Implementación de algoritmos de *Machine Learning* para la identificación de relaciones familiares e identificación de desaparecidos mediante STRs de ADN autosómico.

Juan Antonio Luque Gutiérrez
Máster universitario en Bioinformática y bioestadística UOC-UB
TFM-Bioinformática y Bioestadística. Area 5

Esteban Vegas Lozano
Alexander Sánchez Pla

2 de enero de 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Implementación de algoritmos de Machine Learning para la identificación de relaciones familiares e identificación de desaparecidos mediante STRs de ADN autosómico.*
Nombre del autor:	*Juan Antonio Luque Gutiérrez*
Nombre del consultor/a:	*Esteban Vegas Lozano*
Nombre del PRA:	*Alexander Sánchez Pla*
Fecha de entrega (mm/aaaa):	02/2019
Titulación::	*Máster universitario en Bioinformática y bioestadística UOC-UB*
Área del Trabajo Final:	*TFM-Bioinformática y Bioestadística. Area 5*
Idioma del trabajo:	*Español*
Palabras clave	*Machine Learning, ADN, Identificación de Desaparecidos*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

En el presente Trabajo Fin de Máster, se han implementado y evaluado la efectividad de diversos algoritmos de machine learning para determinar la relación familiar entre dos individuos, con el fin de poder aplicar estos algoritmos en la identificación de desaparecidos en el ámbito forense. Todo el proceso ha sido implementado con R y Rstudio.
Siguiendo la herencia mendeliana, se generaron los perfiles de ADN de marcadores Short Tandem Repeat (STR) autosómicos para un conjunto de grandes familias "sintéticas". Para entrenar y validar los diversos modelos ensayados, se crearon varios conjuntos de datos seleccionando en estas familias sintéticas diversas relaciones familiares entre dos individuos.
Tras la evaluación de los diversos modelos, el algoritmo se ha implementado y entrenado como una red neuronal con keras/tensorflow, de forma que pueda ser empleado posteriormente para la predicción de la relación familiar en base a los perfiles de ADN que se puedan obtener en un suceso de víctimas múltiples para las víctimas y los familiares.
Se han simulado 10 escenarios de sucesos de víctimas múltiples (desde 6 hasta 200 víctimas) con diverso grado de dificultad. La aplicación de la red neuronal a dichos escenarios ha resuelto la mayoría de las identificaciones en pocos minutos. En algunos escenarios con un 100% de efectividad. En el escenario más complejo con relaciones lejanas, la clasificación ha sido del 50%.

Abstract (in English, 250 words or less):

The goal of this Master's Thesis has been to implement and evaluate several machine learning algorithms in order to determine the kinship between two individuals in the context of Disaster Victim Identification (DVI). The development has been done with R and Rstudio.
Following the mendelian transmission, the DNA profiles of autosomal Short Tandem Repeat (STR) markers were generated for a group of extensive synthetic families. The different machine learning models were trained and validated using as input data pairs of individuals selected from these synthetic families, with diverse kinships.
After evaluating several models, a keras/tensorflow neural network was implemented and trained so that it could be used to predict the relationship of two individuals given the multiple DNA profiles obtained in a DVI.
Ten settings of events with multiple victims (from 6 to 200 victims) were simulated, with different levels of difficulty. The inference of the neural network in such settings has solved most of the identifications in a few minutes. In some of the settings, the identifications were solved with a 100% accuracy, while the most complicated setting with the farthest family relationships could only obtain a 50% of accuracy.

Índice

Índice de figuras

1. Contexto y Justificación del Trabajo

1.1. Descripción general.

En el presente Trabajo Fin de Máster, se va a implementar y evaluar la efectividad de diversos algoritmos de *machine learning* para determinar la existencia o no de una cierta relación familiar entre individuos, con el fin de poder aplicar estos algoritmos en la identificación de desaparecidos en el ámbito forense. Como datos de entrada se usará un conjunto de grandes familias "sintéticas" generadas siguiendo la herencia mendeliana, seleccionando en ellas diversas relaciones familiares entre dos individuos. Con el entrenamiento, se comprobará si los algoritmos son capaces de "aprender" las leyes genéticas que subyacen en las relaciones familiares y por tanto confirmar de forma efectiva la relación familiar, en una primera fase como respuesta binaria sí/no y en una segunda fase numéricamente.
Si la aplicación de los algoritmos de *machine learning* tienen una efectividad adecuada se diseñará un clasificador con múltiples categorías de forma que pueda asignar una pareja de individuos a una de las relación familiares investigadas.

1.2. Justificación del TFG.

La identificación de víctimas en casos de desaparecidos tiene limitaciones a medida que las relaciones familiares son lejanas y especialmente en el caso de desastres en masa, cuando el número de víctimas es alto es complicado. El potencial de los algoritmos algoritmos para la clasificación de datos puede ayudar a solventar estos casos. En el ámbito forense ya se han empezado a implementar algoritmos de *machine learning* para la clasificación de haplotipos de marcadores de cromosoma Y [1] y en la deconvolución de mezclas de ADN con marcadores STR [2] [3], pero no en el ámbito del parentesco. Considero que tienen un gran potencial los algoritmos de *machine learning* en este último ámbito y que es necesario confirmar si son capaces de iguales e incluso mejorar los sistemas actualmente establecidos o por contra, su efectividad no es suficiente. En caso de que sean efectivos, abrirían una puerta a la clasificación masiva de datos en casos de desastres en masa que serían de gran ayuda para la resolución de los mismos.

2. Objetivos:

- Objetivo 1. Conseguir un conjunto de datos de marcadores STR en familias con cinco generaciones, adecuados para el entrenamiento de los algoritmos.
- Objetivo 2. Encontrar un algoritmo de *machine learning* que iguale o supere los métodos clásicos de estudio del parentesco para supuestos específicos de parentesco.
- Objetivo 3. Implementar un clasificador que asigne a un par de individuos una relación de parentesco entre ellos y ampliarlo a grandes conjuntos de individuos.

3. Enfoque y método a seguir:

Se han implementado varios algoritmos de *machine learning* en R y Rstudio. El uso de keras y tensorflow se plantea como una herramienta de más compleja implementación pero potencialmente más efectiva. Se descartó el uso de redes bayesianas por la complejidad en su diseño.
La estrategia elegida es la implementación de diversas funciones y algoritmos en R, usando RStudio como interfaz al tener un mejor desempeño con la misma. Adicionalmente, el uso de libretas R (*R notebooks*) con RMarkdown permite conseguir resultados de forma más sencilla y auditable, a la vez que simplifica la redacción y documentación del código en los diversos entregables y la memoria del TFM. Adicionalmente, la

existencia del paquete *Familias* permite realizar los cálculos con los algoritmos clásicos en R directamente desde el mismo conjunto de datos, además de poder usar los objetos definidos para modelizar los pedigrís.

4. Planificación del trabajo:

4.1. Tareas:

- 1.a.1 Aprendizaje de *Familias*. Estudiar el funcionamiento del programa familias y de los objetos que utiliza, especialmente como se define el pedigrí. 3 días.
- 1.a.2 Diseño del árbol familiar. Crear una familia suficientemente grande como para dar cabida a diversas relaciones familiares lejanas y cercanas. 3 días.
- 1.a.3 Seleccionar los marcadores y frecuencias alélicas poblacionales a utilizar. 1 día.
- 1.a.4 Importar las frecuencias a un objeto de R. 2 días.
- 1.a.5 Crear genotipos. Diseñar una función que genere individuos aleatorios para los fundadores del pedigrí y para el resto de individuos siguiendo la trasmisión mendeliana. 10 días.
- 1.a.6 Gráfico de pedigrí. Ajustar los parámetros de la función plot para pedigrís, a fin de conseguir una representación adecuada. 2 días.

- 1.b.1 Calcular LR. Diseñar una función que calcule el LR (Likelihood ratio, valoración del parentesco) para un subconjunto de familiares especificado. 10 días.
- 1.b.2 Datos de entrenamiento (genotipos). Generar un archivo con 40 000 familias de entrenamiento y 40 000 de test con el genotipo correspondiente a todos los individuos. 2 días.
- 1.b.3 Definir el listado de relaciones familiares a investigar. 2 días.
- 1.b.4 Datos de entrenamiento (resultados). Generar la categorización y el valor de LR para cada relación familiar a investigar más la categoría no relacionado. 10 días.

- 1.c.1 Piloto con red neuronal simple. Diseñar una red neuronal simple y entrenar con cantidades crecientes de datos (1000, 5000, 10000, 40000) para evaluar el tiempo requerido y si el equipo utilizado tiene potencia de cálculo suficiente. 4 días.
- 1.c.2 Piloto 2. Repetir el piloto1, si ha sido satisfactorio, con redes más complejas para ver los límites. Estimar los requerimientos de hardware. 3 días.
- 1.c.3 Crear instancia AWS. En caso de que el hardware no sea suficiente, estudiar el funcionamiento de AWS y crear una máquina virtual con potencia suficiente para el proyecto. Estimar si el coste sería asequible. 10 días.

- 1.d.1 Definir la lista de 6 algoritmos para probar. Comprobar si los admite los datos creados. 3 días.
- 1.d.2 Recodificar los datos si hace falta para algún algoritmo. 3 días.

- 2.a.1 Algoritmo 1, entrenamiento básico. Entrenar el algoritmo 1 con relaciones familiares básicas. 2 días.
- 2.a.2 Algoritmo 2, entrenamiento básico. Entrenar el algoritmo 2 con relaciones familiares básicas y 1000 familias. 2 días.
- 2.a.3 Algoritmo 3, entrenamiento básico. Entrenar el algoritmo 3 con relaciones familiares básicas y 1000 familias. 2 días.
- 2.a.4 Algoritmo 4, entrenamiento básico. Entrenar el algoritmo 4 con relaciones familiares básicas y 1000 familias. 2 días.
- 2.a.5 Algoritmo 5, entrenamiento básico. Entrenar el algoritmo 5 con relaciones familiares básicas y 1000 familias. 2 días.
- 2.a.6 Algoritmo 6, entrenamiento básico. Entrenar el algoritmo 6 con relaciones familiares básicas y 1000 familias. 2 días.
- 2.a.7 Seleccionar algoritmos candidatos. 2 días.

- 2.b.1 Entrenamiento y test completo. Realizar el entrenamiento y test con las 40 000 familias para cada una de las relaciones familiares de los algoritmos seleccionados. 10 días.

- 2.c.1 Realizar curvas de discriminación para los algoritmos ML y para algoritmos clásicos. 10 días.
- 2.c.2 Realizar comparación numérica de resultados y graficar. 10 días.

- 3.a.1 Conjunto de datos "multirrelación". Diseñar un conjunto de datos de 40 000 familias en el que estén representadas todas las relaciones familiares. 3 días.
- 3.a.2 Seleccionar 3 algoritmos de clasificación. 4 días.
- 3.a.3 Entrenar clasificación. Clasificar un pull de 200 relaciones con conjunto reducido de datos de entrenamiento (10 000). Mejorar los modelos. 10 días.

- 3.b.1 Entrenamiento final. Realizar el entrenamiento con el conjunto de entrenamiento de 40 000 relaciones. 4 días.
- 3.b.2 Simulación de catástrofes. Hacer conjuntos de muestras simulando escenarios de complejidad creciente en relaciones y número de individuo (de 6 a 1000). 2 días.
- 3.b.3 Clasificación de catástrofes. Hacer el test con los diversos escenarios de catástrofes. Valorar su efectividad. 5 días.

- 3.c.1 Aprender a crear una aplicación Shiny. 15 días.
- 3.c.2 Generar una aplicación Shiny con el conjunto entrenado para uso en laboratorios forenses. 30 días.

- Elaboración del plan de trabajo. 15 días.

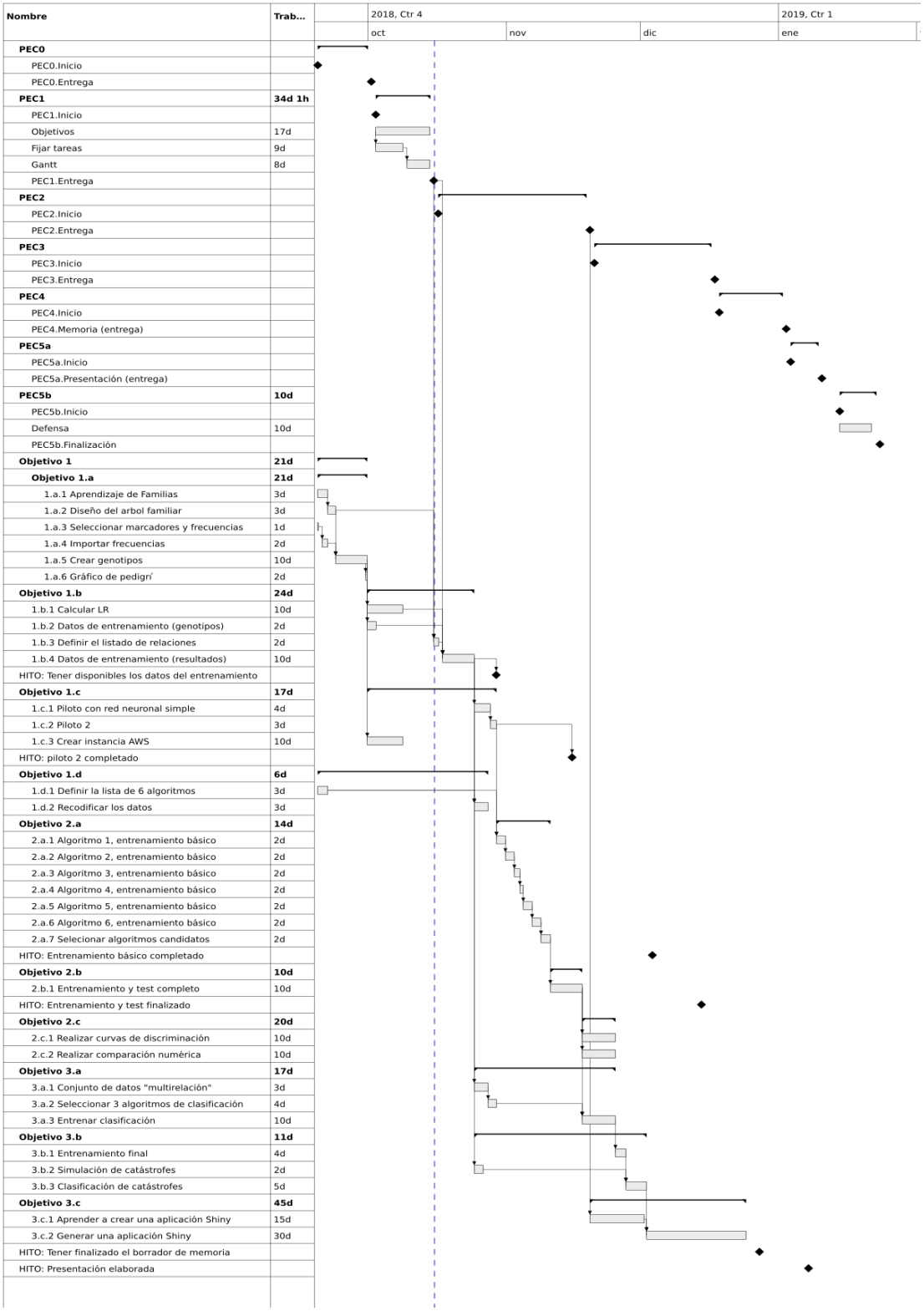
- Elaboración de la memoria. 30 días.

- Elaboración de la presentación. 20 días.

4.2. Hitos

29 octubre 2018. Tener disponibles los datos de entrenamiento.
15 noviembre 2018. Confirmar que los algoritmos y hardware funcionan adecuadamente (piloto 2 completado).
3 diciembre 2018. Entrenamiento básico completo.
14 diciembre 2018. Entrenamiento y test final completo.
27 diciembre 2018. Tener finalizado el borrador de memoria.
7 enero 2017. Presentación elaborada.

4.3. Calendario:



Nombre	Trab...
PEC0	
PEC0.Inicio	
PEC0.Entrega	
PEC1	**34d 1h**
PEC1.Inicio	
Objetivos	17d
Fijar tareas	9d
Gantt	8d
PEC1.Entrega	
PEC2	
PEC2.Inicio	
PEC2.Entrega	
PEC3	
PEC3.Inicio	
PEC3.Entrega	
PEC4	
PEC4.Inicio	
PEC4.Memoria (entrega)	
PEC5a	
PEC5a.Inicio	
PEC5a.Presentación (entrega)	
PEC5b	**10d**
PEC5b.Inicio	
Defensa	10d
PEC5b.Finalización	
Objetivo 1	**21d**
Objetivo 1.a	**21d**
1.a.1 Aprendizaje de Familias	3d
1.a.2 Diseño del arbol familiar	3d
1.a.3 Seleccionar marcadores y frecuencias	1d
1.a.4 Importar frecuencias	2d
1.a.5 Crear genotipos	10d
1.a.6 Gráfico de pedigrí	2d
Objetivo 1.b	**24d**
1.b.1 Calcular LR	10d
1.b.2 Datos de entrenamiento (genotipos)	2d
1.b.3 Definir el listado de relaciones	2d
1.b.4 Datos de entrenamiento (resultados)	10d
HITO: Tener disponibles los datos del entrenamiento	
Objetivo 1.c	**17d**
1.c.1 Piloto con red neuronal simple	4d
1.c.2 Piloto 2	3d
1.c.3 Crear instancia AWS	10d
HITO: piloto 2 completado	
Objetivo 1.d	**6d**
1.d.1 Definir la lista de 6 algoritmos	3d
1.d.2 Recodificar los datos	3d
Objetivo 2.a	**14d**
2.a.1 Algoritmo 1, entrenamiento básico	2d
2.a.2 Algoritmo 2, entrenamiento básico	2d
2.a.3 Algoritmo 3, entrenamiento básico	2d
2.a.4 Algoritmo 4, entrenamiento básico	2d
2.a.5 Algoritmo 5, entrenamiento básico	2d
2.a.6 Algoritmo 6, entrenamiento básico	2d
2.a.7 Selecionar algoritmos candidatos	2d
HITO: Entrenamiento básico completado	
Objetivo 2.b	**10d**
2.b.1 Entrenamiento y test completo	10d
HITO: Entrenamiento y test finalizado	
Objetivo 2.c	**20d**
2.c.1 Realizar curvas de discriminación	10d
2.c.2 Realizar comparación numérica	10d
Objetivo 3.a	**17d**
3.a.1 Conjunto de datos "multirelación"	3d
3.a.2 Seleccionar 3 algoritmos de clasificación	4d
3.a.3 Entrenar clasificación	10d
Objetivo 3.b	**11d**
3.b.1 Entrenamiento final	4d
3.b.2 Simulación de catástrofes	2d
3.b.3 Clasificación de catástrofes	5d
Objetivo 3.c	**45d**
3.c.1 Aprender a crear una aplicación Shiny	15d
3.c.2 Generar una aplicación Shiny	30d
HITO: Tener finalizado el borrador de memoria	
HITO: Presentación elaborada	

5. Descripción entregables

No se realizan entregables. El código está embebido en el presente trabajo.

6. Resumen resto de capítulos

Flujo de pruebas

Creación pedigrí, relaciones familiares y genotipos.
Verificación de las funciones de creación de genotipos.
Creación de los archivos de entrenamiento y test.
Recodificación de ficheros.
Entrenamiento de una red neuronal simple con una única relación de parentesco.
Evaluación de ochos algoritmos de *machine learning* con una única relación de parentesco.
Creación de conjuntos de datos con múltiples relaciones de parentesco.
Evaluación de ochos algoritmos de *machine learning* con el conjunto de múltiples relaciones de parentesco.
Selección del algoritmo final y optimización.
Entrenamiento final del algoritmo optimizado. Creación de 10 escenarios de catástrofe con diverso grado de dificultad.
Predicción de clasificación para cada uno de los escenarios.
Evaluación final de la resolución de los escenarios.

7. Introducción

La (hemo)genética forense ha sido pionera entre las ciencias forenses a la hora de ofrecer a los tribunales una valoración estadística de los resultados de forma objetiva. Ya en 1971 Hummel[4] publicó unos detallados listados con los cálculos para las pruebas de paternidad con gran parte de los marcadores disponibles en la época (principalmente grupos sanguíneos). Hoy en día, se dispone de gran cantidad de herramientas mucho más polimórficas (discriminativas) permitiendo solventar casos bastante complejos[a]. Por lo general, los estudios de parentesco se realizan con muestras "abundantes", por lo que la limitación viene dada por el número de marcadores que tenga validados el laboratorio y por el número y relación de los familiares a estudiar. En el caso de grandes catástrofes (DVI)[b], la situación es más compleja, ya que las muestras no siempre son abundantes ni el número y proximidad genética de los familiares. En este último caso, adicionalmente, hay una necesidad de resolver las identificaciones lo antes posible de cara a la salud psicológica de los familiares, por lo que disponer de herramientas que faciliten dicha identificación de forma sencilla y rápida es fundamental.

¿Protocolo suceso victimas múltiples?

Hay herramientas[c] que resuelven probabilísticamente la identificación genética cuando se dispone de familiares próximos en número suficiente. Sin embargo, en el manejo de catástrofes la mayoría de las herramientas disponibles[d] requieren de una infraestructura suficiente y la resolución no siempre es todo lo rápida que se requiere.

Hoy en día, los algoritmos de **Machine Learning** se están implementando para resolver o mejorar gran número de problemas que con tecnologías *clásicas* eran difíciles o lentos de solucionar, llegando a formar parte de la vida cotidiana. No es raro que en nuestro bolsillo (teléfono móvil) dispongamos de algoritmos

[a] A lo largo de todo el trabajo, salvo mención expresa, se hará uso de marcadores STRs autosómicos analizados con electroforesis capilar, y por tanto, marcadores con alelos codominantes, que serán referenciados por sus repeats, según las recomendaciones de la ISFG[5].

[b] En inglés: *Disaster Victim Identification*

[c] Entre otras Familias (www.familias.name), DNA-VIEW (www.dna-view-com)

[d] CODIS (https://www.fbi.gov/services/laboratory/biometric-analysis/codis), Bonaparte (https://www.bonaparte-dvi.com/familial.php), M-FISys (http://www.genecodesforensics.com/software/)

de *Machine Learning*. En ciencia y tecnología, cada vez son más las implementaciones de estos algoritmos para resolver temas complejos como el reconocimiento facial, detección de objetos, segmentación semántica, reconocimiento de voz/asistentes personales. En el campo de la genética forense, se están empezando aplicar los algoritmos de ML a mezclas.

En el presente estudio se de parentesco se va a explorar la posibilidad de implementar algoritmos de ML en el estudio del parentesco y en identificación de víctimas en catástrofes.

7.1. Parentesco genético[e]

7.1.1. Herencia mendeliana

El genoma de cada individuo procede de su padre y de su madre. Para cada marcador (autosómico[f]) un individuo presenta dos alelos, uno paterno y otro materno. Si los dos alelos son iguales, no hay forma de distinguirlos, por lo que analíticamente sólo se observará uno con el doble de señal, es lo que denominamos homozigoto. Si son diferentes, individuo heterozigoto, se observarán dos alelos, pero no se puede determinar cual procede del padre y cual de la madre.[g]
De forma simplificada y genérica se puede decir que el cálculo de paternidad se basa en calcular la probabilidad del genotipo del hijo (y la de los padres o familiares implicados en la relación de parentesco, concretamente se trata de una probabilidad condicional).

De forma clásica, la forma de calcular cuando el hijo es homozigoto y heterozigoto es diferente. Para un hijo homozigoto (AA), la probabilidad del genotipo del hijo (Pr(AA)) es igual a la probabilidad de que el padre transmita el alelo A ($P \to A$) multiplicado por la probabilidad de que la madre le transmita el alelo A ($M \to A$):

$$Pr(AA) = (P \to A) \times (M \to A)$$

Para un hijo heterozigoto (AB) hay que considerar la opción de que el padre le transmita el alelo A y la madre el B o que el padre le transmita el B y la madre el A:

$$Pr(AB) = (P \to A) \times (M \to B) + (P \to B) \times (M \to A)$$

Estas fórmulas son las que se usan continuamente para valorar las paternidades, y en función de cada tipo de caso, este padre (P) y esta madre (M) variarán y por tanto las probabilidades de transmisión variarán. De forma genérica un progenitor de genotipo conocido homozigoto su probabilidad de trasmisión del alelo es 1, ya que no tiene otra opción, en el 100 % de los casos lo trasmite. Si el progenitor es heterozigoto, la probabilidad de trasmisión de cada alelo es la mitad, o sea 0,5. Si el progenitor no tiene el alelo, la probabilidad de trasmisión será 0.

Si se desconoce el genotipo del progenitor la probabilidad de trasmisión será la frecuencia poblacional para dicho marcador, es decir la probabilidad de trasmisión del alelo A será p_A que por simplicidad anotaremos como "a". En casos de parentesco, dicha probabilidad será la que corresponda según se deduzca de los familiares disponibles.

7.1.2. *Sudoku genético*

Hay una forma gráfica de ver la trasmisión alélica de forma más intuitiva. Esta forma de verla es como se estudian las leyes de Mendel mediante los cuadrados de Punnett, modificando los mismos los mismos

[e] Adaptado de [6]

[f] Como se ha comentado previamente, el presente trabajo hace uso exclusivamente de marcadores STR autosómicos

[g] con las nuevas técnicas de ultra secuenciación (MPS, Massively Parallel Sequencing), esto está cambiando, ya que por diferencias en la secuencia, se pueden distinguir el alelo del padre y el de la madre, incluso en homozigotos.

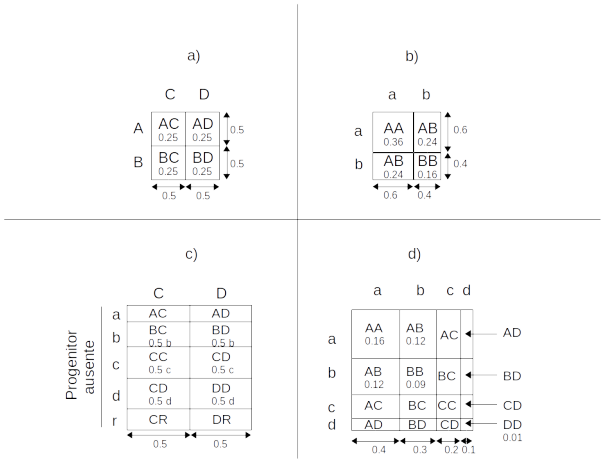


Figura 1: Sudoku genético

considerando cada individuo como un cuadrado de lado con tamaño 1, y por tanto de área 1 (1 x 1), que representa una probabilidad del 100 %. El lado superior representa la probabilidad de trasmisión del padre y el lado izquierdo la de la madre.

Se puede ver en la figura 1a que con dos padres heterozigotos, la probabilidad de trasmisión de cada alelo será la mitad del lado, es decir, 0,5. Por tanto cada hijo tendrá una probabilidad de 0,25 (0,5 x 0,5), es decir 1/4 (una cuarta parte del cuadrado total) o 25 %.

En el caso de homozigotos, en vez de considerar un homozigoto como A (según la notación clásica), es más intuitivo considerarlo como AA, no solo para los humanos, sino también para los sistemas informáticos y algoritmos matemáticos, ya que permite trabajar de forma indistinta con homozigotos y heterozigotos[h]. En este caso, las cuatro casilla son iguales dos a dos, por lo que un 50 % de la descendencia será de un genotipo o de otro[i].

Si ambos padres son homozigotos producirán todos los hijos iguales, heterozigotos si los padres son diferentes y homozigotos si los padres poseen el mismo alelo. Por tanto, la probabilidad es del 100 % (4 x 0.25).

Este mismo concepto, puede servir para calcular la probabilidad de un individuo escogido al azar de la población. En este caso, la probabilidad de trasmisión del padre y de la madre es la de las frecuencias poblacionales para el marcador en cuestión. Por tanto el lado superior y el izquierdo, no se divide en dos sino en tantos fragmentos como alelos, proporcional a la frecuencia. En un marcador con sólo dos alelos, A y B, con frecuencias respectivas a (0,6) y b (0,4) -figura 1b- se obtienen las cuatro posibilidades de hijos con su proporción (frecuencias genotípicas). Los hijos homozigotos AA y BB con frecuencias 0,36 y 0,16 respectivamente (a^2) y los hijos AB dos veces con frecuencia cada uno 0,24, por lo que la probabilidad de tener un hijo heterozigoto es 0,48 ($2ab$). En la diagonal figuran los homozigotos, y en el resto, los heterozigotos duplicados, uno cuando la madre le da el alelo A y el padre el B, y otro cuando la madre le da el B y el padre el A.

En un marcador con cuatro alelos de frecuencias 0,4, 0,3, 0,2 y 0,1 (figura 1d) se puede ver que ocurre lo mismo, la diagonal con los homozigotos, y los heterozigotos duplicados simétricos respecto a dicha diagonal, cada uno proporcional a su frecuencia genotípica.

Si solo se dispone de un progenitor (figura 1c) en un eje figuran los alelos del progenitor presente (equiprobables)

[h]Gran parte de los programas que realizan cálculos de paternidad requieren la entrada de los homozigotos con sus dos alelos. De hecho, la mayoría de los que solo requieren un alelo, internamente lo transforman en dos o usan rutinas específicas para homozigotos diferentes a los heterozigotos

[i]Se puedre considerar que el padre homozigoto en vez de producir dos alelos de 0,5, solo produciría un alelo que ocupara todo el lado del cuadrado, y por tanto se producirían dos rectángulos de 0,5 x 1 (50 %)

y en el otro eje los de un individuo escogido al azar (frecuencias en población). En los casos en que se tengan ancestros de los progenitores ausentes, las frecuencias alélicas serán las condicionadas por dichos ancestros, calculado recurrentemente generación tras generación desde el ancestro hasta el progenitor.

7.1.3. Calculo Bayesiano de la probabilidad de parentesco

En un caso de paternidad se trata de contrastar dos hipótesis alternativas (en el caso de estudio de parentesco, las hipótesis son análogas):

H_1: el presunto padre (PP) es el padre biológico del hijo
H_0: el presunto padre (PP) no es el padre biológico del hijo y por tanto es otro hombre

Estas dos hipótesis tienen una probabilidad (condicionada) en función de la información no genética del caso (I) y del análisis genético (E de evidencia genética). A través del teorema de Bayes se expresa en forma de cociente u "odds":

$$\frac{Pr(H_1|E,I)}{Pr(H_0|E,I)} = \frac{Pr(E|H_1,I)}{Pr(E|H_0,I)} \times \frac{Pr(H_1|I)}{Pr(H_0|I)}$$

Hay tres términos:

$$\text{Odds a posteriori} = \text{LR} \times \text{Odds a priori}$$

El odds a posteriori es lo que intenta determinar el juzgador, es decir, saber en función de la información del caso que tiene, genética (E) y no genética (I), la relación a favor / en contra de la paternidad. Para ello hay que combinar el LR (likelihood ratio o coeficiente de verosimilitud) con los odds a priori. El LR es el valor que puede obtener el laboratorio, que en casos de paternidad se denomina Índice de Paternidad o Índice de Parentesco (IP). Los odds a priori es un valor que debe proporcionar el juzgador, y que no viene a ser otra cosa que la relación a favor en contra de la paternidad que se tiene antes de realizar la prueba genética (con las evidencias no genéticas).

En el caso de las paternidades, como ambas hipótesis son mutuamente excluyentes, deben sumar 1, y por tanto:

$$Pr(H_1|E,I) = 1 - Pr(H_0|E,I)$$
$$Pr(H_1|I) = 1 - Pr(H_0|I)$$

Sustituyendo nos queda

$$Pr(H_1|E,I) = \frac{\text{LR} \times Pr(H_1|I)}{\text{LR} \times Pr(H_1|I) + [1 - Pr(H_1|I)]}$$

Donde Pr(H1|E,I) es la probabilidad de paternidad o W (Wahrscheinlickeit según Essen-Möller[7]) y Pr(H1|I) es la probabilidad de paternidad a priori.

Para los casos de paternidad, donde el LR se suele representar como IP (índice de paternidad o índice de parentesco), tradicionalmente se expresa el IP como el cociente X/Y siendo $X = Pr(E|H_1,I)$ e $Y = Pr(E|H_0,I)$, es decir la probabilidad de encontrar la evidencia genética condicionada a la hipótesis de paternidad (X) y de no paternidad (Y). La evidencia genética (E) son los genotipos del presunto padre, madre e hijo (G_{PP}, G_M, G_H). Por tanto:

$$LR = IP = \frac{X}{Y} = \frac{Pr(E|H_1,I)}{Pr(E|H_0,I)} = \frac{Pr(G_H,G_M,G_{PP}|H_1,I)}{Pr(G_H,G_M,G_{PP}|H_0,I)}$$

Dónde aplicando la tercera ley de la probabilidad

$$IP = \frac{X}{Y} = \frac{Pr(G_H|G_M, G_{PP}, H_1, I)}{Pr(G_H|G_M, G_{PP}, H_0, I)} \times \frac{Pr(G_M, G_{PP}|H_1, I)}{Pr(G_M, G_{PP}|H_0, I)}$$

En el primer término aparece la probabilidad del genotipo del hijo condicionado al genotipo de los padres (conocidos o no) y a las hipótesis. En el segundo término aparecen los genotipos de los padres que son independientes entre si y de las hipótesis. El segundo término, al ser independiente de las hipótesis, serán iguales numerador y denominador y por tanto se anularán (este segundo término valdrá 1).

En los casos de parentesco no se anulará el segundo término, variando en función de los familiares disponibles.

7.1.4. Exclusiones

En el caso de disponer de uno o ambos progenitores, se puede producir lo que se denomina exclusión o inconsistencia para uno o más marcadores por incompatibilidad genética, y por tanto la relación es imposible (sin recurrir a posibles explicaciones alternativas como puede ser la existencia de mutaciones o alelos silentes que pueden ser valorables). En casos sin los progenitores pero con varios familiares, dependiendo de la configuración, puede haber o no la posibilidad de exclusión.

En el caso de identificaciones, en las que solo suele haber un familiar y una víctima (relaciones familiares binarias), sólo habrá posibilidad de exclusión si son un progenitor y un hijo. En el resto de relaciones familiares no habrá exclusiones, y por tanto la compatibilidad se basará únicamente en la valoración estadística. Esta valoración es difícil ya que hay casos de no parientes con un IP>1 y casos de parientes con IP<1, como se verá a lo largo del presente trabajo. Esta circunstancia obliga a usar un gran número de marcadores para separar ambas poblaciones (parientes/no parientes) y un margen de seguridad adecuado. Según los datos que se desprenden de algunos análisis[[8,9]] se puede considerar, para el caso de hermanos, como una indicación fuerte de parentesco un valor de IP (LR) por encima de 10000, y por contra considerar indicativo de no parentesco por debajo de 1/10000[j]. En relaciones más distantes, estos márgenes no son tan claros, por lo que la identificación no tiene un grado de confianza suficiente.

7.1.5. Identificación desaparecidos y catástrofes

En la identificación de desaparecidos hay dos grandes factores limitantes: por una parte la calidad de las muestras a estudiar y por otra disponer de parientes cercanos y en número suficiente para que la identificación sea concluyente.

Respecto a la calidad de las muestras, aunque queda fuera del alcance del presente trabajo, simplemente indicar que en muchas ocasiones, por la antigüedad de los restos encontrados y por su conservación (restos sometidos a la intemperie y degradación microbiana, irradiación solar, condiciones salinas en restos sumergidos, ...), no siempre es posible obtener todos los marcadores necesarios para un estudio completo.

Respecto al número de familiares, como se verá más adelante, es fundamental disponer de familiares lo más cercanos posibles, preferentemente ascendientes o descendientes directos en primer grado, y en caso de no estar disponibles estos, contar al menos con dos o tres familiares (dos abuelos de la misma rama, dos o tres hermanos completos, tres tíos). Como alternativa a los familiares, se puede recurrir a muestras *antemortem* (muestras dejadas en vida) como pueden ser biopsias en hospitales o restos de material celular en diversos efectos personales (cepillo de dientes, maquinilla de afeitar, cartas cerradas con la lengua,...). Estas muestras son muy efectivas, ya que se realiza una identificación directa, sin embargo, hay que asegurarse de la autenticidad de las mismas.

En algunos casos, los desaparecidos, proceden de otros países, por lo que la conexión con los familiares y las circunstancias de la desaparición no es inmediata. Por suerte, hoy en día se dispone en España de una base

[j]En estos casos, siempre que se pueda, deben completarse los análisis con marcadores de linaje (ADN mt y cromosoma Y)

de datos de carácter humanitario para la identificación de desaparecidos (ADNID), que a su vez intercambia información con la mayoría de los estados miembros de la Unión Europea en virtud del tratado de Prüm o con otros países a través de Interpol. Desgraciadamente, hay muchos cadáveres que quedan sin identificar por falta de material de referencia (especialmente los aparecidos tras el naufragio de pateras).

En los sucesos de víctimas múltiples (catástrofes), la situación es más compleja. En primer lugar, la dificultad en la confirmación del parentesco por medio de la genética como se ha comentado anteriormente, aumentado por el hecho de que se trata de una identificación de un gran número de víctimas frente a un gran número de familiares. Dependiendo de que el suceso sea abierto (en el que se desconocen el número e identidad supuesta los implicados, por ejemplo un atentado terrorista en un lugar público o un terremoto) o cerrado (con un número de víctimas cierto y sus identidades conocidas, por ejemplo un accidente aéreo), la identificación es más o menos compleja. Si se añade el hecho de que los cadáveres pueden estar fragmentados y alterados por la violencia del suceso, el escenario es de extremada complejidad. En estos casos, se suele disponer de la mayoría de los familiares o de muestras antemortem, aunque no siempre "cuadran" el número de víctimas reclamadas con las existentes.

No se puede permitir en estos casos la asignación errónea de una víctima a un familiar. En este sentido, el protocolo nacional de sucesos con víctimas múltiples, exige la identificación por al menos dos técnicas confirmativas diferentes (dactiloscopia, fórmula dentaria, ADN). No obstante, tampoco se puede dilatar en el tiempo la resolución de las identificaciones, por lo que la ayuda de sistemas expertos o sistemas de inteligencia artificial puede ser de gran ayuda para encauzar las investigaciones y ahorrar gran parte del trabajo. Es este punto el principal objetivo del presente trabajo, conseguir un algoritmo de *machine learning* o *deep learning* entrenado previamente, que consiga la agrupación de los restos con sus familiares en un tiempo breve.

7.2. Breve descripción de los fundamentos de los algoritmos empleados[k]

7.2.1. Algoritmo de clasificación k-NN

El algoritmo k-NN es uno de los más simples de *Machine Learning*, aunque se usa ampliamente.
Se basa en seleccionar los k vecinos más similares a un valor para clasificarlo. Si representáramos los elementos en un hipotético plano, se mediría la distancia entre los elementos, siendo la línea que divide las categorías nuestro valor de ajuste. Puede que haya elementos mal clasificados a uno y otro lado de la línea, que serían los falsos positivos y los falsos negativos.
Para cuantificar la distancia, usa la distancia euclídea. Es muy importante adecuar los datos para que el algoritmo sea efectivo.
El principal inconveniente, está en determinar cual es el valor de k más adecuado. si se elige un valor de k muy elevado se produce un sobre ajuste, y es menos escalable para futuras predicciones. Se suele usar la raíz cuadrada del número de elementos usados para el entrenamiento, aunque no siempre es el mejor.

Sus fortalezas y debilidades son las siguientes:

Fortalezas	Debilidades
Simple y efectivo	No produce un modelo, limitando la posibilidad de entender como se asocian las clases a las características
No hace suposiciones sobre la distribución de los datos	Requiere la selección de un valor de k adecuado
Fase de entrenamiento rápida	La fase de clasificación es lenta
	Requiere un procesamiento de datos para manejar datos ausentes y las características nominales

[k] Adaptado de [10]

7.2.2. Algoritmo de clasificación Naive-Bayes

El algoritmo Naive Bayes se basa en la aplicación del teorema de Bayes, de forma que a partir de un conjunto de entrenamiento determina las probabilidades a priori de todas las características observadas. A partir de estas, elabora la predicción calculando la probabilidad a posteriori conjunta de todas las características para cada categoría, y así poder asignarle una clasificación.
Sus fortalezas y debilidades son las siguientes:

Fortalezas	Debilidades
Simple, rápido y muy efectivo	No es la mejor para conjuntos de datos con muchas características numéricas
Trabaja bien con datos parciales y corruptos	
Requiere relativamente pocos ejemplos de entrenamiento aunque funciona bien también con grandes conjuntos de entrenamiento	Se basa en la suposición de que las características son independientes y equiparables, pero no siempre es así en las aplicaciones reales
Fácil obtención de la probabilidad estimada para una predicción	Las probabilidades estimadas son menos fiables que las clases predichas

7.2.3. Algoritmo Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (ANN, del inglés **A**rtificial **N**eural **N**etwork) son modelos que relacionan unas señales de entrada con unas señales de salida simulando lo que sería una red de neuronas. Son modelos de caja negra (*black box*) ya que es muy difícil o imposible de interpretar el mecanismo exacto de la transformación de las señales en el interior de la red.
Consisten en varios niveles de nodos, interconectados entre ellos, es decir cada nivel se interconecta con nodos del siguiente nivel, desde el nivel de entrada (*input*) hasta el nivel de salida (*output*), pasando por uno o varios niveles de nodos ocultos (*hidden*) con un número variable de nodos en cada nivel. Adicionalmente, puede haber un nodo por nivel que module la señal del mismo (*bias*) permitiendo un ajuste de la señal. La forma de relacionarse todos estos nodos es lo que conocemos como **topología de la red**, que junto con la función de activación y el algoritmo de entrenamiento, definen la red.
La **función de activación** es el mecanismo que usa cada nodo para trasmitir la señal que le llega. La señal de entrada, es procesada por la función y trasmitida al siguiente nodo, modulada por un peso (*weight*) específico de cada conexión. Precisamente el **algoritmo de entrenamiento** lo que hace es establecer los pesos de forma que el error cometido sea el mínimo. El algoritmo más utilizado, conocido como *backpropagation*, recorre la red hacia adelante hasta que obtiene los valores de salida, y después compara dichos valores con los valores reales y recorre la red en sentido inverso ajustando los valores de los pesos para disminuir el error. Esencialmente se utiliza la técnica de gradiente descendente que utiliza da derivación de la función de activación para determinar en que dirección debe modificar el peso para disminuir el error en una cantidad que se conoce como *indice de aprendizaje*.
La información, en las redes más típicas fluye en dirección de entrada a salida, aunque no es obligatorio que sea así (por ejemplo en redes recurrentes). Igualmente, los nodos suelen conectarse con todos los nodos del siguiente nivel, aunque tampoco es obligatorio.

Sus fortalezas y debilidades son las siguientes:

Fortalezas	Debilidades
* Se pueden abordar problemas de clasificación o de predicción numérica	* Extremadamente exigente computacionalmente y lento de entrenar, especialmente si la topología de la red es compleja
* Capaz de modelar patrones más complejos que prácticamante cualquier algoritmo	* Muy propenso a sobreajustar los datos de entrenamiento
* Hace muy pocas suposiciones sobre las relaciones subyacentes en los datos	* Produce un modelo complejo de caja negra, difícil o prácticamente imposible de interpretar

7.2.4. Algoritmo Support Vector Machine (SVM)

El algoritmo Support Vector Machines (SVM) es un método que podíamos considerar como una combinación de clasificador y regresión linear. Es uno de los modelos de elección en *computer vision* y en identificación del lenguaje. Es también muy efectivo en clasificación de cáncer o enfermedades genéticas con datos de expresión génica.
Se fundamenta en la creación de un hiperplano n-dimensional que separaría las categorías en dos regiones diferenciadas (conocido como hiperplano de margen máximo). Los puntos más cercanos al hiperplano, se conocen como *support vectors*, dando nombre al algoritmo. El objetivo del algoritmo es encontrar la máxima separación entre ellos, de forma homogénea.
Si los datos no son linearmente separables, habrá elementos que no quedarán en el lado del hiperplano correcto. Estos puntos forman en conjunto lo que se conoce como coste (C). La optimización del modelo tiende a disminuir al máximo el coste. Para resolver los espacios no lineales, se usa un ajuste mediante *kernel* que transforma los datos en una dimensión superior, dónde sí sean linearmente separables. Algunos de los *kernel* más conocidos son el linear, el polinomial, el sigmoideo y el gausiano. No hay un criterio objetivo para la elección del modelo y sus parámetros, realizándose mediante prueba/error.

Sus fortalezas y debilidades son las siguientes:

Fortalezas	**Debilidades**
* Se puede usar para clasificación o predicciones numéricas	* Encontrar el mejor modelo requiere comprobar varias combinaciones de kernels y parámetros del modelo
* No se afecta demasiado por datos ruidosos ni es proclive al sobreajuste	* Produce un modelo complejo de caja negra, difícil o prácticamente imposible de interpretar
* Puede ser más fácil de usar que las redes neuronales, especialmente por la existencia de varios algoritmos bien documentados	* Puede ser lento de entrenar, especialmente si los datos de entrenamiento tienen un gran número de características o de entradas
* Está ganando popularidad por su alta precisión y por haber ganado varias competiciones de alto nivel de minado de datos	

7.2.5. Algoritmo Árboles de clasificación

El algoritmo Árboles de Clasificación se basa en elegir reglas que vayan separando sucesivamente los elementos en categorías. Sigue una estructura en forma de árbol, de ahí su nombre. Comienza por un primer nodo que divide los datos en dos (o más) grupos (ramas) mediante una regla de decisión. A su vez, cada rama puede dividirse sucesivamente mediante más reglas hasta alcanzar lo que se conoce como nodos hoja o nodos terminales.
Las divisiones se realizan hasta alcanzar un nivel de homogeneidad establecido (o cualquier otro criterio según el algoritmo). En el caso del algoritmo C5.0 que es de los más usado se utiliza el concepto de entropía que nos indica como de homogéneas son las clases obtenidas en conjunto.
En conjuntos de datos complejos, las divisiones pueden crecer hasta que tengan una clasificación perfecta o

se acaben las características disponibles. Sin embargo, esto tiene el riesgo de sobreajustar el modelo, por lo que se suele limitar el número de ramas en lo que se conoce como poda (*prune*). Puede definirse un tamaño máximo de crecimiento (*pre-pruning*), o crear un árbol grande y después recortarlo a un nivel adecuado o eliminar ramas poco informativas (*post-pruning*).

Sus fortalezas y debilidades son las siguientes:

Fortalezas	Debilidades
* Se puede usar en la mayoría de los problemas	* Los modelos de decisión suelen tener un sesgo hacia carácterísticas con un grán número de niveles
* El proceso de aprendizaje muy automático, permitiendo características numéricas o cualitativas, incluso datos incompletos	* Es fácil sobreajustar o infraajustar el modelo
* Elimina características poco importantes	* Puede tener problemas con el modelado de algunas relaciones por haber divisiones paralelas al eje
* Puede usarse en conjuntos de datos pequeños o grandes	* Pequeños cambios en el entrenamiento pueden producir grandes cambios en la lógica de decisión
* Produce un modelo que puede entenderse sin un amplio conocimiento matemático	* Los árboles grandes suelen ser difíciles de interpretar, y su decisiones pueden parecer antiintuitivas
* Mas eficiente que otros modelos más complejos	

7.2.6. Algoritmo Random Forest

7.2.6.1. Funcionamiento

Este algoritmo se basa en hacer una selección de parte de los datos y parte de las características (al azar, **random**) para crear un árbol de decisiones. Este proceso lo repite de forma independiente para un número prefijado de "árboles", creando un "bosque" (**forest**). La decisión se toma por votación del conjunto de árboles.
Por regla general, se usa la raíz cuadrada del total de características. Tiene la ventaja de que al simplificar los *imputs*, puede trabajar bien con grandes conjuntos de datos. Por contra, no es tan efectivo en conjuntos pequeños ni es fácilmente interpretable.

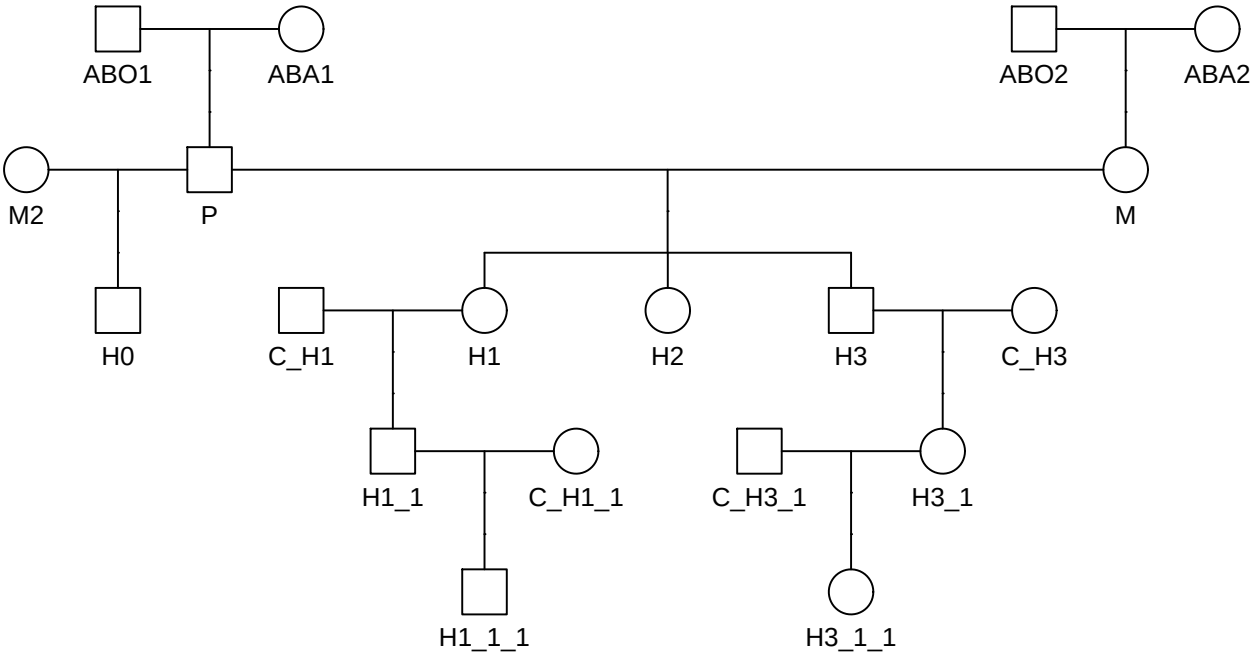
Sus fortalezas y debilidades son las siguientes:

Fortalezas	Debilidades
* Se puede usar en la mayoría de los problemas	* Los modelos no son fácilmente interpretables
* Permite características numéricas o cualitativas, incluso datos incompletos o *ruidosos*	* Necesita dedicar tiempo al ajuste del modelo a los datos
* Selecciona sólo las características más importantes	
* Puede usarse en conjuntos de datos extremadamente grandes o con un grán número de características	

8. Desarrollo del trabajo

8.1. Obtención de datos

El éxito de las predicciones de un algoritmo de *machine learning* viene determinado por un entrenamiento adecuado. Si bien la configuración de los parámetros de entrenamiento son cruciales, nada se puede hacer si no se dispone de suficientes datos representativos del problema a resolver. En el caso de grandes catástrofes, es obvio que no hay suficiente casuística, por lo que desde un primer momento se planificó la creación de grandes conjuntos de datos "*in silico*". Para ello, se diseñó una gran estructura familiar con cinco generaciones, procedentes de los cuatro fundadores (abuelos), que junto con algunos cónyuges, conformaron una familia final con 19 individuos. El siguiente árbol genético muestra la estructura final:




```
## Did not plot the following people: Prandom
```

Esta estructura familiar permite estudiar relaciones tanto próximas como lejanas (hasta cinco generaciones), e incluso varios tipos de relaciones colaterales[1]. Para poder realizar un contraste de hipótesis, tal y como se realiza de forma clásica, se añadió un individuo no relacionado con el resto de la familia (Prandom), que no se representa en el árbol al no tener relación de parentesco con ninguno de los anteriores. Esta muestra se incorporó a los entrenamientos de los diversos algoritmos como ejemplo de valor negativo en contraste con los individuos de relación familiar conocida.

Una vez definida la estructura familiar, es necesario generar los individuos correspondientes a la misma. Para ello, se definieron una serie de funciones, que siguiendo las leyes de transmisión alélica mendeliana generaran los individuos recursivamente, generación tras generación.

```
alelo_aleatorio <- function(alelos) {
    # alelos es lista de frecuencias con nombres
    sample(names(alelos), 1, prob = alelos)
}

alelo_mendel <- function(alelos) {
```

[1]Aunque queda fuera del trabajo del presente trabajo, este diseño familiar y los datos generados, pueden servir para el estudio de relaciones complejas con múltiples parientes

```
    sample(alelos, 1)
}
# alelos es vector de dos alelos del progenitor

genotipo_aleatorio <- function(alelos) {
    # alelos es lista de frecuencias con nombres
    sample(names(alelos), 2, TRUE, alelos)
}
```

Se puede ver que hay una función (`alelo_aleatorio()`) para la generación de alelos cuando no se dispone de uno de los progenitores, seleccionando aleatoriamente los alelos presentes en la población, atendiendo a su abundancia relativa.
En caso de tener un progenitor, se genera el alelo, seleccionando al azar entre los alelos del progenitor, que se pasan como vector ala función (función `alelo_mendel()`).
En caso de que no se disponga de ninguno de los progenitores, directamente se genera el genotipo (ambos alelos) a partir de la lista de frecuencias poblacionales que se pasa a la función `genotipo_aleatorio()`. La lista debe incluir los nombres de los alelos.

Uno de los requisitos para poder generar los individuos, es disponer de unas frecuencias poblacionales, de forma que los resultados sean aplicables a la misma. Para el presente trabajo se han utilizado la frecuencias utilizadas actualmente por los principales laboratorios forenses de España [11,12]. Tras la importación de las frecuencias desde el archivo excel disponible en el material suplementario, se corrigen las mismas para que sumen 1, ya que por el redondeo realizado para la publicación, en algunos marcadores faltan o sobran diezmilésimas. Se a optado por modificar la frecuencia mayoritaria, ya que es dónde menos impacto tiene. Posteriormente se almacenan las frecuencias en un objeto `FamiliasLocus`.

En este punto, es relevante indicar que se ha usado el paquete `Familias` versión 2.4 [13] que es la versión en R del programa del mismo nombre que tiene gran implantación en los laboratorios forenses para el calculo de paternidad y parentesco [14,15]. Entre otras razones, a parte de ser un referente en el campo, la razón para elegir dicho paquete es que contiene objetos para almacenar el pedigrí, las frecuencias, y poder hacer el cálculo de índice de parentesco (IP, LR) como referencia y contraste.

Con todos estos elementos, se puede generar el pedigrí completo para una familia. La función que lo genera, recorre el árbol en busca de individuos sin ancestros, creando su genotipo (para todos los marcadores de la tabla de frecuencias poblacionales). En una segunda pasada, busca individuos con ambos padres, a los que les asigna un genotipo siguiendo la transmisión mendeliana. La función, sigue iterando sobre la familia en busca de nuevos individuos que tengan ambos padres hasta que todos los componentes del árbol están completos. En el anexo 3 puede verse el código correspondiente a esta fase y un ejemplo de verificación de que el proceso funciona correctamente. Todos los objetos y funciones generados se almacenan en ficheros para recuperarlos en fases posteriores.

Una vez comprobado el funcionamiento correcto de las funciones y rutinas, se procede a la creación de los archivos de entrenamiento (train) y prueba (test). Para ello se diseña una rutina que genera los veinte individuos del árbol para 40000 familias de entrenamiento (train) y para 40000 familias de prueba (test). En total se generan 20 individuos para 80000 familias con 21 marcadores por individuo, con dos alelos por marcador.

Como control de calidad del dato, se comprueba que la distribución de frecuencias en los datos simulados se corresponden con los teóricos de partida, y que se cumple la ley de Hardy-Weinberg, es decir, que la proporción se mantiene entre generaciones. Los resultados pueden verse en el anexo 4.

8.2. Relaciones familiares a estudiar

En un primer momento, se planificó el estudio de tres grupos de relaciones familiares:

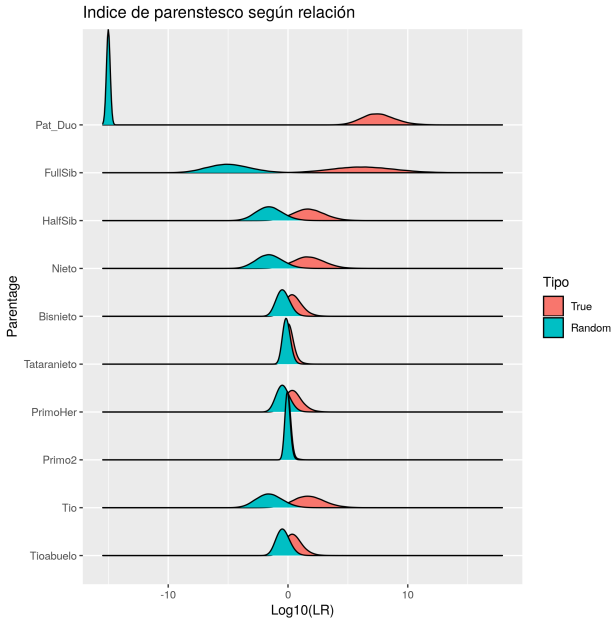


Figura 2: Distribución de LR (IP) de las diversas relaciones familiares

- Relaciones familiares básicas: relaciones sencillas empleadas en los casos de paternidad civil. Incluye paternidad con padre-madre-hijo (Pat_Trio), paternidad con un solo progenitor padre-hijo (Pat_Duo) y paternidad a través de abuelos paternos sin madre disponible (Abuelos). Las relaciones de este grupo se suelen resolver por la existencia de exclusiones si se trata de un individuo falsamente atribuido.

- Relaciones de varios hermanos: relaciones defectivas (sin padre disponible) a través de varios hermanos en diversas combinaciones. Incluye estudio de parentesco con dos hermanos reconocidos (FullSib_2), con dos medio hermanos reconocidos (HalfSib_2), con tres medio hermanos (HalfSib_3), con dos hermanos y su madre (FullSib_2_M) y con tres medio hermanos y su madre (HalfSib_3_M). Todas estas relaciones son complejas. En ocasiones hay exclusiones, pero hay combinaciones que no excluyen a un individuo falsamente atribuido.

- Relaciones binarias: relaciones entre únicamente dos individuos en diverso grado de parentesco. Incluye las relaciones entre dos hermanos de padre y madre (FullSib), dos medio hermanos (HalfSib), abuelo y nieto (Nieto), bisabuelo y bisnieto (Bisnieto), tatarabuelo y tataranieto (Tataranieto), dos primos hermanos (PrimoHer), dos primos segundos (Primo2), tío y sobrino (Tio) y tioabuelo con sobrino-nieto (Tioabuelo). Estas relaciones no presentan exclusión en ningún caso, por lo que no se puede confirmar el no parentesco más allá de una valoración estadística de la certeza. Algunas relaciones son muy lejanas, por lo que la información genética aporta poco valor. En la figura 2 puede verse la distribución del IP (LR). Se aprecia como la separación entre las curvas de relaciones ciertas (True) y falsamente atribuida (Random) se acercan y solapan las relaciones más lejanas.

Tras la codificación de estas relaciones y unas primeras pruebas, se decidió trabajar únicamente con las relaciones del último grupo (relaciones binarias) y la relación Pat_Duo (que también es sólo de dos individuos), en total 10 relaciones diferentes (véase el anexo 2 para visualizar los individuos utilizados). Alguna de las causas son técnicas, como el hecho de que al tener más de dos individuos los dos primeros grupos, la codificación tenía que ser diferente al tercero y el entrenamiento de los algoritmos no sería adecuado. Otras causas son genéticas, como el hecho de que al tener relaciones por ambas líneas ascendentes en la mayoría de los casos, los resultados no serían directamente comparables con el resto. El hecho de que presenten exclusiones también genera problemas a la hora de comparar resultados. Por último hay unas causas prácticas y de enfoque, ya que el objetivo es la generación de un clasificador para utilizar en grandes catástrofes, en las que no suele ser

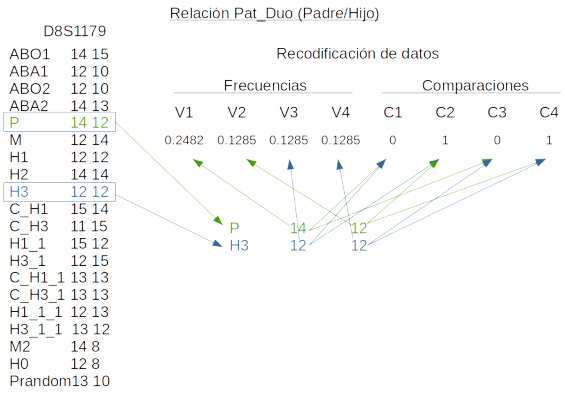


Figura 3: Recodificación de variables

habitual, al menos en un primer momento, analizar relaciones complejas. En cualquier caso, las relaciones de los primeros grupos se pueden descomponer en relaciones binarias, por lo que incluyendo a todos los individuos en los conjuntos de clasificación, estarían también cubiertos. Además es una práctica habitual analizar a los diversos familiares por separado, ya que la realidad aparente, no siempre concuerda con la realidad biológica.
Cabe mencionar que la incorporación de la relación padre-hijo presenta ciertos problemas, ya que es la única de las seleccionadas que presenta exclusiones. No obstante, es la relación de primera elección a la hora de solicitar familiares para la identificación de víctimas en casos de catástrofes, y por tanto tiene que ser incluida.

8.3. Recodificación de datos

Los datos generados, consisten en un listado de alelos para cada individuo para los 21 marcadores analizados, en total 42 alelos. Estos alelos se recogen en variables categóricas, con un número de categorías diferente para cada marcador, oscilando entre 6 y 40. Los valores de cada categoría en los marcadores son diferentes y no guardan relación entre ellos y su incidencia en población varía para cada alelo y cada marcador (ver anexo 4). Los marcadores seleccionados, son: D8S1179, D21S11, D7S820, CSF1PO, D3S1358, TH01, D13S317, D16S539, D2S1338, D19S433, VWA, TPOX, D18S51, D5S818, FGA, D10S1248, D1S1656, D22S1045, D2S441, D12S391, SE33.
CSF1PO y D5S818 están localizados en el cromosoma 5. vWA, D2S1338 y D2S441 están localizados en el cromosoma 2. El resto de marcadores están localizados en cromosomas diferentes, por lo que su transmisión es independiente, y por tanto pueden ser considerados como sucesos independientes. Incluso los marcadores localizados en el mismo cromosoma están lo suficientemente separados como para que a efectos prácticos se consideren igualmente sucesos independientes.
Algunos algoritmos no admiten variables categóricas, por lo que habría que codificar de forma binaria cada marcador en tantas variables booleanas como alelos tenga. Por otra parte, los conjuntos de datos cuentan con información de gran número de individuos que no se usarían en las comparaciones. Se decidió el recodificar los datos para que los mismos tengan más sentido biológico y así facilitar la labor de los algoritmos, a la vez que se simplificaban, lo cual agilizaría su manejo. Para ello, se generaron conjuntos de datos para cada una de las relaciones familiares, seleccionando los dos individuos implicados en la relación. Por cada marcador, tenemos cuatro alelos que convertiremos en 8. En la figura 3 se puede ver el proceso de transformación.

Por una parte, transformaremos los cuatro alelos en su correspondientes frecuencias, lo que permitirá que los algoritmos valoren de forma diferente aquellos alelos más frecuentes en población.
Por otra parte, se compara cada alelo de un individuo con los dos alelos del otro individuo, indicando como 1

si coinciden y como 0 si no coinciden. Con este añadido, los algoritmos valorarán de forma más efectiva las combinaciones únicamente dentro del mismo marcador.

Se realiza la recodificación para las 40000 familias entre los dos individuos implicados y también entre el individuo considerado como víctima y el individuo no relacionado (Prandom), simulando una atribución falsa, obteniendo así dos categorías (True y Random) que indican si la relación es cierta o no. Esta categoría se incorpora como una variable más.

En total, se generan dos conjuntos de datos (train y test) con 80000 entradas, para cada una de las relaciones familiares (10).

8.4. Entrenamiento básico

8.4.1. Relación única

Como primer paso, se realizó el entrenamiento y evaluación de la relación (**Pat_Duo**), la relación con mayor poder de discriminación. Como se ha visto en la estructura de datos recodificados, en este caso, se entrena únicamente con relaciones ciertas padre/hijo e individuos no relacionados (uno de cada por familia), de forma que el clasificador categorice el resultado en emparentado (*True* = padre/hijo) o no relacionados (*Random*). Se utilizaron para el entrenamiento 5000 familias (10000 parejas de individuos, la mitad *True* y la otra mitad *Random*) con una red neuronal simple de una sola capa con 21 nodos (el mismo número de nodos que de marcadores) con ayuda de la función `nnet`. Para la evaluación (*test*) se usaron también 5000 familias (5000 parejas *True* + 5000 parejas *Random*), de otro set de datos diferente (Testset).

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction True Random
##     True   5000      1
##     Random    0   4999
##
##                Accuracy : 1
##                  95% CI : (0.999, 1)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.5
##          Detection Rate : 0.5
##    Detection Prevalence : 0.5
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : True
##
```

Puede verse que solo presenta un fallo en las 10000 comparaciones.

Para comprobar si con otras relaciones menos discriminativas se comporta igualmente, se realiza el mismo entrenamiento (5000+5000) para la relación de hermanos completos (**FullSib**).

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction True Random
##     True   4979     18
##     Random   21   4982
##
##                Accuracy : 0.996
##                  95% CI : (0.995, 0.997)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.992
##  Mcnemar's Test P-Value : 0.749
##
##             Sensitivity : 0.996
##             Specificity : 0.996
##          Pos Pred Value : 0.996
##          Neg Pred Value : 0.996
##              Prevalence : 0.500
##          Detection Rate : 0.498
##    Detection Prevalence : 0.500
##       Balanced Accuracy : 0.996
##
##        'Positive' Class : True
##
```

Los resultados son igualmente excelentes con el conjunto de test (5000 + 5000), con una precisión en este caso del 99,6 % y sólo 39 fallos de 10000 comparaciones.

8.4.2. Comparación de algoritmos en predicción categorica de relación única

Una vez comprobado que, al menos con una red neuronal simple, se obtienen unos resultados aceptables, se pasa a verificar el resto de algoritmos de *machine learning* seleccionados para el presente trabajo knn, naive_bayes, nnet, svmLinear, svmRadial, C5.0, rf. En este caso, se usa el paquete `caret` para el entrenamiento. También se realiza el entrenamiento con una red neuronal con `keras/tensorflow`.
Al igual que en el punto anterior, el entrenamiento se realiza para la relación Pat_Duo y FullSib, en este caso con 10000 familias, es decir, 10000 parejas de relaciones ciertas (*True*) y 10000 parejas no relacionadas (*Random*). Para `keras` se usaron la mitad (5000+5000).

Para la evaluación se usaron igualmente 10000 familias (10000 parejas *True* + 10000 parejas *Random*) En el caso de la relación Pat_Duo los resultados son tan buenos como los del apartado anterior. Únicamente el algoritmo Naive Bayes comete 48/10000 errores en al asignación de la categoría cierta (True). El resto clasifican correctamente esta categoría. Para los individuos no relacionados, cuatro algoritmos clasifican el 100 % y el resto tienen entre 2 y 4 fallos de 10000.

	True	Random	Error(%)
Total de comparaciones	10000	10000	NA
knn	10000	10000	0.000
naive_bayes	9952	10000	0.240
nnet	10000	9996	0.020
svmLinear	10000	9998	0.010
svmRadial	10000	9996	0.020
C5.0	10000	9997	0.015
rf	10000	10000	0.000

Para FullSib, los resultados no son tan buenos, aunque tienen también unas tasas de error por debajo del 1 % salvo el algoritmo *k-nearest network* que presenta un error del 1,66 %.

	True	Random	Error(%)
Total de comparaciones	10000	10000	NA
knn	9670	9998	1.660
naive_bayes	9987	9856	0.785
nnet	9967	9965	0.340
svmLinear	9957	9955	0.440
svmRadial	9971	9989	0.200
C5.0	9960	9963	0.385
rf	9970	9984	0.230

El entrenamiento con `keras` no se puede hacer con el paquete `caret`, sino que requiere una instalación específica que instala un entorno virtual de Python. Para procesadores antiguos y ciertas configuraciones, no se puede instalar la ultima versión de `Tensorflow` porque es incompatible con la versión de `keras`. En el anexo 5 se pueden ver los pasos a ejecutar para solucionar el problema.

Con un entrenamiento básico y una red simple de una capa con 21 nodos ya se consiguen resultados muy buenos, similares al resto de algoritmos. La ventaja es que el entrenamiento requiere aproximadamente un minuto y la predicción pocos segundos, en contraste con el resto que son mucho más lentos.

La configuración utilizada para la prueba es:

```
modelo <- keras_model_sequential()
modelo %>% layer_dense(units = 21, activation = "relu", input_shape = c(168)) %>%
    layer_dropout(rate = 0.4) %>% layer_dense(units = 2, activation = "softmax")

modelo %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(),
    metrics = c("categorical_accuracy"))

set.seed(112358)
entreno <- modelo %>% fit(trainset, cat_train, epochs = 30, batch_size = 128,
    validation_split = 0.2)
plot(entreno)
```

Se puede ver la gráfica de entrenamiento y la matriz de confusión para la relación Pat_Duo (figura 4(izquierda)):

```
##           Reference
## Prediction True Random
##     True   5000      3
##     Random    0   4997
```

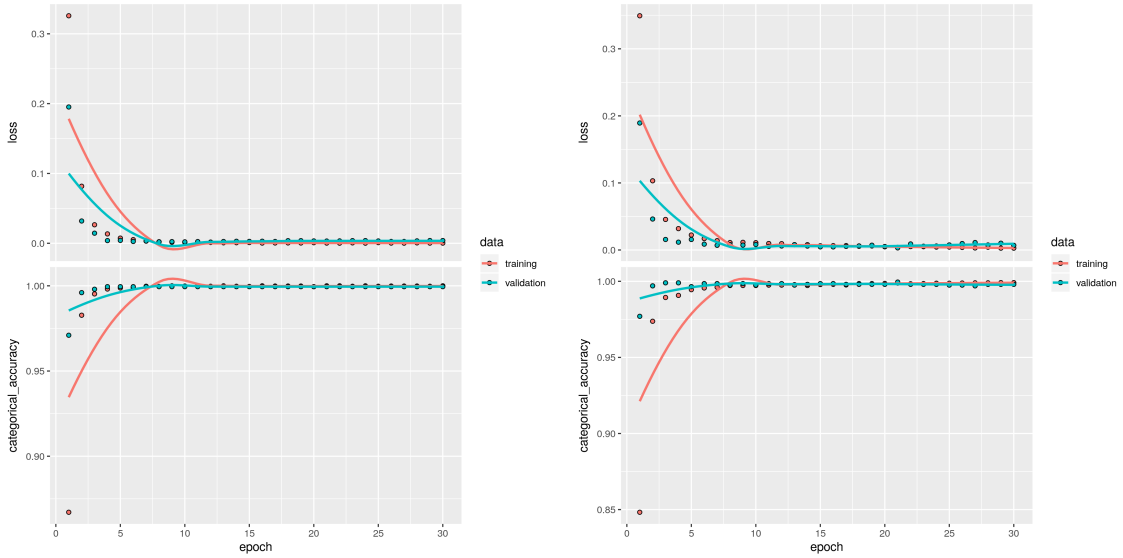


Figura 4: Entrenamiento inicial keras con las relaciones familiares PatDuo (izquierda) y FullSib(derecha)

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper
##         0.9997         0.9994         0.9991         0.9999
##   AccuracyNull AccuracyPValue  McnemarPValue
##         0.5000         0.0000         0.2482
```

Los resultados son muy buenos con solo 3 fallos de 10000 comparaciones. Para FullSib (figura 4(derecha)) los resultados son algo peores, pero aun así es un excelente clasificador:

```
##           Reference
## Prediction True Random
##     True   4984     13
##     Random   16   4987
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper
##         0.9971         0.9942         0.9958         0.9981
##   AccuracyNull AccuracyPValue  McnemarPValue
##         0.5000         0.0000         0.7103
```

8.4.2.1. Comparación de algoritmos en predicción numérica de relación única

Una vez comprobada la efectividad del modelo `keras` como clasificador, se decidió valorar la efectividad predictiva del modelo a nivel numérico. Para ello, se realizó el cálculo del LR (IP) con la función `Familias::FamiliasPosterior()`. Dicha función nos devuelve el valor del índice de parentesco (IP) para el contraste bayesiano entre las hipótesis de parentesco y no parentesco:

H_1: el familiar está relacionado con la víctima con el parentesco declarado.
H_0: el familiar **no** está relacionado con la víctima con el parentesco declarado.

El valor de IP se calcula para todas las entradas de los conjuntos de test y entrenamiento, y se pasa en forma de logaritmo decimal al algoritmo en vez de las categorías en el entrenamiento. Por cada relación familiar se entrena con las 80000 comparaciones del conjunto de entrenamiento (40000 *True* + 40000 *Random*). La predicción se realiza con las 80000 comparaciones del conjunto de test, que será equivalente al logaritmo del IP del conjunto de test. Para comprobar la efectividad, se realizan gráficos de densidad para cada relación familiar. En la figura 5 se pueden ver dichos gráficos. En rojo aparecen los resultados de IP calculados con el paquete `Familias` para el conjunto de test, que denominaremos "bayesiano". En azul aparecen los resultados

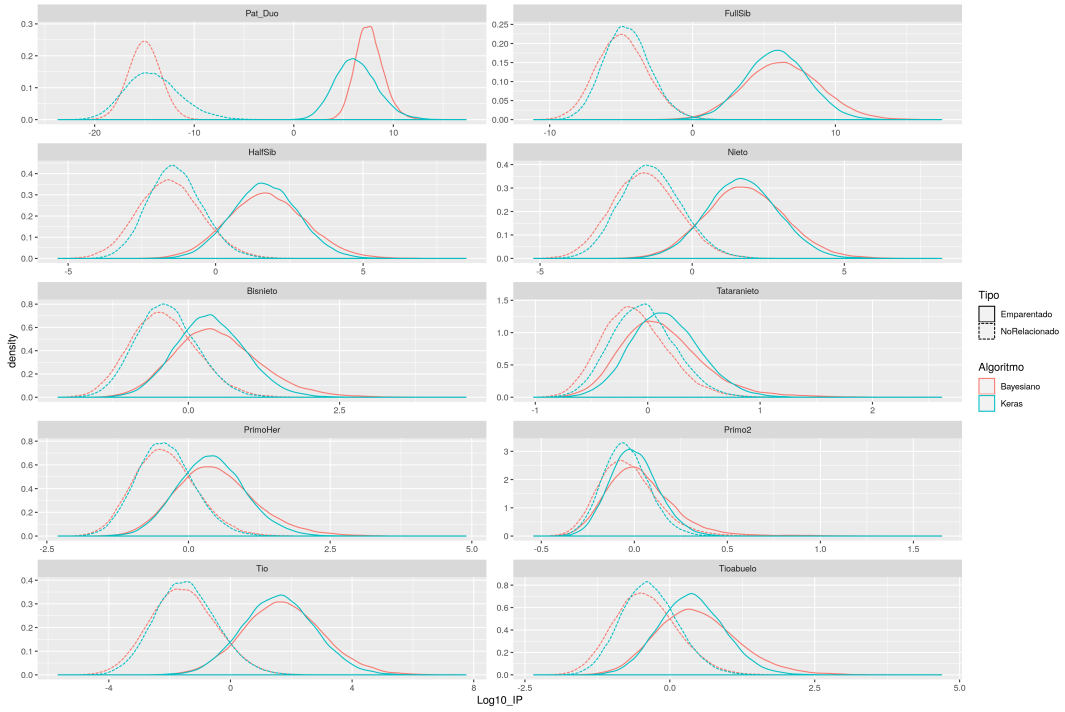


Figura 5: Comparación de la distribución de densidad entre el valor teórico (bayesiano) y la predicción (keras).

de la predicción con el paquete `keras`. Para cada conjunto de test hay dos grupos, uno que corresponde con los individuos que realmente tienen la relación investigada (etiquetados como "Emparentado", con linea continua) y otro grupo de individuos seleccionados al azar, y por tanto sin relación genética (etiquetados como "NoRelacionado", con línea discontinua). Se observa que la distribución de densidad es similar entre el valor teórico (bayesiano) y la predicción (keras) para todas las relaciones.

Para cada relación se observa una separación mayor entre los individuos del grupo emparentado y los no relacionados, concordante con la distancia genética en el árbol (nótese que la escala no es la misma para todas las relaciones).En la figura 6 se recogen los mismos resultados en la misma escala del eje de ordenadas.

La separación entre los dos grupos indica la dificultad en clasificar a los individuos en uno o en otro, lo que se refleja en la efectividad del clasificador dependiendo de la relación familiar. Hay relaciones familiares como bisnieto, tataranieto, primos segundos o tioabuelo, en los que prácticamente se solapan ambas curvas, y por tanto, la capacidad de separación es muy limitada. Con métodos bayesianos, el valor obtenido no es concluyente para estas relaciones.

8.4.2.2. Selección de algoritmos

Inicialmente, al final de esta fase estaba previsto una optimización de todos los algoritmos y la selección de los más efectivos. No obstante, ya que el comportamiento es similar entre todos los algoritmos, y teniendo en cuenta que al tratar con conjuntos de datos con múltiples relaciones el comportamiento podría ser diferente, se decidió continuar con todos los algoritmos en la siguiente fase.

8.4.3. Set de datos con múltiples relaciones

Los individuos a identificar en sucesos de víctimas múltiples, pueden estar relacionados con cualquier grado de parentesco con los familiares. Por tanto habría que hacer una identificación recursiva con los algoritmos

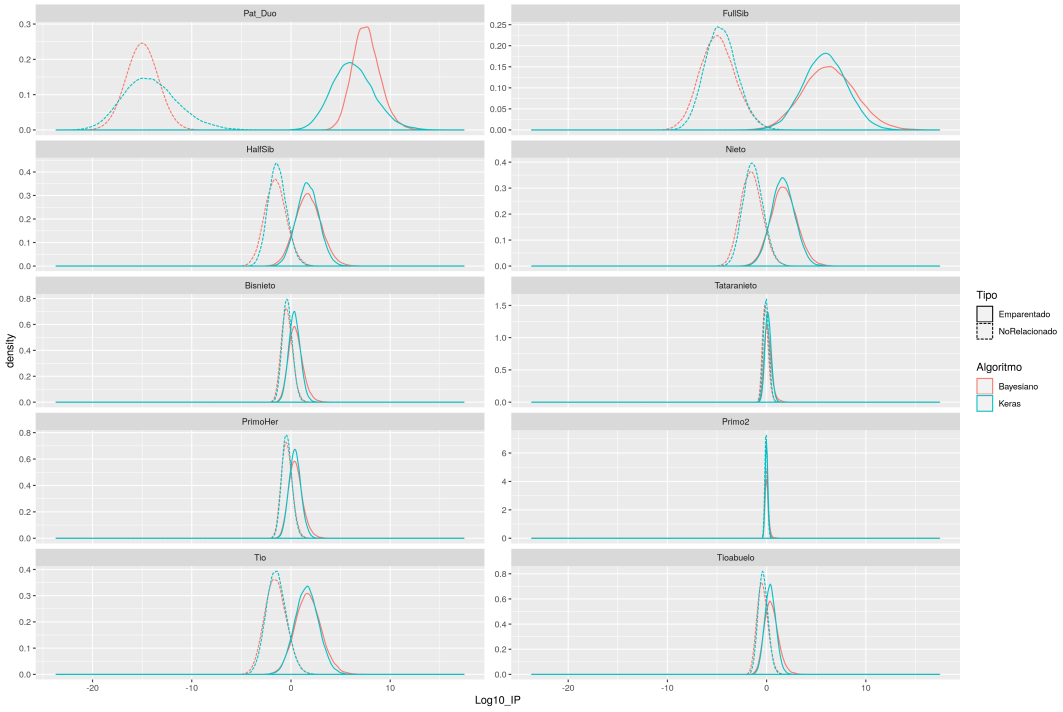


Figura 6: Comparación de la distribución de densidad entre el valor teórico (bayesiano) y la predicción (keras) en la misma escala.

entrenados para cada relación. El problema es que una relación cercana, daría en la mayoría de los casos positiva para otra relación más lejana frente a un individuo al azar. Por ejemplo, dos individuos con relación padre/hijo, es muy probable que den positivo en una relación de hermanos o medio hermanos.

Para solucionar el problema, se puede entrenar a los algoritmos con conjuntos de datos de todas las relaciones familiares, de forma que "aprendan" a clasificar los individuos en la categoría correcta. Para ello, se seleccionaron 4000 relaciones familiares de cada uno de los conjuntos de relación única, teniendo en cuenta que no procedieran de la misma familia original. Se añadió una categoría ramdom con individuos no relacionados genéticamente. Se hicieron dos conjuntos de datos, uno para entrenamiento (conjunto "multitrain") y otra para test (conjunto "multitest"). Se procedió a entrenar todos los algoritmos con dichos conjunto de datos multicategoría con una configuración básica.

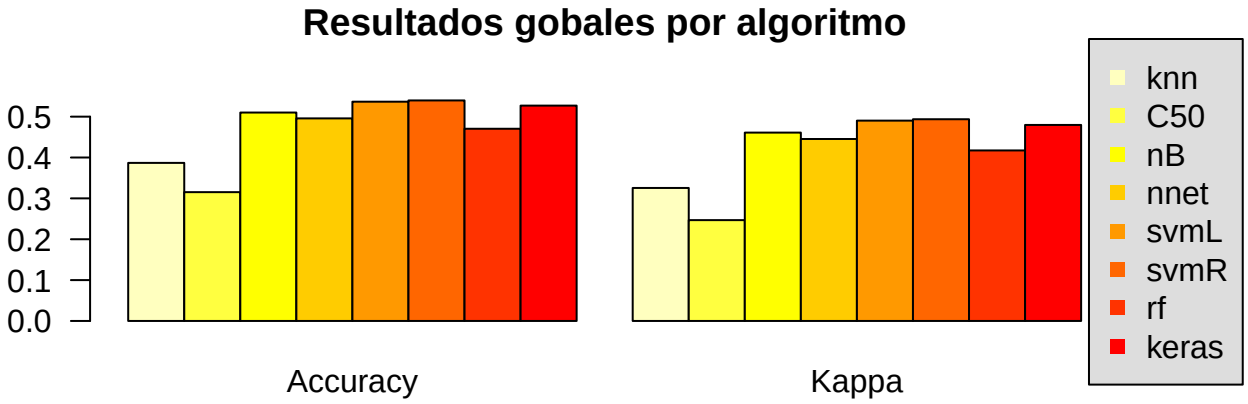


Figura 7: Comparación global por algoritmo con entrenamiento multicategoría

8.4.3.1. Comparación de algoritmos en predicción categorica con entrenamiento multicategoría y test multicategoría

En la tabla siguiente puede observarse el número de aciertos para cada categoría y cada algoritmo.

	Pat_Duo	**FullSib**	**HalfSib**	**Nieto**	**Bisnieto**	**Tataranieto**	**PrimoHer**	**Primo2**	**Tio**	**Tioabuelo**	**Random**	**Acierto(%)**
knn	3486	3168	3189	173	54	64	1291	427	151	109	3360	**35**
C50	3163	2485	1568	789	477	643	875	639	603	733	633	**28**
nB	3543	3557	2980	1117	901	424	2293	909	1068	1970	1637	**46**
nnet	3450	3437	2937	1119	1010	751	2013	924	1173	1829	1185	**45**
svmL	3522	3472	3149	1231	1261	682	2341	1064	1107	2090	1544	**48**
svmR	3586	3524	3178	1189	1220	688	2386	1050	1109	2116	1540	**49**
rf	3606	3524	3292	717	401	599	2488	903	687	1400	1195	**42**
keras	3573	3505	3175	519	964	658	2479	849	1756	2141	1461	**47**

* El número de individuos en cada categoría es de 4000

Los resultados muestran un porcentaje global de acierto bajo, similar en todos los algoritmos. No obstante, se observa que hay relaciones familiares con un alto porcentaje de acierto y otras con bajo porcentaje de acierto, perjudicando el total. Estas relaciones con menor grado de acierto se corresponden con relaciones lejanas con un grado de compartición de información genética bajo. En apartados anteriores, ya se vio que estas relaciones son difíciles de resolver.

Como puede observarse en las figuras 7 y8, hay tres algoritmos que tienen unos resultados ligeramente mejores que el resto: *Support Vector Machines* (SVM) con kernel linear y gausiano y `keras/tensorflow`.

En función de estos resultados, y puesto que el entrenamiento y predicción con los algoritmos SVM era muy costoso en tiempo para el equipo utilizado (en torno a 40 minutos para el conjunto de entrenamiento completo), se tomó la decisión de no entrenar los mismos y continuar el desarrollo del clasificador con `keras`, ya que las prestaciones eran similares y los tiempos de entrenamiento y predicción son mucho menores (en torno a los 3 minutos). Además con keras se pueden realizar multitud de configuraciones de redes neuronales en aras de mejorar el modelo y es un modelo exportable a otros entornos como Python.

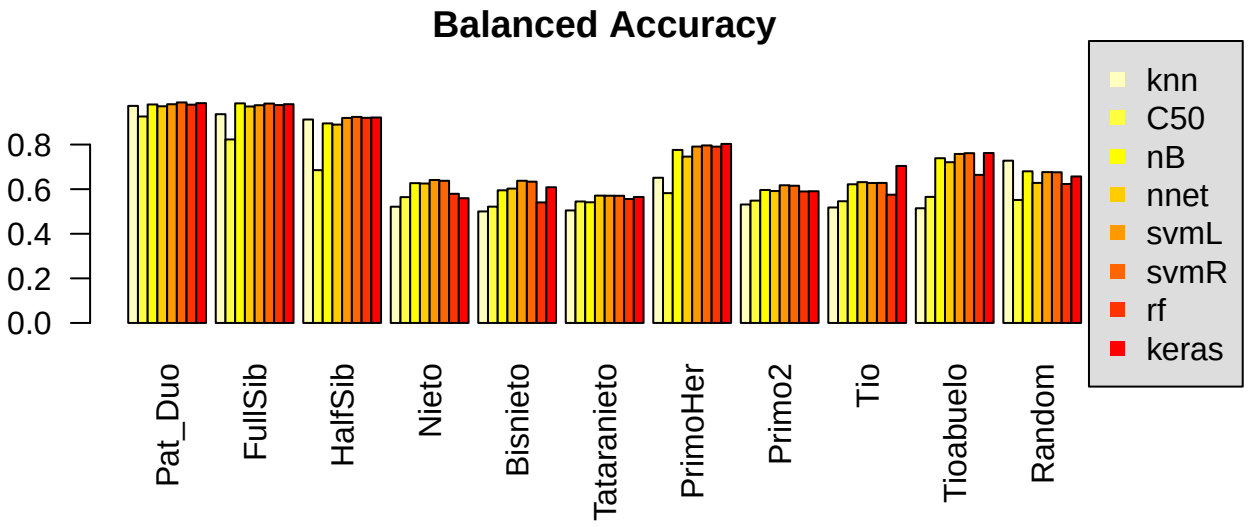


Figura 8: Comparación de precisión de algoritmos por relación familiar con entrenamiento multicategoría

8.5. Mejora de algoritmo/entrenamiento final

El modelo seleccionado, es una red neuronal de capas totalmente interconectadas (`keras_model_sequential()`). Para mejorar el modelo, se realizaron varias pruebas con diferentes números de capas y nodos en cada capa, diferentes funciones de activación y niveles de dropout. Los entrenamientos iniciales parecían indicar que mejoraba la precisión con 2 capas, con un número de nodos elevado (168 como el número de variables) y ligeros valores de dropout. No obstante, las mejoras en un parámetro, al combinarlo con otro no siempre mejoraban el modelo, ya que cada combinación de parámetros genera una configuración única que no es exactamente la combinación de los parámetros individuales. Por tanto se decidió hacer una valoración sistemática haciendo todas las combinaciones de varios parámetros y entrenando cada configuración. Los parámetros seleccionados en base a las primeras pruebas fueron:

- Número de capas: 2
- Numero de nodos por capa: 21, 42, 84, 168 (todas las combinaciones para las dos capas)
- Función de activación: relu, hard_sigmoid, linear
- Dropout: 0, 0.1, 0.2, 0.3 (todas las combinaciones para las dos capas)
- Optimizador: optimizer_rmsprop(), optimizer_adam(), optimizer_nadam(), optimizer_sgd()

Tras el entrenamiento, se observó que se producía sobreajuste (*overfiting*), por lo que se decidió hacer un nuevo entrenamiento de los mejores resultados aplicando regularización para reducir el overfitting[m]. Se aplicó una regularización L2 con un coeficiente de 0,001. Los parámetros para esta nueva ronda de entrenamiento fueron:

- Número de capas: 2
- Numero de nodos por capa: 84, 168 (todas las combinaciones para las dos capas)
- Función de activación: relu, hard_sigmoid, linear
- Dropout: 0, 0.1 (todas las combinaciones para las dos capas)
- Optimizador: optimizer_rmsprop(), optimizer_adam(), optimizer_nadam(), optimizer_sgd()

En el anexo 6 puede consultarse el código empleado.

Tras el entrenamiento se seleccionaron las 20 mejores opciones:

[m] https://keras.rstudio.com/articles/tutorial_overfit_underfit.html

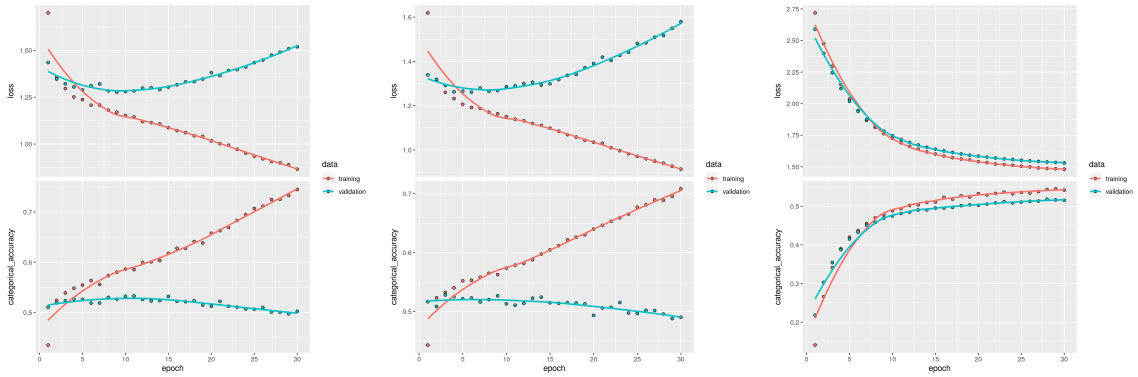


Figura 9: Entrenamiento multicategoría con varias condiciones

Categorical accuracy	Activación	Optimizador	Nodos capa1	Nodos capa2	Dropout1	Dropout2
0.7310	relu	optimizer_adam()	168	168	0	0.1
0.7216	relu	optimizer_adam()	168	168	0	0
0.7200	relu	optimizer_adam()	168	168	0.1	0
0.6975	relu	optimizer_nadam()	168	84	0	0.1
0.6961	relu	optimizer_rmsprop()	168	168	0.1	0.1
0.6961	relu	optimizer_nadam()	168	168	0	0
0.6903	relu	optimizer_nadam()	168	168	0	0.1
0.6895	relu	optimizer_nadam()	84	84	0	0
0.6893	relu	optimizer_adam()	168	84	0	0.1
0.6835	relu	optimizer_adam()	168	168	0.1	0.1
0.6795	relu	optimizer_rmsprop()	168	168	0	0
0.6765	relu	optimizer_rmsprop()	168	168	0	0.1
0.6756	relu	optimizer_nadam()	168	84	0.1	0
0.6745	relu	optimizer_nadam()	168	168	0.1	0
0.6707	relu	optimizer_nadam()	168	168	0.1	0.1
0.6699	relu	optimizer_adam()	168	84	0	0
0.6648	relu	optimizer_nadam()	168	84	0.1	0.1
0.6612	relu	optimizer_adam()	168	84	0.1	0.1
0.6605	relu	optimizer_adam()	84	168	0	0
0.6594	relu	optimizer_adam()	168	84	0.1	0

La mejor combinación es con función de activación `relu`, optimizador `adam` dos capas de 168 nodos y un dropout de 0,1 para la segunda capa. Sin embargo al revisar las curvas de entrenamiento, se ve que hay un claro sobreajuste. Lo mismo ocurre con el segundo, y prácticamente los 20 primeros, prácticamente todos con optimizador `adam` o `nadam`. Revisando las curvas, el primer modelo sin claros signos de sobreajuste fue con función de activación `linear`, optimizador `sgd` (que es menos agresivo), dos capas de 168 nodos y un dropout de 0,1 para la segunda capa. Se pueden ver en la figura 9 las tres curvas para la mejor entrada del optimizador adam, nadam y sgd (de izquierda a derecha):

- relu / adam / 168 x2 / dropout: 0/0,1 (izquierda)
- relu / nadam / 168 84 / dropout: 0/0,1 (centro)
- linear / sgd / 168 x2 / dropout: 0/0,1 (derecha)

La misma operación se realizó, utilizando en este caso la validación con el conjunto de test, seleccionando

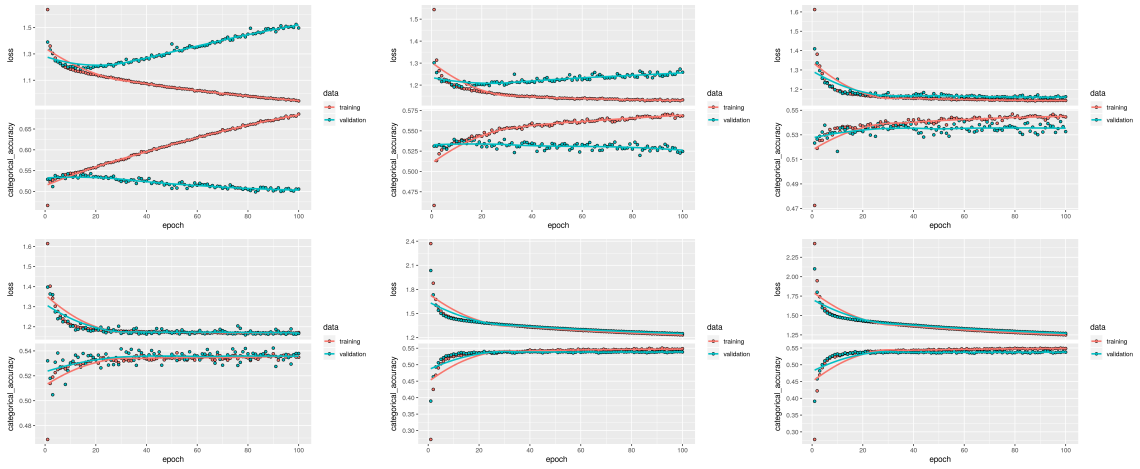


Figura 10: Entrenamiento multicategoría con varias condiciones. Validación con conjunto de test.

otros tres modelos, incluyendo uno con optimizador `rmsprop`. Los seis modelos se llevaron al entrenamiento con el conjunto completo de datos multicategoría. Se empezó con 30 épocas, pero los modelos no acababan de estabilizarse, así que para el entrenamiento final se utilizó un valor de 100 épocas.

El modelo elegido en el primer entrenamiento sigue siendo el más estable y el que muestra menos signos de sobre entrenamiento (abajo a la derecha en la figura 10). Este modelo es el elegido como modelo definitivo. La configuración es la siguiente:

```
# modelo
modelo <- keras_model_sequential()
modelo %>% layer_dense(units = 168, activation = "linear", input_shape = c(168),
    kernel_regularizer = regularizer_l2(l = 0.001)) %>% layer_dropout(rate = 0) %>%
    layer_dense(units = 168, activation = "linear", kernel_regularizer = regularizer_l2(l = 0.001)) %>%
    layer_dropout(rate = 0.1) %>% layer_dense(units = length(categorias),
    activation = "softmax")
# compilar
modelo %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_sgd(),
    metrics = c("categorical_accuracy"))
```

8.5.1. Valoración del modelo

Una vez entrenado con los 44000 comparaciones del conjunto de entrenamiento multicategoría y guardado el modelo, se procede a hacer las predicciones para el conjunto de test multicategoría (44000 comparaciones) y comtrastarlas con las categorías teóricas. Los resultados de la matriz de confusión son los siguientes:

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	3914	4	42	140	2	0	2	0	133	0	0
FullSib	0	3878	70	15	5	3	0	0	13	15	1
HalfSib	44	87	3502	306	178	19	1	22	345	12	35
Nieto	28	6	126	1426	446	8	188	9	1494	10	10
Bisnieto	0	4	137	551	1406	339	396	345	477	167	423
Tataranieto	0	0	3	10	136	653	79	459	8	518	367
PrimoHer	0	0	1	376	706	240	2659	283	406	157	336
Primo2	0	0	5	15	218	815	136	1190	14	519	711
Tio	14	3	82	1121	321	7	176	9	1077	8	9
Tioabuelo	0	18	12	15	127	1068	108	560	13	2307	324
Random	0	0	20	25	455	848	255	1123	20	287	1784

* El número de individuos en cada categoría es de 4000

	x
Accuracy	0.5408
Kappa	0.4949

	Sensitivity	Specificity	Balanced Accuracy
Class: Pat_Duo	0.9785	0.9919	0.9852
Class: FullSib	0.9695	0.9970	0.9832
Class: HalfSib	0.8755	0.9738	0.9246
Class: Nieto	0.3565	0.9419	0.6492
Class: Bisnieto	0.3515	0.9290	0.6403
Class: Tataranieto	0.1632	0.9605	0.5619
Class: PrimoHer	0.6647	0.9374	0.8011
Class: Primo2	0.2975	0.9392	0.6183
Class: Tio	0.2692	0.9562	0.6128
Class: Tioabuelo	0.5768	0.9439	0.7603
Class: Random	0.4460	0.9242	0.6851

En la figura 11 se observa la distribución de aciertos. Los resultados son similares a los vistos anteriormente, con una precisión muy buena para las relaciones más próximas y menos buenas para las más lejanas.

8.5.2. Modelo reducido de relaciones familiares

Como en los sucesos de víctimas múltiples no es habitual utilizar relaciones tan lejanas, se pueden suprimir del conjunto de entrenamiento dichas relaciones y quedarnos con las próximas (Pat_Duo, FullSib, HalfSib, Nieto, PrimoHer, Tio y Random). Por tanto, el entrenamiento lo hará con 28000 comparaciones (4000 por relación familiar). Sin embargo la evaluación se seguirá haciendo con el conjunto completo de test que incluye todas las categorías. Al hacer la predicción, los individuos con las relaciones familiares omitidas en el conjunto de entrenamiento (si los hay), los incorporará a no relacionados (random) o a cualquier otra con similitud genética. En la práctica, la influencia va a ser mínima ya que no suele haber parientes tan lejanos implicados,

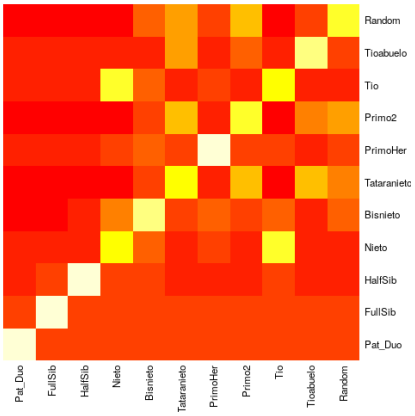


Figura 11: Multicategoría heamap

y en caso de que los hubiera, la incerteza sería tan grande que habría que recurrir a otras técnicas. Por tanto, este conjunto reducido, puede ser más efectivo que el completo en casos reales.

	Pat_Duo	FullSib	HalfSib	Nieto	PrimoHer	Tio	Random
Pat_Duo	3949	7	63	189	4	184	0
FullSib	0	3891	71	21	1	18	2
HalfSib	32	82	3563	358	3	390	55
Nieto	9	7	132	1481	257	1507	73
PrimoHer	0	0	4	424	2856	430	462
Tio	10	3	111	1407	280	1375	58
Random	0	10	56	120	599	96	3350

	x
Accuracy	0.7309
Kappa	0.6860
AccuracyLower	0.7257
AccuracyUpper	0.7361
AccuracyNull	0.1429
AccuracyPValue	0.0000
McnemarPValue	NaN

	Sensitivity	Specificity	Balanced Accuracy
Class: Pat_Duo	0.9872	0.9814	0.9843
Class: FullSib	0.9728	0.9953	0.9840
Class: HalfSib	0.8908	0.9617	0.9262
Class: Nieto	0.3703	0.9173	0.6438
Class: PrimoHer	0.7140	0.9450	0.8295
Class: Tio	0.3438	0.9221	0.6329
Class: Random	0.8375	0.9633	0.9004

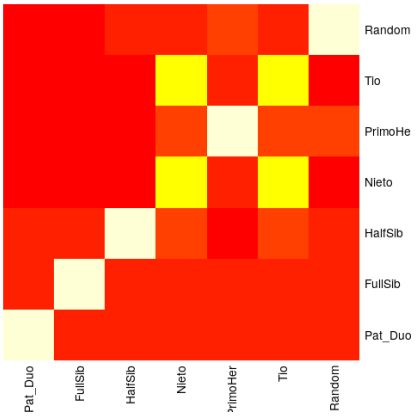


Figura 12: Multicategoría heamap con el conjunto reducido de datos

En la figura 12 se observa mejor la distribución de aciertos para el conjunto de entrenamiento reducido. Con este conjunto de entrenamiento se aprecia que la efectividad es muy buena en todas las relaciones, incluso en los no relacionados (random). Únicamente entre nietos y tíos hay una mala asignación cruzada clarísima. Si se revisan las curvas de distribución de IP, estas dos relaciones muestran una distribución de frecuencias muy similar.

8.6. Escenarios simulados de catástrofes

El objetivo del presente trabajo era la realización de un clasificador para sucesos de víctimas múltiples. Para comprobar como se comportaría en casos reales se han creado diez escenarios simulando sucesos desde el más sencillo hasta otros más complejos. Lo habitual es tener padres, hermanos y abuelos que buscan a sus familiares. En ese sentido, los escenarios tienen una proporción mayor de dichas relaciones, salvo el supuesto 4 que se ha creado expresamente con relaciones lejanas para poner a prueba el clasificador.

Los escenarios se han creado de forma similar a los conjuntos de entrenamiento, salvo que en este caso se han generado todas las comparaciones posibles de familiares con víctimas. Se ha intentado simular una proporción de relaciones familiares de las víctimas similares a los escenario habituales en catastrofes. Cada escenario tiene un número de víctimas (n), y para simplificar se han generado n familiares de relación conocida con la víctima (aunque se puede trabajar con cualquier número de víctimas y familiares). El número de comparaciones, y por consiguiente de entradas que va a categorizar el algoritmo, es de n^2 en cada escenario, ya que se compara cada víctima con todos los familiares. De esas comparaciones, sólo n serán verdaderas relaciones familiares, y el resto serán de indivíduos no relacionados (en el caso de escenarios con víctimas "No relac.", el número de relaciones ciertas serán n-1, ya que la relación de esa víctima debe caer en la categoría random).

Tenemos tres tipos de escenarios:

- Escenarios simples con 6 víctimas. Simulan el equivalente a un accidente de circulación múltiple, la caida de una avioneta o las vístimas de una inundación local. Incluye los escenarios del 1 al 4. Las relaciones son próximas, salvo el 4 como se ha comentado, que suele ser lo habitual. El escenario 3 con una relación de tio y otra de nieto es de complejidad media.

- Escenarios medios con 20 víctimas. Simulan catástrofes más graves como el accidente de un autobus o tren. Incluye los escenarios 5 al 7.

- Escenario graves con 50 o 200 víctimas. Simula grandes catástrofes, por suerte poco frecuentes como grandes atentados terroristas, grandes seismos o tsunamis. No suele haber catástrofes en nuestro ámbito con mayor número de víctimas[n]

[n]Posiblemente el más sonado es el del 11-S (torres gemelas en New York, pentágono y aviones), en el que no se ha podido

Los resultados para las relaciones más proximas son muy buenas. A partir de nietos y tíos, la capacidad de acierto disminuye bastante pero sigue siendo bastante buena. La tasa de falsos negativos es baja. Por contra la tasa de falsos positivos hay que mejorarla.

Uno de los problemas a mejorar es esta última circunstancia (falsos positivos) en la que para las comparaciones que no son ciertas, se siguen clasificando como relaciones familiares (mayoritariamente lejanas) que inteferirían en la identificación, al tener que comprobar más de una asignación a cada víctima. Se espera que futuras mejoras en el modelo, en el entrenamiento y la supresión de familiares lejanos (modelo reducido) puedan reducir la falsa asignación.

Los escenarios 3 y 10 se han evaluado también con el entrenamiento con el con el set de entrenamiento multicategoría reducido, mejorando los resultados.

identificar a la totalidad de víctimas mortales, de las 2996 reconocidas oficialmente. https://en.wikipedia.org/wiki/Casualties_of_the_September_11_attacks#Forensic_identification (visitada 20-12-18)

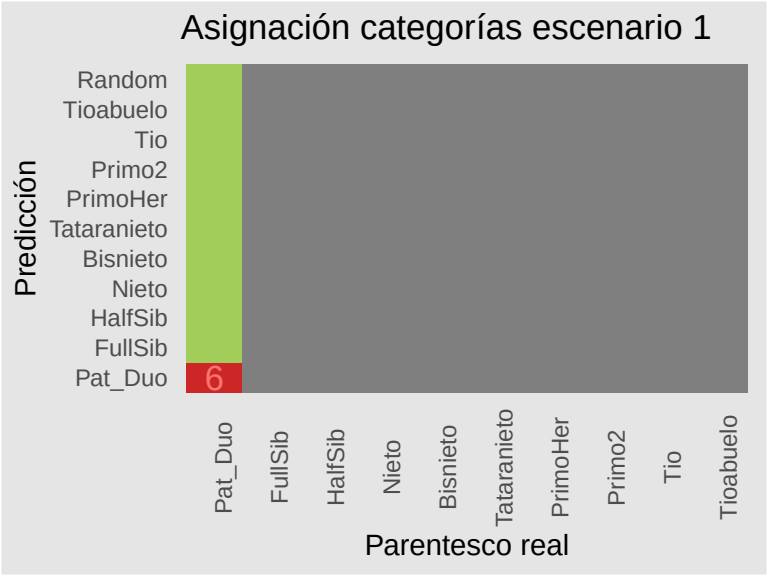


Figura 13: Predicción de categorías del escenario 1.

8.6.1. Escenario 1.

6 individuos con la siguiente proporción de relaciones familiares:

Padres/Hijos
6

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	6	0	0	0	0	0	0	0	0	0	0
FullSib	0	0	0	0	0	0	0	0	0	0	0
HalfSib	0	0	0	0	0	0	0	0	0	0	0
Nieto	0	0	0	0	0	0	0	0	0	0	0
Bisnieto	0	0	0	0	0	0	0	0	0	0	1
Tataranieto	0	0	0	0	0	0	0	0	0	0	6
PrimoHer	0	0	0	0	0	0	0	0	0	0	3
Primo2	0	0	0	0	0	0	0	0	0	0	7
Tio	0	0	0	0	0	0	0	0	0	0	0
Tioabuelo	0	0	0	0	0	0	0	0	0	0	4
Random	0	0	0	0	0	0	0	0	0	0	9

Tasa de aciertos
100 %

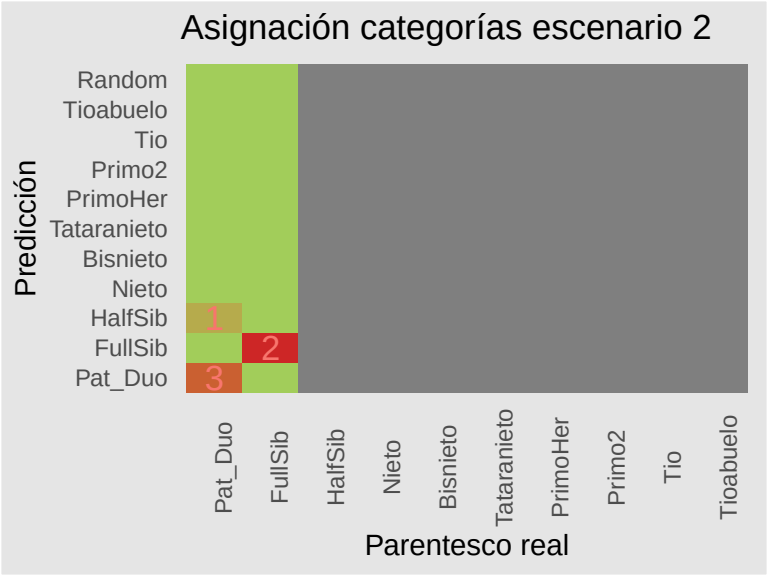


Figura 14: Predicción de categorías del escenario 2.

8.6.2. Escenario 2.

6 individuos con la siguiente proporción de relaciones familiares:

Padres/Hijos	Hermanos
4	2

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	3	0	0	0	0	0	0	0	0	0	0
FullSib	0	2	0	0	0	0	0	0	0	0	0
HalfSib	1	0	0	0	0	0	0	0	0	0	0
Nieto	0	0	0	0	0	0	0	0	0	0	1
Bisnieto	0	0	0	0	0	0	0	0	0	0	3
Tataranieto	0	0	0	0	0	0	0	0	0	0	7
PrimoHer	0	0	0	0	0	0	0	0	0	0	1
Primo2	0	0	0	0	0	0	0	0	0	0	8
Tio	0	0	0	0	0	0	0	0	0	0	0
Tioabuelo	0	0	0	0	0	0	0	0	0	0	0
Random	0	0	0	0	0	0	0	0	0	0	10

Tasa de aciertos
83.33 %

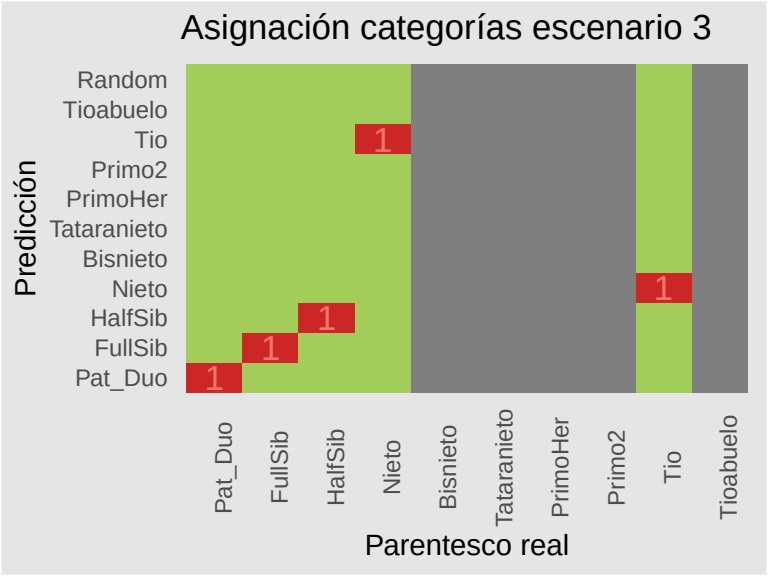


Figura 15: Predicción de categorías del escenario 3.

8.6.3. Escenario 3.

6 individuos con la siguiente proporción de relaciones familiares:

Padres/Hijos	Hermanos	Medio Hermanos	Abuelo/Nieto	Tio/sobrino	No relac.
1	1	1	1	1	1

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	1	0	0	0	0	0	0	0	0	0	0
FullSib	0	1	0	0	0	0	0	0	0	0	0
HalfSib	0	0	1	0	0	0	0	0	0	0	1
Nieto	0	0	0	0	0	0	0	0	1	0	1
Bisnieto	0	0	0	0	0	0	0	0	0	0	5
Tataranieto	0	0	0	0	0	0	0	0	0	0	2
PrimoHer	0	0	0	0	0	0	0	0	0	0	2
Primo2	0	0	0	0	0	0	0	0	0	0	6
Tio	0	0	0	1	0	0	0	0	0	0	0
Tioabuelo	0	0	0	0	0	0	0	0	0	0	1
Random	0	0	0	0	0	0	0	0	0	0	12

Tasa de aciertos
50 %

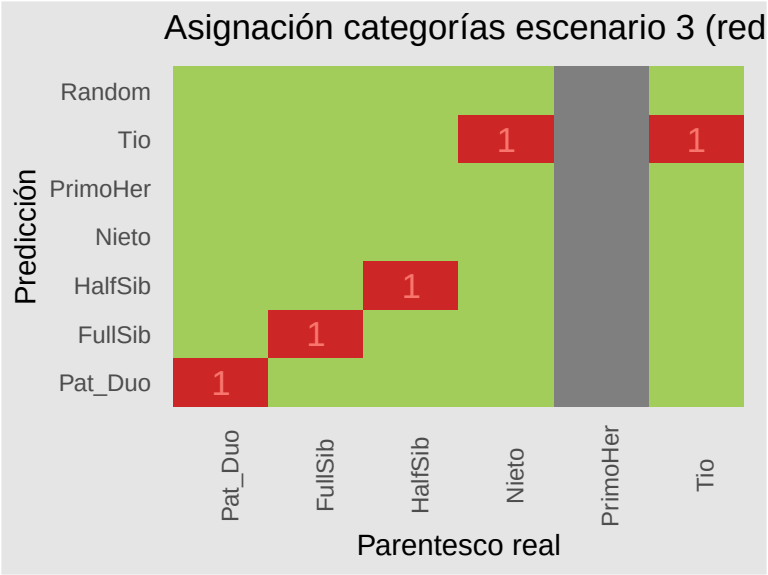


Figura 16: Predicción de categorías del escenario 3 con el set de entrenamiento multicategoría reducido.

Evaluación del escenario con el algoritmo reducido:

	Pat_Duo	FullSib	HalfSib	Nieto	PrimoHer	Tio	Random
Pat_Duo	1	0	0	0	0	0	0
FullSib	0	1	0	0	0	0	0
HalfSib	0	0	1	0	0	0	1
Nieto	0	0	0	0	0	0	1
PrimoHer	0	0	0	0	0	0	4
Tio	0	0	0	1	0	1	0
Random	0	0	0	0	0	0	24

Tasa de aciertos
66.67 %

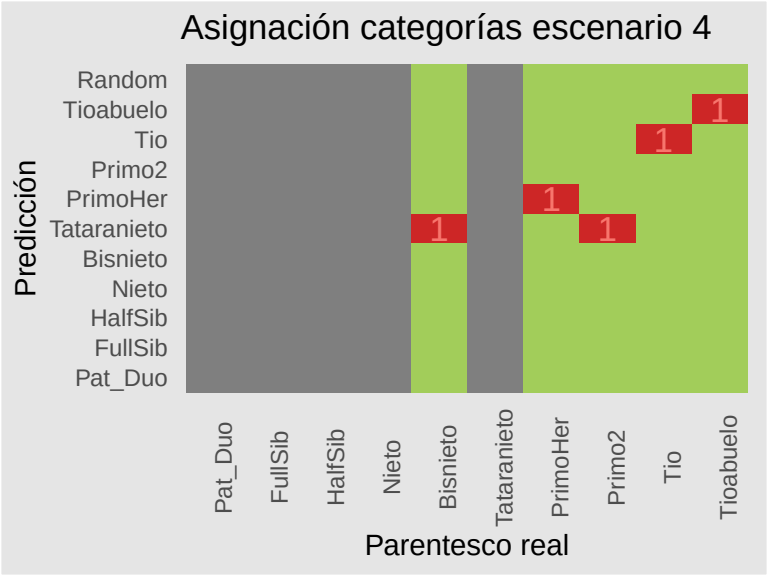


Figura 17: Predicción de categorías del escenario 4.

8.6.4. Escenario 4.

6 individuos con la siguiente proporción de relaciones familiares:

Primos herm.	Primos seg.	Bisabuelo/bisnieto	Tioabuelo/Sobrinonieto	Tio/sobrino	No relac.
1	1	1	1	1	1

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	0	0	0	0	0	0	0	0	0	0	0
FullSib	0	0	0	0	0	0	0	0	0	0	0
HalfSib	0	0	0	0	0	0	0	0	0	0	0
Nieto	0	0	0	0	0	0	0	0	0	0	0
Bisnieto	0	0	0	0	0	0	0	0	0	0	5
Tataranieto	0	0	0	0	1	0	0	1	0	0	3
PrimoHer	0	0	0	0	0	0	1	0	0	0	0
Primo2	0	0	0	0	0	0	0	0	0	0	5
Tio	0	0	0	0	0	0	0	0	1	0	0
Tioabuelo	0	0	0	0	0	0	0	0	0	1	2
Random	0	0	0	0	0	0	0	0	0	0	15

Tasa de aciertos
50 %

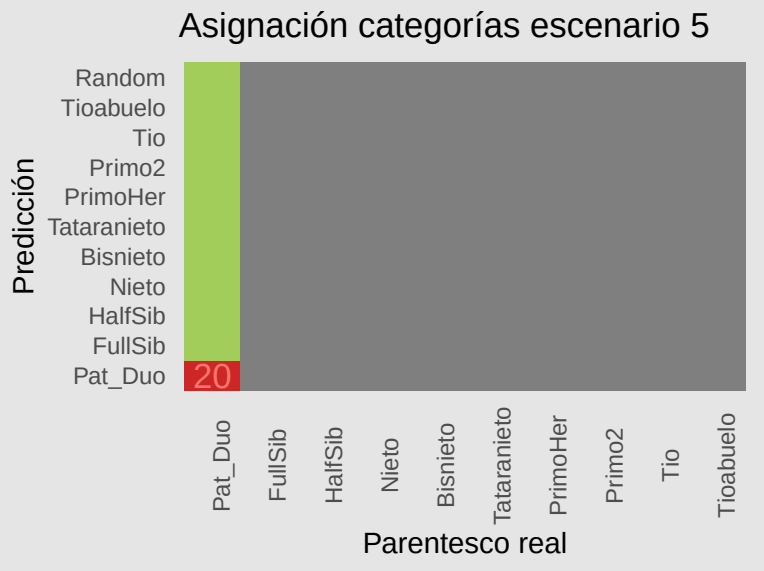


Figura 18: Predicción de categorías del escenario 5.

8.6.5. Escenario 5.

20 individuos con la siguiente proporción de relaciones familiares:

Padres/Hijos
20

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	20	0	0	0	0	0	0	0	0	0	0
FullSib	0	0	0	0	0	0	0	0	0	0	1
HalfSib	0	0	0	0	0	0	0	0	0	0	4
Nieto	0	0	0	0	0	0	0	0	0	0	2
Bisnieto	0	0	0	0	0	0	0	0	0	0	37
Tataranieto	0	0	0	0	0	0	0	0	0	0	59
PrimoHer	0	0	0	0	0	0	0	0	0	0	28
Primo2	0	0	0	0	0	0	0	0	0	0	73
Tio	0	0	0	0	0	0	0	0	0	0	3
Tioabuelo	0	0	0	0	0	0	0	0	0	0	39
Random	0	0	0	0	0	0	0	0	0	0	134

Tasa de aciertos
100 %

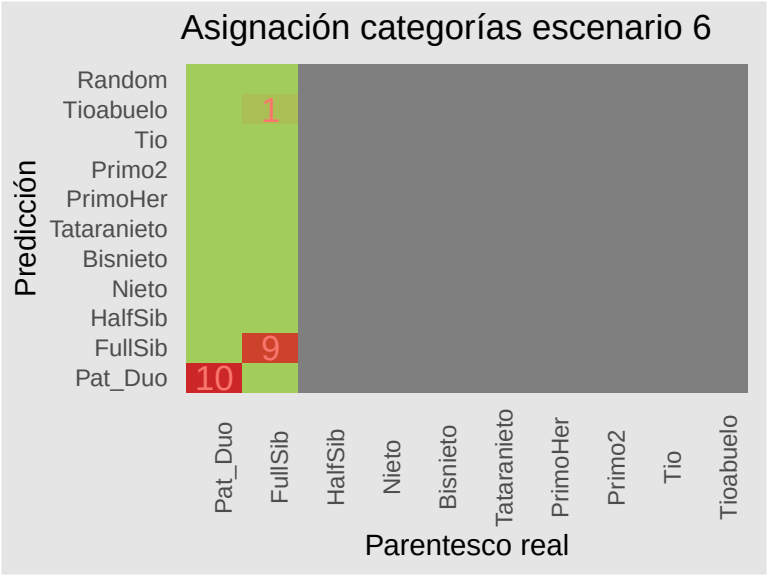


Figura 19: Predicción de categorías del escenario 6.

8.6.6. Escenario 6.

20 individuos con la siguiente proporción de relaciones familiares:

Padres/Hijos	Hermanos
10	10

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	10	0	0	0	0	0	0	0	0	0	0
FullSib	0	9	0	0	0	0	0	0	0	0	0
HalfSib	0	0	0	0	0	0	0	0	0	0	6
Nieto	0	0	0	0	0	0	0	0	0	0	0
Bisnieto	0	0	0	0	0	0	0	0	0	0	41
Tataranieto	0	0	0	0	0	0	0	0	0	0	58
PrimoHer	0	0	0	0	0	0	0	0	0	0	20
Primo2	0	0	0	0	0	0	0	0	0	0	104
Tio	0	0	0	0	0	0	0	0	0	0	0
Tioabuelo	0	1	0	0	0	0	0	0	0	0	27
Random	0	0	0	0	0	0	0	0	0	0	124

Tasa de aciertos
95 %

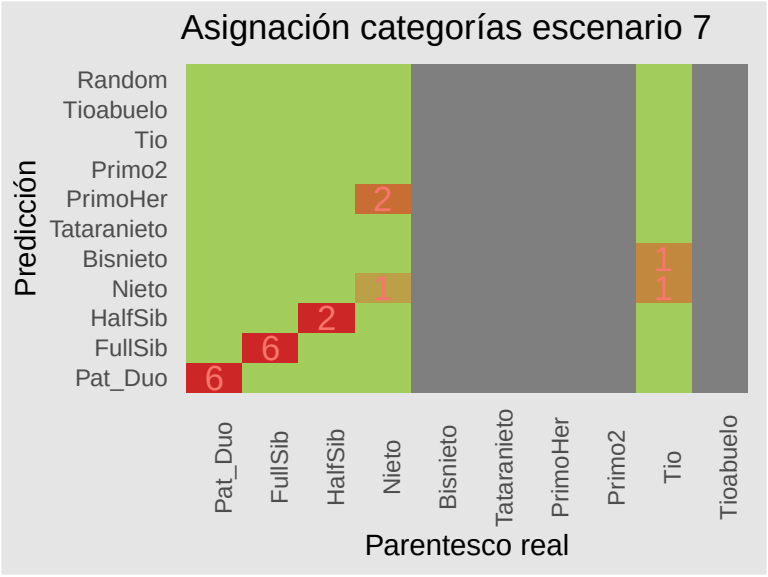


Figura 20: Predicción de categorías del escenario 7.

8.6.7. Escenario 7.

20 individuos con la siguiente proporción de relaciones familiares:

Padres/Hijos	Hermanos	Medio Hermanos	Abuelo/Nieto	Tio/sobrino	No relac.
6	6	2	3	2	1

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	6	0	0	0	0	0	0	0	0	0	0
FullSib	0	6	0	0	0	0	0	0	0	0	0
HalfSib	0	0	2	0	0	0	0	0	0	0	1
Nieto	0	0	0	1	0	0	0	0	1	0	0
Bisnieto	0	0	0	0	0	0	0	0	1	0	46
Tataranieto	0	0	0	0	0	0	0	0	0	0	55
PrimoHer	0	0	0	2	0	0	0	0	0	0	23
Primo2	0	0	0	0	0	0	0	0	0	0	92
Tio	0	0	0	0	0	0	0	0	0	0	1
Tioabuelo	0	0	0	0	0	0	0	0	0	0	39
Random	0	0	0	0	0	0	0	0	0	0	123

Tasa de aciertos
75 %

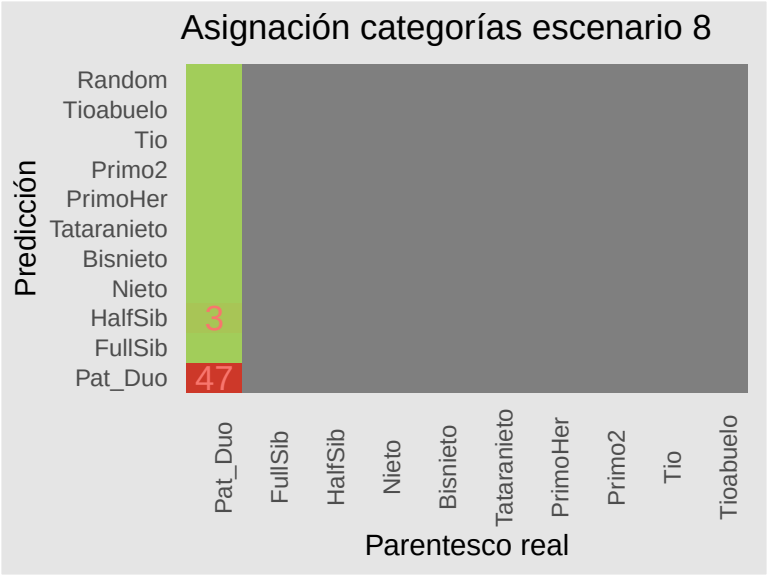


Figura 21: Predicción de categorías del escenario 8.

8.6.8. Escenario 8.

50 individuos con la siguiente proporción de relaciones familiares:

Padres/Hijos
50

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	47	0	0	0	0	0	0	0	0	0	0
FullSib	0	0	0	0	0	0	0	0	0	0	2
HalfSib	3	0	0	0	0	0	0	0	0	0	15
Nieto	0	0	0	0	0	0	0	0	0	0	9
Bisnieto	0	0	0	0	0	0	0	0	0	0	243
Tataranieto	0	0	0	0	0	0	0	0	0	0	393
PrimoHer	0	0	0	0	0	0	0	0	0	0	155
Primo2	0	0	0	0	0	0	0	0	0	0	571
Tio	0	0	0	0	0	0	0	0	0	0	15
Tioabuelo	0	0	0	0	0	0	0	0	0	0	229
Random	0	0	0	0	0	0	0	0	0	0	818

Tasa de aciertos
94 %

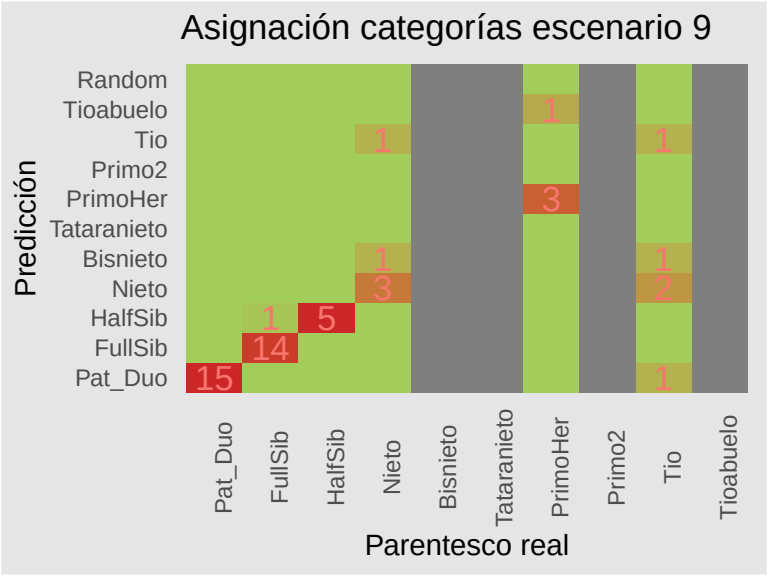


Figura 22: Predicción de categorías del escenario 9.

8.6.9. Escenario 9.

50 individuos con la siguiente proporción de relaciones familiares:

Padres/Hijos	Hermanos	Medio Hermanos	Abuelo/Nieto	Tio/sobrino	Primos herm.	No relac.
15	15	5	5	5	4	1

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	15	0	0	0	0	0	0	0	1	0	0
FullSib	0	14	0	0	0	0	0	0	0	0	1
HalfSib	0	1	5	0	0	0	0	0	0	0	22
Nieto	0	0	0	3	0	0	0	0	2	0	7
Bisnieto	0	0	0	1	0	0	0	0	1	0	290
Tataranieto	0	0	0	0	0	0	0	0	0	0	349
PrimoHer	0	0	0	0	0	0	3	0	0	0	176
Primo2	0	0	0	0	0	0	0	0	0	0	611
Tio	0	0	0	1	0	0	0	0	1	0	10
Tioabuelo	0	0	0	0	0	0	1	0	0	0	208
Random	0	0	0	0	0	0	0	0	0	0	776

Tasa de aciertos
82 %

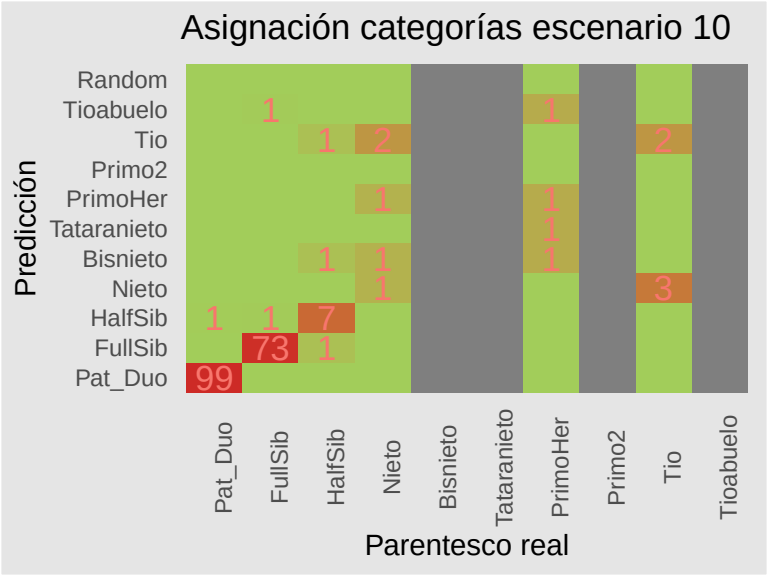


Figura 23: Predicción de categorías del escenario 10.

8.6.10. Escenario 10.

200 individuos con la siguiente proporción de relaciones familiares:

Padres/Hijos	Hermanos	Medio Hermanos	Abuelo/Nieto	Tio/sobrino	Primos herm.	No relac.
100	75	10	5	5	4	1

	Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo	Random
Pat_Duo	99	0	0	0	0	0	0	0	0	0	0
FullSib	0	73	1	0	0	0	0	0	0	0	13
HalfSib	1	1	7	0	0	0	0	0	0	0	238
Nieto	0	0	0	1	0	0	0	0	3	0	115
Bisnieto	0	0	1	1	0	0	1	0	0	0	4524
Tataranieto	0	0	0	0	0	0	1	0	0	0	6009
PrimoHer	0	0	0	1	0	0	1	0	0	0	2712
Primo2	0	0	0	0	0	0	0	0	0	0	9206
Tio	0	0	1	2	0	0	0	0	2	0	121
Tioabuelo	0	1	0	0	0	0	1	0	0	0	3525
Random	0	0	0	0	0	0	0	0	0	0	13337

Tasa de aciertos
91.5 %

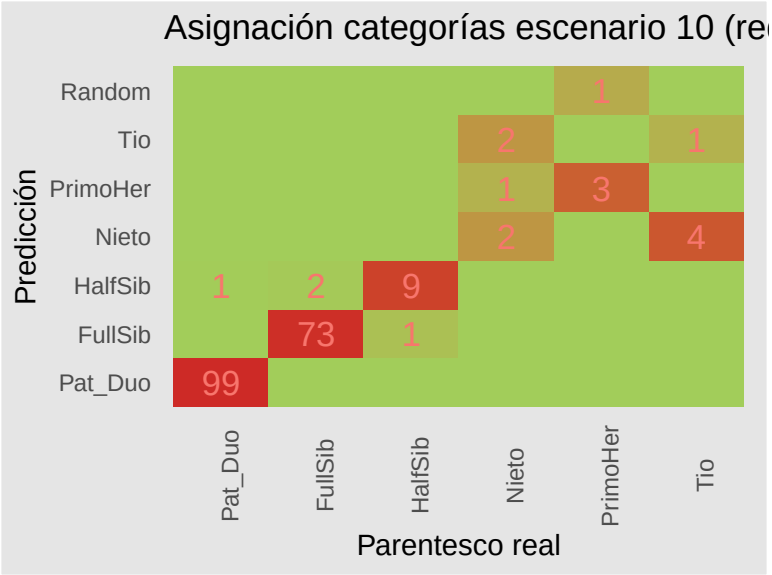


Figura 24: Predicción de categorías del escenario 10 con el set de entrenamiento multicategoría reducido.

Evaluación del escenario con el algoritmo reducido:

	Pat_Duo	FullSib	HalfSib	Nieto	PrimoHer	Tio	Random
Pat_Duo	99	0	0	0	0	0	0
FullSib	0	73	1	0	0	0	29
HalfSib	1	2	9	0	0	0	494
Nieto	0	0	0	2	0	4	721
PrimoHer	0	0	0	1	3	0	4313
Tio	0	0	0	2	0	1	618
Random	0	0	0	0	1	0	33625

Tasa de aciertos
93.5 %

9. Conclusiones

Se han conseguido los tres objetivos marcados: se ha desarrollado un conjunto de datos de marcadores STR en familias con cinco generaciones, adecuados para el entrenamiento de los algoritmos, se ha desarrollado un algoritmo de *machine learning* que al menos iguala los métodos clásicos de estudio del parentesco y se ha implementado un clasificador que asigne a un par de individuos una relación de parentesco entre ellos aplicable a grandes conjuntos de individuos.

Los modelos ensayados, han demostrado ser tan efectivos como los métodos habituales utilizados en los laboratorios de genética forense para establecer la relación entre dos personas. No obstante, la aplicación en casos judiciales está limitado por el hecho de que los algoritmos empleados son de difícil explicación y comprensión. Sin embargo en el caso de sucesos de víctimas múltiples cualquier ayuda que agilice y facilite la identificación es de gran utilidad.

Adicionalmente, cabe destacar que todo el proceso se ha realizado en un equipo de bajas prestaciones, lo que permitiría prácticamente a cualquier laboratorio forense implementar el modelo e incluso entrenarlo.

Se ha logrado implementar un modelo preeentrenado para la identificación de victimas en sucesos múltiples. Este modelo permite la clasificación en pocos minutos de la mayoría de las víctimas respecto a sus familiares. En los escenarios más habituales de víctimas múltiples suelen estar involucrados padre e hijos, hermanos, y en algunas ocasiones tíos o abuelos. En este tipo de relaciones, el modelo presenta una alta tasa de acierto, del 100 % para la mayoría de casos en los que sólo hay padres e hijos implicados. En aquellos casos en los que la relación familiar es lejana, debido a la poca información genética que comparten, la tasa de acierto es más baja en cuanto a establecer la relación familiar pero si produce una asociación de los individuos. Aun así supone una indicación fuerte del parentesco, para completar los estudios con otros tipos de técnicas como pueden ser la identificación odontológica o lofoscópica.

La planificación se ha seguido, incluso se ha ido por delante en los objetivos 1 y 2. Para el objetivo 3 (desarrollo del conjunto multicategoría), debido a la potencia limitada del equipo, los tiempos de entrenamiento para todos los algoritmos (excepto para keras) ha requerido más tiempo del planificado, por lo que toda la planificación sufrió un retraso de unos diez días, repercutiendo en el tiempo para elaborar la presente memoria. El disponer de un equipo más potente y haber decidido, tras el entrenamiento con una sola relación familiar, seguir sólo con keras, hubiera ajustado la ejecución a lo planificado.

Un error sistemático cometido ha sido generar ficheros html con los resultados y gráficas parciales. A la hora de elaborar la memoria, ha requerido generar las gráficas y tablas de nuevo, con el consiguiente tiempo empleado para cada versión de prueba de la memoria. Hubiera sido preferible ir generando figuras y resultados de tablas en ficheros listos para incorporar.

No se ha logrado desarrollar una aplicación con Shiny debido a la falta de tiempo para el aprendizaje y la elaboración. En la planificación inicial ya se planteaba como como riesgo debido a lo apretado del programa. Queda pendiente de elaborar junto con una mejora del modelo, de cara a facilitar una herramienta más amigable a los laboratorios forenses.

De cara al futuro, quedaría pendiente una mejora del modelo explorando las diversas alternativas disponibles para keras/tensorflow, de cara a reducir la asignación de parentesco en las relaciones al azar, que todavía sigue siendo elevado. Tras la mejora del modelo, además de la aplicación Shiny mencionada anteriormente, se podría desarrollar un notebook con Python ya que el modelo de keras sería reutilizable en Python directamente. Todo ello se podría publicar en revistas especializadas de genética forense para darle difusión.

10. Bibliografía

1. Schlecht J, Kaplan ME, Barnard K, Karafet T, Hammer MF, Merchant NC. Machine-learning approaches for classifying haplogroup from Y chromosome STR data. PLoS computational biology. 2008;4(6):e1000093.

2. Marciano MA, Adelman JD. PACE: Probabilistic Assessment for Contributor Estimation—A machine learning-based assessment of the number of contributors in DNA mixtures. Forensic Science International: Genetics. 2017;27:82-91.

3. Marciano MA, Williamson VR, Adelman JD. A hybrid approach to increase the informedness of CE-based data using locus-specific thresholding and machine learning. Forensic Science International: Genetics. 2018;35:26-37.

4. Hummel K, Ihm P, Schmidt V. Biostatistical Opinion of Parentage Based Upon the Results of Blood Group Tests: Biostatistische Abstammungsbegutachtung Mit Blutgruppenbefunden. G. Fischer; 1971.

5. Bär W, Brinkmann B, Budowle B, Carracedo A, Gill P, Lincoln P, etal. DNA recommendations. Further report of the DNA Commission of the ISFH regarding the use of short tandem repeat systems. International Society for Forensic Haemogenetics. Int J Legal Med. 1997;110(4):175-6.

6. Luque JA. Genética forense. Del laboratorio a los Tribunales. En: Crespillo M, Barrio PA, editores. Diaz de Santos; 2018. pp. 351-82.

7. Essen-Möller E. Die Beweiskraft der Ahnlichkeit im Vaterschaftsnachweis – theoretische Grundlagen. Mitt Anthropol Ges. 1938;68:9-53.

8. Luque J, Valverde J. Indice de hermanidad I. Estudio y valoración mediante STR's. En: IV Jornadas de Genética Forense La Gomera España. 1999.

9. Luque J, Valverde J. Indice de hermanidad II. Estudio y valoración mediante STR's. En: VI Jornadas de Genética Forense Córdoba Argentina. 2001.

10. Lantz B. Machine learning with R. Packt Publishing Ltd; 2015.

11. García O, Alonso J, Cano J, García R, Luque G, Martín P, etal. Population genetic data and concordance study for the kits Identifiler, NGM, PowerPlex ESX 17 System and Investigator ESSplex in Spain. Forensic Science International: Genetics. 2012;6(2):e78-9.

12. García O, Alonso J, Cano J, García R, Luque G, Martín P, etal. Corrigendum to «"Population genetic data and concordance study for the kits Identifiler, NGM, PowerPlex ESX 17 System and Investigator ESSplex in Spain"»[Forensic Sci. Int.: Genet. 6 (2012), e78–e79]. Forensic Science International: Genetics. 2014;9:192.

13. Mostad P, Egeland T, Simonsson I. [Internet]. 2016. Disponible en: https://CRAN.R-project.org/package=Familias

14. Egeland T, Mostad PF, Mevåg B, Stenersen M. Beyond traditional paternity and identification cases: selecting the most probable pedigree. Forensic Science International. 2000;110(1):47-59.

15. Kling D, Tillmar AO, Egeland T. Familias 3–Extensions and new functionality. Forensic Science International: Genetics. 2014;13:121-7.

11. Anexos

Anexo 1 Modelos de datos

Modelo de datos original

			M1		**M2**		**M3**		**...**	**M19**		**M20**		**M21**	
Familia	**Nombre**	Id	D8S1179.1	D8S1179.2	D21S11.1	D21S11.2	D7S820.1	D7S820.2	resto_de_marcadores	D2S441.1	D2S441.2	D12S391.1	D12S391.2	SE33.1	SE33.2
Familia1	**ABO1**	1	14	15	31	31.2	9	8	...	10	12	17	23	28.2	15
	ABA1	2	12	10	32.2	31	11	8	...	11	15	15	18	21	31.2
	ABO2	3	12	10	29	31.2	10	9	...	14	11	20	18	15	19
	ABA2	4	14	13	32.2	32	8	7	...	12	14	19	19	29.2	17
	P	5	14	12	31.2	31	9	8	...	10	11	23	15	15	21
	M	6	12	14	31.2	32.2	10	7	...	14	12	18	19	19	17
	H1	7	12	12	31	32.2	9	7	...	10	12	23	19	21	17
	H2	8	14	14	31.2	32.2	8	7	...	10	12	15	19	15	19
	H3	9	12	12	31.2	31.2	8	10	...	11	12	15	18	21	17
	C__H1	10	15	14	29	30	10	11	...	11	11.3	19	21	22.2	17
	C__H3	11	11	15	28	32.2	11	9	...	11.3	15	18	15	21	24.2
	H1__1	12	15	12	30	32.2	11	7	...	11.3	12	21	19	17	21
	H3__1	13	12	15	31.2	32.2	10	9	...	12	11.3	15	18	17	24.2
	C__H1__1	14	13	13	30	31	10	8	...	11.3	14	17.3	18	21	26.2
	C__H3__1	15	13	13	30	29	10	11	...	14	11	18.3	18.3	15	18
	H1__1__1	16	12	13	32.2	31	7	8	...	12	11.3	19	17.3	21	26.2
	H3__1__1	17	13	12	30	32.2	11	9	...	11	11.3	18.3	18	15	17
	M2	18	14	8	25	32.2	10	11	...	11	11	23	21	31.2	30.2
	H0	19	12	8	31.2	25	9	10	...	11	11	15	21	15	30.2
	Prandom	20	13	10	31.2	30	8	11	...	14	11	19	22	17	14
Familia2	**ABO1**	1	10	13	30	30	9	10	...	10	11	21	22	21	30.2
	ABA1	2	13	14	30	30	11	10	...	10	11	23	18	20	17
	ABO2	3	11	13	29	28	10	10	...	10	11	20	21	27.2	18
	ABA2	4	11	14	30.2	31.2	11	9	...	15	10	17	19	17.3	27.2
	P	5	10	14	30	30	10	11	...	10	11	21	18	30.2	20
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

* Las cabeceras marcadas en negrita no forman parte de la tabla

El conjunto de datos original se estructura en una línea por individuo, 20 individuos por familia, 40000 familias. Cada individuo lleva un índice de su posición secuencial en la familia. Consta (columnas) de dos alelos por marcador, con el nombre del marcador finalizado en .1 y .2 respectivamente (este formato es compatible con el paquete `Familias`). Los individuos homozigotos tienen ambos alelos iguales. El formato para los alelos es un factor, siendo diferentes en número y descripción para cada marcador. No tienen relación entre marcadores aunque tengan la misma etiqueta.

Modelo de datos recodificado

	Marcador1				**Marcador2**						**Marcador1**				**Marcador2**				
	Frecuencias				**Frecuencias**				**...**	**Categoria**	**Comparación**				**Comparación**				**...**
	V1	V2	V3	V4	V5	V6	V7	V8	V9_84	cat_train	C1	C2	C3	C4	C5	C6	C7	C8	C9_84
Fam1_T	0.2482	0.1285	0.1285	0.1285	0.1109	0.1109	0.0616	0.1109	...	True	0	1	0	1	1	0	1	0	...
Fam2_T	0.0739	0.2482	0.2482	0.2482	0.2763	0.2763	0.2763	0.1056	...	True	0	1	0	1	1	1	0	0	...
Fam3_T	0.0739	0.088	0.088	0.2482	0.2763	0.2095	0.2095	0.1056	...	True	0	1	0	0	0	1	0	0	...
Fam4_T	0.2853	0.2482	0.2482	0.2482	0.2763	0.2763	0.2095	0.2763	...	True	0	1	0	1	1	0	1	0	...
Fam5_T	0.1144	0.088	0.088	0.2853	0.1127	0.1127	0.2763	0.2763	...	True	0	1	0	0	1	0	0	1	...
6-40000	...	...	...	...	...	...	...	...	...	NA	...	...	...	...	...	...	...	...	...
Fam1_R	0.2482	0.2853	0.1285	0.0739	0.1109	0.1109	0.0616	0.2763	...	Random	0	0	0	0	1	0	0	0	...
Fam2_R	0.0739	0.2482	0.2482	0.2853	0.2763	0.0335	0.2763	0.0616	...	Random	0	1	0	0	0	0	0	0	...
Fam3_R	0.0739	0.2482	0.088	0.2482	0.2763	0.2763	0.2095	0.2095	...	Random	0	0	0	0	1	0	0	1	...
Fam4_R	0.2853	0.088	0.2482	0.2853	0.2763	0.2763	0.2095	0.1056	...	Random	0	0	1	0	1	0	0	0	...
Fam5_R	0.1144	0.2482	0.088	0.2853	0.1127	0.2095	0.2763	0.2763	...	Random	0	0	0	0	0	0	0	1	...
40006...	...	...	...	...	...	...	...	...	...	NA	...	...	...	...	...	...	...	...	...

* Las cabeceras marcadas en negrita no forman parte de la tabla

El conjunto modificado tiene una entrada por familia (es por tanto más reducido que el conjunto de datos original, 1 a 20). por contra, se duplican las variables (columnas), y se duplica el set completo al incorporar entradas para relación cierta y entradas para una relación del individuo cuestionado con un individuo al azar (Prandom). Para los cuatro alelos (dos de cada individuo, en el ejemplo P y H3), hay una columna por cada uno con la frecuencia en población de dicho alelo (variables (columnas) V1 a V84). A su vez, hay otras 4 columnas con la comparación de cada alelo de un individuo con los dos del otro individuo con valor 1 o 0 según coincidan o no (variables C1 a C84). Adicionalmente se ha introducido una columna (85) con la variable `cat_train` que tendrá el valor `True` o `Random`, según corresponda a la relación cierta o al azar (las primera mitad `True` y la segunda mitad `Random`). En el caso del conjunto de test, no se ha incluido esta última variable. Se tienen por tanto 40000 entradas `True` y 40000 entradas `Random`, para es conjunto de entrenamiento y otras tantas para el conjunto de test por cada relación familiar a investigar.

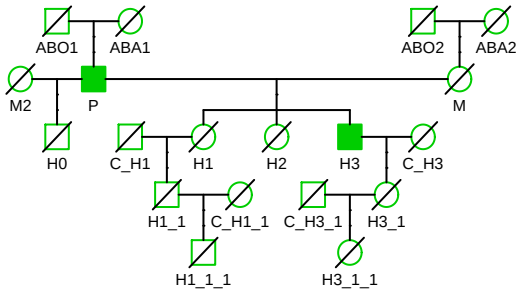
Anexo 2 Detalle de los arboles familiares según el tipo de relación

Familiares analizados en cada relación familiar.

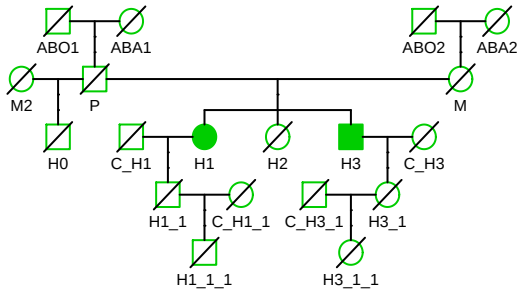
Pat_Duo	FullSib	HalfSib	Nieto	Bisnieto	Tataranieto	PrimoHer	Primo2	Tio	Tioabuelo
P	H3	H3	ABO1	ABO1	ABO1	H1_1	H1_1_1	H1	H1
H3	H1	H0	H3	H3_1	H3_1_1	H3_1	H3_1_1	H3_1	H3_1_1

Representación para cada estructura familiar seleccionada.

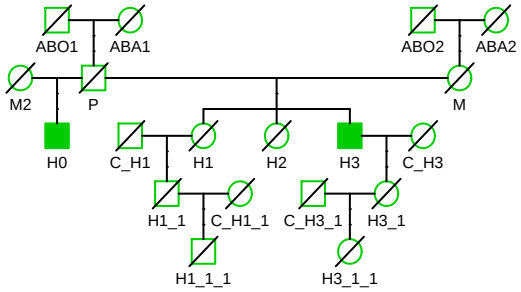
Pat_Duo



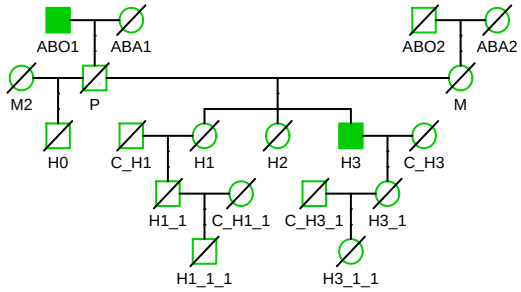
FullSib



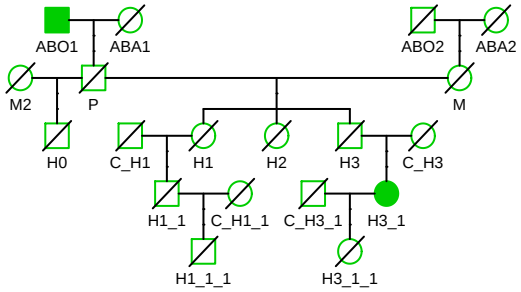
HalfSib



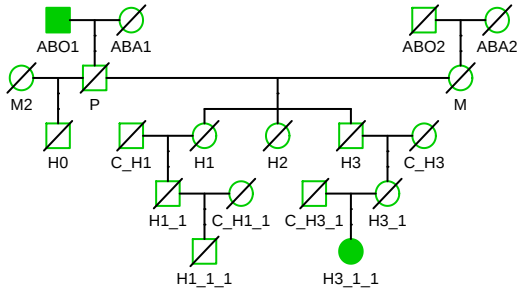
Nieto



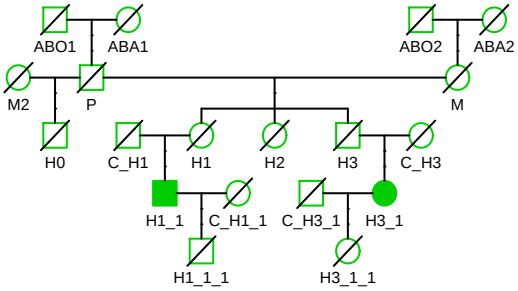
Bisnieto



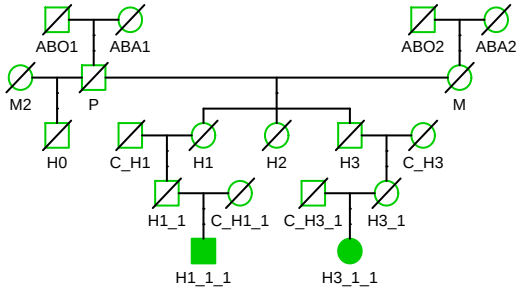
Tataranieto



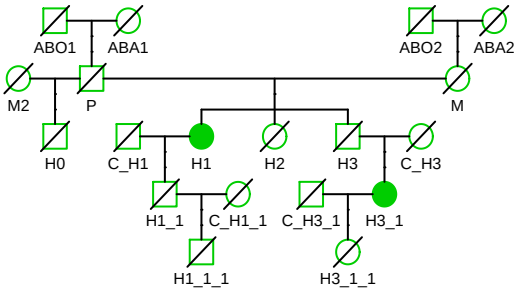
PrimoHer



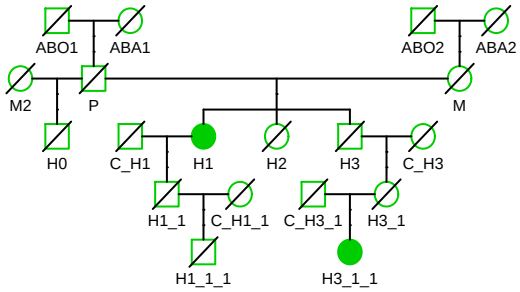
Primo2



Tio



Tioabuelo



Anexo 3. Creación de los pedigrís

Creación del árbol genético y de las funciones para completar los genotipos del mismo:

```
require(Familias)
require(readxl)
```

Se crean la funciones para trasmitir alelos entre generaciones

```
alelo_aleatorio <- function(alelos) {
    # alelos es lista de frecuencias con nombres
    sample(names(alelos), 1, prob = alelos)
}

alelo_mendel <- function(alelos) {
    sample(alelos, 1)
}
# alelos es vector de dos alelos del progenitor

genotipo_aleatorio <- function(alelos) {
    # alelos es lista de frecuencias con nombres
    sample(names(alelos), 2, TRUE, alelos)
}

# Se guardan los objetos utilizados para recuperarlos más adelante
saveRDS(alelo_aleatorio, "./data/falelo_aleatorio.rds", ascii = TRUE, compress = FALSE)
saveRDS(alelo_mendel, "./data/falelo_mendel.rds", ascii = TRUE, compress = FALSE)
saveRDS(genotipo_aleatorio, "./data/fgenotipo_aleatorio.rds", ascii = TRUE,
    compress = FALSE)
```

Se crea la función que puebla un árbol familiar usando la tabla de frecuencias

```
poblarped <- function(frec, arbol) {
    # Función para poblar una estructura familiar con todos los marcadores
    # dela tabla de frecuencias Se verifica que frec es FamiliasLocus y arbol
    # es FamiliasPedigree
    if ((class(arbol) == "FamiliasPedigree") & (class(frec[[1]]) == "FamiliasLocus")) {
        nmarcadores <- length(frec)
        nindividuos <- length(arbol$id)
        genomatriz <- matrix(NA, nindividuos, nmarcadores * 2)
        control <- vector(length = nindividuos)
        colnames(genomatriz) <- paste(rep(names(frec), each = 2), c("1",
            "2"), sep = ".")
        n <- 0
        # se recorren todos los individuos buscando los que no tienen padre ni
        # madre
        for (i in 1:nindividuos) {
            if (arbol$findex[i] + arbol$mindex[i] == 0) {
                # Asigna pedigri random
                for (m in 1:nmarcadores) {
                  genomatriz[i, (m * 2 - 1):(m * 2)] <- genotipo_aleatorio(frec[[m]][[2]])
                }
                # se marca como completo
```

```
                control[i] <- TRUE
            }
        }
        # se completa el resto de individuos hasta que todos estén completos
        while (sum(control) < nindividuos) {
            n <- n + 1
            for (i in which(control == FALSE)) {
                # Se van seleccionandoS los que tienen los dos padres completos (o uno
                # random y el otro completo)
                if ((control[arbol$findex[i]] | arbol$findex[i] == 0) & (control[arbol$mindex[i]] |
                  arbol$mindex[i] == 0) == TRUE) {
                  # Asigna cada alelo random o mendel segun padres
                  for (m in 1:nmarcadores) {
                    # alelo paterno
                    genomatriz[i, (m * 2 - 1)] <- ifelse(arbol$findex[i] ==
                      0, alelo_aleatorio(frec[[m]][[2]]), alelo_mendel(genomatriz[arbol$findex[i],
                      (m * 2 - 1):(m * 2)]))
                    # alelo materno
                    genomatriz[i, (m * 2)] <- ifelse(arbol$mindex[i] == 0,
                      alelo_aleatorio(frec[[m]][[2]]), alelo_mendel(genomatriz[arbol$mindex[i],
                        (m * 2 - 1):(m * 2)]))
                  }
                }
                # se marca como completo
                control[i] <- TRUE
            }

            if (n > nindividuos^2)
                return("ERROR: No se puede completar la familia")
            ######### mejorar finalizar bucle con error#######################
        }
        return(genomatriz)
    }
    err <- ""
    err <- paste(quote(arbol), ifelse(class(arbol) != "FamiliasPedigree",
        "no es un objeto FamiliasPedigree", ""))
    err <- paste(err, "\\", quote(frec), ifelse(class(frec[[1]]) != "FamiliasLocus",
        "no es un objeto FamiliasLocus", ""))
    return("ERROR: " & err)  ######### mejorar finalizar bucle con error#######################
}

saveRDS(poblarped, "./data/fpoblarped.rds", ascii = TRUE, compress = FALSE)
```

Se importan las frecuencias como matriz

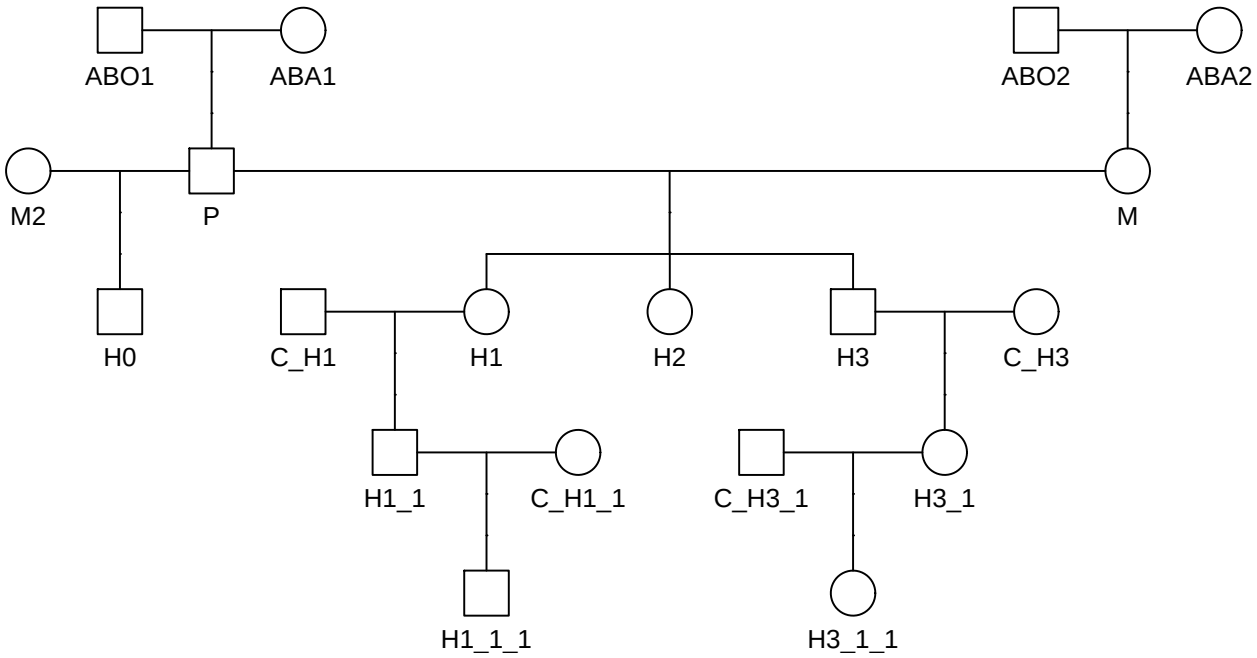
```
library(readxl)
ES_Freq <- as.matrix(read_excel("./input/PIIS1872497313001774.xls", col_names = TRUE,
    col_types = "numeric", skip = 2, n_max = 65))
rownames(ES_Freq) <- ES_Freq[, 1]

# Se ajustan las frecuencias para que sumen 1 (faltan restos por redondeo
# que se añaden al mayoritario)
ajuste <- 1 - colSums(ES_Freq[, -1], na.rm = TRUE)
for (i in 2:dim(ES_Freq)[2]) {
```

```
    a <- which.max(ES_Freq[, i])
    ES_Freq[a, i] <- ES_Freq[a, i] + ajuste[i - 1]
}
# Se convierte a FamiliasLocus
frec_list <- apply(ES_Freq[, -1], 2, function(i) {
    FamiliasLocus(na.omit(i))
})
# Se añade el nombre de cada marcador
for (i in 1:length(frec_list)) {
    frec_list[[i]]$locusname <- names(frec_list[i])
}

saveRDS(frec_list, "./data/frec_list.rds", ascii = TRUE, compress = FALSE)
```

```
# creamos una matriz con todos los indivíduos del árbol.
arbol <- matrix(c("ABO1", NA, NA, "male", "ABA1", NA, NA, "female", "ABO2",
    NA, NA, "male", "ABA2", NA, NA, "female", "P", "ABO1", "ABA1", "male",
    "M", "ABO2", "ABA2", "female", "H1", "P", "M", "female", "H2", "P", "M",
    "female", "H3", "P", "M", "male", "C_H1", NA, NA, "male", "C_H3", NA,
    NA, "female", "H1_1", "C_H1", "H1", "male", "H3_1", "H3", "C_H3", "female",
    "C_H1_1", NA, NA, "female", "C_H3_1", NA, NA, "male", "H1_1_1", "H1_1",
    "C_H1_1", "male", "H3_1_1", "C_H3_1", "H3_1", "female", "M2", NA, NA,
    "female", "HO", "P", "M2", "male", "Prandom", NA, NA, "male"), 20, 4,
    byrow = T)
# Se convierte a FamiliasPedigree
ped <- Familias::FamiliasPedigree(arbol[, 1], arbol[, 2], arbol[, 3], arbol[,
    4])

# Sw guardan los objetos utilizados para recuperarlos más adelante
saveRDS(ped, "./data/ped.rds", ascii = TRUE, compress = FALSE)
```

Se muestra el árbol generado para ver que todo es correcto

```
plot(ped, symbolsize = 1, cex = 0.8, branch = 1, packed = FALSE, angle = 0,
    align = c(8, 0), pconnect = 0.5)
```




```
## Did not plot the following people: Prandom
```

Se prueban las funciones para trasmitir alelos

```
set.seed(112358)
alelo_aleatorio(frec_list[[1]][[2]])
```

```
## [1] "14"
```

```
alelo_mendel(c("17", "18"))
```

```
## [1] "18"
```

```
genotipo_aleatorio(frec_list[[1]][[2]])
```

```
## [1] "10" "15"
```

Ya se puede crear la tabla de alelos

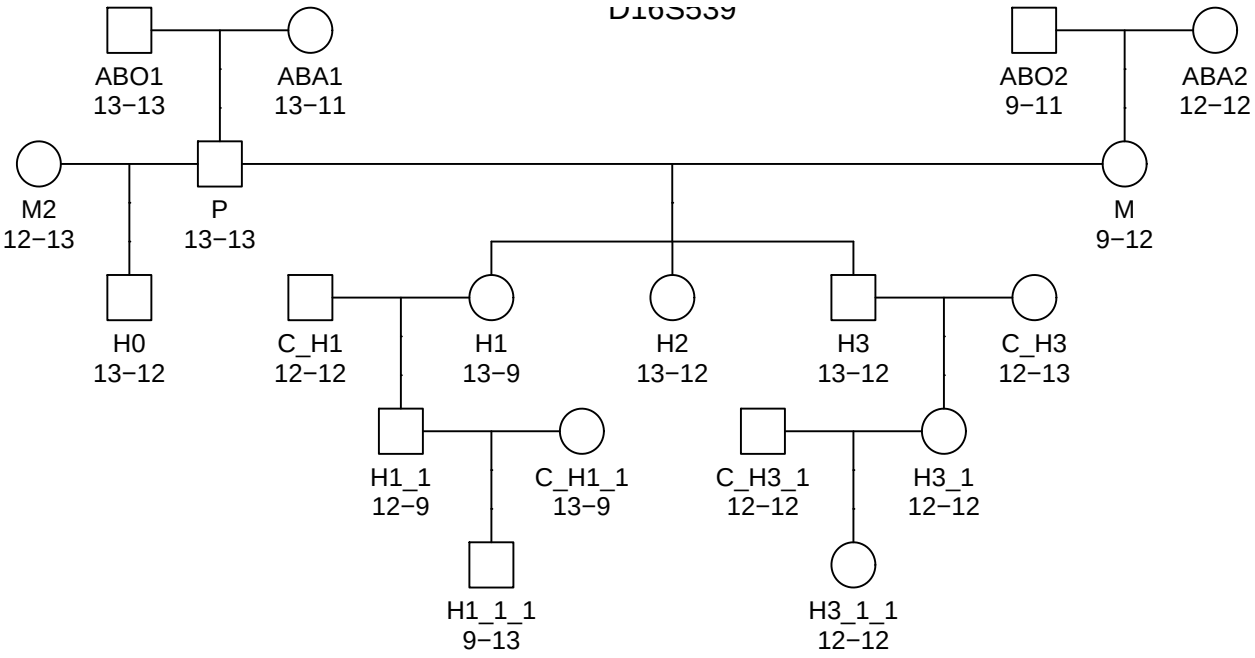
```
set.seed(112358)
j <- poblarped(frec_list, ped)
```

Se genera un árbol con los genotipos uno de los marcadores para ver que es correcto

```
n <- 8  #cambiar el número de marcador para ver otro
plot(ped, id = paste(ped$id, paste(j[, n * 2 - 1], j[, n * 2], sep = "-"),
    sep = "\n"), symbolsize = 1, cex = 0.8, branch = 1, packed = FALSE, angle = 0,
    align = c(8, 0), pconnect = 0.5, main = names(frec_list[n]))
```

```
## Did not plot the following people: Prandom
```

```
text(3.5, 1, names(frec_list[n]))
```




```
# guarda los objetos utilizados para recuperarlos más adelante
ll <- list(ped, frec_list, poblarped, alelo_aleatorio, alelo_mendel, genotipo_aleatorio)
save(ll, "./data/tfm_objetos.rds")
```

Creación de los archivos de genotipos de entrenamiento y test (40000 familias completas cada uno).

```
require(Familias)
```

```
# Importar funciones, frecuencias y estructura familiar archivadas
# Cambiar nombres de archivos .rds en parámetros # para otros conjuntos
# de datos de estructura familiar y frecuencias#
ped <- readRDS(params$ped)  #Estructura familiar
frec_list <- readRDS(params$frec_list)  #Frecuencias
poblarped <- readRDS("./data/fpoblarped.rds")
genotipo_aleatorio <- readRDS("./data/fgenotipo_aleatorio.rds")
alelo_aleatorio <- readRDS("./data/falelo_aleatorio.rds")
alelo_mendel <- readRDS("./data/falelo_mendel.rds")
```

```
IP <- function(pedigri, relatives, victim, Prandom, freq, profiles) {
    # Función para calcular el Índice de Parentesco entre los familiares y la
    # víctima Debe haber un individuo no relacionado en el pedigri (con
    # progenitores NA) que será Prandom

    # Se busca las posiciones de la víctima y el individuo alternativo no
    # relacionado
    unk <- which(pedigri$id == Prandom)
    vv <- which(pedigri$id == victim)
    pedigri$sex[unk] <- pedigri$sex[vv]
    # Se crea un pedigrí alternativo intercambiando a la víctima
    ped0 <- pedigri
    ped0$id[unk] <- victim
```

```
    ped0$id[vv] <- Prandom
    # Se seleccionan únicamente los genotipos de los implicados
    datos <- profiles[pedigri$id %in% c(relatives, victim), ]
    # return(datos)
    ff <- FamiliasPosterior(list(pedigri, ped0), freq, datos, ref = 2)$LR[1]
    return(ff)
}

# IP(ped, c('M', 'H3'), 'P', 'Prandom', frec_list, datos)
```

```
extrae <- function(datobruto, n_ped, ped_id) {
    # Función para extraer los genotipos de la familia con número de orden
    # n_ped datobruto es data.frame con los genotipos de todas las familias
    # ped_id es el listado de individuos de la familia (generalmente se
    # obtiene del vector id del pedigrí usado para crear dato bruto)
    p <- length(ped_id)
    n <- n_ped
    a <- (n - 1) * p + 1
    b <- a + p - 1
    datos <- datobruto[a:b, -1]
    rownames(datos) <- ped_id
    return(datos)
}
# extrae(fff,3,ped$id)
```

```
saveRDS(extrae, "./data/fextrae.rds", ascii = TRUE, compress = FALSE)
```

```
arbol_plot <- function(pedigri, relatives, victim, Prandom) {
    # Función para generar un plot del arbol genético
    nn <- length(pedigri$id)
    unk <- which(pedigri$id == Prandom)
    vv <- which(pedigri$id == victim)
    rr <- which(pedigri$id %in% relatives)
    personas <- c(rr, vv)
    ped0 <- pedigri
    ped0$id[unk] <- victim
    ped0$id[vv] <- "Unknown"
    par(mfrow = c(1, 2))
    relleno <- rep(1, nn)
    relleno[personas] <- 2
    tachado <- rep(1, nn)
    tachado[personas] <- 0
    pedigri$id[vv] <- paste(pedigri$id[vv], "=\nVíctima")
    ped0$id[vv] <- paste(ped0$id[unk], "=\nUnknown")
    plot(pedigri, cex = 0.8, affected = relleno, status = tachado, branch = 1,
        packed = FALSE, angle = 0, align = c(8, 0), pconnect = 0.5, col = 3)
    title(paste("La víctima es familiar de", paste(relatives, collapse = ", ")))
    tachado[vv] <- 1
    plot(ped0, cex = 0.8, affected = relleno, status = tachado, branch = 1,
        packed = FALSE, angle = 0, align = c(8, 0), pconnect = 0.5, col = 3)
    title(paste(victim, "no es la víctima"))
}
# arbol_plot(ped, c('M','H2'), 'P', 'Prandom')
```

```r
arbol_plot_alt <- function(pedigri, relatives, victim, Prandom) {
    # Función para crear el arbol genético de la hipótesis alternativa
    nn <- length(pedigri$id)
    unk <- which(pedigri$id == Prandom)
    vv <- which(pedigri$id == victim)
    rr <- which(pedigri$id %in% relatives)
    personas <- c(rr, vv)
    ped0 <- pedigri
    # pedigri$id[unk] <- victim pedigri$id[vv] <- 'Random'
    par(mfrow = c(1, 2))
    relleno <- rep(1, nn)
    relleno[personas] <- 2
    tachado <- rep(1, nn)
    tachado[personas] <- 0
    ped$id[vv] <- paste(ped$id[vv], "=\nRandom")
    ped0$id[vv] <- paste(pedigri$id[vv], "=\nUnknown")
    plot(pedigri, cex = 0.8, affected = relleno, status = tachado, branch = 1,
        packed = FALSE, angle = 0, align = c(8, 0), pconnect = 0.5, col = 3)
    title(paste("Familiar de", paste(relatives, collapse = ", "), "individuo al azar"))
    tachado[vv] <- 1
    plot(ped0, cex = 0.8, affected = relleno, status = tachado, branch = 1,
        packed = FALSE, angle = 0, align = c(8, 0), pconnect = 0.5, col = 3)
    title(paste(victim, "no es conocido"))
    ped
    ped0
}
# arbol_plot(ped, c('M','H2'), 'P', 'Prandom')
```

```r
# lista con las relaciones básicas
parentage <- list("Pat_Trio", "Pat_Duo", "Abuelos")
relatives <- list(c("P", "M"), "P", c("ABO1", "ABA1"))
hijo <- list("H3", "H3", "H3")
# random <- rep(list('Prandom'),4)
rel_basicas <- list(parentage = parentage, relatives = relatives, victim = hijo)
```

```r
# lista con las relaciones de varios hermanos
parentage <- list("FullSib_2", "HalfSib_2", "HalfSib_3", "FullSib_2_M", "HalfSib_3_M")
relatives <- list(c("H1", "H2"), c("H1", "H2"), c("H1", "H2", "H3"), c("H1",
    "H2", "M"), c("H1", "H2", "H3", "M"))
victim <- list("H3", "HO", "HO", "H3", "HO")
# random <- rep(list('Prandom'),5)
rel_multihermanos <- list(parentage = parentage, relatives = relatives, victim = victim)
```

```r
# lista con las relaciones de dos individuos (Pat_Duo en relaciones
# básicas)
parentage <- list("FullSib", "HalfSib", "Nieto", "Bisnieto", "Tataranieto",
    "PrimoHer", "Primo2", "Tio", "Tioabuelo")
relatives <- list("H3", "H3", "ABO1", "ABO1", "ABO1", "H1_1", "H1_1_1", "H1",
    "H1")
victim <- list("H1", "HO", "H3", "H3_1", "H3_1_1", "H3_1", "H3_1_1", "H3_1",
    "H3_1_1")
# random <- rep(list('Prandom'),9)
rel_binarias <- list(parentage = parentage, relatives = relatives, victim = victim)
# añadir Pat_Duo
```

Se generan los ficheros con los genotipos para múltiples familias

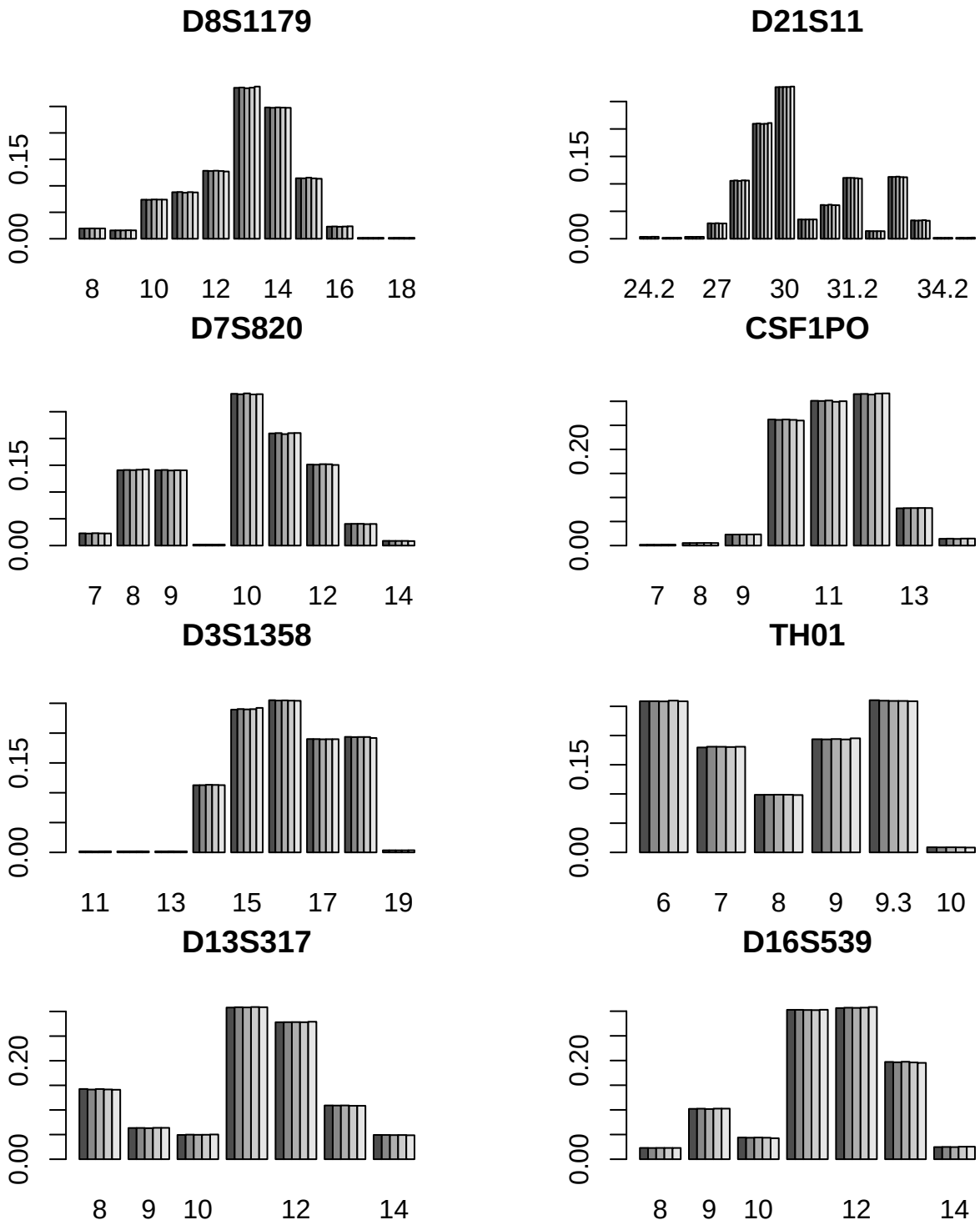
```
fichtr <- "./output/genotypes_train.txt"
# l <- 1000 #Descomentar línea para hacer pruebas
l <- 40000  #Comentar línea para hacer pruebas
# Se fija una semilla para obtener resultados reproducibles
set.seed(112358)
j <- poblarped(frec_list, ped)
# Se genera la primera familia para poblar las cabeceras del fichero
write.table(t(c("Id", colnames(j))), fichtr, sep = ",", append = FALSE, row.names = FALSE,
    col.names = FALSE)
write.table(j, fichtr, sep = ",", append = TRUE, col.names = FALSE, row.names = TRUE)
# Se generan el resto de familias
for (i in 1:(l - 1)) {
    # set.seed(112358) #no activar porque generaría el primer registo igual
    # que el creado anteriormente
    j <- poblarped(frec_list, ped)
    write.table(j, fichtr, sep = ",", append = TRUE, col.names = FALSE, row.names = TRUE)
}
# Se hace lo mismo para el conjunto de test
fichte <- "./output/genotypes_test.txt"
write.table(t(c("Id", colnames(j))), fichte, sep = ",", append = FALSE, row.names = FALSE,
    col.names = FALSE)
for (i in 1:l) {
    # set.seed(112358) #no activar porque generaría un conjunto de test igual
    # que train
    j <- poblarped(frec_list, ped)
    write.table(j, fichte, sep = ",", append = TRUE, col.names = FALSE, row.names = TRUE)
}

# rm(poblarped, genotipo_aleatorio, alelo_aleatorio, alelo_mendel)
```

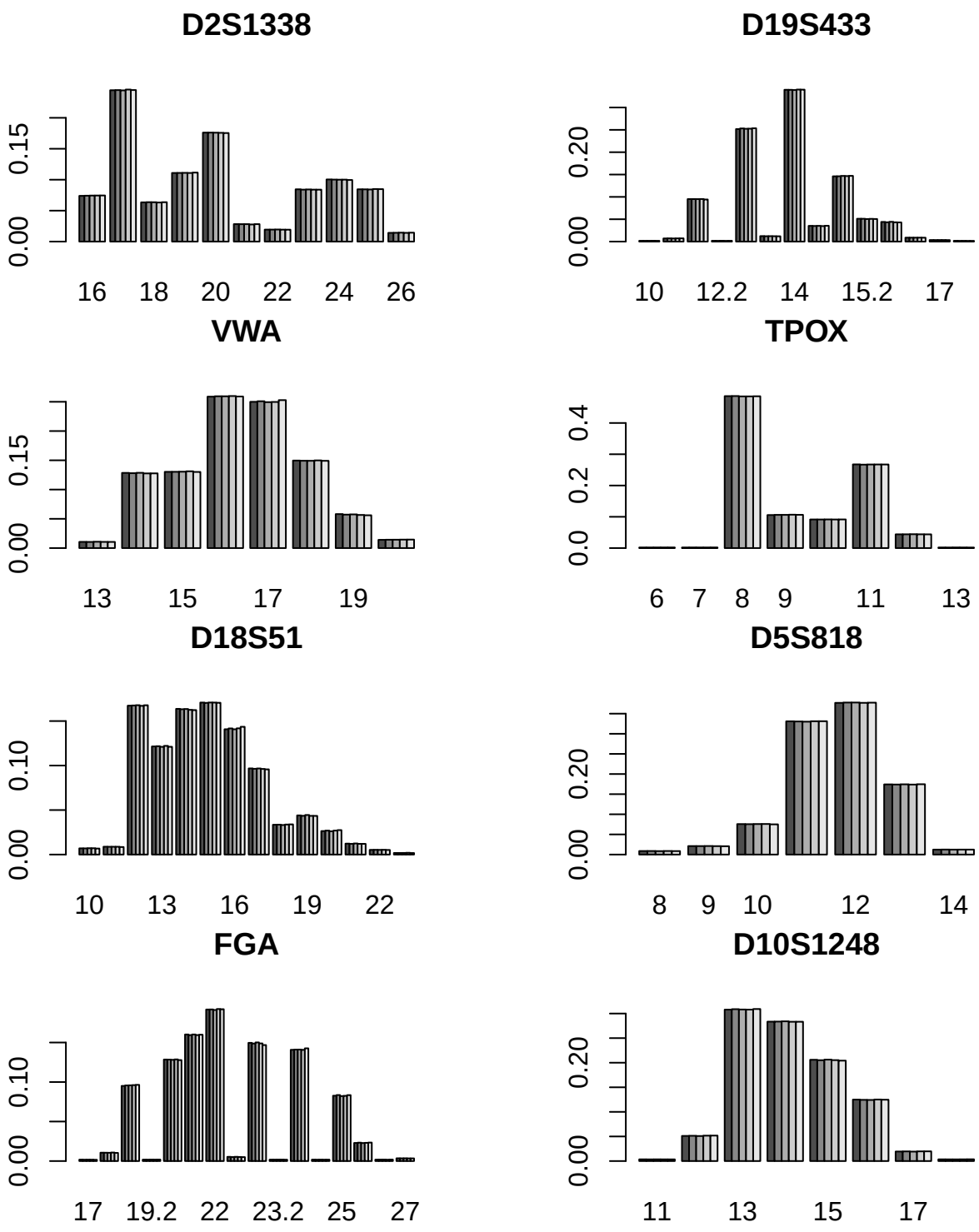
```
# IP para identificaciones
IPv <- function(pedigri, relatives, victim, freq, profiles) {
    # Función para calcular el Índice de Parentesco entre los familiares y
    # una víctima desaparecida No válido si hay relaciones indubitadas en la
    # víctima

    # Se busca las posiciones de la victima
    vv <- which(pedigri$id == victim)
    # Se crea un pedigrí alternativo eliminando las relaciones paternas
    ped0 <- pedigri
    ped0$mindex[vv] <- 0
    ped0$findex[vv] <- 0
    # Se seleccionan únicamente los genotipos de los implicados
    datos <- profiles[pedigri$id %in% c(relatives, victim), ]
    # return(datos)
    ff <- FamiliasPosterior(list(pedigri, ped0), freq, datos, ref = 2)$LR[1]
    return(ff)
}

# IPv(ped, c('M', 'H3'), 'P', frec_list, datos)
```

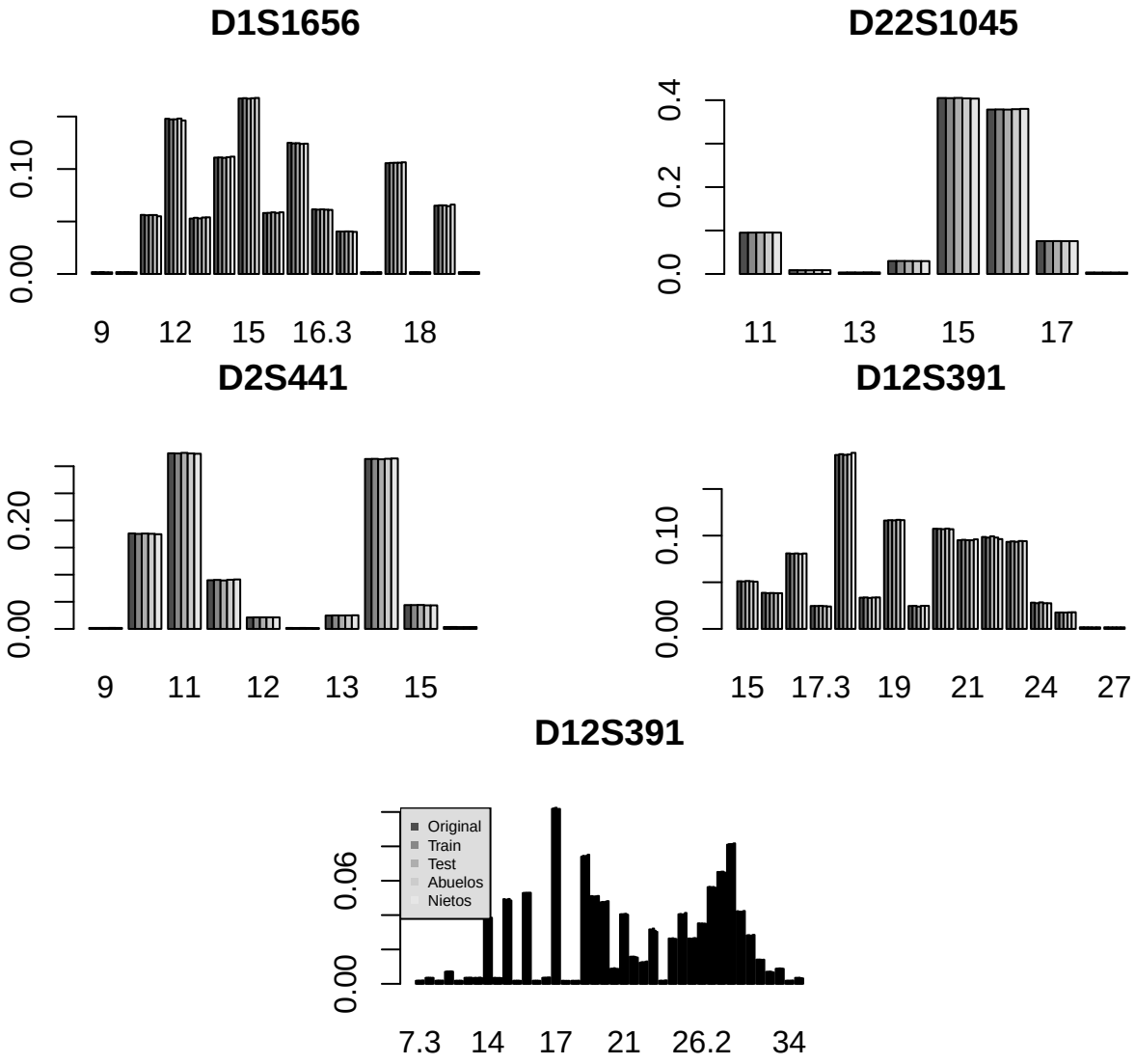
```
combo <- mapply(c, rel_basicas, rel_multihermanos, rel_binarias, SIMPLIFY = FALSE)
saveRDS(combo, "./data/relaciones_familiares.rds", ascii = TRUE, compress = FALSE)
```

```
# Generación de IP para todas las combinaciones
for (t in c("train", "test")) {
    # Se recupera la tabla de genotipos de todas las familias
    f_in <- paste("./output/genotypes_", t, ".txt", sep = "")
    geno <- read.csv(f_in, sep = ",")
    # Se harán tantas familias como haya en el fichero
    nfam <- dim(geno)[1]/length(ped$id)
    tiempo <- NA
    # Se realiza para todas las combinaciones de combo Restringir a menos si
    # se desea hacerlo por lotes
    for (rr in 1:length(combo[[1]])) {
        # Para cada relación se seleccionan los familiares y la persona problema
        relatives <- combo$relatives[[rr]]
        victim <- combo$victim[[rr]]
        # nfam <- 1 #Descomentar par hacer pruebas con una sola familia Se crea
        # una matriz con cuatro columnas Se recoge el índice de la familia, el
        # parentesco y el IP para la relación cierta y para un individuo no
        # relacionado al azar
        ip_data <- matrix(NA, nfam, 4)
        colnames(ip_data) <- c("nfam", "parentage", "IP", "IPrandom")
        # Se hace un control de tiempo empleado
        inicio <- proc.time()
        for (i in 1:nfam) {
            # Usando la función extrae se recuperan los alelos de la familia
            alelos <- extrae(geno, i, ped$id)
            ip_data[i, 1] <- i
            ip_data[i, 2] <- combo$parentage[[rr]]
            # Se calcula el IP del pedrigrí verdadero (columna 3) y del alternativo,
            # random (columna 4)
            ip_data[i, 3] <- IPv(ped, relatives, victim, frec_list, alelos)
            # Para identificaciones, se sutituye el genotipo de la persona
            # cuestionada por un individuo al azar
            alelos[which(ped$id == victim), ] <- alelos[which(ped$id == "Prandom"),
                ]
            ip_data[i, 4] <- IPv(ped, relatives, victim, frec_list, alelos)
        }
        tiempo <- paste(tiempo, (((proc.time() - inicio)/60)[3]))
        # Se guardan los resultados en un objeto R en texto plano por si se
        # quiere usar en otra aplicación
        f_out <- paste("./output/ip_", t, "_", combo$parentage[[rr]], ".rds",
            sep = "")
        saveRDS(ip_data, f_out, ascii = TRUE, compress = FALSE)
    }
}

# rm(ip_data, geno, combo)
```

Anexo 4. Distribución de alelos (HWE)




D2S1338
0.00
0.15
16 18 20 22 24 26
D19S433
0.00
0.20
10 12.2 14 15.2 17
VWA
0.00
0.15
13 15 17 19
TPOX
0.0
0.2
0.4
6 7 8 9 11 13
D18S51
0.00
0.10
10 13 16 19 22
D5S818
0.00
0.20
8 9 10 12 14
FGA
0.00
0.10
17 19.2 22 23.2 25 27
D10S1248
0.00
0.20
11 13 15 17


D1S1656
0.00
0.10
9
12
15
16.3
18
D22S1045
0.0
0.2
0.4
11
13
15
17
D2S441
0.00
0.20
9
11
12
13
15
D12S391
0.00
0.10
15
17.3
19
21
24
27
D12S391
Original
Train
Test
Abuelos
Nietos
0.00
0.06
7.3
14
17
21
26.2
34

Anexo 5. Instalación de keras

La documentación oficial de keras.rstudio.com indica realizar los siguientes pasos con la última versión que está en Git Hub:

```
devtools::install_github("rstudio/keras")
library(keras)
install_keras()
```

Estas instrucciones instalan también tensorflow y un entorno virtual de Python

```
# modelo
model <- keras_model_sequential()
model %>% layer_dense(units = 5, activation = "relu", input_shape = c(168)) %>%
    layer_dropout(rate = 0.4) %>% # layer_dense(units = 128, activation = 'relu') %>% layer_dropout(rat
# 0.3) %>%
layer_dense(units = 1, activation = "softmax")
```

```
# compilar
model %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(),
    metrics = c("accuracy"))
```

Tras estas instrucciones, se produce un error y un reinicio de R.

Al parecer, según la referencia https://github.com/tensorflow/tensorflow/issues/17411, el problema se produce porque la instalación por defecto instala la última versión disponible de TensorFlow (1.10.0), que es incompatible con la implementación actual del paquete (viene ocurriendo desde la versión 1.6.0.)

Con un par de instrucciones en la terminal, desinstalamos TensorFlow e instalamos una versión compatible:

```
~$ ./.virtualenvs/r-tensorflow/bin/pip uninstall tensorflow

~$ ./.virtualenvs/r-tensorflow/bin/pip install tensorflow==1.5
```

No obstante se puede proceder desde el principio con la instalación de la versión adecuada de tensorflow.

```
# Primero instalar tensorflow desde de CRAN:
install.packages("tensorflow")
# Después instalar TensorFlow con la versión adecuada
library(tensorflow)
install_tensorflow(method = "virtualenv", version = "1.5")
```

Verificar que ha ido bien:

```
sess = tf$Session()
hello <- tf$constant("Hello, TensorFlow!")
sess$run(hello)
```

```
## [1] "Hello, TensorFlow!"
```

Por último instalar el paquete `keras` de Cran

```
# ejecutar sólo una vez
install.packages("keras")
library(keras)
```

```
model <- keras_model_sequential()
model %>% layer_dense(units = 5, activation = "relu", input_shape = c(168)) %>%
    layer_dropout(rate = 0.4) %>% # layer_dense(units = 128, activation = 'relu') %>% layer_dropout(rat
# 0.3) %>%
```

```
layer_dense(units = 1, activation = "softmax")
```

```
model %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(),
    metrics = c("accuracy"))
```

Ahora ya no produce error.

También se puede instalar el paquete `kerasR` que proporciona una interfaz R para keras.

```
library(kerasR)
```

Los comandos son similares en contenido pero con una semántica diferente.

```
modelo <- Sequential()
modelo$add(Dense(units = 5, activation = "relu", input_shape = 168))
modelo$add(Dense(units = 10, activation = "softmax"))
```

```
keras_compile(modelo, loss = "categorical_crossentropy", metrics = c("categorical_accuracy"),
    optimizer = RMSprop())
```

Anexo 6. Entrenamiento de keras con múltiple parámetros

```
library(tensorflow)
library(keras)
library(caret)

combo <- readRDS("./data/relaciones_familiares.rds")
categorias <- c(unlist(combo$parentage[])[c(2, 9:17)], "Random")
testset <- readRDS(paste("./output/multi_test.rds", sep = ""))
trainset <- readRDS(paste("./output/multi_train.rds", sep = ""))
set.seed(112358)
trainset <- trainset[sample(nrow(trainset)), ]
testset <- testset[sample(nrow(testset)), ]
cat_train <- to_categorical(factor(trainset[, length(trainset)], levels = categorias,
    labels = 10:0), length(categorias))
trainset <- as.matrix(trainset[, -length(trainset)])
cat_test <- factor(testset[, length(testset)], labels = categorias)
testset <- as.matrix(testset[, -length(testset)])
```

```
cmlist <- list()
evaldf <- data.frame(condiciones = NA, loss = NA, categorical_accuracy = NA)
activa <- c("relu", "hard_sigmoid", "linear")
# nodo1 <- c(21, 42, 84, 168)
nodo1 <- c(84, 168)
# nodo2 <- c(21, 42, 84, 168)
nodo2 <- c(84, 168)
# drop1 <- c(0, 0.1, 0.2, 0.3)
drop1 <- c(0, 0.1)
# drop2 <- c(0, 0.1, 0.2, 0.3)
drop2 <- c(0, 0.1)
opt <- c(optimizer_rmsprop(), optimizer_adam(), optimizer_nadam(), optimizer_sgd())
nfam <- 11000
i <- 0
for (act in activa) {
    for (n1 in nodo1) {
        for (n2 in nodo2) {
            for (d1 in drop1) {
                for (d2 in drop2) {
                  for (o in opt) {
                    cc <- paste(act, substring(o, 19, 21), n1, n2, d1, d2,
                      sep = "_")
                    i <- i + 1
                    inicio <- proc.time()
                    # modelo
                    modelo <- keras_model_sequential()
                    modelo %>% layer_dense(units = n1, activation = act,
                      input_shape = c(168), kernel_regularizer = regularizer_l2(l = 0.001)) %>%
                      layer_dropout(rate = d1) %>% layer_dense(units = n2,
                      activation = act, kernel_regularizer = regularizer_l2(l = 0.001)) %>%
                      layer_dropout(rate = d2) %>% layer_dense(units = length(categorias),
                      activation = "softmax")
                    # compilar
                    modelo %>% compile(loss = "categorical_crossentropy",
```

```
                    optimizer = o, metrics = c("categorical_accuracy"))
                  # entrenar
                  set.seed(112358)
                  entreno <- modelo %>% fit(trainset[1:nfam, ], cat_train[1:nfam,
                    ], epochs = 30, batch_size = 128, validation_split = 0.2)
                  plot(entreno)
                  ggsave(paste("./fig/traink2r_", cc, ".png", sep = ""))
                  # evaluar
                  set.seed(112358)
                  evaldf[i, 1] <- cc
                  evaldf[i, 2:3] <- modelo %>% evaluate(trainset[1:nfam,
                    ], cat_train[1:nfam, ])
                  # predecir
                  set.seed(112358)
                  cat_keras <- modelo %>% predict_classes(testset[1:nfam,
                    ])
                  # Se convierten las categorías obtenidas a factores para poder usarlos
                  # con confusionMatrix
                  cat_keras <- factor(cat_keras, levels = 10:0, labels = categorias)

                  # Se evalua el rendimiento
                  cmlist[[i]] <- list(cc, tiempo = ((proc.time() - inicio)/60)[3],
                    cm = confusionMatrix(cat_keras, cat_test[1:nfam], positive = "Random")$overall[1:4

                }
              }
            }
          }
        }
}
saveRDS(cmlist, "./output/3a3_2apasada_reg_cm.rds")
saveRDS(evaldf, "./output/3a3_2apasada_reg_eval.rds", sep = "")
```