Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Generación de azar en humanos: modelo computacional

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Sergio Gastón Romano

Directores:

- Figueira, Santiago - sfigueir@dc.uba.ar
- Sigman, Mariano - sigman@df.uba.ar

Jurados:

- Agustín Gravano - gravano@dc.uba.ar
- Guillermo Cecchi - gcecchi@us.ibm.com

Buenos Aires, 2012

GENERACIÓN DE AZAR EN HUMANOS

Diversos estudios han afirmado que los seres humanos somos incapaces de generar secuencias aleatorias. Sin embargo, poco se ha avanzado en construir un modelo formal que permita estudiar cómo funciona el proceso de intentar generar cadenas al azar en nuestras mentes y cómo varía ante diversos contextos o características poblacionales. Nuestro trabajo propone un modelo computacional basado en las definiciones de complejidad de Kolmogorov, la aleatoriedad de Martin-Löf y la inducción de Solomonoff. Si bien estas herramientas teóricas convierten al problema de definir la aleatoriedad en un problema no computable, trasladaremos estas nociones a un modelo computable.

Luego, verificaremos este modelo en dos experimentos de generación de secuencias aleatorias con personas y comprobaremos cómo este lenguaje formal se muestra efectivo para diferenciar las secuencias generadas por humanos, de aquellas generadas por algoritmos pseudo-aleatorios o por ruido blanco. Mostraremos también su utilidad para medir efectos tales como la fatiga o la estabilidad de la calidad de las cadenas aleatorias para un participante en diversos intentos.

Palabras claves: Aleatoriedad, Generación y Percepción del Azar, Complejidad de Kolmogorov, Aleatoriedad de Martin-Löf, Inducción de Solomonoff.

RANDOM GENERATION IN HUMANS

Several studies have claimed that humans are unable to generate random sequences. However, little progress has been made in building a formal model that allows us to study how this process of generation of random strings works in our mind and how it changes with different contexts or population features. Our work proposes a computational model based on definitions such us Kolmogorov complexity, Martin-Löf randomness and Solomonoff induction. While these theories make the problem of defining randomness not computable, we will transfer these notions to our own computable model.

Later, we will verify this model in two experiments about random sequences generation with people, and we will prove how this formal language is effective to differentiate the sequences generated by humans, from those produced by pseudo random number generators or by white noise processes. We will also show its usefulness for measuring effects such as fatigue or random quality stability of strings produced by participants in different attempts.

Keywords: Randomness, Random generation and perception, Kolmogorov Complexity, Martin-Löf Randomness, Solomonoff induction.

AGRADECIMIENTOS

A mi hermana, por siempre ser un ejemplo y una dulce compañía. Creo que nunca supo cuanto admiraba sus logros en el secundario y en la universidad, aprovecho esta hoja para dejárselo en claro.

A mi papá y a mi mamá, quienes trabajaron mucho toda su vida para que nunca me falte nada, incluso en épocas difíciles. Gracias a ellos tuve mi primer computadora que me trajo hasta esta tesis, y por sobre todo, gracias a ellos aprendí la importancia de ser libre y seguir los sueños que me hacen feliz.

A toda mi familia, por haber soportado mis ausencias entre el trabajo, el estudio, la militancia y la tesis. Los casamientos y los nacimientos hacen que cada día seamos más y no pueda nombrarlos a todos sin cometer alguna torpeza. No duden que todos están incluidos en este renglón. Sí me gustaría mencionar a mi abuela quien siempre me pidió sólo una cosa. Hoy te cumplo esa promesa, espero que lo estés viendo.

A mis grandes amigos que me acompañan desde el secundario: Abuelo, Dieguito, Gonza, Indio, Juan, Kultor, Nico, Pepo, Pía, Pichu, Seba y Yagui, gracias por compartir secretos, locuras y alegrías cada día en nuestra *Pizarra de Noticias*.

A todos mis profesores del secundario. Especialmente a los dos que mejor representan las decisiones que tomé en mi vida: Nicolas Triolo, quien en su primer clase de historia nos leyó la introducción de las *Venas Abiertas de Latinoamérica* y luego la respuesta de Vargas Llosa en su *Manual del Perfecto Idiota Latinoamericano*. Ese debate, marcó por siempre mi manera de entender la realidad. Y a Gonzalo Zabala, quien no sólo me contagió el gusto por seguir esta carrera en la facultad de Exactas, sino también me mostró que para conquistar sueños descomunales sólo se necesita animarse a montar a Rocinante.

A Andy quien en estos largos años de trabajo en Assembla me enseñó todo lo que no se puede aprender en la universidad sobre el mundo de la computación y de los negocios. Gracias por tu locura y por tu confianza. Fue un gusto enorme haber trabajado tantos años a la par tuya. Y no puedo dejar de mencionar al otro gran amigo que me dio Assembla: Paco. Gracias por acompañarme día a día, darme ánimos y enseñarme el mundo del diseño, un mundo totalmente desconocido para mí que sólo con tu ayuda pudo ser tan sencillo de descubrir.

A todos los profesores de la facultad. Especialmente, a Fernando Schapachnik y a Carlos Diuk, quienes sin saberlo, me devolvieron con sus clases la pasión por esta profesión cuando más lo necesitaba.

A Herman y a Kivi, que me acompañaron y ayudaron en las últimas materias y finales de la carrera. Y obviamente, a Pato y Alejandro, los otros dos grandes amigos que me dio esta facultad con quienes pasamos noches enteras preparando entregas, y días enteros descubriendo juntos por donde caminar en este mundo de la computación.

A mis amigos del grupo En La Tecla que insisten, contra viento y marea, en levantarse temprano todos los sábados para hacer de este un mundo más justo y contagiar el placer

por desentrañar los misterios de las computadoras entre quienes más lo necesitan. Gracias Pablo y Sole por haber empezado a hacer rodar esa aventura colectiva.

A todos los argentinos que día a día, con su esfuerzo y trabajo, sostienen este modelo de país y me permitieron tener una educación pública de excelencia. Espero poder devolverles algo de todo lo que me dieron.

A Néstor y a Cristina, que con sus aciertos y errores devolvieron a mi generación el valor de la política. Gracias a ellos y a todos los que intentan hacer de este un país más justo, equitativo e inclusivo. Especialmente, a mis amigos y compañeros de Nuevo Encuentro.

A Santiago y a Mariano, mis directores de tesis, por haberme dedicado tanto de su tiempo y de su conocimiento. Gracias nuevamente por la humildad y el compromiso con el que me acompañaron en estos meses. Otra vez, gracias.

Por último a Alejandra, quien supo estar a mi lado soportando alegrías, malhumores y tristezas a lo largo de tantos años. Sin su ayuda, esta tesis nunca hubiera terminado. Sin su amor, no hubiera descubierto el otro lado de las cosas.

A Alejandra, mi compañera de la vida

DILBERT *By* Scott Adams

Índice general

1. INTRODUCCIÓN

Desde que en 1934 Reichenbach [Rei34] afirmó que las personas somos incapaces de generar secuencias aleatorias, distintas investigaciones han estudiado el fenómeno y coincidido en destacar las limitaciones de la mente humana para producir o detectar cadenas aleatorias (ver [Wag72] o [Nic02] por una revisión de distintos estudios sobre el tema). Sin embargo, pocos estudios (entre los que se puede destacar [Gri01]) han avanzado en modelar el proceso que seguimos al intentar producir dichas secuencias.

Estudios recientes de ingeniería reversa del aprendizaje humano y su desarrollo cognitivo entienden al aprendizaje como un tipo de problema computable, y a la mente humana como una máquina natural que ha evolucionado para resolverlos (ver [Ten11]). Dichos estudios coinciden en explicar al aprendizaje como modelos probabilísticos jerárquicos basados sobre estructuras simbólicas formales para representar el conocimiento tales como: grafos, gramáticas, esquemas relacionales, programas funcionales, lógica de primer orden. Qué estructura utilizar para captar cómo representamos el conocimiento depende del dominio y la tarea que estemos estudiando.

El objetivo de nuestra tesis es introducir un modelo que permita un mayor entendimiento del proceso de generación de secuencias aleatorias en los seres humanos y de las estructuras simbólicas formales que captan cómo representamos y entendemos al azar.

Para lograr nuestro cometido, presentaremos en el capítulo 2 la noción de Complejidad de Kolmogorov (sección 2.1). Dicha función nos permitirá definir la complejidad de las secuencias de manera universal y estudiar su relación con el concepto de aleatoriedad de Martin-Löf (sección 2.2) que define lo que entendemos por azar tanto para secuencias finitas como infinitas. En la sección 2.3, presentaremos la inducción de Solomonoff, la cual nos brindará una distribución universal para predecir cualquier secuencia generada por un proceso computable, y servirá de base para nuestro modelo de predicción de las secuencias generadas por las personas.

Sin embargo, el problema principal de todas las herramientas teóricas que presentaremos es que convierten a la noción de complejidad de una secuencia, su aleatoriedad o la predicción del siguiente bit, en problemas no computables. En el capítulo 3, buscaremos trasladar estas nociones a un modelo computable que permita entender la algoritmia que utilizan los seres humanos para representar y generar azar. Introduciremos la hipótesis de cómo podría ser la gramática que las personas utilizan para representar al azar, la cual da lugar a una familia de funciones de complejidad computables que varían en distintos parámetros ajustables. A su vez, cada una de estas funciones de complejidad nos permitirá instanciar un algoritmo distinto de generación de azar para predecir los caracteres de las secuencias generadas por las personas y que también contará con sus propios parámetros ajustables.

En el capítulo 4, mostraremos los primeros intentos por entender y calibrar los parámetros del algoritmo de predicción y del de complejidad antes de ser utilizados en los experimentos para medir la *aleatoriedad* de los participantes.

En el capítulo 5, presentaremos el diseño del primer experimento que hemos desarrollado. Los resultados obtenidos en dicho experimento no fueron los esperados, motivo por el cual, presentamos también un análisis sobre las posibles fallas en el diseño que nos llevaron a obtener esos resultados.

Los cambios al primer experimento son presentados en el capítulo 6, allí explicaremos el nuevo experimento que corrige las principales falencias del anterior y, en este caso, veremos una mejora significativa en los resultados que nos permitirá profundizar en el análisis de los datos obtenidos. Estudiaremos entonces las cadenas generadas por los participantes, la complejidad de estas según nuestro modelo, los efectos de la fatiga y cómo nuestro predictor nos da una evidencia del modelo que utilizamos los seres humanos para representar el azar en nuestras mentes e intentar generar secuencias aleatorias.

2. MARCO TEÓRICO

Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin
–John von Neumann, Various Techniques Used in Connection with Random Digits, 1949

2.1. Complejidad de Kolmogorov

2.1.1. Intuición

Supongamos que queremos describir a un objeto con un lenguaje de descripción y que cada descripción describe a un sólo objeto. De todas las descripciones válidas en el lenguaje para describir a un objeto, hay una que será la más corta. Sería intuitivo decir que un objeto es *simple* si puede ser descripto por una descripción corta, y que un objeto es *complejo* si requiere de una descripción mucho más larga. Por ejemplo, el número 10^{100} podría decirse que es simple ya que lo podemos describir con la descripción *"diez elevado a la cien"*. Sin embargo, para otros números con la misma cantidad de cifras necesitaríamos más símbolos para describirlos.

Si nos detenemos a pensar en lo anterior, podríamos notar que la complejidad de un objeto, es decir, el costo de describirlo puede ser favorecido por el lenguaje de descripción que elijamos. A priori, esto es un problema ya que buscamos una medida de complejidad que dependa de los objetos en sí y no del lenguaje de descripción elegido. Veremos que la teoría de la complejidad de Kolmogorov nos facilita un lenguaje de descripción universal que cumplirá nuestras necesidades.

Nos interesa formalizar la noción de descripción para evitar caer en contradicciones como la de la paradoja de Berry. Según esta paradoja, existe una contradicción en la frase: *el menor entero positivo que no se puede definir con menos de quince palabras*. Notemos que existe un conjunto finito de enteros positivos, o naturales, que se pueden describir con menos de quince palabras. Como es un conjunto finito hay, al menos, un entero positivo que no está contenido en dicho conjunto. De todos los números naturales que no se pueden describir con menos de quince palabras, uno de ellos es el menor. Este número es entonces *el menor entero positivo que no se puede definir con menos de quince palabras*. Sin embargo, dicha frase contiene catorce palabras. Por lo tanto, lo hemos definido con menos de quince palabras.

2.1.2. Formalización

Denotemos a S como el conjunto de los objetos a describir y asumamos que existe una función $n : \ S \to \mathbb{N}$ que asigna a cada objeto $x \in S$ un número natural $n(x)$. Estamos interesados en ver si existen maneras más cortas de describir a x que utilizando $n(x)$.

Para poder describir a los objetos necesitaremos un método de descripción o especificación. Supongamos que existe una cantidad finita m de funciones parciales $f_1, f_2, \cdots, f_m$ donde cada $f_i : \mathbb{N} \to \mathbb{N}$ representa un método distinto de especificación. Fijado un f_i, buscamos para cada $x \in S$ un $p \in \mathbb{N}$ que lo describa según el método de descripción f_i, es decir, un p tal que $f_i(p) = n(x)$.

La complejidad de un objeto $x \in S$ con respecto al método de especificación f_i estará dada por la función $C_{f_i} : S \to \mathbb{N}$.

$$C_{f_i}(x) := \begin{cases} \text{mín}\{p : f_i(p) = n(x)\} & \text{si existe tal } p \\ \infty & \text{si no} \end{cases}$$

Definiremos ahora una función general de complejidad $C_f : S \to \mathbb{N}$ que no dependa del f_i elegido, de manera que sólo exceda en una constante c ($c \approx \log m$) al mínimo $C_{f_1}(x), C_{f_2}(x), \cdots, C_{f_m}(x)$.

$$\text{En este caso, } C_f(x) := \begin{cases} \text{mín}\{p : f(p) = n(x)\} & \text{si existe tal } p \\ \infty & \text{si no} \end{cases}.$$

Para definir C_f necesitamos entonces definir al método de descripción general $f : \mathbb{N} \to \mathbb{N}$. Sea $C_{f_j}(x)$ la mínima complejidad para un objeto $x \in S$, esta complejidad está asociada al método de descripción f_j. El método general $f(p)$ reservará entonces los $\lfloor \log m \rfloor$ bits más significativos de p para indicar cuál es método de especificación f_j que se debe seguir y los restantes bits de p se utilizarán como el parámetro para ese f_j de manera de poder describir a x.

De esta manera, vale que $\forall i,\ C_f(x) \leq C_{f_i}(x) + c$, ya que $C_f(x)$ se quedará con el método de especificación más pequeño que describa a x dentro de los posibles f_i, al que se le agrega un costo c para codificarlo como describimos anterioremente en el método de descripción general f. Si tomásemos una cantidad infinita numerable de métodos de especificación $f_1, f_2, \cdots$, tendríamos que $\forall i\ \exists c_i, \forall x\ C_f(x) \leq C_{f_i}(x) + c_i$.

Consideremos ahora el problema particular de describir a las cadenas binarias en términos de programas que son, a su vez, secuencias binarias. Tenemos entonces $S = \{0,1\}^*$ y $f_i : \{0,1\}^* \to \{0,1\}^*$. Nos interesa remarcar que la restricción a un alfabeto binario no implica pérdida de generalidad. Cambiar el tamaño del alfabeto mantiene las demostraciones invariantes hasta un factor logarítmico multiplicativo dependiente del tamaño del nuevo alfabeto (ver [LiV08, Cap 2.1]).

La idea de la complejidad de Kolmogorov para este problema es tomar a las máquinas de Turing como los métodos de especificación f_i. Dado que el conjunto de las máquinas de Turing es un conjunto infinito numerable, tomemos la numeración $M_1, M_2, \cdots$ de manera que $M_i = f_i$, es decir, cada f_i será la función parcial computada por la i-ésima máquina de Turing. Como el dominio de M_i (o f_i) está definido sobre programas que son cadenas finitas de 0's y 1's, necesitamos una medida para seleccionar al *mínimo* programa p que computa x en M_i sin parámetros adicionales, para esto utilizaremos la longitud de los programas. Entonces, $C_{M_i}(x) : \{0,1\}^* \to \mathbb{N}$ estará definido como la menor longitud de un programa p que computa x en M_i sin parámetros adicionales.

Para definir la complejidad general $C_U : \{0,1\}^* \to \mathbb{N}$, trabajaremos con una máquina universal de Turing U, es decir, de todas las C_{M_i} definidas, nos interesa aquella donde $M_i = U$. La máquina U espera un input p' que puede entenderse como la codificación de

una tupla $\langle i, p\rangle$ tal que la primera componente es un código auto-delimitado que codifica la máquina M_i (que será simulada en U) y la segunda componente, el programa p que se espera ejecutar en M_i. Ejemplo, $\langle i, p\rangle = \underbrace{11\cdots1}_{i}0p$

Es decir, $U(p') = U(\langle i, p\rangle) = M_i(p)$ y, por consiguiente, $\forall x\ C_U(x) \leq C_{M_i}(x) + c_{M_i}$ con $c_{M_i} = i + 1$.

Esta función C_U la notaremos a partir de ahora como C y es la llamada **Complejidad de Kolmogorov**, de manera que $C : \{0,1\}^* \to \mathbb{N}$ está definida como:

$C(x) := \text{mín}\{|p| : U(p)\downarrow = x\}$ donde $|p|$ es la longitud de la cadena binaria p.

A efectos prácticos, la desventaja principal de esta complejidad es que resulta no computable ya que no podemos conocer todos los programas que paran y computan a x.

2.2. Aleatoriedad de Martin-Löf

2.2.1. Definiciones Históricas

Existen diversos enfoques en la literatura para definir algorítmicamente el azar (ver [Dow09, Chapter 5]), los cuales se pueden agrupar en tres grandes paradigmas:

- **El paradigma de Teoría de la Medida [Mis19,Löf66]:** Las secuencias aleatorias son aquellas que pasan todo test estadístico efectivo, es decir, las que no tienen ninguna propiedad *rara* que nos dé algún grado de conocimiento extra sobre la secuencia.

- **El paradigma de Impredictibilidad [Sch71]:** Las secuencias aleatorias son aquellas que resultan computacionalmente impredecibles. Es decir, que aún conociendo los primeros $n-1$ bits de una secuencia, no podríamos predecir mediante métodos efectivos cuál sería el n-ésimo bit (al menos no con más seguridad que el resultado de arrojar una moneda no cargada al aire).

- **El paradigma Computacional [Lev74, Cha75]:** Las secuencias aleatorias son aquellas difíciles de describir (o de comprimir) algorítmicamente. En otras palabras, se trata de identificar al azar con lo que no tiene estructura, regularidades, ni patrones reconocibles, es decir, con aquellas secuencias que resultan incompresibles.

Estos tres paradigmas coinciden, bajo ciertas hipótesis, en lo que se entiende por una secuencia aleatoria. Esto significa que existe una noción robusta de aleatoriedad, la cual es conocida como *aleatoriedad de Martin-Löf* y que presentaremos a continuación. Si bien no es la única, es la más conocida y estudiada de las variantes propuestas, y es la que utilizaremos para nuestro trabajo.

2.2.2. Aleatoriedad para cadenas finitas

Empezaremos por una definición de aleatoriedad para cadenas finitas ya que, si bien la definición de aleatoriedad de Martin-Löf es para cadenas infinitas, ambas comparten la idea de relacionar la aleatoriedad de una cadena con la complejidad de Kolmogorov que presentamos en la sección 2.1.

Podríamos decir que una cadena finita x es C-aleatoria si es aleatoria o incompresible respecto de su representación en una máquina universal de Turing U, es decir, $C(x) \geq |x|$. Sin embargo, es más razonable, dada la elección arbitraria de U, relajar la definición anterior agregando una constante d. Decimos entonces que una cadena x es C, d-aleatoria si $C(x) \geq |x| - d$.

Notemos que la constante d actúa, intencionalmente, como un *grado* de aleatoriedad de manera que no seamos categóricos en la definición de una secuencia finita como azarosa o no. También es importante resaltar que efectivamente existen cadenas que cumplen con la condición de C, d-aleatoriedad (la demostración puede consultarse en [Sol64] o en [Kol65]).

2.2.3. Aleatoriedad para cadenas infinitas

El contenido de la sección 2.2.3 no es necesario para la comprensión del modelo propuesto en esta tesis, pero se incluye pues la definición de aleatroiedad de Martin-Löf se torna más interesante para cadenas infinitas

En el caso de cadenas infinitas, la definición es más interesante ya que será una definición absoluta. Es decir, una cadena infinita será aleatoria o no lo será.

Para esta definición, introduciremos la complejidad K. Esta complejidad es igual a la complejidad C con la que estábamos trabajando sólo que en este caso la enumeración de las máquinas de Turing $M_1, M_2, \cdots$ que utilizamos anteriormente para definir la complejidad son ahora todas máquinas libres de prefijos, es decir, si $M_i(p)$ está definida, entonces $M_i(q)$ se indefine $\forall q$ que sea una extensión de p.

Sea X una secuencia infinita de ceros y unos, decimos que X es Martin-Löf aleatoria si $\exists\ c/\ \forall n, K(X \upharpoonright n) > n - c$, donde $X \upharpoonright n$ denota los primeros n símbolos de X.

El cambio de C por K es un cambio técnico producto de que se puede demostrar que no existe una secuencia que cumpla con la condición de que $\exists c/\ \forall n, C(X \upharpoonright n) > n - c$. Esto se debe a las oscilaciones de la complejidad C (ver [Löf71] o [Lev74, Teorema 2.5.1]). Lo importante es que podemos hacer este cambio sin pérdida de generalidad, ni de las propiedades que nos interesan, ya que para toda máquina de Turing existe otra equivalente que es libre de prefijos.

2.3. Inducción de Solomonoff

2.3.1. Intuición

Ya hemos introducido las nociones teóricas de complejidad y de aleatoriedad, las cuales nos permitirán definir en el capítulo 3 un modelo computable para representar la complejidad de las cadenas. Sin embargo, necesitamos otra herramienta más que nos permita avanzar en nuestro objetivo de generar un modelo que nos permita predecir la generación de las secuencias aleatorias. Supongamos que tenemos un conjunto de observaciones correspondientes a los primeros caracteres de una secuencia x producida por una persona, $x = x_1, x_2, \cdots$, y tenemos un conjunto de hipótesis $H = h_1, h_2, \cdots$ sobre los posibles procesos que pueden generar x, tal que una es la hipótesis del proceso verdadero que genera x. ¿Cómo elegimos h_i para que sea la hipótesis del proceso más probable que explique a x? ¿Cómo elegimos h_i con la mínima cantidad de datos necesarios?

Una solución es la propuesta por la regla de Bayes: $P(h_i|x) = \dfrac{P(x|h_i) \cdot P(h_i)}{P(x)}$. Para computar las probabilidades relativas de cada h_i podemos descartar a $P(x)$ de la ecuación porque es una constante independiente. Por otro lado, calcular $P(x|h_i)$ suele ser también bastante directo. El problema de la regla de Bayes es que necesitamos asignar la probabilidad a priori de h_i, es decir, **antes de observar los datos**. En 1960, Ray J. Solomonoff [Sol60a, Sol60b] crea la teoría de la probabilidad algorítmica para dar una solución teórica a este problema. Decimos que es una solución teórica ya que veremos que esta probabilidad no es computable y que, en la práctica, sólo pueden utilizarse ciertas aproximaciones.

Según el paradigma de Occam, entre todas las teorías que explican los datos, la mejor es la más pequeña, la que comprima los datos que observamos de un fenómeno de la manera más corta, es decir, la que dé la explicación más simple será la que demuestre nuestro mayor conocimiento sobre el fenómeno y la que mejor prediga nuevos datos. Para darle más rigurosidad a esta creencia, Solomonoff utiliza la descripción efectiva más corta de un objeto, es decir, su complejidad de Kolmogorov. En otras palabras, veremos que de todas las hipótesis posibles que explican el fenómeno, aquella con la menor complejidad de Kolmogorov es la más probable.

Las secciones 2.3.2 y 2.3.3 formalizan estas ideas y presentan una distribución universal M que, utilizando la noción de complejidad de Kolmogorov, nos permite predecir cualquier distribución computable μ casi tan bien como la verdadera distribución. Dado que dichas secciones tienen una fuerte carga de conceptos que pueden desalentar a un lector sin conocimientos previos en el tema, nos interesa remarcar que el corolario que utilizaremos de dicha teoría es que suele ser una buena heurística para predecir los siguientes caracteres de una secuencia x generada por una distribución computable μ, encontrar la extrapolación y que minimize la complejidad de Kolmogorov de xy.

2.3.2. Probabilidad universal a priori discreta

Intentaremos encontrar una distribución que nos permita calcular la probabilidad a priori de nuestras hipótesis para el caso de variables discretas. Supongamos un proceso

desconocido p que produce una cadena binaria de veinte unos. La probabilidad de que un proceso aleatorio arroje esa cadena es de $2^{-20} \approx 10^{-6}$, al igual que cualquier otra cadena de la misma longitud. Sin embargo, tenemos la intuición de que hay una diferencia entre esa cadena fácilmente reconocible y distinguible para nosotros con la gran mayoría de las cadenas de esa longitud, lo que nos lleva a intuir que p no es un proceso aleatorio, sino un proceso determinístico más sencillo.

Formalizando, sea p un programa que al ser ejecutado en una máquina universal libre de prefijos U devuelve la cadena binaria finita x como resultado. Y dado que p es, a su vez, una cadena binaria, podemos definir la probabilidad universal a priori discreta, $m : \{0,1\}^* \to [0,1]$, como la probabilidad de que la salida de U sea x cuando los datos de entrada de la cinta son generados por el lanzamiento de una moneda no cargada. Esta medida puede escribirse como:

$$m(x) \ := \sum_{p \,:\, U(p)\downarrow \,=x} 2^{-|p|}$$

Por la desigualdad de Kraft, la serie converge ($m(x) \leq 1$) si la sumatoria es sobre el conjunto de los programas válidos de una máquina libre de prefijos. De lo contrario, si la sumatoria fuese sobre el conjunto de los programas válidos de una máquina de Turing universal plana, entonces podría diverger (ver [LiV08, Teorema 1.11.1]). Por lo cual, al igual que en la sección 2.2.2, el uso de una máquina universal libre de prefijos U es necesario para el cálculo de $m(x)$.

Nos interesa remarcar que esta distribución está fuertemente relacionada con la Complejidad de Kolmogorov, en tanto que el máximo término de la sumatoria es $2^{-K(x)}$ y, por lo tanto, $m(x) \geq 2^{-K(x)}$. Más aún, según el Teorema de Codificación de Levin [Lev74], $-\log\ m(x) \ = \ K(x) + O(1)$.

Por otro lado, como toda cadena binaria finita puede ser computada por, al menos, un programa p en U, se muestra que m asigna una probabilidad no nula a cualquier hipótesis computable y, por lo tanto, se respeta el principio de Epicurus (342? B.C. - 270 B.C.) según el cual se debe considerar todas las hipótesis que sean consistentes con los datos. Más aún, si medimos la calidad de cada hipótesis con su complejidad de Kolmogorov, vemos que las hipótesis con las cadenas más simples tienen una probabilidad más alta en m que aquellas con cadenas más complejas. Por lo cual, podemos decir que m también formaliza el principio de Occam que mencionábamos en la introducción.

Decimos que m es una probabilidad universal, ya que Leivin también demuestra que si p es una *semi-medida semi-computable desde abajo*, entonces, existe una constante c tal que $\forall x,\ c \cdot m(x) \geq p(x)$ [Lev74]. Por lo cual, m es en algún sentido *la mayor* de las *semi-medidas semi-computables desde abajo*.

Sin embargo, el problema principal es que m **no es computable**, ya que no es posible calcular la sumatoria sobre el conjunto de los programas que paran en U porque conocer dicho conjunto es un problema indecidible.

2.3.3. Probabilidad universal a priori continua

La distribución m del punto anterior está definida sobre cadenas binarias finitas. Si queremos trabajar con cadenas infinitas necesitamos una nueva medida $M : \{0,1\}^* \to [0,1]$ que definiremos como la probabilidad de que la salida de una máquina de Turing monótona universal U empiece con la cadena x cuando los datos de entrada de la cinta son generados por el lanzamiento de una moneda no cargada.

La variante que ahora U sea monótona significa que, dada una entrada σ, si el resultado del computo de M en algún instante es τ y en un instante posterior es $\widehat{\tau}$, entonces $\tau \preccurlyeq \widehat{\tau}$. En este caso diremos que $U(p) \downarrow= x*$ si, después de leer exactamente p de la cinta de entrada, U se detiene o continúa computando, pero devuelve x o una extensión de x en la cinta de salida.

Formalizamos a M como: $M(x) := \sum_{p\,:\,U(p)\downarrow\,=x*} 2^{-|p|}$

Se puede demostrar que M es una *semi-medida semi-computable*, y que por otro teorema de Levin [Lev74], si μ es una *semi-medida semi-computable desde abajo*, entonces existe una constante c tal que $\forall x,\ c \cdot M(x) \geq \mu(x)$.

Nuevamente, podemos enumerar todas las máquinas de Turing monótonas: $M_1, M_2, \cdots$ de manera de usar el mismo método que veníamos usando para construir una máquina universal y, por lo tanto, definir la complejidad monótona de Kolmogorov, $Km : \{0,1\}^* \to \mathbb{N}$, como

$Km(x) := \text{mín}\{|p| : U(p) \downarrow= x*\}$

Levin también demuestra que M tiene propiedades similares a m que la vinculan con la complejidad de Kolmogorov, ya que $-\log M(x) = K(x) - O(\log|x|)) = Km(x) - O(\log|x|)$

La propiedad que más nos interesa es que si la secuencia binaria infinita X está distribuida de acuerdo a una medida computable μ, entonces la distribución de predicción $M(x_{n+1}|x_1...x_n) := M(x_1...x_{n+1})/M(x_1...x_n)$ converge rápidamente a $\mu(x_{n+1}|x_1...x_n) := \mu(x_1...x_{n+1})/\mu(x_1...x_n)$ con μ-probablidad 1.

Esto quiere decir que M **predice casi tan bien como la verdadera distribución** μ. Aunque, nuevamente, la distribución M resulta **no computable**.

Una de las principales aplicaciones de M es la llamada **inducción de Solomonoff**. Definamos $S_n(a) : \{0,1\} \to \mathbb{R}$ como

$$S_n(a) := \sum_{|x|=n-1} \mu(x)(M(a|x) - \mu(a|x))^2$$

$$\text{y } S_n := \sum_{a\in\{0,1\}} S_n(a)$$

Es decir, S_n es el error cuadrático en la predicción del n-ésimo bit de una secuencia infinita $x = x_1x_2x_3\cdots x_{n-1}x_nx_{n+1}\cdots$ distribuida de acuerdo a una medida computable μ, cuando se la predice utilizando la *semi-medida semi-computable* M.

Como μ es una medida computable, existe al menos un programa que la computa. Sea

$K(\mu)$ la complejidad de Kolmogorov de μ, es decir, la longitud del menor programa que computa a μ. Puede demostrarse que $\sum_n S_n \leq K(\mu)(\ln 2)/2$ ([LiV08, Teorema 5.2.1]). En otras palabras, la suma total de los errores cuadráticos de predicción al utilizar la distribución M en vez de μ, está acotado por una constante y típicamente decrece más rápido que $\frac{1}{n}$.

Esto significa que tenemos una distribución universal M que nos permite predecir cualquier secuencia producida por un proceso computable con el mínimo absoluto de datos necesarios. El problema principal, una vez más, es que M **no es computable**.

Un corolario interesante de la inducción de Solomonoff es lo que se conoce como predicción por compresión de datos (ver [LiV08, Chapter 5.2.4]). Sea w una cadena binaria infinita, tal que $w = xy\cdots$ con x, y cadenas binarias finitas, entonces:

$$\lim_{|x|\to\infty} \log \frac{1}{\mu(y|x)} = Km(xy) - Km(x) + O(1) < \infty$$

Por lo tanto, es una buena heurística para predecir la extrapolación y dado x, es decir $\mu(y|x)$, minimizar $Km(xy)$. En otras palabras, para predecir los siguientes caracteres de $w = x\cdots$, conociendo sólamente a x, suele ser buena idea elegir a y como la cadena generada por el programa más corto que devuelve $xy\cdots$.

3. MODELO PROPUESTO

El azar no perdona al que lo quiera explicar
–Zambayonny, Búfalo de agua, 2011

La idea de entender al aprendizaje humano y sus capacidades cognitivas como problemas computacionales, y a la mente humana como una máquina natural que ha evolucionado para resolverlos, viene alcanzando mayores consensos en estudios recientes (ver [Ten11]). Estos trabajos entienden al proceso de aprendizaje en términos de inferencias probabilísticas sobre una jerarquía de estructuras simbólicas formales (con ciertos parámetros ajustables) que representan nuestro conocimiento. La mejor manera de capturar la representación de este conocimiento varía según el dominio y la tarea, por ejemplo, a través de grafos, gramáticas, esquemas relacionales y programas funcionales.

Para representar nuestro conocimiento sobre la aleatoriedad, buscaremos trasladar los acercamientos teóricos que vimos en el capítulo 2 sobre aleatoriedad o predicción que utilizan la noción de complejidad de Kolmogorov. Sin embargo, el problema de aplicar las herramientas que se desprenden de dicha teoría es que formalizan la noción de aleatoriedad con conceptos no computables.

Por lo tanto, en el presente capítulo, buscaremos trasladar estas nociones a un modelo computable que permita entender la algoritmia que utilizan los seres humanos para representar el azar y para tratar de generarlo.

Gramática G

Definiremos un lenguaje de programación con una gramática G tal que todo programa hecho con nuestro lenguaje termine y arroje una secuencia de caracteres de un alfabeto binario.

$S = A|\lambda$

$A = 0|1|AA|(A)^n$

Máquina de Turing M_G

Todo programa bien definido por nuestra gramática G se ejecutará en una máquina de Turing M_G la cual computará una secuencia binaria de la siguiente manera:

$M_G(\lambda) = \lambda$

$M_G(0) = 0$

$M_G(1) = 1$

$$M_G(S_1S_2) = M_G(S_1) \bullet M_G(S_2)$$
$$M_G(S^n) = \underbrace{M_G(S) \bullet M_G(S) \bullet \cdots \bullet M_G(S)}_{n}$$

donde $\bullet$ es la concatenación de las secuencias

Medida $\| \cdot \|_G$

Necesitaremos también de una función $\| \cdot \|_G$ para medir el tamaño de cada programa escrito en nuestro lenguaje.

$\| \lambda \|_G = 0$

$\| 0 \|_G = \| 1 \|_G = G_{base}$

$\| S_1S_2 \|_G = \| S_1 \|_G + \| S_2 \|_G$

$\| S^n \|_G = G_{rep} * \lfloor \log_{10}(n) \rfloor + \| S \|_G$

Donde $G_{base}, G_{rep} \in \mathbb{R}$ son parámetros de nuestro modelo que calibraremos empíricamente en los experimentos de los capítulos 4, 5 y 6. La intuición detrás de estos parámetros es que G_{base} representa el costo base de registrar un caracter y $G_{rep} * \lfloor \log_{10}(n) \rfloor$ es lo que cuesta registrar cuantas veces se repite una secuencia (asumiendo que los números se representan/recuerdan en nuestro alfabeto de diez caracteres en la mente humana, aunque igualmente configurable por el uso del parámetro G_{rep} a cualquier alfabeto).

Complejidad C_G de una cadena

Ahora que tenemos una función para medir el tamaño de nuestros programas, $\| \cdot \|_G$, tomaremos la idea de complejidad de Kolmogorov y definiremos la complejidad C_G de una secuencia de caracteres como el tamaño del menor programa que la genera. La diferencia principal con esta medida teórica radicará en que, con nuestro lenguaje, encontrar el menor programa que genera una secuencia es un problema computable.

Sea x la secuencia que computa nuestra máquina de Turing M_G cuando toma como entrada un programa p de nuestro lenguaje, definimos entonces la complejidad de una cadena x como: $C_G : \{0,1\}^* \to \mathbb{N}$

$$C_G(x) := \text{mín}\{\| p \|_G : M_G(p) = x\}$$

Para calcular el mínimo programa que computa una secuencia x, utilizaremos el algoritmo *1*, ProgramaMínimo. Si bien la manera más natural de resolverlo es calculando los programas mínimos de cada subsecuencia recursivamente, utilizaremos una técnica de programación dinámica. Es decir, guardaremos las soluciones parciales de los programas mínimos para cada subsecuencia y así evitaremos la repetición de cálculos. Esto nos lleva a reducir la complejidad del algoritmo de un orden exponencial, $O(2^n)$, a un orden cuadrático, $O(n^2)$, con $n = |x|$. Esto se debe a que sólo necesitaremos calcular una vez al programa mínimo para cada subsecuencia de x.

También podemos notar en el pseudo-código la llamada a los métodos constructores de distintas subclases de la clase *Programa*: *ProgramaVacío*, *ProgramaImprimeUnCaracter*, *ProgramaConcatenación* y *ProgramaRepetición*, cada uno asociado al modelo de cómputo que definíamos para las máquinas de Turing M_G que ejecutan nuestra gramática.

$M_G(\lambda) \simeq$ ProgramaVacío()

$M_G(0) \simeq$ ProgramaImprimeUnCaracter(0)

$M_G(1) \simeq$ ProgramaImprimeUnCaracter(1)

$M_G(S_1S_2) \simeq$ ProgramaConcatenación($Programa_{S_1}, Programa_{S_2}$)

$M_G(S^n) \simeq$ ProgramaRepetición($Programa_S, n$)

Predictor

Nos interesa que nuestro modelo permita estudiar cómo funciona el proceso de generación de secuencias *aleatorias* en los seres humanos. Por ejemplo: ¿Utilizamos alguna noción de complejidad? ¿Cuántos caracteres de la secuencia retiene la mente humana para decidir el siguiente caracter? ¿Cuántos caracteres a futuro tiene en cuenta? ¿Cómo los pondera? ¿Cómo varían estos parámetros si la persona tiene que generar la secuencia en menos tiempo? ¿Cómo varían respecto de determinadas características de la población de estudio? Ahora que hemos definido nuestra medida de complejidad y podemos encontrar el programa mínimo que genera una secuencia, estamos listos para definir un algoritmo que nos permita predecir los caracteres de las secuencias generadas por las personas al intentar producir cadenas aleatorias.

En la sección 2.3.3, remarcámos que la **inducción de Solomonoff** nos provee una buena heurística para predecir los siguientes caracteres de una cadena infinita, habiendo conocido los primeros $n-1$ caracteres. Sin embargo, esta heurística depende de la complejidad de Kolmogorov K_m lo que la vuelve no computable. Dado que contamos con una medida de complejidad computable en nuestro modelo, podemos también avanzar en definir nuestro algoritmo de predicción tomando la idea de esta heurística.

Formalizando, supongamos que existe una cadena binaria $s \in \{0,1\}^*$ de la cual conocemos los primeros $n-1$ caracteres, es decir $s = x_1x_2 \cdots x_{n-1}x_n \cdots$, y nos interesa calcular el n-ésimo caracter. Llamemos w a la subsecuencia conocida de s, entonces $s = wx_n \cdots$. Sea $pred : \{0,1\}^* \to \{0,1\}$ la función de predicción del siguiente caracter de una cadena binaria, entonces $x_n \simeq pred(w) = pred(x_1x_2 \cdots x_{n-1})$.

$$pred(w) := \begin{cases} 0 & \text{si } \min\{C_G(w0y) : y \in \{0,1\}^*\} < \min\{C_G(w1y) : y \in \{0,1\}^*\} \\ 1 & \text{si } \min\{C_G(w0y) : y \in \{0,1\}^*\} > \min\{C_G(w1y) : y \in \{0,1\}^*\} \\ desempate(w) & \text{si no} \end{cases}$$

Es decir, el caracter $pred(w)$ será el primer caracter de aquella cadena de extrapolación que minimice la complejidad entre todas las extrapolaciones posibles de la cadena w.

Notemos que, para predecir x_n, utilizamos la complejidad C_G. Esto implica que nuestra

Algorithm 1 ProgramaMínimo: Calcula el menor programa que genera a la cadena x

```
function programaMínimo(x = x_1, x_2, ··· , x_n)
    if TablaDeMínimos(x) ≠ null then
        return TablaDeMínimos(x);
    end if
    progConc ⇐ mínimoConConcatenación(x);
    progRep ⇐ mínimoConRepetición(x);
    if progRep ≠ null ∧ progRep.tamaño() ≤ progConc.tamaño() then
        mínimo ⇐ progRep;
    else
        mínimo ⇐ progConc;
    end if
    TablaDeMínimos(x) ⇐ mínimo;
    return mínimo;
end function

function mínimoConConcatenación(x = x_1, x_2, ··· , x_n)
    programaInicial ⇐ programaMínimo(x_1);
    programaFinal ⇐ programaMínimo(x_2, x_3, ··· , x_n);
    mínimoActual ⇐ new ProgramaConcatenación(programaInicial,programaFinal);
    n ⇐‖ x ‖;
    for all i ∈ [2, n − 1] do
        programaInicial ⇐ programaMínimo(x_1, x_2, ··· , x_i);
        programaFinal ⇐ programaMínimo(x_{i+1}, x_{i+2}, ··· , x_n);
        programa ⇐ new ProgramaConcatenación(programaInicial,programaFinal);
        if programa.tamaño() < mínimoActual.tamaño() then
            mínimoActual ⇐ programa;
        end if
    end for
    return mínimoConConctenación;
end function

function mínimoConRepetición(x = x_1, x_2, ··· , x_n)
    mínimoActual ⇐ null;
    n ⇐‖ x ‖;
    for all i ∈ divisoresPositivosNoTriviales(n) do
        if (x_1 ··· x_i) * (n/i) = (x_1 ··· x_n) then
            progRep ⇐ new ProgramaRepetición(programaMínimo(x_1 ··· x_i),n/i));
            if mínimoActual = null ∨ progRep.tamaño() < mínimoActual.tamaño() then
                mínimoActual ⇐ programa;
            end if
        end if
    end for
    return mínimoActual;
end function
```

Nota 1: TablaDeMínimos empieza inicializado con *null* para todos los valores, a excepción de los valores $\lambda, 0$ y 1, para los cuales empieza con ProgramaVacío(), ProgramaImprimeUnCaracter(0) y ProgramaImprimeUnCaracter(1), respectivamente.
Nota 2: El método *tamaño* devuelve el tamaño del programa, es decir, $\| \cdot \|_G$.
Nota 3: El método *divisoresPositivosNoTriviales* devuelve los divisores positivos de un número n, a excepción de 1 y n.
Nota 4: El operador $*$ para cadenas, devuelve la repetición de una cadena. Es decir, $0 * 3$ devuelve la cadena 000.

función de predicción va a depender entonces de los parámetros G_{base}, y G_{rep} de la medida $\| \cdot \|_G$. A estos parámetros, les sumaremos otros dos. Por un lado, estudiaremos la máxima longitud de las cadenas de extrapolación y ya que, en este caso, no utilizaremos cadenas infinitas como en la hipótesis de la inducción de Solomonoff. A esta longitud la llamaremos $G_{futuros}$. El otro parámetro será $G_{anteriores}$, la longitud de caracteres hacia atrás a utilizar al momento de predecir x_n, ya que en realidad no siempre empezaremos desde x_1 pues nos interesa estudiar cuántos caracteres contempla la mente humana para hacer su predicción.

Por lo tanto, la función final de predicción que utilizaremos será $pred' : \{0,1\}^* \to \{0,1\}$, y está definida como:

$$pred'(w = x_1 x_2 \cdots x_{n-1}) := \begin{cases} 0 & \text{si } min0 < min1 \\ 1 & \text{si } min0 > min1 \\ desempate(x_{n-i} \cdots x_{n-2} x_{n-1}) & \text{si no} \end{cases}$$

$$min0 = \text{mín}\{C_G(x_{n-i} \cdots x_{n-2} x_{n-1} 0y) : y \in \{0,1\}^* \wedge \| y \| \leq G_{futuros}\}$$

$$min1 = \text{mín}\{C_G(x_{n-i} \cdots x_{n-2} x_{n-1} 1y) : y \in \{0,1\}^* \wedge \| y \| \leq G_{futuros}\}$$

$$i = G_{anteriores}$$

El pseudo-código del algoritmo que implementa la función $pred'$ se muestra en algoritmo 2, PREDECIR, y cuenta con los siguientes parámetros ajustables:

- $G_{base} \in \mathbb{R}$**:** costo del caso base de la medida $\| \cdot \|_G$ utilizada en C_G
- $G_{rep} \in \mathbb{R}$**:** costo de las repeticiones de la medida $\| \cdot \|_G$ utilizada en C_G
- $G_{futuros} \in \mathbb{N}$**:** longitud máxima de la secuencia de extrapolación y.
  $\| y \| \leq G_{futuros}$, $\| y' \| \leq G_{futuros}$
- $G_{anteriores} \in \mathbb{N}$**:** longitud de caracteres de x anteriores a contemplar.
  $G_{anteriores} = \| x_{n-i} \cdots x_{n-2} x_{n-1} \|$

Por último, la función *desempate* es una función determinística con el objetivo de simplificar el análisis posterior de los resultados. La idea detrás de esta función (cuyo pseudo-código puede verse en el algoritmo 3, DESEMPATE) es devolver el caracter que más se repitió en la cadena o, si ambos caracteres se hubieran repetido la misma cantidad de veces, el último caracter de la secuencia.

Algorithm 2 PREDECIR: Predice el caracter x_n

procedure PREDECIR($x = x_1, x_2, \cdots x_{n-1}$)
  i $\Leftarrow G_{anteriores}$
  xAcotado $\Leftarrow x_{n-i} \cdots x_{n-2} x_{n-1}$
  complejidadMínimaCon0 $\Leftarrow$ ValorMáximo
  complejidadMínimaCon1 $\Leftarrow$ ValorMáximo
  for all $y \in \{0,1\}^* \wedge \parallel y \parallel \leq G_{futuros}$ **do**
    complejidadCon0 $\Leftarrow$ programaMínimo(xAcotado • 0 • y).tamaño()
    if complejidadCon0 < complejidadMínimaCon0 **then**
      complejidadMínimaCon0 $\Leftarrow$ complejidadCon0
    end if
    complejidadCon1 $\Leftarrow$ programaMínimo(xAcotado • 1 • y).tamaño()
    if complejidadCon1 < complejidadMínimaCon1 **then**
      complejidadMínimaCon1 $\Leftarrow$ complejidadCon1
    end if
  end for
  if complejidadMínimaCon0 < complejidadMínimaCon1 **then**
    return 0
  else if complejidadMínimaCon0 > complejidadMínimaCon1 **then**
    return 1
  else
    return desempate(xAcotado)
  end if
end procedure

Nota: La constante ValorMáximo es el mayor valor numérico que puede tomar el tipo de datos con el que se guarde la complejidad de un programa.

Algorithm 3 DESEMPATE: Calcula al caracter x_n para el caso del desempate

procedure DESEMPATE($x = x_1, x_2, \cdots x_{n-1}$)
  if $x = \lambda$ **then**
    return 0
  else
    if $|x|_0 > |x|_1$ **then**
      return 0
    else if $|x|_0 < |x|_1$ **then**
      return 1
    else
      return x_{n-1}
    end if
  end if
end procedure

4. CALIBRANDO EL MODELO

Random numbers should not be generated with a method chosen at random
–Donald Knuth, The Art of Computer Programming, 1969

Antes del primer experimento, estudiaremos el comportamiento del algoritmo de predicción con el fin de obtener un primer ajuste de los parámetros descriptos en el capítulo 3: $G_{base}, G_{rep}, G_{futuros}, G_{anteriores}$.

Lo que buscamos con esta calibración no son los parámetros óptimos del algoritmo de predicción para una serie de secuencias de base, ni tampoco someter sus resultados a los tests estadísticos que se utilizan para determinar si un algoritmo es bueno para generar números pseudo-aleatorios. Lo que esperamos es obtener una versión que pueda predecir *lo suficientemente bien* secuencias con cierta estructura para establecer un puntaje a los participantes del experimento durante su ejecución. Más adelante, analizaremos los parámetros nuevamente con las cadenas generadas *al azar* por los sujetos de prueba.

En uno de los papers más citados de la historia de la psicología, Miller sostiene que la memoria a corto plazo está limitada por un número mágico cercano a siete items [Mil56]. Queremos entonces que nuestro algoritmo funcione bien para predecir caracteres de secuencias que sean la repetición de subsecuencias fáciles de recordar en la memoria; por eso, probaremos cuántos de los últimos caracteres generados es necesario contemplar para poder predecir fácilmente secuencias de período menor o igual a 7.

La figura 4.1 corresponde al análisis del algoritmo de predicción para todas las cadenas de longitud 100 generadas a partir de la repetición sucesiva de una secuencia de período menor o igual a 7 caracteres. Para cadenas que puedan ser generadas por distintos períodos (ejemplo, 0101), las tomaremos siempre como correspondientes al período mínimo que las genera. En el eje Y, vemos el promedio de aciertos que tuvo el algoritmo para cada valor de $G_{anteriores}$ al predecir los 100 caracteres de todas las cadenas de ese período.

El resto de los parámetros utilizados en la figura 4.1 fueron establecidos en los siguientes valores: $G_{base} = 1$, $G_{rep} = \frac{1}{\log_2(10)}$ y $G_{futuros} = 5$.

En el análisis de la figura 4.1 notamos que a partir de los dieciséis caracteres, el algoritmo se comporta como esperamos y el promedio de aciertos se vuelve estable y mayor a 0.9 para casi todos los períodos. Basándonos en el análisis anterior, elegiremos veinte caracteres como ventana para el experimento donde tendremos un algoritmo que le dirá a los participantes qué tan *azarosas* fueron sus cadenas generadas.

En una nueva prueba, fijamos $G_{anteriores}$ en veinte y, utilizando los mismos valores para G_{base} y G_{rep}, variamos ahora el parámetro $G_{futuros}$. El resultado que puede apreciarse en la figura 4.2 es que a partir de los cinco caracteres ya se alcanza un promedio estable y alto de aciertos. Dado que la cantidad de cadenas a analizar por el algoritmo de predicción crece exponencialmente de acuerdo al tamaño de $G_{futuros}$, sumado a la necesidad de que el algoritmo sea veloz para devolverle una respuesta inmediata a los participantes del

experimento, decidimos establecer el valor de $G_{futuros}$ en cinco para el predictor a utilizar en el primer experimento.

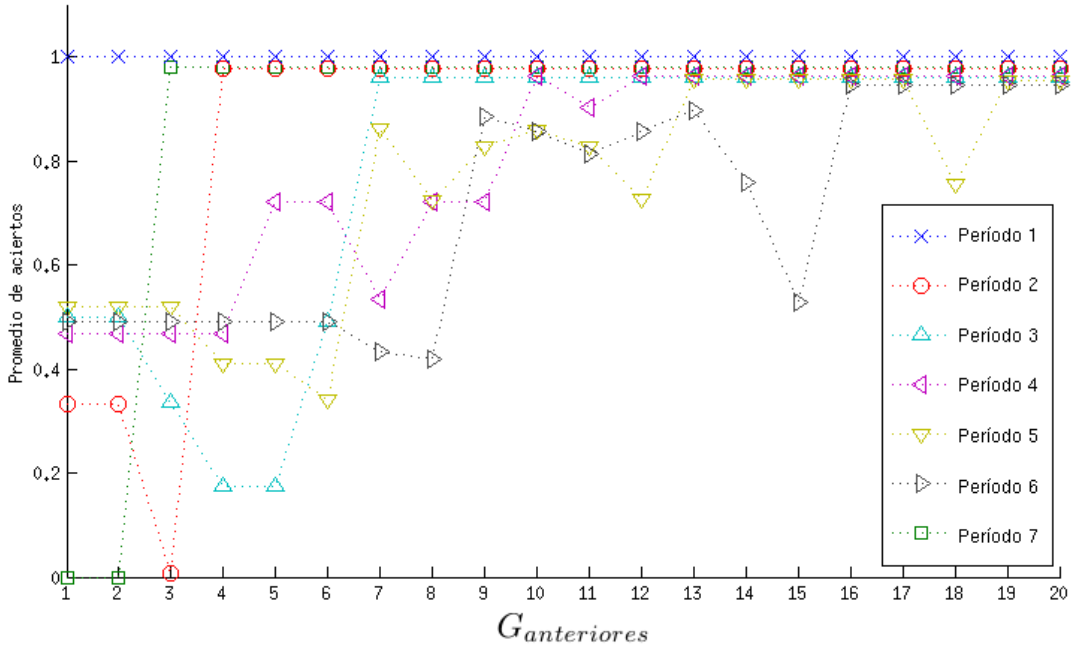

Fig. 4.1: Promedio de aciertos según distintos valores de $G_{anteriores}$

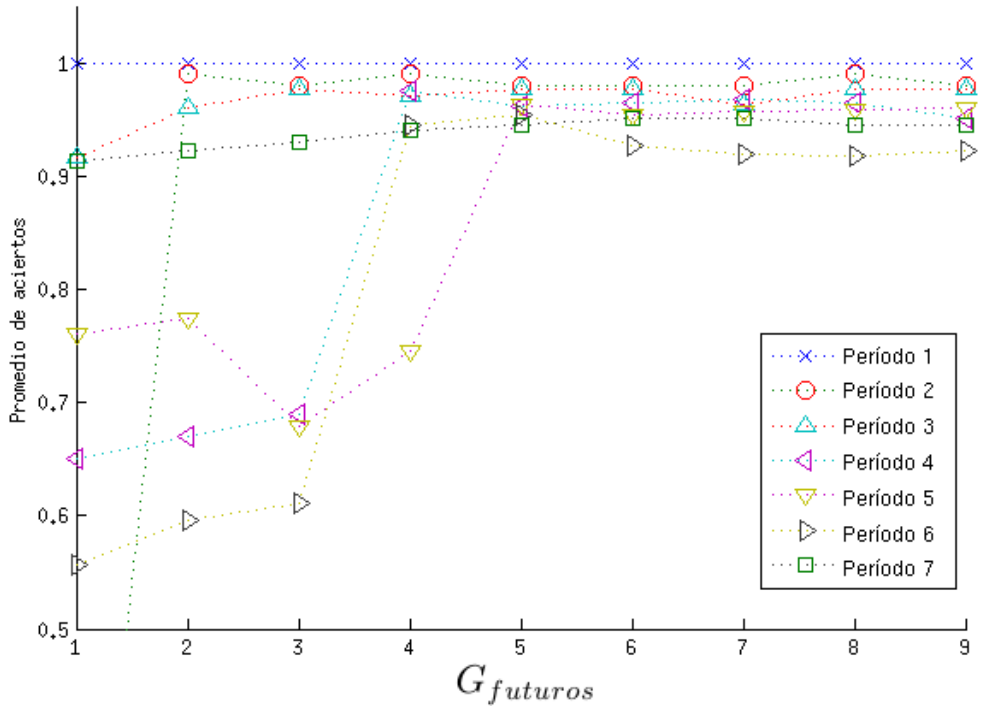

Fig. 4.2: Promedio de aciertos según distintos valores de $G_{futuros}$

5. PRIMER EXPERIMENTO

Comment oser parler de lois du hasard ? Le hasard n'est-il pas l'antithèse de toute loi ?
–Joseph Bertrand, Calcul des probabilités, 1889

5.1. Diseño del experimento

El primer experimento consistió en el diseño de un juego en el cual los participantes debían presionar las teclas izquierda y derecha de sus teclados *al azar*. No se les daba una explicación de qué es *al azar*, tan sólo se les daba esa consigna laxa y se les explicaba que, cada vez que la computadora predijera correctamente lo que ellos estaban por presionar, iban a restar un cierto puntaje y, cuando la computadora no predijera correctamente lo que ellos iban a presionar, iban a sumar un cierto puntaje.

La historia del juego consistía entonces en ocho niveles en los que el participante iba acompañando la evolución del planeta generando azar para favorecer la evolución. Esta historia pretendía mantener a los participantes motivados en avanzar por los niveles y conscientes de su progreso.

Interfaz del juego (ver figura 5.1):

- Nombre del nivel (año de la evolución del planeta).
- Odómetro que indicaba el puntaje actual del jugador.
- Imagen alusiva al nivel/año de evolución.
- Mensaje de penalidad si presionaba muy rápido las teclas.

Funcionamiento del juego:

- Cada vez que el usuario presionaba una flecha izquierda o derecha, la aguja del odómetro avanzaba si era lo contrario a lo que había calculado el predictor o retrocedía si era igual a lo calculado por el predictor.
- Cuando la flecha del odómetro se mantenía más de 5 segundos en la zona *Impredecible* (pintada de verde), el usuario avanzaba de nivel.
- Cuando la flecha del odómetro se mantenía más de 5 segundos en la zona *Predecible* (pintada de rojo), el usuario perdía el juego.
- Si el usuario no presionaba ninguna flecha en más de 2.5 segundos, retrocedía la aguja del odómetro la misma cantidad de puntaje que si hubiese sido predicho.

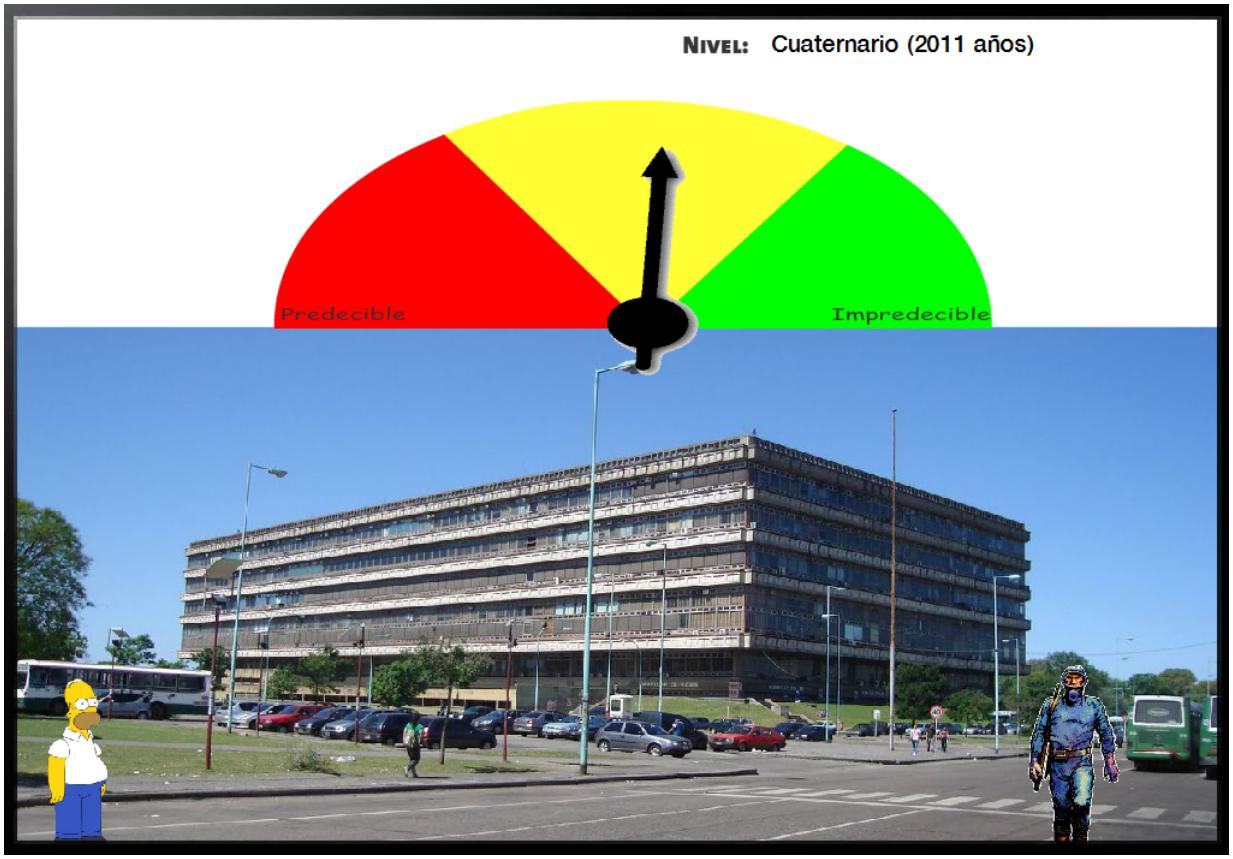

Fig. 5.1: Ejemplo de la vista de un nivel del juego utilizado para el experimento

- Cada vez que el usuario presionaba una tecla, la flecha del odómetro pasaba de negro a blanco. Si el usuario presionaba la siguiente tecla demasiado rápido, sin esperar los 450 milisegundos donde la aguja del odómetro se ponía nuevamente en negro, se lo penalizaba con un retroceso.[1]

El experimento registraba las teclas presionadas por el participante, los instantes en las que fueron presionadas, las predicciones de nuestro predictor online y los tiempos de cada evento producido.

Todos los participantes eran personas mayores de 18 años con estudios secundarios completos.

[1] La decisión de forzar al usuario a esperar un determinado tiempo entre cada tecla se debió a limitaciones de la tecnología utilizada para el juego: Flash (y su lenguaje ActionScript). Dado que los cálculos del predictor requerían un cierto poder de cálculo y que ActionScript fue ideado para animaciones en la web donde suele ser más importante devolverle una respuesta al usuario que registrar todos los eventos producidos por este, debíamos forzar un cierto tempo para no perder ninguna de las teclas presionadas por el usuario mientras el predictor calculaba el próximo caracter.

5.2. Resultados del primer experimento

La cantidad de participantes en el primer experimento fue de 36 personas. El rendimiento del predictor online no fue bueno como puede verse en la figura 5.2. El promedio de aciertos del predictor para los jugadores fue del 50.3 %, el primer cuartil de 45.6 %, la mediana de 48.6 % y el tercer cuartil de 53.3 %. Es decir, **en la mitad de los casos el predictor no acertó siquiera uno de cada dos caracteres de la secuencia.**

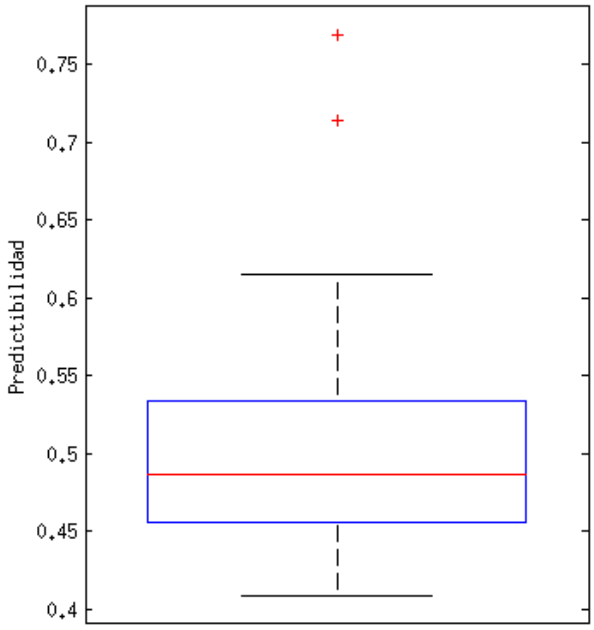

Fig. 5.2: Predictibilidad media por jugador

En principio estos resultados no eran tan desalentadores ya que esperábamos optimizar los parámetros del predictor una vez obtenidas las secuencias del usuario. Volvimos entonces a evaluar todas las cadenas obtenidas modificando los parámetros del algoritmo del predictor.

Se analizaron exhaustivamente todas las combinaciones de los siguientes valores para cada parámetro:

- G_{base}**:** Valores discretos entre 0 y 100, con una diferencia de 0.1 entre cada uno
- G_{rep}**:** Valores discretos entre 0 y 100, con una diferencia de 0.1 entre cada uno
- $G_{futuros}$**:** Valores entre 1 y 8.
- $G_{anteriores}$**:** Valores entre 1 y 15.

Sin embargo, tras optimizar los parámetros del algoritmo, la mejora fue muy leve; llevando la predictibilidad media de 50.3 % a 52.4 %. Este resultado se dio con:

- $G_{base} = 1$
- $G_{rep} = 1$
- $G_{futuros} = 7$
- $G_{anteriores} = 9$

Análisis de las posibles fallas del primer experimento

Respuesta inmediata

Nuestra principal hipótesis para explicar el mal rendimiento del predictor fue la *respuesta inmediata* que recibían los participantes. Dado que obtenían una respuesta inmediata para saber si su elección fue predicha o no, esto podía generar un efecto de aprendizaje o adaptación al algoritmo del predictor por parte del participante. Cuando el participante notaba que su algoritmo de generación de azar no era efectivo para continuar avanzando, seguramente intentaba adaptarlo para no seguir retrocediendo. Esto también puede verse al analizar las *rachas* del predictor, es decir la cantidad de caracteres consecutivos bien predichos, el promedio de las rachas fue cercano a 2 caracteres (1.98).

Varianza de niveles

Otro de los puntos que buscamos resolver para el siguiente experimento fue la varianza en los niveles que completaron los participantes. Si bien el predictor online era el mismo en cada nivel, la dificultad de cada uno era diferente ya que se variaba los puntos que el usuario avanzaba o retrocedía por cada acierto del predictor. Como la manera de actuar del participante podía variar de acuerdo a su percepción de la dificultad del nivel e influir entonces en cómo generaba las cadenas, se buscó para el siguiente experimento obtener resultados más homogéneos donde todos participaran de los mismos niveles.

Varianza de longitud de secuencia

La longitud de la secuencia de teclas presionadas por cada usuario variaba notablemente ya que dependían del rendimiento de cada participante en cada nivel. Como puede verse en el boxplot de la figura 5.4, el 75 % de las secuencias fue de una longitud igual o mayor a 25, el 50 % tuvo una longitud mayor a 41, y sólo el 25 % superó los 89 caracteres (o teclas). Esta varianza dificultaba el análisis de la efectividad del predictor para cada secuencia por lo que preferimos para el siguiente experimento homogeneizar la longitud de las cadenas de los participantes y asegurarnos también longitudes mayores de las cadenas.

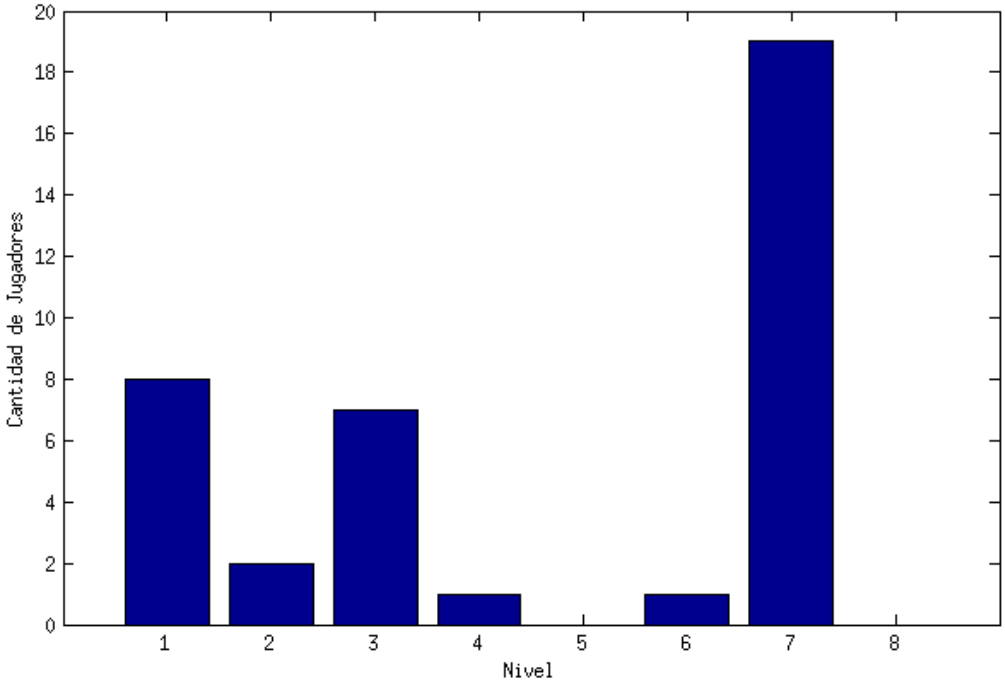

Fig. 5.3: Último nivel alcanzado por los jugadores

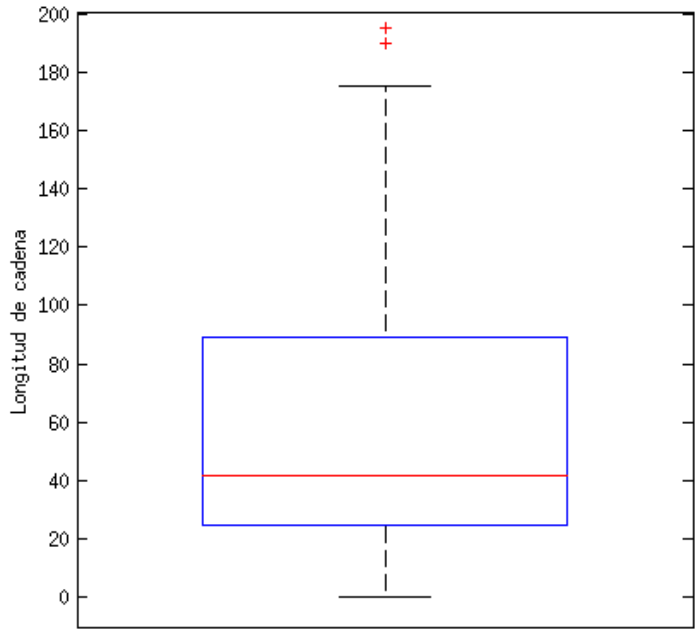

Fig. 5.4: Longitud de las cadenas generadas por los participantes

6. EXPERIMENTO FINAL

Le hasard est un mot vide de sens; rien ne peut exister sans cause
–Voltaire, Dictionnaire philosophique, 1764

6.1. Diseño del experimento

En el nuevo experimento la consigna a los participantes se mantuvo: presionar las teclas izquierda y derecha de sus teclados *al azar*, sin ninguna explicación más detallada de lo que esto significaba. Todos los participantes fueron personas mayores de 18 años con estudios secundarios completos y ciertas nociones de programación y/o computabilidad (aunque no necesariamente el mismo grupo que el experimento anterior). La diferencia principal en el experimento era que, en esta ocasión, los participantes ya no recibían una respuesta inmediata al presionar cada tecla, sino que recibían un puntaje al completar cada nivel.

El experimento consistía en cuatro niveles/oportunidades donde el participante debía presionar ciento veinte veces las teclas izquierda o derecha.

Interfaz del juego (ver figura 6.1):

- Número de intentos
- Cuatro filas delimitadas con líneas
- Cada fila con 30 columnas no delimitadas

Funcionamiento del experimento:

- Cada vez que el usuario presionaba una flecha izquierda o derecha, un casillero se pintaba de verde.
- Una vez completados los ciento veinte casilleros de cada nivel, el participante obtenía un puntaje llamado *calidad de azar*, el promedio de desaciertos del predictor.
- El experimento se detenía una vez que el participante completaba los cuatro niveles/intentos (de 120 casilleros cada uno).

El experimento registraba las teclas presionadas por el participante, los instantes en los que fueron presionadas y las predicciones de nuestro predictor para cada nivel. Como ahora las predicciones para la cadena eran calculadas al finalizar cada nivel, y no al presionar cada tecla, ya no era necesario poner una penalidad si el participante presionaba las teclas demasiado rápido pues no había ningún cálculo complejo a realizar durante la generación de las cadenas por los usuarios que afectase el rendimiento del reproductor de Flash para registrar los eventos.

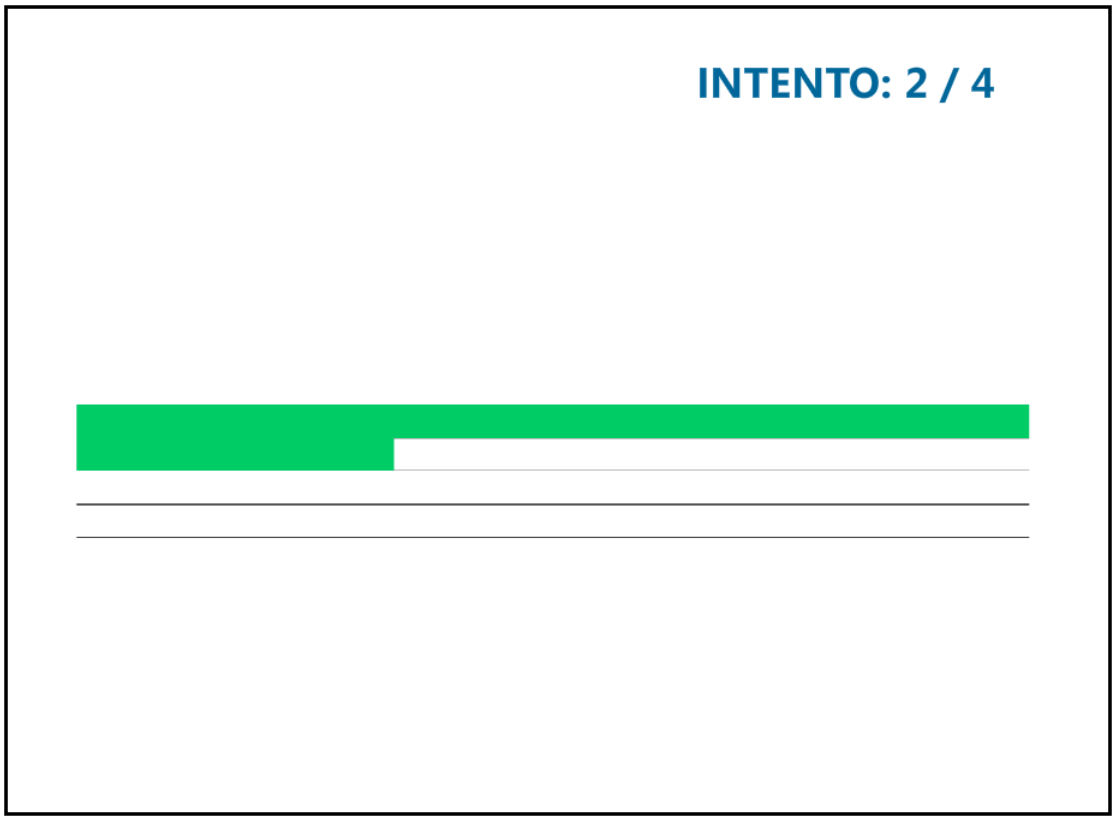

Fig. 6.1: Ejemplo de la vista del segundo intento en el nuevo experimento

6.2. Resultados del experimento final

De los 58 participantes que abrieron la página para realizar el experimento, sólo 38 completaron los cuatro niveles previstos. Tomamos a estos 38 participantes como los sujetos de estudio. Cada uno de los participantes produjo cuatro secuencias (una en cada nivel/intento), lo que da un total de 152 secuencias.

6.2.1. Predictor Online

Los parámetros utilizados para el predictor online del último experimento fueron los mismos que para el primero, a excepción de $G_{futuros}$ que se redujo de 7 a 3. Esto se debió a que ahora se calcularían todos los aciertos del predictor para cada caracter de la secuencia ingresada por el participante una vez terminado cada nivel, y no resultaba conveniente hacer esperar demasiado tiempo a los participantes para poder ver su puntaje de *calidad de azar*. Vale destacar que esto no genera ninguna dificultad en el estudio de los parámetros ya que una vez obtenida las secuencias generadas por los participantes en el experimento, procedimos a analizarlas con distintas variantes de los parámetros. Esta configuración era simplemente utilizada para devolver un puntaje de *calidad de azar* a los participantes al finalizar cada nivel. Los parámetros utilizados para el último experimento fueron entonces:

- $G_{base} = 1$
- $G_{rep} = 1$
- $G_{futuros} = 3$
- $G_{anteriores} = 9$

El promedio de acierto de las 152 cadenas fue de 57.9 %, la mediana se ubicó en el 55.7 % de aciertos, el primer cuartil en 51.0 % y el tercer cuartil en 61.5 %, tal cual puede verse en la figura 6.2. Estos resultados ya muestran una diferencia considerable respecto del 50.3 % de aciertos que el predictor obtuvo en las cadenas generadas en el primer experimento. El objetivo principal de este breve análisis fue eliminar los outliers (o valores atípicos), es decir, aquellas cadenas donde el predictor online obtuvo un rendimiento atípicamente bueno ya que estas cadenas eran excesivamente triviales producto de errores de los participantes al presionar las teclas, desgano o interés de estos en verificar que el nivel de calidad de azar sea una medida efectiva. Quedaron finalmente 138 de las 152 cadenas: 37 correspondientes al primer intento, 33 del segundo, 34 del tercero y 34 del cuarto. Retomaremos luego, con más detalle, el análisis de los resultados del predictor en la sección 6.2.5, por ahora, nos centraremos en estudiar las cadenas generadas por los participantes.

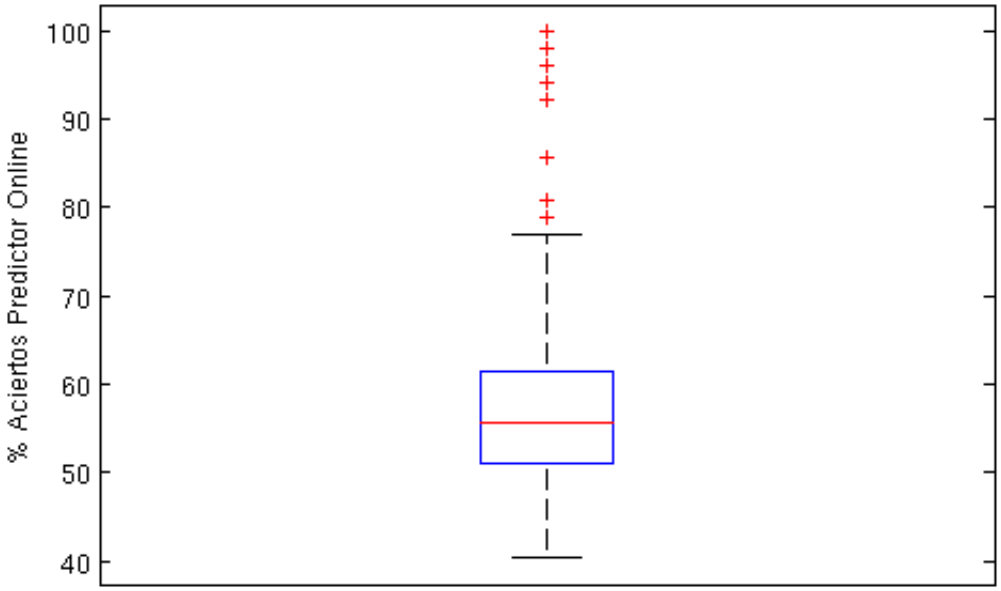

Fig. 6.2: Promedio de aciertos del predictor online en el Experimento Final

6.2.2. Cadenas de los participantes

Empezamos por analizar estas 138 cadenas de 120 caracteres generadas por los participantes con las que trabajaremos. Diversos estudios sostienen que el principal error que cometen las personas al intentar generar cadenas aleatorias es producir más alternaciones

de los símbolos que las que se esperaría de un proceso aleatorio. Una revisión de dichos trabajos puede verse en [Fak97] donde se define la *probabilidad de alternación* de una secuencia binaria x, $P(x) : \{0,1\}^{\geq 2} \to [0,1]$, como una medida del grado de alternación de una secuencia para comparar los resultados de cada estudio. $P(x)$ se calcula como $P(x) = \frac{r-1}{n-1}$ donde n es la cantidad de símbolos de la secuencia y r el número de corridas (subsecuencias de un mismo símbolo). Para una secuencia sin alternaciones, es decir compuesta por un sólo símbolo, la *probabilidad de alternación* será igual a 0, mientras que para una secuencia con una alternancia constante entre 0 y 1, su valor será igual a 1. En el caso de secuencias aleatorias, donde se espera que las frecuencias de los símbolos sean equiprobables, el valor esperado de la *probabilidad de alternación* es 0,5.

Para el caso de las secuencias generadas por nuestros participantes, podemos ver el histograma de la probabilidad de alternación para las cadenas en la figura 6.3. El promedio de la *probabilidad de alternación* de todas las cadenas es de 0.51, la mediana de 0.53, y casi el 57 % de las cadenas se ubican por encima del 0.5 de alternancia. Sin embargo, al realizar un t-test para descartar que esta diferencia sea esperable al tomar una muestra de datos de una distribución de probabilidades de alternación donde la media sea 0.5, notamos que la diferencia no es significativa ya que el p-valor es igual a 0.31, el intervalo de confianza = [0,49, 0,53] y el t-valor en 1,01. Esto quiere decir que, para nuestros participantes, la *probabilidad de alternación* no es una buena medida para modelar la baja calidad de las secuencias supuestamente aleatorias producidas por ellos. Tal vez esto se deba a que los participantes del experimento tienen cierto conocimiento de programación y/o computabilidad que los lleva a evitar el error más común que producen otras personas de sobrestimar las alternaciones que son esperables en un proceso aleatorio.

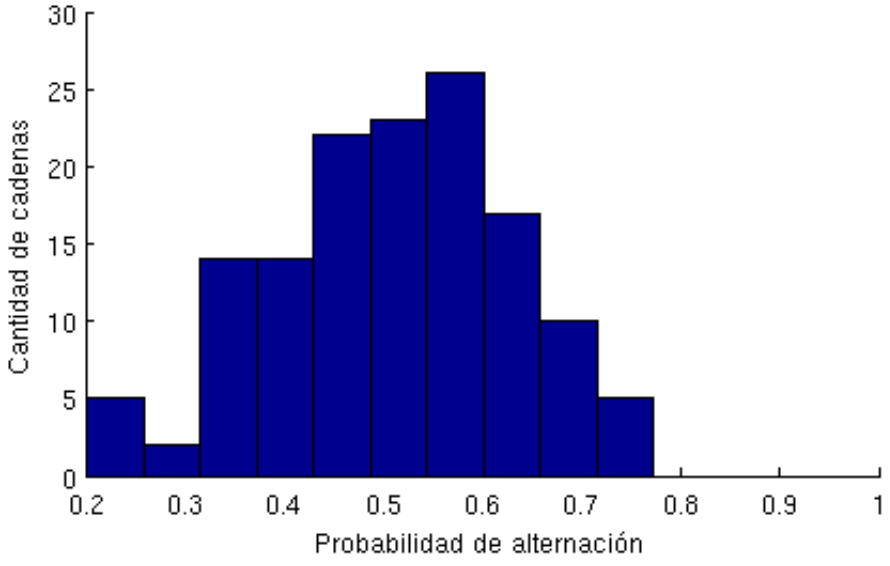

Fig. 6.3: Histograma de la probabilidad de alternación de las cadenas

Analizamos luego la frecuencia de todas las subsecuencias de longitud entre 1 y 4 para estudiar si las frecuencias de cada subsecuencia son equiprobables, como se esperaría por la ley de los grandes números. Tal como puede verse en la figura 6.4, las subsecuencias

que implican una mayor cantidad de alternaciones para cada longitud están entre las que más se repiten (*01* y *10* para el caso de longitud 2, *010* y *101* para longitud 3 y *0101* y *1010* para longitud 4). Por otro lado, para todas las longitudes, también se repite una gran cantidad de veces las secuencias *11*, *111* y *1111*. Tal cual lo planteaban los estudios que mencionábamos al principio de esta sección, estos datos muestran que las personas, al tratar de generar secuencias aleatorias, tienden a favorecer las alternaciones más allá de lo esperado por un proceso aleatorio. Pero por otro lado, al ser participantes con cierto conocimiento de aleatoriedad producto de su formación en computabilidad y/o programación, notamos la repetición de una cadena sin alternaciones que intenta equilibrar este exceso de alternaciones. Sin embargo, notemos que pese a esta preocupación, el resto de las cadenas no mantienen una frecuencia cercana como se esperaría en el caso de un proceso aleatorio.

En resumen, las personas efectívamente no generan cadenas aleatorias de buena calidad porque las subsecuencias generadas no son equiprobables ya que tienden a favorecer la aparición de subsecuencias con un mayor número de alternaciones. Sin embargo, la probabilidad de alternación no resulta una medida efectiva para distinguir las frecuencias verdaderamente aleatorias de aquellas generadas por los participantes, ya que los participantes intentan mantener la probabilidad de alternación cercana a lo esperado en un proceso aleatorio generando secuencias sin repeticiones que compensan esta probabilidad.

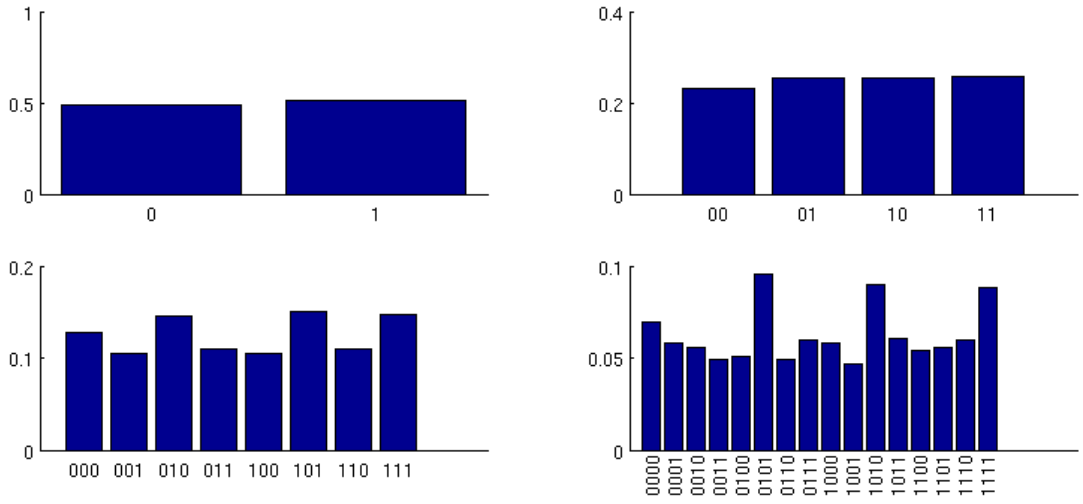

Fig. 6.4: Frecuencia de aparición de subsecuencias

6.2.3. Complejidad C_G

Vimos que la *probabilidad de alternación* no resultó una medida efectiva para medir lo que entendemos por azar. Veamos entonces cómo se comportan las medidas de nuestro modelo. Para eso, nos interesa analizar la complejidad C_G, tal cual la definimos en el capítulo 3, de las cadenas generadas por los participantes. Para esto, tomamos 3 grupos de datos de 138 cadenas de 120 caracteres de longitud cada una.

1. Generadas por los participantes de nuestro experimento (sin outliers)

2. Generadas por un PRNG (PseudoRandom Number Generator)

3. Generadas por ruido blanco

Como PRNG utilizamos el algorítmo **Mersenne Twister** desarrollado por Makoto Matsumoto y Takuji Nishimura [M&N98], popular por su calidad y velocidad para simulaciones estadísticas (específicamente, la segunda revisión del año 2002 que se encuentra actualmente implementada en la librería *GNU Scientific Library*).

El ruido blanco fue tomando del sitio random.org, propiedad de *Randomness and Integrity Services Limited Company.*

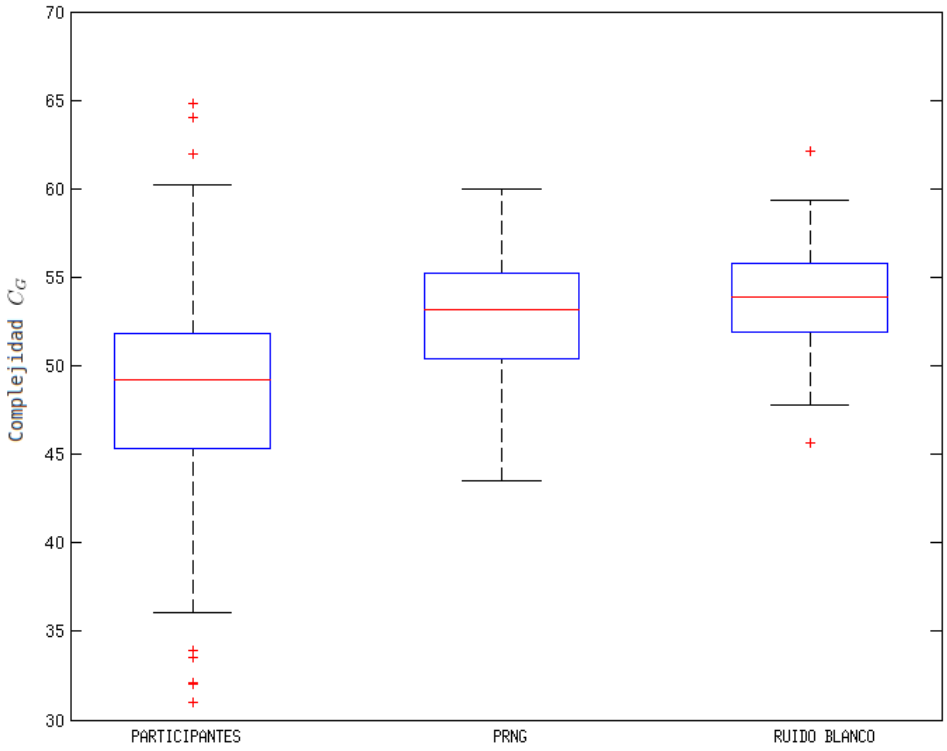

Fig. 6.5: Complejidad C_G de: participantes, PRNG y ruido atmosférico

	Participantes	PRNG	Ruido blanco
Promedio	48.43	52.99	53.88
Desviación Estandard	6.62	3.06	2.87
1er Cuartil	45.30	50.42	51.88
Mediana	49.23	53.15	53.85
3er Cuartil	51.79	55.21	55.79

Tab. 6.1: Valores de complejidad C_G de participantes, PRNG y ruido blanco para las secuencias

Como puede verse en la figura 6.5 y en la tabla 6.1, tanto el promedio, como la mediana de la complejidad C_G aumentan entre las secuencias generadas por los participantes y las

	Personas vs PRNG	Personas vs Ruido blanco	PRNG vs Ruido blanco
p-valor	$4{,}9 \times 10^{-11}$	$1{,}2 \times 10^{-15}$	$1{,}4 \times 10^{-2}$
t-valor	$-7{,}1$	$-9{,}1$	$-2{,}5$
Int. de confianza	$[-5{,}8, -3{,}3]$	$[-6{,}6, -4{,}2]$	$[-1{,}6, -0{,}2]$

Tab. 6.2: T-test complejidad C_G de participantes, PRNG y ruido blanco para las secuencias

generadas por el PRNG o el ruido blanco, y a su vez, la desviación estándar disminuye a casi la mitad. Realizamos luego un t-test entre cada par de distribuciones para descartar la posibilidad de que sean simplemente dos instancias de distribuciones que compartan la misma mediana. En la tabla 6.2 se puede notar que hay una diferencia muy significativa entre las cadenas generadas por los participantes y las generadas por el PRNG (p-valor $= 4{,}9 \times 10^{-11}$) y por ruido blanco (p-valor$= 1{,}2 \times 10^{-15}$). Sin embargo, la diferencia entre la distribución de las cadenas de PRNG y la de ruido blanco es muy poco significativa (p-valor $= 1{,}4 \times 10^{-2}$). Esto implica que, tal cual era de esperarse, no encontramos una diferencia significativa entre la complejidad del PRNG y la del ruido blanco, pero sí la hubo entre estas y las complejidades de las cadenas generadas por los participantes. En otras palabras, nuestra función de complejidad C_G encontró patrones en las secuencias de personas que las hacen más sencillas que las de PRNG o las de ruido blanco según nuestra medida, es decir, los participantes no fueron capaces de producir cadenas tan complejas o azarosas como las que provienen del ruido blanco o de un PRNG, y nuestra medida de complejidad pudo modelar esa diferencia.

6.2.4. Fatiga

Algunos autores han estudiado cómo la fatiga afecta la capacidad de la memoria a corto plazo [Jon99], nos interesaba entonces evaluar si el modelo y el experimento actual mostraban algún síntoma de fatiga en los participantes que disminuyera la complejidad de las cadenas producidas.

Estudiamos entonces la complejidad de las 138 cadenas generadas por los participantes respecto de la fatiga. Para esto, medimos la complejidad de todas las subsecuencias de longitud 30 que componen la cadena. Es decir, para cada cadena $x = x_1, x_2, \cdots, x_n$ medimos $C_G(x_i, x_{i+1}, \cdots, x_{i+29})$ con $i \in [1, 100]$. De esta manera, por cada cadena podíamos analizar la complejidad de 90 subcadenas.

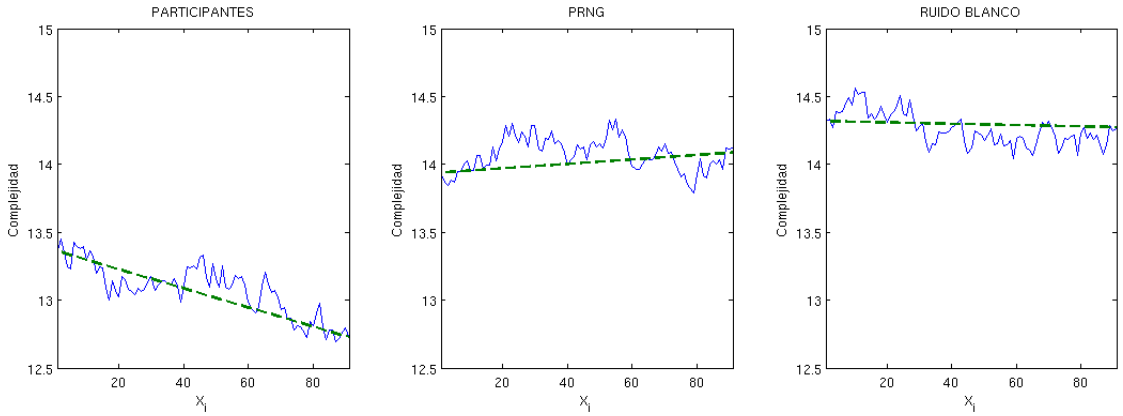

Fig. 6.6: Complejidad C_G según x_i (posición de inicio de la subcadena)

En la figura 6.6, promediamos la complejidad de las cadenas respecto del x_i inicial. A su vez, las comparamos realizando el mismo estudio para las cadenas generadas por el PRNG y por ruido blanco. Si bien en esta figura analizamos secuencias de 30 caracteres de longitud (en vez de las originales de 120 caracteres), notemos que nuevamente la complejidad de las secuencias generadas por los participantes es menor que la de las cadenas generadas por el PRNG o ruido blanco. Sin embargo, lo que más nos interesa destacar en la figura es, en realidad, la pendiente de cada gráfico. Para el caso de los participantes, la pendiente aproximada fue de $-0{,}007$, para PRNG $0{,}0016$ y para el ruido blanco $-0{,}0005$. Es decir, la pendiente más pronunciada apareció en las cadenas generadas por los participantes y efectivamente esta pendiente tuvo un signo negativo que mostraría cierta evidencia de que el cansancio (o fatiga) influye en la complejidad de las cadenas que podemos producir.

Quedaba por ver si esta pendiente podía considerarse significativa. Para eso, estimamos el gradiente de cada una de las 138 cadenas producidas por los participantes. El promedio de estas pendientes fue, como mencionábamos, de $-0{,}007$ y, al realizar un t-test para asegurarnos que la distribución de las pendientes que estimamos efectivamente tuviera una diferencia significativa sobre una distribución con media cero, notamos que la diferencia era significativa con un p-valor de $0{,}02$, un intervalo de confianza entre $-0{,}01$ y $-0{,}001$, y un t-valor de $-2{,}32$. Estos valores pueden verse en la tabla 6.3 donde también se muestran los valores del t-test para las cadenas producidas por el PRNG y por ruido blanco. En estos dos últimos casos, la diferencia con una distribución de media cero no resultó significativa.

	Participantes	PRNG	Ruido Blanco
Promedio	$-0{,}007$	$0{,}0016$	$-0{,}0005$
p-valor	$0{,}02$	$0{,}5$	$0{,}8$
t-valor	$-2{,}3$	$0{,}7$	$-0{,}2$
Intervalo de confianza	$[-0{,}01, -0{,}001]$	$[-0{,}003, 0{,}006]$	$[-0{,}005, 0{,}004]$

Tab. 6.3: T-test sobre la presencia de fatiga en las cadenas

Estos resultados muestran que la calidad del azar o la complejidad de las cadenas producidas por las personas disminuye con la fatiga y que esta diferencia puede considerarse

significativa. Sería interesante profundizar este tipo de análisis comparando las cadenas producidas por un grupo de participantes con síndrome de fatiga crónica respecto de otro grupo de control para investigar con mayor profundidad la hipótesis en trabajos futuros.

6.2.5. Predictibilidad

Al principio del capítulo, remarcábamos que el promedio de aciertos del predictor para las 152 cadenas en el experimento online fue de 57.9 %. Este promedio, obtenido con los parámetros $G_{base} = 1, G_{rep} = 1, G_{futuros} = 3$ y $G_{anteriores} = 9$, nos sirvió para descartar a las cadenas donde el predictor obtuvo valores excepcionalmente buenos, y poder estudiar la medida de complejidad.

Nos interesaba luego estudiar los parámetros del algoritmo de predicción para cada persona con el fin de entender mejor el lenguaje que utilizamos para generar azar. Empezamos por analizar la predictibilidad, es decir, el promedio de aciertos, para distintas combinaciones de los parámetros del predictor. Las tuplas que analizamos en una primera instancia fueron todas las combinaciones de los siguientes valores de cada parámetro:

- $G_{futuros}$**:** Valores entre 1 y 7.
- $G_{anteriores}$**:** Valores entre 1 y 14.
- Los valores de G_{rep} y G_{base} se fijaron en 1 ya que no se analizaron en este trabajo.

Dado que nuestra motivación es buscar los parámetros que optimizan la predictibilidad para cada participante, tomaremos una medida de base que nos permita comparar qué tan efectiva es la optimización de los parámetros. Para esto, empezamos por destacar el desempeño general del predictor para cada uno de los 38 participantes promediando los aciertos de los cuatro intentos/niveles por participante para todas las combinaciones de los parámetros. Luego, repetiremos este procedimiento para calcular el promedio general con 38 tuplas de 4 secuencias generadas por el PRNG y con la misma cantidad de secuencias generadas por ruido blanco para obtener otros dos conjuntos de datos con los que comparar los valores.

En la figura 6.7 y en la tabla 6.4 puede verse esta comparación donde el porcentaje de aciertos promedio para los participantes llegó al 56,2 % y sólo un 25 % de participantes obtuvieron un porcentaje de aciertos menor a 50 %, mientras que para el caso de PRNG y ruido blanco los promedios de aciertos fueron de 50,7 % y 49,5 % respectivamente. Además, el porcentaje de secuencias donde el porcentaje de aciertos fue menor a 50 % fue casi la mitad para PRNG y cerca del 75 % para ruido blanco. Esto nos muestra que el predictor efectivamente pudo predecir las secuencias de los participantes, pero no las producidas por PRNG o ruido blanco. Por otra parte, en la tabla 6.5 puede verse el valor de los aciertos promedio para los participantes desagregado por cada nivel, en la cual no se observa ninguna diferencia significativa entre los niveles.

A continuación, hicimos un t-test para ver si la diferencia entre las medias de la distribución de los promedios de predicciones para los 38 participantes era significativa respecto de las 38 tuplas de PRNG y de las de ruido blanco, notemos que el p-valor de las distribuciones para las personas en comparación con PRNG o con ruido blanco nos muestra una

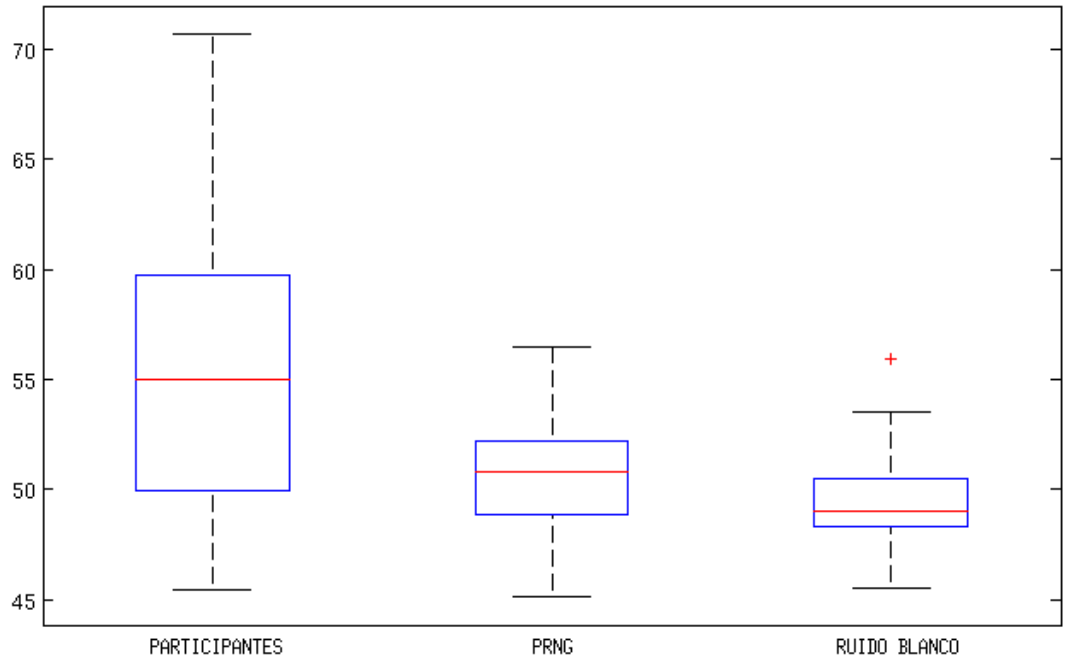

Fig. 6.7: Promedio de aciertos de todas las combinaciones estudiadas de los parámetros

	Participantes	PRNG	Ruido Blanco
Promedio	56,2 %	50,7 %	49,5 %
1^{er} cuartil	50,0 %	48,8 %	48,3 %
Mediana	55,0 %	50,8 %	49,0 %
3^{er} cuartil	59,7 %	52,2 %	50,5 %

Tab. 6.4: Promedio de aciertos de todas las combinaciones estudiadas de los parámetros

diferencia mucho más significativa que la que hay entre ruido blanco y PRNG tal cual se muestra en la tabla 6.6.

El siguiente paso fue estudiar los parámetros del predictor que optimizan el promedio de aciertos. Para esto, seleccionamos los parámetros que optimizaban la predictibilidad en el 50 % de los caracteres de las secuencias para cada persona, y luego medimos la predictibilidad sobre el 50 % restante de manera de poder evaluar el resultado de la optimización en un conjunto de datos independientes de los cuales optimizamos los valores. Los resultados pueden verse en la figura 6.8; para el caso de los participantes el promedio de aciertos asciende de 56,2 % a 58,1 %, mientras que para PRNG pasa de 50,7 % a 51,2 % y para ruido blanco de 49,5 % a 49,2 %. Si hacemos un t-test para evaluar qué tan significativa es la diferencia entre los promedios de todos los parámetros y los parámetros óptimos

	General	Nivel 1	Nivel 2	Nivel 3	Nivel 4
Participantes	56,2 %	54,4 %	56,6 %	56,5 %	57,0 %

Tab. 6.5: Promedio de aciertos de todas las combinaciones estudiadas de los parámetros

	Personas vs PRNG	Personas vs Ruido blanco	PRNG vs Ruido blanco
p-valor	$7{,}4 \times 10^{-5}$	$5{,}4 \times 10^{-7}$	$3{,}9 \times 10^{-2}$
t-valor	4,4	6,0	2,1
Int. de confianza	$[0{,}03, 0{,}08]$	$[0{,}04, 0{,}09]$	$[0, 0{,}02]$

Tab. 6.6: T-test de distribuciones de los promedios de aciertos de todas las combinaciones estudiadas de los parámetros

para cada participante, veremos que la diferencia es significativa con un p-valor= $1{,}8 \times 10^{-3}$ y un intervalo de confianza = $[0{,}008, 0{,}032]$. Sin embargo, como era de esperarse, la diferencia para los parámetros óptimos de las cadenas generadas por PRNG respecto de su promedio no resulta significativa (p-valor= 0,3 e intervalo de confianza = $[-0{,}004, 0{,}014]$), y lo mismo sucede para el caso de ruido blanco (p-valor= 0,2 e intervalo de confianza = $[-0{,}012, 0{,}003]$).

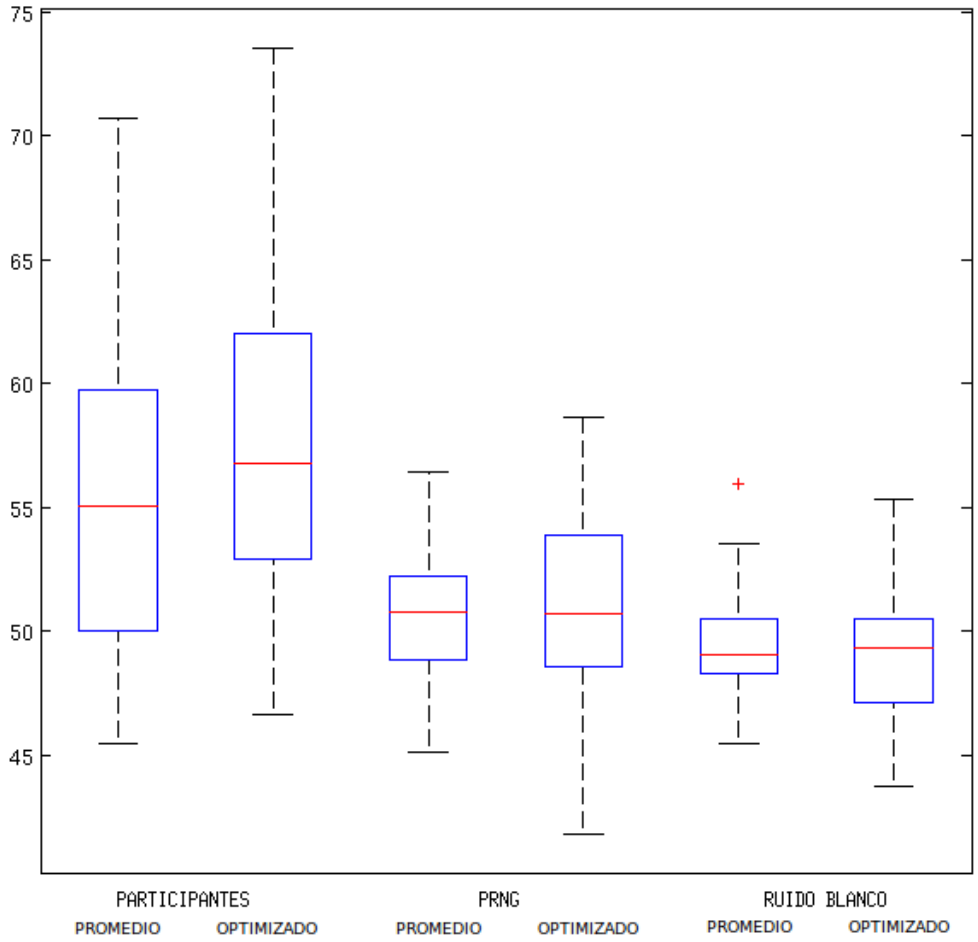

Fig. 6.8: Promedio vs Optimización de parámetros

Hasta aquí habíamos elegido los parámetros que optimizaban los aciertos del predictor para los cuatro niveles/intentos de cada participante. Sin embargo, ningún parámetro

Intento	1^{er}	2^{do}	3^{er}	4^{to}
1^{er}	1	0,24	0,16	0,20
2^{do}	0,24	1	0,24	0,13
3^{er}	0,16	0,24	1	0,14
4^{to}	0,20	0,13	0,14	1

Tab. 6.7: Promedio de matrices de correlación de la predictibilidad para cada nivel

pareciera destacarse sobre el resto, ni tampoco encontramos evidencia de que el parámetro que optimiza a cada intento se mantenga invariante entre los niveles para cada persona. Es decir, eran distintos los parámetros que obtenían la mayor cantidad de aciertos en cada participante según el intento. A pesar de esto, podemos encontrar evidencia de cierta correlación entre los valores elegidos para los parámetros y la predictibilidad para cada nivel.

La tabla 6.7 nos muestra el promedio de 38 matrices de correlación, una matriz por cada participante donde las variables que estudiamos son la predictibilidad en cada nivel. Es decir, la matriz sobre la que se calcula la correlación es una matriz donde cada variable/columna se corresponde a la predictibilidad en uno de los cuatro niveles, y cada dato/fila una combinación de los parámetros $\langle G_{anteriores}, G_{futuros} \rangle$ del predictor. Notemos que los valores de correlación son altos por lo que la predictibilidad entre cada intento de un participante es linealmente dependiente de los otros. Estos valores de correlación resultan significativos ya que la probabilidad de obtenerlos si los intentos de cada participante fueran variables independientes es baja como puede verse experimentalmente en la figura 6.9. Para esta figura, realizamos el mismo procedimiento del cálculo de la correlación promedio pero para 1000 permutaciones distintas de cada una de las 38 matrices sobre las que se calcula la correlación, y graficamos la cantidad de veces que se repite cada valor en las matrices de correlación calculadas sobre estas permutaciones. Los valores de la tabla 6.7 se muestran en la misma figura 6.9 con líneas punteadas y, tal cual puede apreciarse, están muy alejados de los valores de los datos permutados. Esto significa que la probabilidad de que los valores de correlación de la predictibilidad entre los niveles para cada tupla $\langle G_{anteriores}, G_{futuros} \rangle$ hayan sido producto de algún ruido estadístico es pequeña.

Otra pregunta que nos hicimos fue si también existía una correlación entre la predictibilidad óptima para cada nivel. Para estudiar esto, analizamos la correlación entre los promedios optimizados de los cuatro intentos para los 38 participantes. En otras palabras, en este caso las variables/columas de la matriz sobre la que se calcula la correlación es la predictibilidad optimizada para cada nivel y los datos/filas son cada uno de los 38 participantes. La matriz de correlación puede verse en la tabla 6.8, notemos que la correlación es muy fuerte entre el tercer intento y el último. Para los otros intentos no hemos encontrado valores que evidencien una correlación tan fuerte, sin embargo, la media de la matriz de correlación (sin la diagonal) es de 0,22, un valor que evidencia algún tipo de relación lineal entre la predictibilidad de los niveles. Es importante destacar que una correlación cercana a cero tampoco significa que los intentos sean necesariamente independientes, sólo que tal vez no sea lineal la relación entre ellos. Por otro lado, en la tabla 6.9 podemos ver el p-valor asociado a cada elemento de la matriz de correlación, esto es la probabilidad de obtener una correlación tan alta como la que se observó por azar. Allí podemos ver nuevamente que

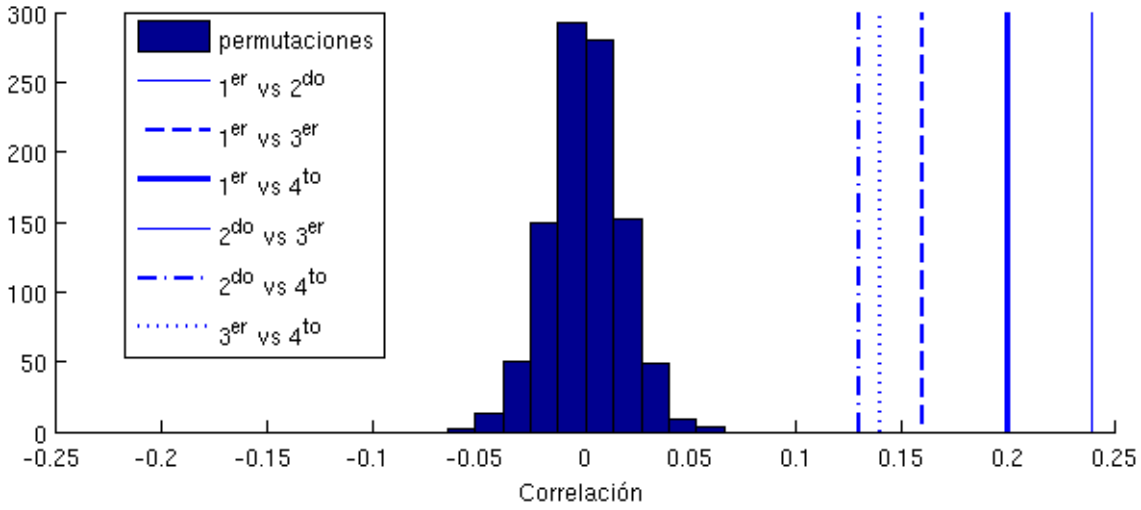

Fig. 6.9: Valores de correlación de 1000 permutaciones y valores de la matriz de la tabla 6.7

Intento	1^{er}	2^{do}	3^{er}	4^{to}
1^{er}	1	0,29	0,14	0,25
2^{do}	0,29	1	−0,07	0,07
3^{er}	0,14	−0,07	1	0,49
4^{to}	0,25	0,07	0,49	1

Tab. 6.8: Matriz de correlación de la predictibilidad óptima para cada nivel

la correlación obtenida entre los dos últimos niveles es fuertemente significativa ya que la probabilidad de obtenerla por azar es tan sólo de 0,002. En la figura 6.10, estudiaremos la probabilidad de haber obtenido una media de los valores de correlación igual a 0,22. Para esto, realizamos el mismo procedimiento de medir la correlación de la matriz de valores óptimos de los 38 participantes pero para 1000 permutaciones de ésta. En ese caso, la probabilidad de haber obtenido 0,22 como valor de correlación promedio es aproximadamente 0,12. Esto significativa que la predictibilidad óptima podría tener alguna relación lineal entre los niveles, aunque los resultados no son tan fuertes o significativos como para el caso de la estabilidad de la predictibilidad que estudiamos antes. Sin embargo, esto tampoco descarta otro tipo de relación no lineal entre los parámetros que optimizan la predictibilidad para cada persona en cada nivel.

Intento	1^{er}	2^{do}	3^{er}	4^{to}
1^{er}	1	0,08	0,41	0,12
2^{do}	0,08	1	0,70	0,66
3^{er}	0,41	0,70	1	0,002
4^{to}	0,12	0,66	0,002	1

Tab. 6.9: Matriz de p-valores asociados a la matriz de correlación de la tabla 6.8

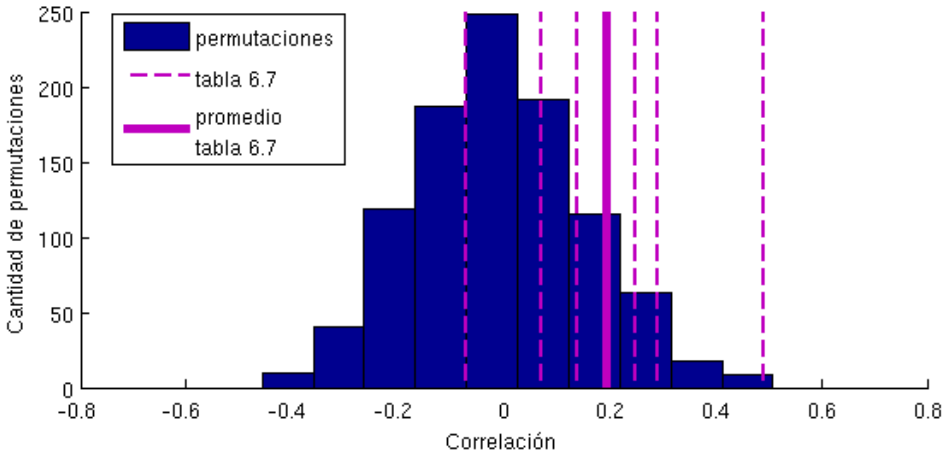

Fig. 6.10: Valores de correlación de 1000 permutaciones y valores de la matriz de la tabla 6.8

7. CONCLUSIONES

Quand une regle est fort composée, ce qui luy est conforme, passe pour irrégulier.
–Leibniz, Discours de métaphysique, 1686

El objetivo de nuestro trabajo es introducir un modelo que permita un mayor entendimiento del proceso de generación de secuencias aleatorias en los seres humanos y de las estructuras simbólicas formales que puedan captar cómo representamos y entendemos el azar.

Los mejores intentos de modelar cómo representamos en nuestras mentes al conocimiento que vamos adquiriendo del mundo que nos rodea, suelen ser versiones sencillas de cómo los científicos representan esos mismos conceptos en la teoría. En nuestro caso, buscamos trasladar las nociones no computables de la teoría de aleatoriedad computacional, a un modelo computable para comprobar si existe alguna relación entre estas y cómo se representa el azar en nuestras mentes.

Si bien el rendimiento del predictor puede ser igualado o superado por otros basados en redes neuronales o test estadísticos, no es nuestra intención crear un predictor óptimo, sino encontrar un lenguaje formal que sea efectivo para estudiar las variantes en la generación de secuencias aleatorias por los seres humanos que puedan presentarse ante distintos contextos o diversas características de la población en estudio.

Hemos destacado, al analizar las cadenas generadas en el último experimento (sección 6.2.2), que los participantes tenían un cierto conocimiento más formal de aleatoriedad que los llevaba a *intentar* evitar el error más común de alternar en exceso los símbolos de la cadena. Decimos *intentar* ya que, si bien la alternancia entre símbolos era prácticamente equiprobable, la frecuencia de aparición de subsecuencias de mayores longitudes evidenciaban una tendencia a favorecer aquellas con máxima alternación. Sin embargo, pese a esta dificultad, el modelo se muestra efectivo para diferenciar las cadenas generadas por los participantes de aquellas generadas por algoritmos pseudo-aleatorios o por ruido blanco. Esta efectividad la destacamos, tanto en la medida de complejidad donde los participantes tenían una diferencia significativamente menor en la complejidad de las cadenas que podían generar en comparación a los otros procesos (sección 6.2.3), como en la efectividad del algoritmo para predecir los símbolos de las cadenas de cada proceso (sección 6.2.5).

El poder contar con un lenguaje formal que se muestra efectivo para diferenciar la generación de las cadenas, nos provee una herramienta para estudiar las variables que afectan la generación del azar en los humanos. Un ejemplo de esto puede verse en el estudio sobre los efectos de la fatiga en la generación de las cadenas de la sección 6.2.4, aquí el modelo nos permite encontrar una diferencia significativa en la complejidad de las cadenas que evidencian una reducción de la complejidad con el paso del tiempo. También resulta

importante destacar la estabilidad de la predictibilidad de las secuencias de una persona ante sucesivos intentos como resaltamos al estudiar la correlación entre la predictibilidad de los participantes en cada intento según los distintos valores de los parámetros $G_{anteriores}, G_{futuros}$. Esta relación lineal entre los intentos, que se torna más significativa al fijar un valor que al analizar la predictibilidad óptima, hablan de una cierta estabilidad del proceso de generación de secuencias aleatorias en las personas que hacen más interesante al modelo como herramienta para estudiar variaciones producto de diversos efectos. Es una tarea para un trabajo futuro inferir cómo los parámetros o valores del modelo varían ante diversos contextos o características poblacionales.

Por otro lado, los resultados de nuestro predictor muestran que hay indicios de una relación entre cómo representamos la aleatoriedad en nuestras mentes y las definiciones teóricas de aleatoriedad de Martin-Löf y de la complejidad de Kolmogorov. Es decir, que efectivamente nuestra idea intuitiva del azar está relacionada, de algún modo, a la compresibilidad o complejidad de las cadenas en algún lenguaje interno.

8. TRABAJOS FUTUROS

The generation of random numbers is too important to be left to chance.
–Robert R. Coveyou

Como mencionábamos en las conclusiones del capítulo 7, ahora que contamos con un lenguaje formal que se muestra efectivo para diferenciar las cadenas generadas por personas de aquellas producidas por un generador de números pseudo-aleatorios o por ruido blanco, y que este modelo se muestra estable ante diversos intentos de los participantes, es necesario estudiar cómo varía la predictibilidad de las secuencias aleatorias generadas por las personas (o su complejidad) ante diversas situaciones tales como el tiempo disponible para generarlas, eventos de distracción, o ante distintas características de la población como la edad, enfermedades, etc. Especialmente, sería interesante profundizar el estudio en pacientes con fatiga crónica para analizar si la complejidad de las cadenas producidas por estos pacientes son menos complejas que las de un grupo de control y así avanzar en los indicios que remarcábamos en la sección 6.2.4 sobre cómo el cansancio puede degradar la complejidad o aleatoriedad de las cadenas que podemos producir.

Por otro lado, sería importante encontrar un vínculo entre el número mágico de Miller que mencionábamos en el capítulo 4, el cual sostiene que la memoria a corto plazo está limitada por un número cercano a siete items [Mil56], y los estudios más recientes [Bad74, Bra09, Bro75, C&S73, Cow01, Est72, Gob04, Hal05, Hal98, Luc97, Pyl88] que ubican a este límite en 4 ± 1 *trozos* de información. La definición de *trozo* es, hasta la actualidad, un poco vaga e implica a un conjunto de objetos que son tratados conjuntamente como una sola unidad. En los últimos meses, Fabien Mathy y Jacob Feldman [Mat12] proponen una definición derivada de la noción de complejidad de Kolmogorov de *trozo* como una unidad en algún código máximamente comprimido. Su estudio muestra que el límite de 4 *trozos* de los estudios actuales es equivalente a los 7 items sin comprimir de Miller, aunque no ahondan en una función general para calcular cuantos *trozos* tiene algún objeto genérico ni en cómo podría ser el método que agrupe a los objetos en *trozos*. Creemos que alguna variante del modelo presentado en esta tesis, podría descubrir alguna evidencia que ayude a reforzar la relación entre los 7 items sin comprimir que según Miller podemos recordar y los cuatro *trozos* de los estudios actuales.

Finalmente, creemos que sería interesante el diseño de un algoritmo de aprendizaje automático que pueda calibrar sus parámetros con el objetivo de maximizar la tasa de predicción del algoritmo mientras la cadena es generada por una persona o ante diversos intentos de ésta por producir nuevas secuencias.

BIBLIOGRAFÍA Y REFERENCIAS

[Bad74] Baddeley, A., & Hitch, G. *Working memory. In G. H. Bower (Ed.), Recent advances in learning and motivation.* New York, NY: Academic Press. 647-667. 1974

[Bra09] Brady, T. F., Konkle, T., & Alvarez, G. A. *Compression in visual working memory: Using statistical regularities to form more efficient memory representations.* Journal of Experimental Psychology: General, 138:487–502. 2009

[Bro75] Broadbent, D. *The magic number seven after fifteen years.* Studies in long-term memory. New York: Wiley. 1975

[C&S73] Chase, W.G., & Simon, H.A *Perception in chess.* Cognitive Psychology, 4:55–81. 1973

[Cha75] Chaitin, G. *A theory of program size formally identical to information theory.* Journal of the ACM, 22:329-340, 1975.

[Cow01] Cowan, N. *The magical number 4 in short-term memory: A reconsideration of mental storage capacity.* Behavioral and Brain Sciences, 24:87–185. 2001.

[Dow09] Downey, R. *Algorithmic Randomness and Complexity.* 2009

[Est72] Estes, W. K. *An associative basis for coding and organization in memory.* Coding processes in human memory (pp. 161–190). Washington, DC: Winston. 1972.

[Fak97] Falk, R. & Konold, C. *Making sense of randomness: Implicit encoding as a bias for judgment.* Psychological Review 104:301-318. 1997.

[Gob04] Gobet, F., & Clarkson, G. *Chunks in expert memory: Evidence for the magical number four. . . or is it two?.* Memory, 12:732–747. 2004.

[Gri01] Griffiths, T. & Tenenbaum, J. *Randomness and Coincidences: Reconciling Intuition and Probability Theory.* Proceedings of the 23rd Annual Conference of the Cognitive Science Society, 370-375. 2001

[Hal05] Halford, G. S., Baker, R., McCredden, J. E., & Bain, J. D. *How many variables can humans process?.* Psychological Science, 16:70–76. 2005.

[Hal98] Halford, G. S., Wilson, W. H., & Phillips, W. *Processing capacity defined by relational complexity: Implications for comparative, developmental and cognitive psychology.* Behavioral and Brain Sciences, 21:803–831. 1998.

[Jon99] Jongman, L., Meijman T. & De Jong, R. *The Working Memory Hypothesis of Mental Fatgiue.* 1999

[Kol65] Kolmogorov, A. N. *Three approaches to the quantitative definition of information.* Problems of Information Transmission. 1965.

[Lev74] Levin, L. A. *Laws of information conservation (non-growth) and aspects of the foundation of probability theory.* Problems of Information Transmission, 10:206-210, 1974.

[LiV08] Li, M., & Vitányi, P. *An introduction to Kolmogorov complexity and its applications.* Springer, New York, 3er edition, 2008.

[Löf66] Martin-Löf, P. *The definition of random sequences.* Information and Control, 9:602-619, 1966.

[Löf71] Martin-Löf, P. *Complexity oscillations in infinite binary sequences.*Z. Wahrscheinlichkeitstheorie verw. Gebiete. 1971.

[Luc97] Luck, S. J., & Vogel, E. K. *The capacity of visual working memory for features and conjunctions.* Nature, 390:279–281. 1997.

[Mat12] Mathy, F., & Feldman, J. *What's magic about magic numbers? Chunking and data compression in short-term memory.* Cognition, 122:346–362. 2012.

[M&N98] Matsumoto M., & Nishimura, T. *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator.* ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30. 1998.

[Mil56] Miller, G. A. *The magical number seven, plus or minus two: Some limits on our capacity for processing information.* Psychological Review 63 (2):81–97. 1956

[Mis19] Mises, R. von. *Grundlagen der Wahrscheinlichkeitsrechnung.* Mathematische Zeitschrift, 5:52-99, 1919.

[Nic02] Nickerson, R.S. *The production and perception of randomness.* Psychological Review, 109(2):330–357. 2002

[Pyl88] Pylyshyn, Z., & Storm, R. *Tracking multiple independent targets: Evidence for a parallel tracking mechanism.* . Spatial Vision, 3:1–19. 1988.

[Rei34] Reichenbach, H. *The Theory of Probability.* The theory of probability. University of California Press, Berkeley. 1934/1949.

[Sch71] Schnorr, C.P. *A unified approach to the definition of a random sequence.* Mathematical Systems Theory, 5:246-258, 1971.

[Sol60a] Solomonoff, R. Paper from *Conference on Cerebral Systems and Computers.* California Institute of Technology. 1960.

[Sol60b] Solomonoff, R. *A Preliminary Report on a General Theory of Inductive Inference.* Report V-131, Zator Co., Cambridge. 1960.

[Sol64] Solomonoff, R. *A formal theory of inductive inference, I and II. Information and Control.* 1964.

[Ten11] Tenenbaum, J. & Kemp, C. & Griffiths, J. & Goodman, N. *How to Grow a Mind: Statistics, Structure, and Abstraction.* Science 11 March 2011, 331:1279-1285. 2011

[Wag72] Wagenaar, W.A. *Generation of random sequences by human subjects: A critical survey of literature.* Psychological Bulletin, 77:65–72. 1972

8. ANEXO

Código fuente:

https://subversion.assembla.com/svn/juegopredictores.ruby/trunk/lib/

Tablas de correlación de precitibilidad por nivel:

Las siguientes tablas de correlación para cada participante son las utilizadas para calcular los valores promedio de la tabla 6.7

Intento	1^{er}	2^{do}	3^{er}	4^{to}
1^{er}	1	0,58	0,47	0,27
2^{do}	0,58	1	0,81	0,52
3^{er}	0,47	0,81	1	0,75
4^{to}	0,27	0,52	0,75	1
1^{er}	1	0,92	0,92	0,85
2^{do}	0,92	1	0,90	0,86
3^{er}	0,92	0,90	1	0,82
4^{to}	0,85	0,86	0,82	1
1^{er}	1	−0,45	−0,24	0,29
2^{do}	−0,45	1	−0,10	−0,07
3^{er}	−0,24	−0,10	1	−0,15
4^{to}	0,29	−0,07	−0,15	1
1^{er}	1	0,58	0,63	0,65
2^{do}	0,58	1	0,91	0,80
3^{er}	0,63	0,91	1	0,72
4^{to}	0,65	0,80	0,72	1
1^{er}	1	0,32	0,00	−0,08
2^{do}	0,32	1	0,00	−0,66
3^{er}	0,00	0,00	1	0,00
4^{to}	−0,08	−0,66	0,00	1

Intento	1^{er}	2^{do}	3^{er}	4^{to}
1^{er}	1	0,41	0,08	0,72
2^{do}	0,41	1	0,53	0,40
3^{er}	0,08	0,53	1	0,12
4^{to}	0,72	0,40	0,12	1
1^{er}	1	–	0,33	0,06
2^{do}	–	–	–	–
3^{er}	0,33	–	1	0,38
4^{to}	0,06	–	0,38	1
1^{er}	1	−0,31	−0,39	−0,30
2^{do}	−0,31	1	0,57	0,66
3^{er}	−0,39	0,57	1	0,78
4^{to}	−0,30	0,66	0,78	1
1^{er}	1	0,65	0,82	0,72
2^{do}	0,65	1	0,53	0,61
3^{er}	0,82	0,53	1	0,81
4^{to}	0,72	0,61	0,81	1
1^{er}	1	−0,50	−0,38	−0,25
2^{do}	−0,50	1	0,16	0,60
3^{er}	−0,38	0,16	1	−0,20
4^{to}	−0,25	0,60	−0,20	1

Intento	1^{er}	2^{do}	3^{er}	4^{to}
1^{er}	1	−0,29	−0,36	−0,07
2^{do}	−0,29	1	0,77	0,34
3^{er}	−0,36	0,77	1	0,44
4^{to}	−0,07	0,34	0,44	1
1^{er}	1	0,68	−0,35	−0,04
2^{do}	0,68	1	−0,55	−0,58
3^{er}	−0,35	−0,55	1	0,43
4^{to}	−0,04	−0,58	0,43	1
1^{er}	1	−0,56	0,56	−0,11
2^{do}	−0,56	1	−0,51	0,02
3^{er}	0,56	−0,51	1	−0,03
4^{to}	−0,11	0,02	−0,03	1
1^{er}	1	−0,32	0,48	0,44
2^{do}	−0,32	1	−0,29	−0,52
3^{er}	0,48	−0,29	1	0,33
4^{to}	0,44	−0,52	0,33	1
1^{er}	1	0,39	0,29	0,48
2^{do}	0,39	1	0,40	−0,01
3^{er}	0,29	0,40	1	0,44
4^{to}	0,48	−0,01	0,44	1
1^{er}	1	0,93	0,83	0,65
2^{do}	0,93	1	0,79	0,53
3^{er}	0,83	0,79	1	0,53
4^{to}	0,65	0,53	0,53	1
1^{er}	1	−0,06	−0,16	−0,23
2^{do}	−0,06	1	−0,09	0,12
3^{er}	−0,16	−0,09	1	−0,28
4^{to}	−0,23	0,12	−0,28	1
1^{er}	1	−0,15	0,06	−0,10
2^{do}	−0,15	1	0,42	−0,36
3^{er}	0,06	0,42	1	−0,82
4^{to}	−0,10	−0,36	−0,82	1
1^{er}	1	0,17	−0,32	−0,09
2^{do}	0,17	1	0,12	−0,21
3^{er}	−0,32	0,12	1	−0,09
4^{to}	−0,09	−0,21	−0,09	1
1^{er}	1	−0,48	−0,34	−0,18
2^{do}	−0,48	1	0,20	0,08
3^{er}	−0,34	0,20	1	0,71
4^{to}	−0,18	0,08	0,71	1

Intento	1^{er}	2^{do}	3^{er}	4^{to}
1^{er}	1	0,50	0,21	0,66
2^{do}	0,50	1	−0,01	0,64
3^{er}	0,21	−0,01	1	0,011
4^{to}	0,66	0,64	0,11	1
1^{er}	1	0,96	0,86	0,80
2^{do}	0,96	1	0,90	0,74
3^{er}	0,86	0,90	1	0,68
4^{to}	0,80	0,74	0,68	1
1^{er}	1	0,12	−0,63	0,49
2^{do}	0,12	1	−0,43	0,22
3^{er}	−0,63	−0,43	1	−0,49
4^{to}	0,49	0,22	−0,49	1
1^{er}	1	0,13	−0,09	0,41
2^{do}	0,13	1	−0,47	0,32
3^{er}	−0,09	−0,47	1	−0,09
4^{to}	0,41	0,32	−0,09	1
1^{er}	1	0,85	0,30	−0,19
2^{do}	0,85	1	0,29	−0,46
3^{er}	0,30	0,29	1	−0,16
4^{to}	−0,19	−0,46	−0,16	1
1^{er}	1	0,74	0,19	0,42
2^{do}	0,74	1	0,39	0,41
3^{er}	0,19	0,39	1	−0,16
4^{to}	0,42	0,41	−0,16	1
1^{er}	1	0,54	−0,19	0,59
2^{do}	0,54	1	0,02	−0,10
3^{er}	−0,19	0,02	1	0,00
4^{to}	0,59	−0,10	0,00	1
1^{er}	1	0,01	−0,02	−0,32
2^{do}	0,01	1	0,05	−0,07
3^{er}	−0,02	0,05	1	−0,35
4^{to}	−0,32	−0,07	−0,35	1
1^{er}	1	0,49	−0,01	0,52
2^{do}	0,49	1	−0,13	0,87
3^{er}	−0,01	−0,13	1	−0,39
4^{to}	0,52	0,87	−0,39	1
1^{er}	1	0,08	−0,39	–
2^{do}	−0,58	1	0,47	–
3^{er}	−0,39	0,47	1	–
4^{to}	–	–	–	–

Intento	1^{er}	2^{do}	3^{er}	4^{to}
1^{er}	1	0,87	0,94	−0,65
2^{do}	0,87	1	0,88	−0,48
3^{er}	0,94	0,88	1	−0,51
4^{to}	−0,65	−0,48	−0,51	1
1^{er}	1	0,00	−0,10	−0,24
2^{do}	0,00	1	0,01	0,45
3^{er}	−0,10	0,01	1	0,07
4^{to}	−0,24	0,45	0,07	1
1^{er}	1	−0,32	−0,24	0,25
2^{do}	−0,32	1	0,50	−0,28
3^{er}	−0,24	0,50	1	−0,39
4^{to}	0,25	−0,28	−0,39	1
1^{er}	1	0,85	0,94	−0,37
2^{do}	0,85	1	0,75	−0,57
3^{er}	0,94	0,75	1	−0,25
4^{to}	−0,37	−0,57	−0,25	1

Intento	1^{er}	2^{do}	3^{er}	4^{to}
1^{er}	1	−0,19	−0,35	−0,14
2^{do}	−0,19	1	0,31	−0,25
3^{er}	−0,35	0,31	1	0,63
4^{to}	−0,14	−0,25	0,63	1
1^{er}	1	0,56	0,29	0,56
2^{do}	0,56	1	0,29	0,57
3^{er}	0,29	0,29	1	0,22
4^{to}	0,56	0,57	0,22	1
1^{er}	1	−0,51	0,64	0,58
2^{do}	−0,51	1	−0,61	−0,77
3^{er}	0,64	−0,61	1	0,61
4^{to}	0,58	−0,77	0,61	1
1^{er}	1	0,41	0,50	0,26
2^{do}	0,41	1	0,19	0,27
3^{er}	0,50	0,19	1	0,12
4^{to}	0,26	0,27	0,12	1