjunto D^+ y el conjunto D^-, respectivamente).
- Presentación basada en un oráculo, se basa en la capacidad de realizar, durante el proceso de aprendizaje, distintos tipos de preguntas sobre el lenguaje objetivo a un oráculo o *teacher* (esta aproximación se conoce también como *active learning*).

La primera, la presentación positiva, consiste en una sobregeneralización, luego tiene el inconveniente de restringir las clases del lenguaje objetivo. En la tercera, la presentación basada en un oráculo, se pueden realizar distintos tipos de consultas. Las consultas

de subconjunto, las de pertenencia y las de equivalencia son algunos ejemplos en este área. Las consultas de subconjunto consisten en preguntar si, dado un lenguaje, una hipótesis determinada es un subconjunto del mismo. Las consultas de pertenencia se caracterizan por preguntar si una determinada palabra forma parte de un lenguaje. Y, por último, las consultas de equivalencia consisten en que, tras proponer una gramática, el oráculo responda si es equivalente a la gramática objetivo y, en caso de que no, aportaría un contraejemplo. Estas dos formas de presentación no se verán en nuestro estudio, ya que en el trabajo desarrollado se hará uso en todo momento de una presentación completa de la información.

A lo largo de los últimos años se han hecho investigaciones en el área de la inferencia gramatical tratando de encontrar la forma de diseñar o usar algoritmos de aprendizaje logrando la mínima tasa de error posible. Para ello, se han hecho pruebas con variaciones en las formas de presentación de la información y se han usado todo tipo de métodos de inferencia.

Los primeros estudios sobre este tema se ubican entre las décadas de 1960 y 1970, con el trabajo de E.M. Gold *Language Identification in the Limit* [1] (1967) como referencia de todos los conceptos, problemas básicos y reglas de la inferencia gramatical. Este trabajo fue esencial pues permite comprobar la corrección de los algoritmos de inferencia. Entre otras aportaciones del propio autor también se encuentra el hecho de demostrar que, para un lenguaje dado, el problema de encontrar un DFA (autómata finito determinista) minimal con respecto al número de estados y con presentación completa es NP-duro [2].

Paralelamente, Trakhtenbrot y Barzdin realizan un trabajo similar al de Gold [3]. El algoritmo que propusieron era parecido pero con dos diferencias principales: en primer lugar, la manera en que representan la información y, en segundo, el hecho de considerar limitaciones en el tamaño de las palabras del lenguaje mediante un determinado número entero.

Una década después, Angluin demuestra que el problema de la obtención del autómata minimal que ya fue planteado por Gold es también NP-duro incluso para autómatas de dos estados [4]. Además, demuestra que esa misma complejidad también la tiene el problema cuando en el algoritmo se pregunta a un oráculo sobre la pertenencia de muestras al lenguaje [5] y cuando se pregunta sobre la equivalencia de las hipótesis y el lenguaje [6]. Sin embargo, los resultados hasta ese momento siguen sin ser demasiado positivos.

En la década de los 90, se publican nuevos resultados en el campo de la inferencia gramatical que despiertan optimismo en los investigadores. Gracias a ello, se produce un aumento considerable del número de estudios en este campo. En 1992, aparecen los trabajos de Oncina y Garcia [7] con el algoritmo *RPNI*, y de Lang [8], del que se extrae un algoritmo al que llama *Traxbar*, que propone una modificación respecto al algoritmo de Trakhtenbrot y Barzdin de 1973.

En 1996, De la Higuera estudia cómo afecta a la convergencia el orden en que se realiza la fusión determinista de estados [9] y, al final de la misma década, se desarrolla el algoritmo Blue-Fringe *EDSM* que plantea la estrategia Red-Blue para decidir los estados candidatos a fusionarse [10].

En las últimas dos décadas, surgen aún más estudios por parte de diferentes investigadores. Concretamente, mencionar el trabajo de P. García, M. Vázquez de Parga, D. López y J. Ruiz en [11], en el que se propone el algoritmo GRBM (Generalized Red-Blue Merging) y que servirá como referencia para nuestro estudio.

Hasta ahora no se había planteado el uso generalizado de técnicas de muestreo en estudios relacionados con esta área de investigación. Algunos de los ejemplos de este tipo

de técnicas son las de *bagging*, las de *boosting* o las de *bootstrapping*. El objetivo de estas técnicas es aprovechar mejor la información disponible mediante unos criterios predeterminados aplicados a una serie de muestras o subconjuntos de un conjunto de datos. Es por ese motivo por el cual se propone en este trabajo hacer uso del último tipo mencionado, con tal de comprobar la repercusión que tendría el hecho de aplicarlas al ámbito del aprendizaje de lenguajes formales.

1.1 Motivación y objetivo

En el trabajo desarrollado se plantea el estudio de la influencia de aplicar técnicas de *bootstrapping* a un conjunto de datos de entrada de un algoritmo de inferencia gramatical. El algoritmo utilizado en el estudio será el Generalized Red-Blue Merging (GRBM), que hace uso de la estrategia Red-Blue mencionada anteriormente.

El desarrollo del trabajo consistirá en aplicar este tipo de técnicas de muestreo sobre cada uno de los conjuntos de entrenamiento del corpus de datos. Posteriormente, se generará mediante el algoritmo GRBM un equipo de autómatas que será utilizado para comprobar la capacidad de aprendizaje de los modelos obtenidos para cada lenguaje objetivo.

El objetivo principal del trabajo es estudiar si la aplicación de técnicas de *bootstrapping* tiene un efecto positivo en el aprendizaje de lenguajes formales, o dicho de otra manera, si se acerca o supera, en términos de tasa de acierto, a los mejores resultados obtenidos hasta la fecha, los cuales provienen de estudios que no consideran el uso de este tipo de técnicas.

1.2 Estructura de la memoria

La memoria del trabajo se estructura en 5 capítulos contando esta introducción. En el Capítulo 2 se explicará el algoritmo Generalized Red-Blue Merging y además se explicará previamente el procedimiento seguido para la fusión determinista de estados, que resulta ser parte fundamental del algoritmo. En el Capítulo 3 se hará una explicación de las técnicas de *bootstrapping* y *bagging* en el contexto concreto de este estudio.

A continuación, en el Capítulo 4, se explicará la parte de experimentación del trabajo, donde se expondrá el corpus de datos y el tratamiento del mismo. Además, se mostrarán los resultados obtenidos y se ofrecerán comparativas con los estudios previos. Finalmente, se hará una serie de valoraciones del trabajo realizado con las correspondientes conclusiones en el último capítulo, el Capítulo 5.

CAPÍTULO 2

Algoritmo Generalized Red-Blue Merging (GRBM)

La inferencia gramatical es un proceso que requiere del uso de métodos o algoritmos para generar modelos que representen un lenguaje objetivo y, como se menciona en la introducción, en este estudio concretamente se hará uso del algoritmo Generalized Red-Blue Merging [11].

La tarea de este algoritmo es recibir como entrada un conjunto de palabras que deberían pertenecer al lenguaje objetivo (D^+) y otro conjunto de palabras que no (D^-), y devolver una representación del lenguaje objetivo, usualmente un autómata finito determinista A o una gramática G. En este trabajo, debido a las ventajas a la hora de implementar el algoritmo, la representación del lenguaje objetivo será mediante una Máquina de Moore (MM).

El objetivo del algoritmo es, dada una entrada (D^+, D^-) generar un autómata equivalente con el mínimo número de estados posibles. Esto principalmente se consigue mediante fusiones de los mismos. Por tanto, es necesaria una explicación de los conceptos básicos usados, del método de fusión determinista que se ha aplicado en el trabajo y del algoritmo de inferencia mencionado que determinará el orden que se seguirá en la fusión de estados.

2.1 Notación

Un *autómata finito determinista (DFA)* es una máquina de estados definida como una tupla $A = (Q, \Sigma, \delta, q_0, F)$, donde Q es un conjunto finito de estados, Σ es un alfabeto, $q_0 \in Q$ es el estado inicial, $F \subseteq Q$ es el conjunto de estados finales y $\delta : Q \times \Sigma \to Q$ es la función de transición. Esta función se puede extender a cadenas de símbolos, de manera que $\hat{\delta} : Q \times \Sigma^* \to Q$ permitirá reconocer un lenguaje L.

Una *Máquina de Moore (MM)* es una máquina de estados definida como una tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, \Phi)$, donde Σ es el alfabeto de entrada, Γ el alfabeto de salida, δ es una función parcial que aplica $Q \times \Sigma \to Q$ y Φ es la función de salida que aplica $Q \to \Gamma$.

Una Máquina de Moore $M = (Q, \Sigma, \Gamma, \delta, q_0, \Phi)$ es consistente con (D^+, D^-) si $\forall x \in D^+$ se tiene $\Phi(x) = 1$ y $\forall x \in D^-$ se tiene $\Phi(x) = 0$. Se puede deducir dado que un autómata finito determinista $A = (Q, \Sigma, \delta, q_0, F)$ puede simularse mediante una Máquina de Moore $M = (Q, \Sigma, \Gamma, \delta, q_0, \Phi)$ donde $\Phi(q) = 1$ si $q \in F$ y $\Phi(q) \neq 1$ si no. Luego, el lenguaje definido por M sería $L(M) = \{x \in \Sigma^* : \Phi(\delta(q_0, x)) = 1\}$.

Dados dos conjuntos de palabras D^+ y D^- se puede definir la *Máquina de Moore aceptora de prefijos (PTMM)* como la Máquina de Moore que tiene $\Gamma = \{0, 1, ?\}, Q = Prefijos(D^+ \cup D^-)$, $q_0 = \lambda$ y $\delta(u, a) = ua$ si $u, ua \in Q$ y $a \in \Sigma$. Para definir el valor de la función salida asociada a u es 1, 0 o ? dependiendo de la pertenencia de u a D^+, D^- o a $Q - (D^+, D^-)$, respectivamente. El pseudocódigo para generar la PTMM dados los conjuntos D^+ y D^- está representado en el algoritmo 2.1.

Algoritmo 2.1 Algoritmo de construcción de una PTMM.

Entrada: Dos conjuntos finitos: D^+ y D^-
Salida: Máquina de Moore aceptora de prefijos equivalente

```
1:  P := Prefijos(D+) ∪ Prefijos(D−)                    // Ordenado canónicamente
2:  Q := {}
3:  Σ es el conjunto de símbolos distintos de P
4:  δ := {}
5:  i := 1, s := 1
6:  for all p ∈ P do
7:     Q := Q || {i}
8:     if p ∈ D+ then
9:        Φ(i) := 1
10:    else if p ∈ D− then
11:       Φ(i) := 0
12:    else
13:       Φ(i) :=?
14:    end if
15:    for all a ∈ Σ do
16:       aux := p || {i}
17:       if aux ∈ P then
18:          s := s + 1
19:          δ := δ ∪ {i, a, s}
20:       end if
21:    end for
22:    i := i + 1
23: end for
24: M := {Q, Σ, {1, 0, ?}, δ, {1}, Φ}
25: Return(M)
```

2.2 Fusión determinista de estados

Una fusión de estados consiste en una generalización del lenguaje aceptado por una Máquina de Moore. El objetivo es obtener una representación ajustada del lenguaje mediante una serie de fusiones de estados. Según la teoría, dada una Máquina de Moore, en una fusión determinista dos estados (q_1, q_2) solamente son susceptibles de ser fusionados si tienen la misma función de salida ($\Phi(q_1) \equiv \Phi(q_2)$) o si alguno de los estados a fusionar tiene función de salida ? ($\Phi(q_1) = ?$ o $\Phi(q_2) = ?$).

Además, aún siendo compatibles dos estados, es posible que la fusión lleve a indeterminismo en el nuevo autómata, luego son necesarias nuevas fusiones para solucionarlo. Estos nuevos intentos pueden dar lugar a alguna incompatibilidad de fusión, y entonces se deberían revertir la cadena de fusiones hechas y quedarnos con el autómata original.

El algoritmo que simula el comportamiento de esta función está representado en la figura 2.2.

Algoritmo 2.2 Algoritmo para la fusión determinista de estados (detmerge).

Entrada: Una Máquina de Moore $M = (Q, \Sigma, \{1, 0, ?\}, \delta_M, q_0, \Phi_M)$
Entrada: Dos estados p y q
Salida: La Máquina de Moore M' con los estados p y q fusionados (si es posible)
Salida: La Máquina de Moore M de la entrada (si no es posible)

1: **if** $\{\Phi_M(p), \Phi_M(q)\} = \{0, 1\}$ **then**
2: Return(M) *// No es posible la fusión, se devuelve MM original*
3: **end if**
4: $M' = M$;
5: **if** $\Phi_{M'}(p) =?$ **then**
6: $\Phi_{M'}(p) = \Phi_M(q)$;
7: **end if**
8: Sustituir con p cada referencia a q en M'
9: **for all** $a \in \Sigma$ **do**
10: **if** $\delta_M(p, a)$ y $\delta_M(q, a)$ están ambas definidas **then**
11: $M' = detmerge(M', \delta_M(p, a), \delta_M(q, a))$
12: **if** $M' = M$ **then**
13: $Return(M)$ *// No es posible la fusión, se devuelve MM original*
14: **end if**
15: **end if**
16: **end for**
17: Return(M') *// Ha sido posible la cadena de fusiones, se devuelve la MM modificada*

Para explicar mejor el concepto se van a realizar una serie de ejemplos. En ellos, los nodos con línea discontinúa harán referencia a los estados de clase ?, los que tienen doble círculo pertenecerán a clase 1 o + (estados de aceptación) y los nodos simples pertenecerán a clase 0 o − (estados de rechazo).[1]

A continuación, se muestran los ejemplos que ilustran de una forma más visual el comportamiento del algoritmo 2.2.

Ejemplo 3.1: dados los conjuntos de aceptación $D^+ = \{a, bb\}$ y rechazo $D^- = \{\lambda\}$, se obtiene la PTMM de la figura 2.1.

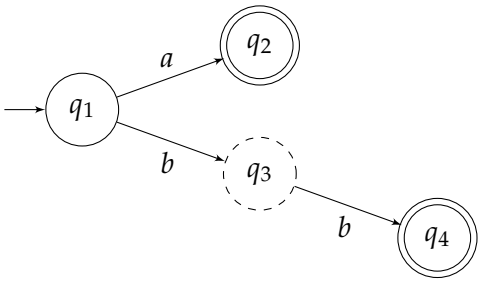

Figura 2.1: Máquina de Moore aceptora de prefijos dados los conjuntos $D^+ = \{a, bb\}$ y $D^- = \{\lambda\}$.

[1] Usamos el término 'clase de un estado' para hacer referencia a la función de salida Φ que tiene el mismo.

Si quisiéramos hacer una fusión de los estados q_1 y q_2, no podríamos pues no tienen la misma clase. Sin embargo, sí se puede fusionar el estado q_2 con q_3 pues son compatibles, y la Máquina de Moore resultante sería la representada en la figura2.2.

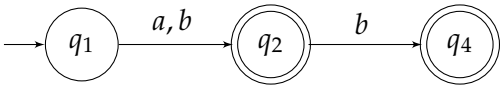

Figura 2.2: Máquina de Moore resultante tras la fusión de estados q_2 y q_3 en el ejemplo 2.1.

Ejemplo 3.2: dados los conjuntos de aceptación $D^+ = \{a, b, ab\}$ y rechazo $D^- = \{bb\}$, se obtiene la PTMM equivalente en la figura 2.3.

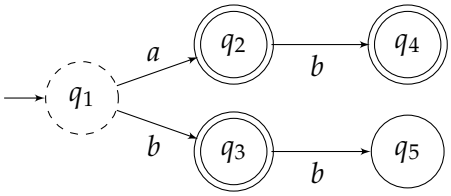

Figura 2.3: Máquina de Moore aceptora de prefijos dados los conjuntos $D^+ = \{a, b, ab\}$ y $D^- = \{bb\}$.

En este ejemplo, si se quisiera fusionar q_2 con q_3 veríamos que ambos son compatibles, pero implicaría fusionar entre sí los estados alcanzables con mismo símbolo, que en este caso serían q_4 y q_5 (transición con b), pero estos estados son incompatibles de fusionar, luego ninguna fusión podría darse. Sin embargo, sí se podría fusionar, por ejemplo, q_1 con q_2, y como se puede comprobar en la figura 2.4, la fusión de q_1 y q_2 ha implicado también la fusión de q_3 y q_4.

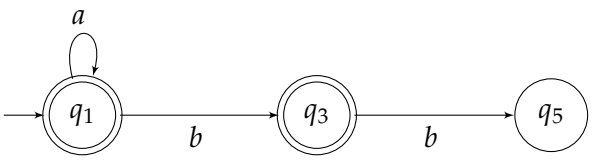

Figura 2.4: Máquina de Moore resultante tras la fusión de estados q_1 y q_2 en el ejemplo 2.2.

Ejemplo 3.3: dados los conjuntos $D^+ = \{a, aba, abba\}$ y $D^- = \{ab, ba, abab\}$, se obtiene la PTMM(D^+, D^-) equivalente de la figura 2.5.

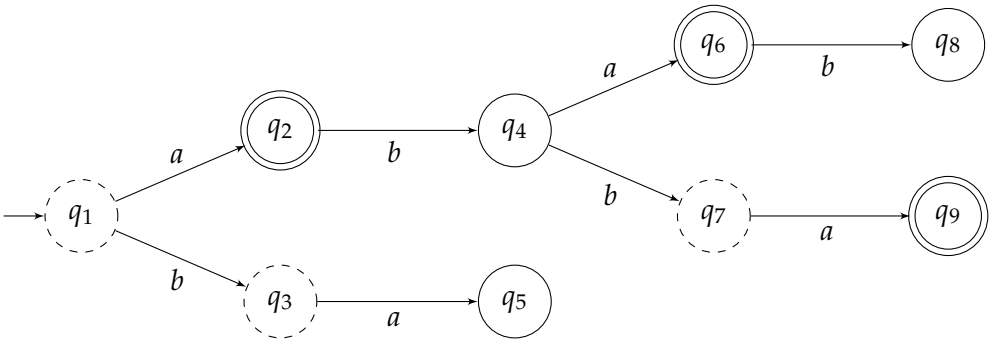

Figura 2.5: Máquina de Moore aceptora de prefijos para $D^+ = \{a, aba, abba\}$ y $D^- = \{ab, ba, abab\}$.

De la misma manera que en el ejemplo anterior, se puede deducir que si se desea fusionar q_1 y q_2, esto implica una fusión de q_3 con q_4 y después de q_5 con q_6, pero como estos últimos son incompatibles, ninguna fusión se podrá completar. Del mismo modo, si se probara con q_1 y q_3 se comprobaría que es imposible fusionarlos porque implica hacer lo propio con q_2 y q_5, que son estados incompatibles. En cambio, si la fusión se intenta hacer con los estados q_2 y q_3, se vería que sí hay compatibilidad tal y como se aprecia en la figura 2.6.

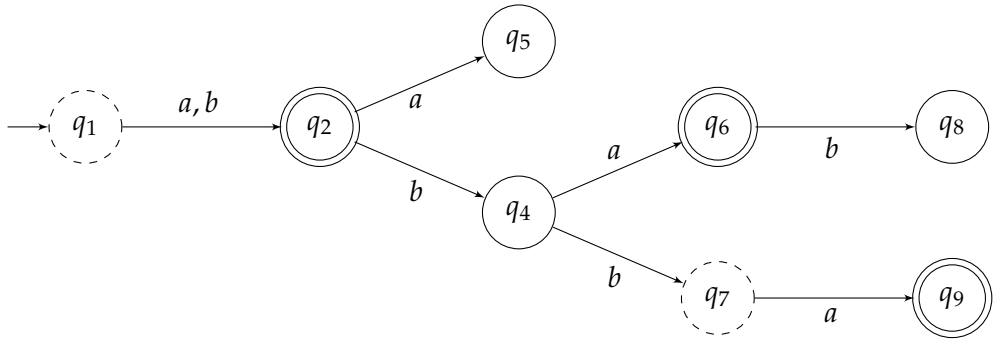

Figura 2.6: Máquina de Moore resultante tras fusionar q_2 y q_3 en el ejemplo 2.3.

Y, sobre esta misma, se podrían hacer aún más fusiones, como la del estado q_1 y q_5, que resultaría en la máquina de estados de la figura 2.7.

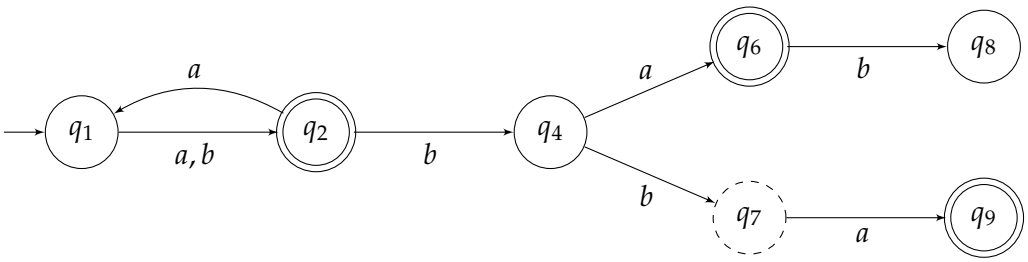

Figura 2.7: Máquina de Moore resultante tras fusionar q_1 y q_5 en el ejemplo 2.3.

2.3 Funcionamiento y estructura del algoritmo GRBM

En el apartado anterior se explicaba la fusión determinista de estados que es pieza clave para este algoritmo, pero ahora se explicará cómo se decide el orden de fusion siguiendo la estrategia Red-Blue.

El funcionamiento del algoritmo es el siguiente: primero genera una Máquina de Moore M tomando las palabras pertenecientes al lenguaje y no pertenecientes al lenguaje de los conjuntos de entrada. A continuación, se crean dos sets R y B, el primero (Red) contiene el estado inicial y el segundo (Blue) contiene aquellos estados que se pueden alcanzar desde el estado inicial [11].

Una vez se tienen los dos conjuntos inicializados se procede a realizar iteraciones donde, en cada una de ellas, R contiene los estados que formarán parte de la salida y

B los estados alcanzables desde los estados de R mientras no estén ya en R. En cada iteración se escoge un elemento arbitrario q del conjunto B y se intenta hacer una fusión determinista con cada estado p del conjunto R, si se pueden fusionar entonces se actualiza M, se recalcula el conjunto B y se repite el procedimiento asignando otra vez q. Si esa fusión no se puede hacer para ningún elemento de R, entonces q se añade al conjunto R, se recalcula el conjunto B y se vuelve a repetir lo realizado escogiendo el elemento q de nuevo. El algoritmo termina cuando el conjunto B resulte vacío.

Lo que se acaba de explicar se resume en el algoritmo siguiente:

Algoritmo 2.3 Algoritmo GRBM.

Entrada: Dos conjuntos finitos: D+ y D-
Salida: Una máquina de Moore consistente

$M := PTMM(D+, D-) = (Q, \sum, \{0, 1, ?\}, \delta, q_0, \Phi)$
$R := \{q_0\}$
$B := \{q \in Q : \delta(q_0, a) = q, a \in \sum\}$
while $B \neq 0$ **do**
  for $q \in B$ (in arbitrary order) **do**
    $merged := false$
    for $p \in R$ (in arbitrary order) **do**
      if $detmerge(M, p, q) \neq M$ **then**
        $merged := true$
        $M = detmerge(M, p, q)$
        break()
      end if
    end for
    if $\neg merged$ **then**
      $R := R \cup \{q\}$
    end if
    $B := \{q \in Q | \delta(p, a) = q, p \in R, a \in \sum\} - R$
  end for
end while
Return M

Ahora se mostrará mediante un ejemplo la estrategia seguida en el algoritmo de forma que se aprecie de una forma más intuitiva el procedimiento del mismo.

Ejemplo 3.4: dados los conjuntos $D^+ = \{a, aba, bab, abba\}$ y $D^- = \{ba, baa, abab, baa\}$, se obtiene la PTMM equivalente, nombrada como M, mostrada en la figura 2.8.

Una vez generado M, se inicializa el conjunto R y B, donde $R = \{q_1\}$ y $B = \{q2, q3\}$, se procede con las iteraciones. La primera iteración asigna de manera arbitraria $q = q_2$ del conjunto B en primer lugar, y después $p = q_1$ del conjunto R. Al no poder completar la fusión entre p y q (explicación en Sección 2.2), se intenta reasignar p con el siguiente valor (en orden arbitrario) de R, pero como no hay más valores se añade $q = q_2$ al conjunto R y se recalcula B.

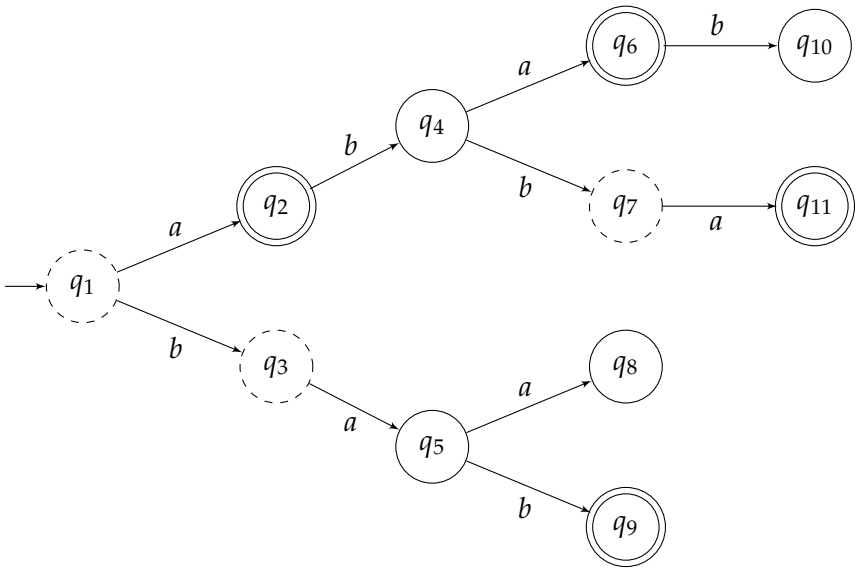

Figura 2.8: M generada al inicio del algoritmo GRBM para el ejemplo 3.4.

En la segunda iteración se tiene $R = \{q_1, q_2\}$ y $B = \{q_3, q_4\}$, y de forma arbitraria se escoge primero $q = q_3$ de B y luego $p = q_1$ de R. Como no se pueden fusionar, se asigna a p otro estado del conjunto R de forma arbitraria, luego obtenemos $p = q_2$. En este caso sí se puede proceder a la fusión y obtenemos una nueva PTMM M, representada en la figura 2.9.

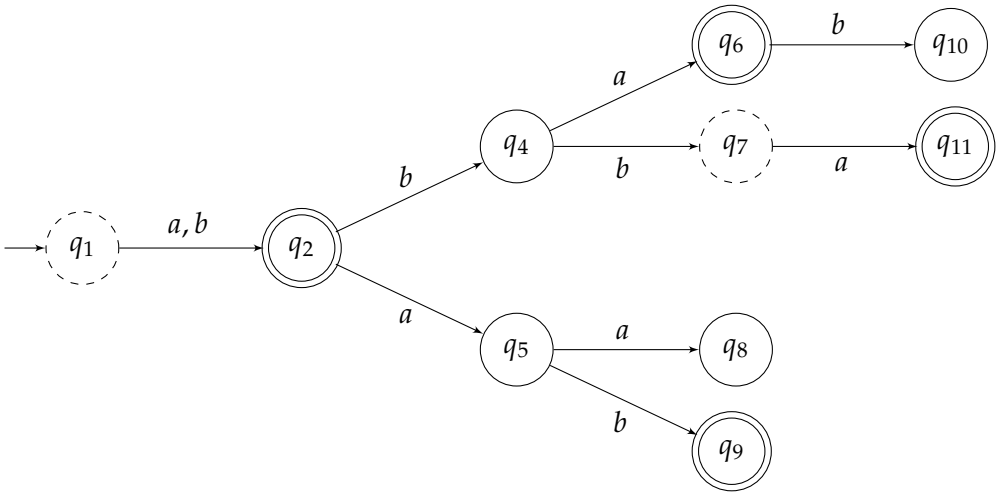

Figura 2.9: M tras la fusión de q_2 y q_3 en el ejemplo 3.4.

En la siguiente iteración, tras recalcular B, se tiene $B = \{q_4, q_5\}$ y R se mantiene igual ($R = \{q_1, q_2\}$). En este caso, primero a q se le asigna el estado q_4 y luego a p el estado q_1. Como no son compatibles de fusionar, se debe coger otro estado de R, y se asigna tal que $p = q_2$. Se comprueba que tampoco es posible una fusión, por tanto, al no quedar más estados por intentar fusionar en R, se añade el estado q a R. En consecuencia, ahora $R = \{q_1, q_2, q_4\}$ y $B = \{q_5, q_6, q_7\}$.

Como B sigue sin ser vacío, se debe seguir iterando. Se escoge arbitrariamente primero $q = q_7$ y después $p = q_4$, y se procede a intentar fusionar ambos estados. Se puede

comprobar que son compatibles y, tras la fusión, se actualiza M, cuyo resultado está representado en la figura 2.10.

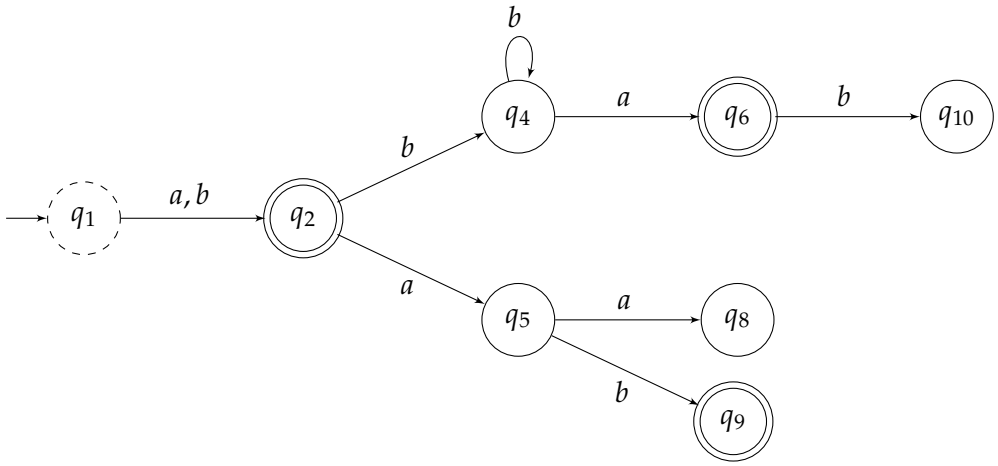

Figura 2.10: M tras la fusión de q_4 y q_7 en el ejemplo 3.4.

Ahora es el momento de recalcular B tras la actualización de M, y se obtiene $B = \{q_5, q_6\}$ (R se mantiene sin cambios). Arbitrariamente se escoge $q = q_6$ y $p = q_2$ y se procede a intentar la fusión de ambos estados. La fusión se efectúa y el resultado se muestra en la figura 2.11.

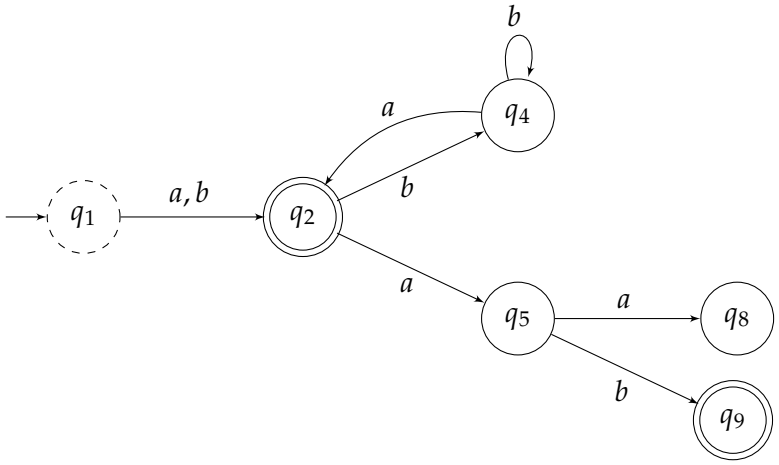

Figura 2.11: M tras la fusión de q_2 y q_6 en el ejemplo 3.4.

Tras esta última fusión, los conjuntos adoptan los siguiente valores: $B = \{q_5\}$ y $R = \{q_1, q_2, q_4\}$. El valor de q en esta iteración es q_5 y se puede comprobar que para cualquier valor de R es imposible completar una fusión, luego q_5 se añade al conjunto R y se recalcula B.

Los nuevos valores de los conjuntos ahora serán: $B = \{q_8, q_9\}$ y $R = \{q_1, q_2, q_4, q_5\}$. A continuación, se asigna q_8 a q de manera arbitraria, y posteriormente se asigna $p = q_5$ (como en todo el algoritmo, también escogido de forma aleatoria). Se puede comprobar que la fusión es posible y su efecto se encuentra en la actualización de M en la figura 2.12.

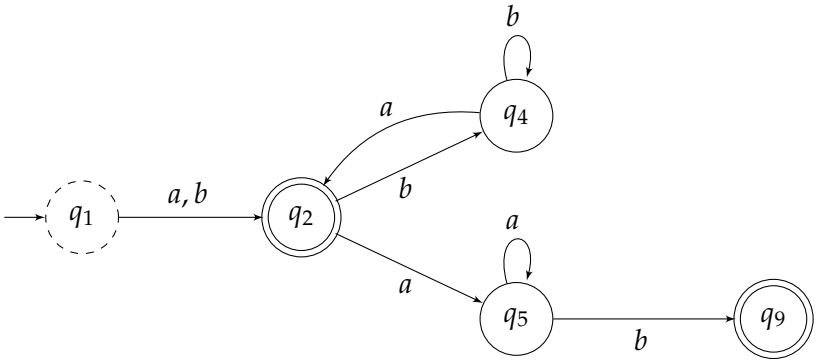

Figura 2.12: M tras la fusión de q_5 y q_8 en el ejemplo 3.4.

Ahora se recalcula B, tal que $B = \{q_9\}$ (R se mantiene igual), y se intenta fusionar q con p tomando como valores de ambas: $q = q_9$ y $p = q_2$. Se efectúa la fusión de manera satisfactoria y, tras actualizar M, se puede comprobar que B queda vacío y el algoritmo termina. Luego la Máquina de Moore M que devuelve el algoritmo GRBM en este ejemplo se puede ver en la figura 2.13.

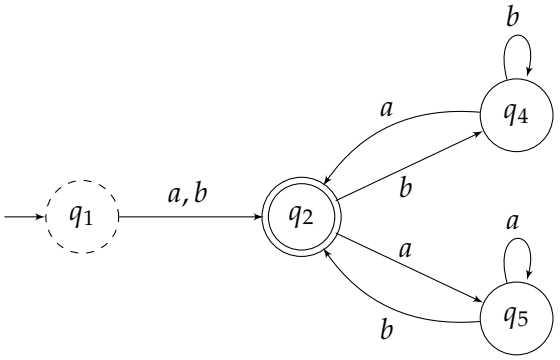

Figura 2.13: M resultante al finalizar el algoritmo GRBM en el ejemplo 3.4.

Como se ha visto en el ejemplo, el modelo resultante depende en gran medida de los estados que se escojan para combinar en cada iteración. Por tanto, como el criterio de orden selección de estados a fusionar es arbitrario, los resultados que se pueden obtener con este algoritmo pueden diferir bastante si se realizan varias ejecuciones sobre el mismo conjunto de datos.

CAPÍTULO 3
Técnicas de muestreo: bootstrapping

Como bien se deduce del título del trabajo, el objetivo del mismo es estudiar la repercusión que tiene utilizar técnicas de *bootstrapping* en el aprendizaje de lenguajes formales. Además, en la parte final del estudio se va a hacer uso de técnicas de *bagging* para el análisis de los modelos generados. Sin embargo, esta última fase es un procedimiento que ya se ha ido realizando en estudios previos aunque en ellos no se reconociera con el propio nombre de *bagging*.

Bootstrapping es un término con múltiples significados y matices según el campo en el que se esté aplicando. Concretamente, en este estudio se concibe dicha técnica como la obtención de subconjuntos escogidos de forma arbitraria sobre un conjunto de datos original sin elementos repetidos.

La técnica de *bagging* (o *bootstrap aggregating*) también tiene diferentes interpretaciones según el área de aplicación. En este estudio se concibe el concepto como la técnica de combinación de una serie de modelos, ponderándolos con unos pesos determinados. En nuestro caso, los modelos a los que se aplicará esta técnica serán los autómatas generados por el algoritmo GRBM, explicado en el capítulo 2, y el resultado será la tasa de acierto del conjunto de autómatas generados.

En la figura 3.1 se muestra un esquema de las partes en las que se aplicarán las técnicas mencionadas en el proceso de aprendizaje de cada uno de los lenguajes objetivo.

Se han explicado las técnicas que se usarán pero no el por qué de hacerlo. Se puede pensar que el objetivo es obtener mejores resultados con menor cantidad de datos, pero no es así, ya que la capacidad de aprendizaje de los algoritmos de inferencia siempre será directamente proporcional a la cantidad de información que reciba en la entrada. Luego, el motivo principal de este estudio es comprobar si es posible aprovechar mejor la información disponible, que suele ser casi siempre escasa.

Como se ha explicado anteriormente, en la fase en la que se aplica la técnica de *bootstrapping* se obtienen subconjuntos aleatorios de un conjunto de datos. Para escoger esa serie de subconjuntos se requiere de unas premisas previas, que en nuestro caso estarán relacionadas con el tamaño de dichos subconjuntos. Como se comentaba en la introducción, este trabajo utiliza una presentación completa de la información, es decir, dentro del conjunto de entrenamiento las palabras estarán etiquetadas como pertenecientes a una de las dos clases: si pertenecen al lenguaje objetivo o si no.

Existen distintas alternativas a la hora de utilizar técnicas de *bootstrapping* como pueden ser la selección de subconjuntos aleatorios sobre el total o una selección con la pre-

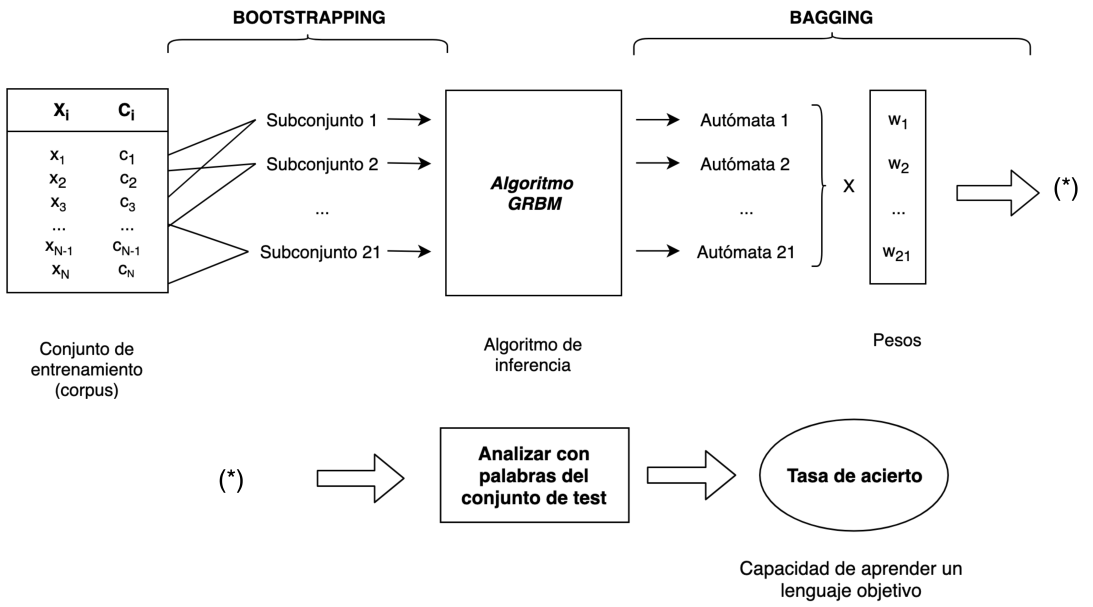

Figura 3.1: Esquema de aplicación de técnicas de muestreo en el proceso de inferencia.

misa de un mínimo de peso, pero hay otras muchas más. En el caso de nuestro estudio, la premisa que se seguirá será la siguiente:

- Se extraerán 21 subconjuntos del conjunto de entrenamiento de los cuales:
  - 10 serán escogidos con todos los elementos que pertenecientes al lenguaje (clase 1) y con el 75 % de los elementos que no pertenezcan al lenguaje (clase 0), escogidos arbitrariamente.
  - 11 serán escogidos con todos los elementos no pertenecientes al lenguaje (clase 0) y con el 75 % de los elementos que sí pertenezcan al lenguaje (clase 1), escogidos arbitrariamente.

De manera que se ilustre mejor lo explicado, se muestra un esquema en la figura 3.2.

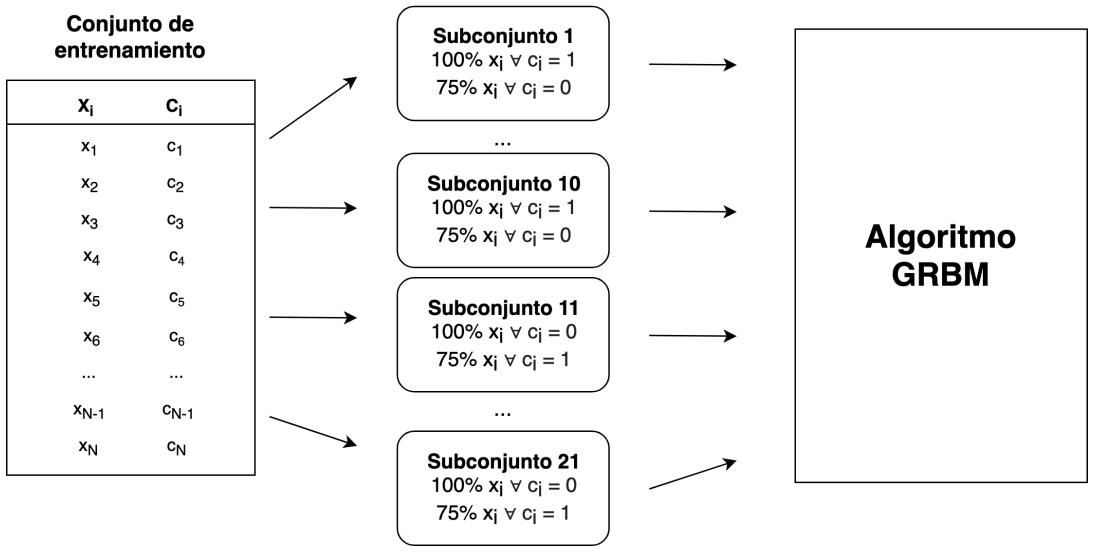

Figura 3.2: Esquema de la fase de *bootstrapping* que se realiza en este trabajo.

Como se mostraba en el esquema general (3.1), en la fase final del proceso de inferencia se aplicará la técnica de *bagging*. Se buscará potenciar los modelos generados por los subconjuntos extraídos en la fase de *bootstrapping* en base a una serie de diferentes criterios. El objetivo final es obtener la mejor tasa de acierto (en el aprendizaje) posible.

Como se comentaba antes, la forma de potenciar unos modelos sobre otros se hará otorgando un peso determinado a cada modelo. Los criterios de ponderación que se considerarán están basados en el criterio de la navaja de Ockham, que considera que los modelos más breves son los más verosímiles, y en nuestro caso un modelo más breve equivale a un autómata más pequeño. Los diferentes criterios de ponderación usados en nuestro trabajo se encuentran en la siguiente lista:

- Voto equitativo, todos los modelos obtenidos (autómatas finitos) tienen el mismo peso.
- Voto ponderado, los modelos tienen el peso inversamente proporcional al tamaño de los mismos (número de estados del autómata). Con lo cual, se potenciarán los autómatas más pequeños.
- Voto ponderado al cuadrado, esta aproximación sigue la misma línea que la anterior pero potencia aún más los autómatas más pequeños, ya que los pesos serán inversamente proporcionales al tamaño de los modelos al cuadrado.
- Voto del autómata más pequeño, el autómata generado con menor número de estados es el único que tiene peso distinto de cero, es decir, es el único que decide.

En la figura 3.3 se muestra de una forma más clara la aplicación de estos criterios dentro de la fase de *bagging*.

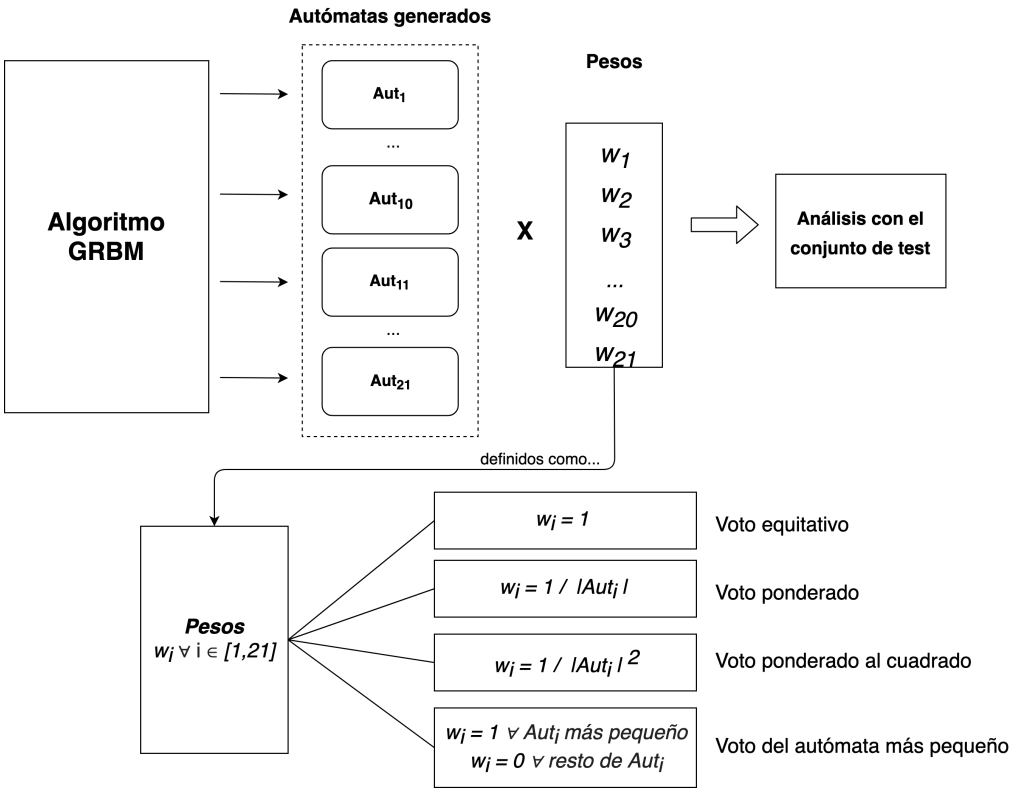

Figura 3.3: Esquema de la fase de *bagging* en el estudio.

Las ponderaciones se aplicarán sobre los modelos generados de manera que el peso otorgado a cada autómata influirá proporcionalmente en la decisión final de si una palabra deberá pertenecer o no al lenguaje objetivo. El análisis se hace con las palabras del conjunto de test, comprobando si la decisión de los autómatas (modelos) generados es acertada o no.

Finalmente, como ya se explica más detalladamente en el capítulo dedicado exclusivamente a la experimentación (Capítulo 4), tras aplicar el proceso de *bagging* con diferentes criterios de ponderación se evalúa con cuál de ellos se obtienen mejores resultados.

CAPÍTULO 4
Experimentación

A lo largo de este capítulo se va a exponer un estudio comparativo entre los resultados obtenidos y los resultados de un estudio previo con el mismo algoritmo que se usa en nuestro estudio (*GRBM*). Es de relevancia comentar que el trabajo se desarrolla mediante el software y lenguaje de programación de *Wolfram Mathematica*.

Dentro del trabajo realizado se encuentran tareas como el desarrollo de módulos en el lenguaje mencionado, además de la creación de un corpus de datos a partir de un conjunto de autómatas dado, y su posterior manejo en el desarrollo del proceso de inferencia. Este conjunto de autómatas se ha obtenido de la *European Conference on Machine Learning 2003*.

4.1 Descripción y uso del corpus

El corpus de datos es un componente muy importante del trabajo, y se compone de los siguientes elementos:

1. Los lenguajes objetivos del aprendizaje, representados por autómatas.
2. Un conjunto de entrenamiento por cada lenguaje objetivo, que contiene un número de palabras determinado y la clase de cada una de ellas (si pertenecen al lenguaje o si no).
3. Un conjunto de test por cada lenguaje objetivo, que contiene también un número de palabras determinado y la clase de cada una de ellas y será utilizado para comprobar la tasa de acierto del método de inferencia posteriormente (conjunto formado por palabras distintas a las del conjunto de entrenamiento).

En este trabajo se han escogido 90 lenguajes objetivo obtenidos del corpus propuesto en el trabajo de F. Coste y D. Fredouille [12]. Si bien la representación de los lenguajes es uniforme (mediante DFA), el corpus distingue 3 lotes dependiendo del método para su generación: autómatas finitos deterministas (DFA), autómatas finitos no deterministas (NFA) y expresiones regulares (ER), y a partir de ellos, se generan y clasifican los conjuntos de entrenamiento y test.

En primer lugar, habrá que generar las palabras de forma arbitraria con las siguientes restricciones:

1. La longitud máxima de cada palabra será de 30 símbolos.

2. Los símbolos utilizados en la generación de palabras deben pertenecer al alfabeto $\{0, 1\}$.

3. La cantidad total de palabras a generar para los conjuntos de entrenamiento será de 500, y de 1000 para el conjunto de test (comprobando en este último conjunto que cada palabra a generar no coincida con alguna del conjunto de entrenamiento).

Posteriormente, se clasifican las palabras una a una (de ambos conjuntos) para cada lenguaje objetivo [1] y se dividen los lotes en varios "sublotes" haciendo cortes progresivos del conjunto de entrenamiento teniendo en cuenta el tamaño (de 100 en 100, hasta 500). En otras palabras, se dispone de lotes de DFA con 100, 200, 300, 400 y 500 palabras en el conjunto de entrenamiento, y lo mismo para NFA y ER. Esto permitirá ver cómo afecta al método de inferencia la cantidad de palabras que se usan para inferir autómatas minimales equivalentes.

Por otro lado, para el conjunto de test siempre se dispone de 1000 palabras, y no se realizan cortes progresivos como es el caso del conjunto de entrenamiento. Toda esta información, tanto los autómatas como los conjuntos, se almacenan en unos ficheros de texto divididos por lotes (cada lote un directorio). Con tal de que se vea de una forma más gráfica, se muestra en la siguiente figura el esquema:

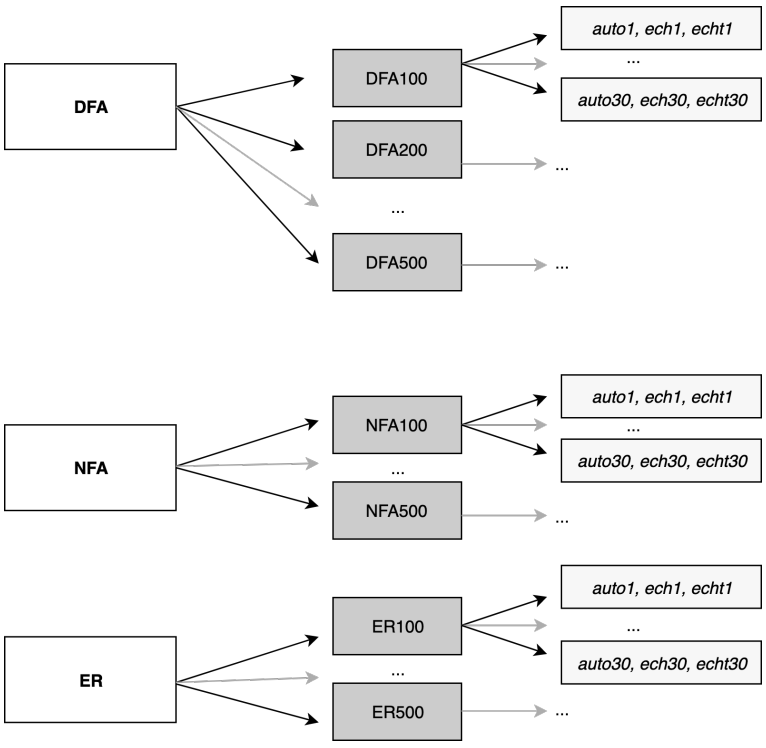

Figura 4.1: Esquema del corpus de datos.

En dicho esquema, *auto* hace referencia al autómata que representa el lenguaje objetivo, *ech* al conjunto de entrenamiento y *echt* al conjunto de test. Como se puede apreciar,

[1] Las palabras del conjunto de entrenamiento son las mismas para todos los autómatas que las clasifican, no se generan conjuntos arbitrarios independientes para cada lenguaje objetivo.

habrá 30 autómatas por cada lote, es decir, 90 autómatas distintos en total[2]. La forma de representar la información en cada fichero se muestra a continuación. En primer lugar, un ejemplo de fichero correspondiente al autómata que representa un lenguaje se encuentra en la figura 4.2.

```
# Alfabeto
{0,1}
# Número de estados
20
# Estados iniciales (precedidos por el número)
2 0 7
# Estados terminales (precedidos por el número)
3 1 9 17
# Descripción de transiciones
42
0 0 0
0 0 2
0 1 3
...
```

Figura 4.2: Fichero que contiene la información correspondiente al autómata que representa el lenguaje objetivo.

En segundo lugar, se muestra en la figura 4.3 un ejemplo de fichero que contiene el conjunto de palabras (de entrenamiento o test) de un lenguaje objetivo. El ejemplo se corresponde al conjunto de entrenamiento de 200 palabras, pero el esquema seguido por el conjunto de test es el mismo.

```
# Número de palabras y clases
200 2
# Palabras del conjunto (clase, tamaño, secuencia de símbolos)
0 8 0 1 1 1 0 1 1 1
0 27 1 0 0 1 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 0 0 0 1
1 11 1 0 0 0 0 0 0 1 1 0 0
0 6 1 0 1 0 0 1
0 18 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 1
0 30 1 1 0 0 0 1 0 0 1 1 1 1 1 1 0 1 0 1 0 0 1 1 0 0 0 1 0 1 0 0
0 1 0
1 16 1 1 1 1 1 0 1 0 1 1 1 0 0 1 0 0
0 17 0 0 0 1 0 1 0 1 1 0 0 0 1 1 1 1 0
0 2 0 1
...
```

Figura 4.3: Fichero que contiene el conjunto de palabras (sea de entrenamiento o de test) de un lenguaje objetivo.

Una vez se dispone de los conjuntos de entrenamiento y test, se procede a la generación de los autómatas equivalentes mediante el algoritmo anteriormente mencionado.

El número de lenguajes objetivos del corpus utilizados para la inferencia del equipo de autómatas es de 30, y el número de autómatas que se generan por cada uno de ellos es de 21 (como ya se mostraba en la figura 3.1 del capítulo 3).

Para generar cada uno de los autómatas se pasa una entrada distinta al algoritmo Generalized Red-Blue Merging (2.3). Como se mostraba en dicho algoritmo, la entrada son dos conjuntos, D^+ y D^-, que representan las palabras que debería aceptar y rechazar, respectivamente, el autómata generado en la salida. Estos 21 subconjuntos que se pasarán

[2]Los autómatas del mismo lote son los mismos independientemente del sublote en el que se encuentren, es decir, el *auto1* de DFA100 será el mismo que el *auto1* de DFA200 o DFA500, pero diferente a los de NFA y ER, ya que son tipos de autómatas distintos.

como entrada al algoritmo, serán producto de aplicar técnicas de *bootstrapping* sobre el conjunto de entrenamiento (como se explicaba también el capítulo 3).

Tras la obtención de los 21 autómatas para cada uno de los lenguajes objetivos, se procede a analizar la calidad de estos modelos con los conjuntos de test. Esta fase de análisis consiste en comprobar, para cada palabra del conjunto, si el autómata que ha generado el algoritmo acierta respecto a si la palabra pertenece al lenguaje objetivo. Posteriormente, se calcula la media para obtener la tasa de acierto del autómata en cuestión.

Después de obtener la tasa de acierto de cada autómata, se aplican técnicas de *bagging* otorgando un peso determinado a cada uno de los autómatas de cada lenguaje objetivo (como se muestra en el esquema 3.3 del capítulo 3), para así obtener la tasa de acierto media para cada lenguaje objetivo aplicando los diferentes sistemas de voto o ponderación propuestos para este estudio: voto equitativo, voto ponderado, voto ponderado al cuadrado y voto del autómata más pequeño.

Una vez se tiene la tasa de acierto de cada lenguaje objetivo de cada lote, se hace una media para sacar la tasa de acierto media de cada lote. Posteriormente, se generan una serie de tablas y gráficas para establecer comparativas entre los resultados de este estudio y los de estudios previamente realizados con otros algoritmos de inferencia.

Es de relevancia comentar que por el tiempo disponible para la realización de un trabajo de fin de grado se han tenido que hacer una serie de limitaciones frente al estudio previo con el que se harán comparativas posteriormente. En primer lugar, no se disponen del mismo número de autómatas en los equipos generados pues aumentaba considerablemente el tiempo de espera hasta obtener resultados. En segundo lugar, no se obtienen resultados con un tamaño del conjunto de entrenamiento acorde con el utilizado, ya que se usan técnicas de *bootstrapping* que reducen las dimensiones de este conjunto. Estos factores suponen una desventaja importante que será considerada a la hora de realizar las valoraciones finales.

4.2 Resultados

En esta sección se van a exponer los resultados de la experimentación. En primer lugar, se van a mostrar los resultados obtenidos de un artículo en el que se expone un estudio anterior [11]. Este estudio demuestra que el algoritmo GRBM aporta resultados positivos en el área de la inferencia gramatical.

Se disponen en la tabla 4.1 los resultados obtenidos en dicho artículo, donde se muestra la aplicación del algoritmo (*GRBM*) utilizando equipos de 81 autómatas. Se muestran los resultados tras el análisis de los equipos con sistemas de voto ponderado ($v.p.$), de voto del autómata más pequeño ($\downarrow Aut$), y con sistemas de voto ponderado al cuadrado ($v.p.^2$) teniendo en cuenta tanto todos los autómatas del equipo como sólo los que tengan tamaño ($\#Aut$) inferior a la media.

En la tabla 4.2 se muestran los resultados obtenidos en nuestro estudio con las técnicas mencionadas: voto equitativo ($v.e.$), voto ponderado ($v.p.$), voto ponderado al cuadrado ($v.p.^2$) y voto del autómata más pequeño ($\downarrow Aut$).

Set	GRBM (81 Aut)					
	% $v.p.$	% $\downarrow Aut$	$\lvert\downarrow Aut\rvert$	sin selec.	selec. $\lvert Aut\rvert$ inferior a la media	
				$v.p.^2$	$v.p.^2$	# Aut
er100	94,50	94,64	7,28	95,19	95,55	50,00
er200	98,14	97,94	7,85	98,32	98,45	58,24
er300	98,67	98,59	8,39	98,78	98,87	60,58
er400	99,16	99,02	8,54	99,24	99,32	63,45
er500	98,14	99,27	8,75	99,42	99,49	65,91
nfa100	77,08	71,23	17,11	77,24	77,45	40,51
nfa200	80,70	74,80	28,18	80,96	81,25	41,28
nfa300	83,19	77,14	37,26	83,46	83,77	41,26
nfa400	85,01	79,01	44,95	85,14	85,50	41,47
nfa500	86,57	80,65	51,84	86,71	86,98	42,25
dfa100	76,45	70,32	17,21	76,68	76,83	40,48
dfa200	82,10	80,69	24,66	83,13	84,04	36,00
dfa300	88,47	91,29	23,02	90,04	91,59	36,00
dfa400	93,37	96,62	19,60	95,24	96,48	36,31
dfa500	96,76	98,84	17,99	98,16	98,68	37,69

Tabla 4.1: Resultados con el algoritmo GRBM en el estudio anterior.

Set	GRBM (21 Aut, con bootstrapping)			
	% $v.e.$	$v.p.$	$v.p.^2$	$\lvert\downarrow Aut\rvert$
er100	75,93	77,72	78,92	83,86
er200	83,17	85,07	86,20	89,66
er300	85,71	87,20	88,23	90,93
er400	89,10	90,67	91,75	93,65
er500	91,10	92,81	94,03	95,67
nfa100	62,93	63,15	63,35	65,42
nfa200	65,33	65,75	65,94	67,62
nfa300	66,74	67,11	67,48	69,67
nfa400	68,15	68,44	68,74	71,32
nfa500	69,05	69,45	69,80	72,69
dfa100	61,28	61,39	61,50	63,38
dfa200	63,18	63,54	63,88	66,65
dfa300	66,46	67,07	67,61	69,49
dfa400	68,72	69,30	69,86	74,06
dfa500	69,85	71,05	72,59	78,16

Tabla 4.2: Resultados con el algoritmo GRBM en nuestro estudio.

A continuación se muestran los resultados obtenidos en nuestro estudio comparados con los mejores resultados del estudio anterior (tablas 4.1 y 4.2). Dicha comparativa se separa en una serie de tablas diferentes según la técnica de *bagging* aplicada en el análisis. El orden de las tablas según el sistema de voto empleado es el siguiente: en la 4.3 con voto ponderado ($w.v.$), en la 4.4 con voto ponderado al cuadrado ($w.v.^2$) y, en último lugar, la tabla 4.5 con los resultados con voto del autómata más pequeño ($\downarrow Aut$). No se aporta tabla comparativa para el sistema de voto equitativo pues en el estudio anterior no se muestran resultados utilizando esta técnica.

Set ($\%v.p.$)	Anterior estudio	Este estudio
er100	94,50	77,72
er200	98,14	85,07
er300	98,67	87,20
er400	99,16	90,67
er500	98,14	92,81
nfa100	77,08	63,15
nfa200	80,70	65,75
nfa300	83,19	67,11
nfa400	85,01	68,44
nfa500	86,57	69,45
dfa100	76,45	61,39
dfa200	82,10	63,54
dfa300	88,47	67,07
dfa400	93,37	69,30
dfa500	96,76	71,05

Tabla 4.3: Resultados en nuestro estudio usando sistema de voto ponderado.

Set ($\%v.p.^2$)	Anterior estudio	Este estudio
er100	95,19	78,92
er200	98,32	86,20
er300	98,78	88,23
er400	99,24	91,75
er500	99,42	94,03
nfa100	77,24	63,35
nfa200	80,96	65,94
nfa300	83,46	67,48
nfa400	85,14	68,74
nfa500	86,71	69,80
dfa100	76,68	61,50
dfa200	83,13	63,88
dfa300	90,04	67,61
dfa400	95,24	69,86
dfa500	98,16	72,59

Tabla 4.4: Resultados en nuestro estudio usando sistema de voto ponderado al cuadrado.

Set (% ↓ *Aut*)	Anterior estudio	Este estudio
er100	94,64	83,86
er200	97,94	89,66
er300	98,59	90,93
er400	99,02	93,65
er500	99,27	95,67
nfa100	71,23	65,42
nfa200	74,80	67,62
nfa300	77,14	69,67
nfa400	79,01	71,32
nfa500	80,65	72,69
dfa100	70,32	63,38
dfa200	80,69	66,65
dfa300	91,29	69,49
dfa400	96,62	74,06
dfa500	98,84	78,16

Tabla 4.5: Resultados en nuestro estudio usando el voto del autómata más pequeño.

Y ahora se muestran unas gráficas comparativas que resumen los resultados de una forma más visual. Hay una gráfica para cada lote de representación de lenguajes sobre el que se ha hecho el estudio: ER (expresiones regulares, en la figura 4.4), NFA (autómatas finitos no deterministas, en la figura 4.5), DFA (áutomatas finitos deterministas, en la figura 4.6). En estas gráficas, se muestra la variación (negativa) de ratio de reconocimiento de nuestro estudio respecto al estudio anterior.

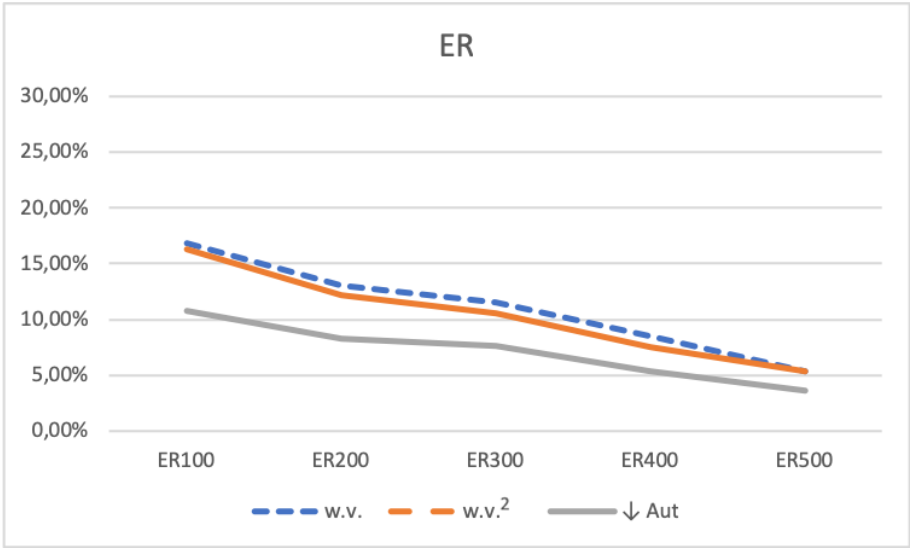

Figura 4.4: Gráfica comparativa de resultados con lenguajes representados por expresiones regulares.

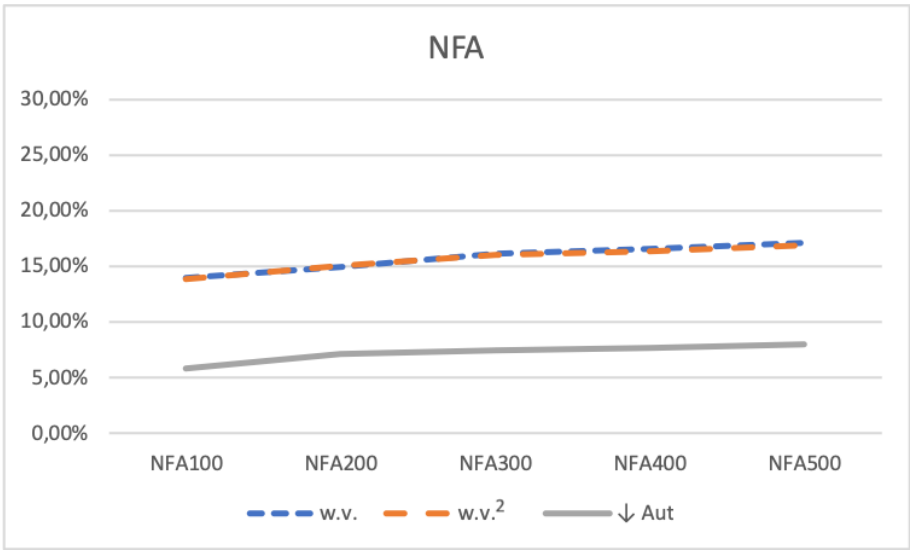

Figura 4.5: Gráfica comparativa de resultados con lenguajes representados por autómatas finitos no deterministas.

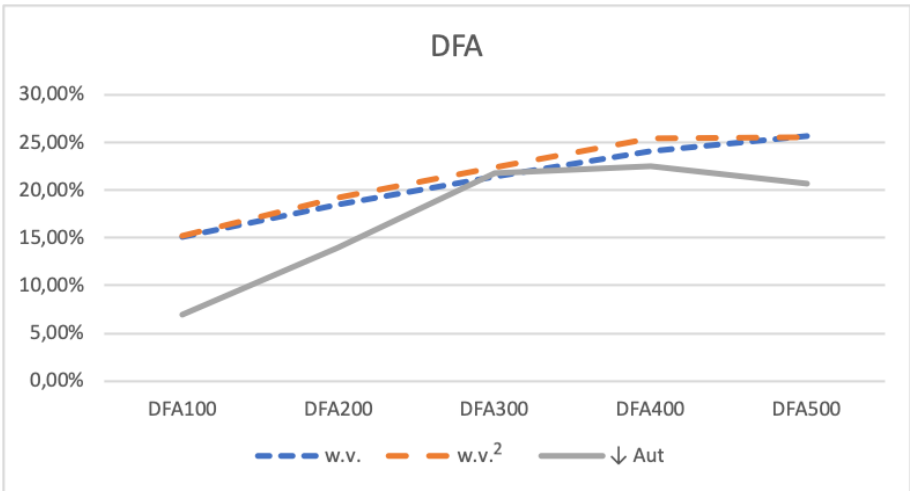

Figura 4.6: Gráfica comparativa de resultados con lenguajes representados por autómatas finitos deterministas.

Tras la lectura y el correspondiente análisis de los resultados obtenidos, se puede comprobar que, dentro del conjunto de diferentes técnicas de *bagging* usadas en la fase de análisis de los equipos de autómatas generados, la que usa el sistema de voto del autómata más pequeño es la técnica con la que se obtienen los mejores resultados en la mayoría de los casos. Esto demuestra que, como indica el principio de *la navaja de Ockham*, la explicación más simple es la más probable.

Sin embargo, aún siendo ésta la técnica con la que se obtienen los mejores resultados de nuestro estudio, se puede apreciar que prácticamente en ningún lote los resultados se acercan a los mejores resultados obtenidos con el algoritmo GRBM sin usar técnicas de *bootstrapping*. Para que los resultados fueran positivos deberíamos tener una diferencia de aproximadamente un 5 % en la tasa de acierto, sin embargo, en la mayoría de los casos se obtienen resultados superiores a ese umbral. De todos modos, algunos hechos que se deben destacar son los siguientes:

- En el caso del lote NFA (autómatas finitos no deterministas) se obtienen resultados considerablemente más cercanos al estudio anterior con el sistema de voto del autómata más pequeño que con los demás.
- En el caso de ER (expresiones regulares), los resultados a partir de conjuntos de entrenamiento de 400 palabras obtienen una variación (negativa) inferior al 5 % respecto a los resultados sin *bootstrapping*, que podría considerarse como un resultado positivo.

Pese a estos puntos, los resultados en líneas generales no se acercan de forma suficiente a los resultado sin aplicar técnicas de *bootstrapping*, luego se concluye que el estudio realizado no aporta optimismo en el uso de este tipo de técnicas. Sin embargo, se podría considerar que las limitaciones mencionadas en la sección previa (ver último párrafo de la sección 4.1) pueden haber influido negativamente en nuestros resultados.

CAPÍTULO 5
Conclusiones

En este trabajo de fin de grado se aborda un estudio sobre el uso de técnicas de *bootstrapping* en procesos de aprendizaje de lenguajes formales. En el área de la inferencia gramatical se habían hecho estudios hasta ahora proponiendo, en su mayoría, variaciones a los algoritmos usados en procesos de inferencia de estudios previos. Sin embargo, hasta ahora no se había planteado la posibilidad de aplicar técnicas que redujeran el tamaño de los datos de entrada del algoritmo pretendiendo aprovechar mejor la información, casi siempre escasa, de la que se dispone.

La hipótesis de partida era la de estudiar si la aplicación de técnicas de *bootstrapping* sobre el conjunto de entrenamiento aportaba conclusiones positivas en términos de capacidad de aprendizaje en los métodos de inferencia gramatical. Para que las conclusiones fueran positivas, se debían tener resultados cercanos (al menos en un 5 %) a los obtenidos sin usar este tipo de técnicas.

Dado el ámbito del trabajo, en el que se presupone un tiempo de desarrollo de aproximadamente 300 horas, se tuvo que hacer una serie de limitaciones en el estudio como ha sido la reducción del tamaño de los equipos de autómatas. Otro aspecto a considerar que influye también en nuestro trabajo es la diferencia en el tamaño de los conjuntos de entrenamiento comparados. Al aplicar *bootstrapping*, el tamaño de los mismos se reduce y esta reducción limita la capacidad de aprendizaje, luego es lógico que la tasa de acierto de nuestros equipos de autómatas sea inferior. Sin embargo, aún teniendo en cuenta estas desventajas importantes, los resultados no son positivos pues son demasiado inferiores en términos de reconocimiento. En resumen, se sacan las siguientes conclusiones:

1. El uso de técnicas de *bootstrapping* no logra resultados suficientemente cercanos a los obtenidos en estudios previos que no las aplicaban.
2. Se descarta que la aplicación de este tipo de técnicas aporte conclusiones que conduzcan al optimismo sobre su aplicación en el campo de la inferencia gramatical.

En principio se podía pensar que el uso de este tipo de técnicas abriría nuevas puertas en un área de investigación como es el del aprendizaje de lenguajes formales, sin embargo, el estudio nos permite comprobar que la información no es más aprovechable usando técnicas de *bootstrapping* de lo que ya es sin su aplicación, luego la hipótesis de partida era errónea.

Bibliografía

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

[2] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[3] B.A. Trakhtenbrot and Ya. M. Barzdin. *Finite automata. Behavior and Synthesis.* North-Holland Pub. Co., 1973.

[4] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[5] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Control*, 75:87–106, 1987.

[6] D. Angluin. Negative Results for Equivalence Queries. *Machine Learning*, 5(2):121–150, 1990.

[7] J. Oncina and P. García. *Pattern recognition and image analysis, volume 1, chapter Inferring regular languages in polynomial updated time, pages 49–61.* World Scientific, 1992.

[8] K.J. Lang. *Random DFA's can be approximately learned from sparse uniform examples. In Proceedings of the fifth annual workshop on Com- putational learning theory, pages 45–52.* 1992.

[9] C. de la Higuera, J. Oncina, and E. Vidal. Identification of dfa: data- dependent vs data-independent algorithms. *LNAI*, 1147:313–325, 1996.

[10] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *LNAI*, 1433:1–12, 1998. 4th International Colloquium, ICGI-98.

[11] P. García, M. Vázquez de Parga, D. López, J. Ruiz. Learning Automata Teams. *LNAI 6339*, 52–65, septiembre, 2010.

[12] F. Coste, D. Fredouille. Unambiguous Automata Inference by Means of State-Merging Methods. *LNAI 2837,* pp. 60–71, 2003.