Aarón Murillo Lort

PREDICCIÓN DE GÉNERO EN MENSAJES DE TEXTO MEDIANTE APRENDIZAJE SUPERVISADO

TRABAJO DE FIN DE GRADO

dirigido por Antonio Moreno Ribas

Grado de Ingeniería Informática

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2021

Resum.

Les relacions interpersonals estan canviant cada vegada més amb el creixement de les xarxes socials i la tecnologia mòbil. Al comunicar-nos tantíssim mitjançant missatges de text, és senzill recopilar una gran quantitat de dades per poder analitzar-les amb fins científics, polítics, lingüístics, etcètera.

En aquest projecte, he documentat el funcionament de la ciència de dades al tractar de predir el gènere dels autors de missatges de text mitjançant algorismes d'aprenentatge supervisat. Primer, he estudiat i comprès la teoria dels algorismes utilitzats. Després, he après com implementar-los amb la llibreria Scikit-learn. Finalment, he executat les proves per obtenir les prediccions i les seves precisions. Els resultats han sigut més que satisfactoris.

Aquest projecte forma part d'un més gran d'ànima lingüística. Un projecte del Damián Morales, que treballa en un doctorat amb l'objectiu de trobar un patró que sigui capaç de determinar si un missatge de text és d'un home o d'una dona.

Resumen.

Las relaciones interpersonales están cambiando cada vez más con el auge de las redes sociales y la tecnología móvil. Al comunicarse tantísimo mediante mensajes de texto, resulta sencillo recopilar una cantidad grande de datos para analizarlos con fines científicos, políticos, lingüísticos, etcétera.

En este proyecto, he documentado el funcionamiento de la ciencia de datos al tratar de predecir el género de los autores de ciertos mensajes de texto mediante algoritmos de aprendizaje supervisado. Primero, he estudiado y comprendido la teoría detrás de los algoritmos utilizados. Después, he aprendido cómo implementarlos con la librería Scikit-learn. Finalmente, he ejecutado las pruebas para obtener las predicciones y sus precisiones. Los resultados han sido más que satisfactorios.

Este proyecto forma parte de uno más grande de alma lingüística. Un proyecto de Damián Morales, cuyo doctorado trata de encontrar un patrón capaz de determinar si un mensaje de texto es de un hombre o de una mujer.

Abstract.

Interpersonal relationships are increasingly changing with the rise of social media and mobile technology. By communicating so much through text messages, it is easy to collect a large amount of data to analyse it for scientific, political, linguistic and many more goals.

In this project, I have documented how data science works by trying to predict the gender of the authors of certain text messages using supervised learning algorithms. First, I understood the theory behind them. Later, I have learned their implementation with the Scikit-learn library. Finally, I have run many tests to get the predictions and their accuracy. The results have been more than satisfactory.

Resum

This project is part of a larger linguistic one. A project by Damián Morales, whose PhD tries to find a pattern that can tell if a text message has been written by a man or a woman.

Índice

Índice de tablas

Índice de figuras

1 Introducción

Hoy en día, debido al auge tecnológico en el que nos encontramos y al auge de las redes sociales y de la comunicación a distancia (por no mencionar la situación pandémica que acabamos de vivir), una gran parte de las comunicaciones interpersonales se realiza a través de una pantalla. Hace veinte años, esto era totalmente impensable. Las personas se comunicaban por vía telemática para enviarse correos, quizás mediante el envío de SMS's, pero no era posible mantener conversaciones que durasen días o incluso semanas con todo lujo de detalles.

Sin embargo, a causa de aplicaciones como WhatsApp, Facebook, Instagram, etcétera, es muy sencillo encontrar textos que reflejen no sólo la opinión escrita de una persona sobre cualquier tema, sino textos que reflejen su comportamiento, sus creencias, sus deseos y aspiraciones. En definitiva, muchísimos fragmentos escritos que definen a una persona. Así pues, ¿Pueden, de alguna manera, reflejar nuestro sexo estos escritos? ¿Hay algún patrón infalible para determinar si el autor de un fragmento es de sexo masculino o femenino? Y, si este patrón existe, ¿Qué implicaciones podría tener? ¿Qué significaría este patrón, y qué diría del comportamiento y la psicología de los sexos, qué hecho estaría reflejando?

En este trabajo de fin de grado, nos centraremos en la parte informática de este problema. Dada la naturaleza de este, se necesitan muchísimos datos para encontrar dicho patrón. Por lo tanto, nos centraremos en el análisis de estos usando herramientas de **inteligencia artificial (IA)**, más concretamente, herramientas de **aprendizaje supervisado**[1].

En primer lugar, hablaremos detalladamente del proyecto en el que participamos, en qué consiste y qué interés tiene la diferenciación de género en textos. En segundo lugar, estudiaremos los conceptos teóricos en los que se basan las herramientas utilizadas. Estos conceptos son necesarios para poder darle sentido a los resultados que obtendremos y para conseguir un mayor entendimiento de las herramientas que usaremos. Finalmente, expondremos las pruebas realizadas y extraeremos conclusiones de ellas.

1.1 Descripción del proyecto

El proyecto en el que he tenido la suerte de participar es, en realidad, un proyecto con alma lingüística. Damián Morales es un doctorando en filología hispánica cuyo doctorado se basa en intentar observar algún patrón lingüístico que pueda determinar si un mensaje de texto en alguna red social es de autor femenino o masculino.

Este estudio puede ser de interés para diversos campos. Por un lado, puede dar indicios sobre el comportamiento y psicología de cada sexo. En la época actual, hay un amplio abanico de personalidades, comportamientos, gustos, estamos en plena era de libertad individual y eso conlleva muchísimas cosas. Puede resultar interesante entonces estudiar si, en el campo de la lengua escrita, hay alguna forma de determinar el sexo del autor. Y, si la encontramos, determinar por qué esa forma funciona y no cualquier otra.

Por otro lado, por lo que respecta a la informática, este proyecto nos puede dar un mejor entendimiento de cómo las grandes empresas analizan nuestros datos con tal de comercializar con ellos y con nuestra identidad. Se sabe sobradamente que compañías y empresas como Google, Amazon, Facebook, Twitter, entre muchas otras, ofrecen servicios

[1] En el campo de la inteligencia artificial, el aprendizaje supervisado es una técnica que permite clasificar objetos gracias a un previo entrenamiento con datos clasificados previamente.

gratuitos a cambio de información concerniente a nuestra identidad. De esta forma, al disponer de una cantidad ingente de datos, les es más sencillo clasificarnos en ciertos grupos según nuestros gustos, sexos, ideologías políticas, aficiones, etcétera, con tal de ofrecernos publicidad acorde a lo que podríamos necesitar, ganando así el capital que necesitan. Este TFG, al tratarse de un proyecto centrado en algoritmos de aprendizaje supervisado y predicción y clasificación de mensajes de texto, nos permite entender en profundidad cómo, a partir de datos como qué leemos, qué escribimos en nuestras redes y qué buscamos, cualquiera puede saber todo sobre nosotros y clasificarnos de manera automática.

Por lo tanto, el doctorando Morales ha estado trabajando, en primer lugar, en el análisis de decenas de miles de mensajes de texto extraídos de redes sociales y, en segundo lugar, en el estudio de los datos y evidencias extraídas a partir del análisis.

1.2 Análisis de mensajes de texto

Si se quiere encontrar algún tipo de patrón o estructura que pueda dar pistas sobre el sexo del autor, es lógico pensar que el primer paso a efectuar es estudiar cada uno de los mensajes con tal de extraer toda la información posible.

Seguidamente, esta información debe ser correctamente estructurada y organizada con tal de asegurar su legibilidad y facilitar su uso. Como se nos ha enseñado a lo largo de los cuatro años que dura el grado de ingeniería informática, siempre que se deban tratar de alguna manera los datos, es de vital importancia que estos estén estructurados de tal forma que su uso y estudio se pueda hacer de forma sencilla. De esta manera, no se pierde tiempo en entender la estructura de los datos, no existe la necesidad de reestructurar los datos dentro del programa, no se genera overhead a causa de funciones y código extra que se podría haber evitado con otra disposición de la información… En síntesis, tanto el buen entendimiento de un programa como su rendimiento dependen en gran medida de la forma en que los datos son recibidos. Pero del código hablaremos más adelante.

En este proyecto, los mensajes se han traducido a vectores numéricos que los definen en función de ciertas **dimensiones de análisis**. Cada una de las dimensiones de análisis representa una parte de la lingüística, además de dimensiones que representan simbología más reciente como puede ser el uso de Emojis,GIF’s, entre otros. Disponemos entonces de un total de siete dimensiones de análisis:

- **Digital**: deja constancia del lenguaje digital usado en cada mensaje
- **Léxico**: determina el conjunto de palabras que se ha usado en cada mensaje, clasificando dichas palabras según la emoción, la información que transmiten o el tipo de palabra que son. Por ejemplo, dentro de esta dimensión, hay atributos[2] que representan palabras que denotan un sentimiento de Alegría, de Enojo, de Miedo, de Repulsión, Verbos auxiliares, Adjetivos de probabilidad, y un largo etcètera.

[2] En el campo del análisis de datos, un atributo es una especificación o una característica cuantificable que define una propiedad de un objeto.

- **Morfológico y morfológico simplificado**: campo lingüístico que estudia las reglas que rigen la composición y la derivación de las palabras, es decir, la propia forma de las palabras.
- **Ortográfico**: estudio de las normas de escritura en una lengua (en este caso, la lengua castellana).
- **Pragmático**: parte de la lingüística que estudia el lenguaje en función del contexto. Es la parte más humana de la lengua, en que no se estudia la forma en sí, sino su significado y sus implicaciones según el contexto en el que el mensaje ha sido escrito y los estados emocionales y/o físicos del autor y del receptor del mensaje.
- **Sintáctico**: campo de la gramática que estudia los principios que rigen la estructura y la formación de frases.

Entonces, disponemos de un total de siete conjuntos de ficheros en formato Excel, representando cada uno de estos conjuntos una de las dimensiones de análisis. En otras palabras, cada uno de los mensajes ha sido analizado y clasificado en siete campos distintos. Si el lector desea saber más sobre los atributos de cada dimensión de análisis, no dude en dirigirse al *Capítulo 6*, donde se haya el anexo.

Concerniente al sexo de los autores, quisiera recalcar el hecho de que son conjuntos de ficheros, y no ficheros individuales. Esto se debe a que no sólo se ha tenido en cuenta el sexo de los autores, sino que se ha tenido en cuenta la franja de edad de dichos autores y se han creado ficheros distintos para cada una de las franjas y sexos/géneros. Así pues, para cada dimensión de análisis, se dispone de doce ficheros que se pueden dividir en tres grupos. Por un lado, dentro de cada uno de estos, hay un fichero para los objetos[3] de entre diez y veinte años, otro para los objetos de entre veinte y treinta años, un tercero para las personas de más edad y finalmente uno que engloba todos los anteriores. Por otro lado, un grupo toma en consideración ambos sexos, y los otros dos estudia únicamente uno de los sexos. En total, disponemos de 74.901 objetos, siendo 37.448 de ellos mujeres, y el resto hombres.

1.3 Análisis de datos

La parte del análisis de datos es la parte que nos concierne en este trabajo de final de grado, la parte donde la inteligencia artificial adquiere un papel fundamental. Como se ha mencionado en el apartado anterior, hay muchísima información con la que tratar e intentar llevar a cabo este proyecto sin hacer uso de ninguna herramienta informática es prácticamente imposible. Por esto, el objetivo principal de este trabajo es la de ver si es posible predecir el sexo del autor de un mensaje de texto mediante el uso de inteligencia artificial. Más concretamente, se usarán algoritmos de aprendizaje supervisado como **árboles de decisión** y **Random Forests**, modelos de los que hablaremos en profundidad más adelante.

Entonces, este documento se centrará en cómo usar ambos algoritmos de aprendizaje supervisado para analizar los datos de los que disponemos y en el análisis de los datos en sí.

[3] En el campo del análisis de datos, un objeto es uno de los casos presentes en el conjunto de datos estudiado.

Para poder hacerlo, se han llevado a cabo ciertas tareas previas imprescindibles. En primer lugar, se ha estudiado el funcionamiento de los algoritmos, incluyendo ventajas y desventajas de cada uno. En segundo lugar, se ha analizado concienzudamente la implementación de estos modelos en la librería que usaremos. Esto requiere un profundo entendimiento no sólo de la teoría, sino de la parametrización y de los efectos que puede tener cada parámetro sobre el modelo. Finalmente, respecto al análisis de datos en sí, se han ejecutado un gran número de pruebas con tal de obtener las mejores predicciones posibles. Una vez obtenidas, las predicciones y los árboles de decisión han sido analizados para poder sacar conclusiones pertinentes para el proyecto de Damián y para este TFG.

Me gustaría mencionar que este proyecto no es único, ni mucho menos. Este análisis se ha estado haciendo durante años alrededor del mundo. Un ejemplo de ello son algunas de las competiciones internacionales que ha habido en este ámbito. Los resultados obtenidos pueden verse en el enlace [1], en los análisis "Authorship análisis" de nombre "Author Profiling" y "Bots and Gender Profiling". La web PAN es una web donde se almacenan eventos científicos y tareas llevadas a cabo en el ámbito de el estudio forense del texto digital y la estilometría. Uno de esos estudios forenses del texto digital tiene relación con la predicción del género de los autores de textos escritos en redes sociales. En PAN, podemos ver que ha habido competiciones desde el 2013 hasta el 2019. Veamos cuán precisas son algunas de las predicciones ejecutadas en estas competiciones.

En 2013, se acertó como máximo en el 38,94% de las ocasiones. En el año 2014, se acertó de media notablemente menos. No obstante, en el 2015, la media máxima de aciertos fue del 84,04%, una media parecida a la de los años 2017, 2018 y 2019. De todos los años en los que se ha realizado esta competición, la mejor media de aciertos obtenida es la del año 2017, con una media de aciertos del 85,99%. Veremos más adelante si somos capaces de conseguir esos resultados.

Para resumir, mientras que el proyecto en sí es un proyecto con objetivos lingüísticos, la metodología a seguir para poder cumplir con nuestros objetivos es esencialmente **Data Science** y algoritmos de inteligencia artificial.

1.4 Síntesis

Este trabajo de fin de grado se basa en el análisis de datos de un proyecto correspondiente al campo de la filología y la lengua. Damián Morales, doctorando en filología hispánica, tiene como objetivo encontrar la forma de determinar si un mensaje de texto ha sido escrito por un hombre o por una mujer.

Para poder encontrar tal cosa, ha reunido y descompuesto decenas de miles de mensajes de texto de hombres y mujeres de distintas edades en vectores numéricos. Así pues, a partir de un mensaje de texto, obtenemos un total de siete vectores numéricos correspondientes a distintas dimensiones de análisis lingüístico: digital, léxico, morfológico, morfológico simplificado, ortográfico, pragmático y sintáctico.

En este documento, nos centraremos en el análisis de estos datos para la obtención del patrón buscado por el doctorando Morales. Buscar un patrón con esta cantidad de datos sin hacer uso de la informática o la computación es prácticamente imposible. Lo más común es usar herramientas propias del campo del análisis de datos. Concretamente, veremos el funcionamiento y los resultados que devuelven algunas de las herramientas más comunes en este campo, herramientas de IA. Estas herramientas son los árboles de decisión y los bosques aleatorios, siendo ambos algoritmos de aprendizaje supervisado.

En primer lugar, repasaremos los conceptos teóricos de las herramientas que usaremos para poder entender a la perfección su funcionamiento. Así, podremos también hacer un buen análisis de los resultados obtenidos. En segundo lugar, expondremos todas las pruebas ejecutadas y sus resultados. Finalmente, trataremos de darle sentido a los resultados devueltos gracias a los conceptos teóricos vistos en el siguiente capítulo.

2 Estudio previo

En un primer momento, con tal de analizar los vectores numéricos generados por Damián Morales, se hizo uso de un programa llamado Orange[4]. Sin embargo, se trata de un entorno bastante simple orientado a no informáticos. Tal y como se puede observar en la *Figura 1*, Orange es un entorno basado en un sistema donde únicamente se usan cajas y flechas para crear programas, pero no da la opción de programar absolutamente nada.

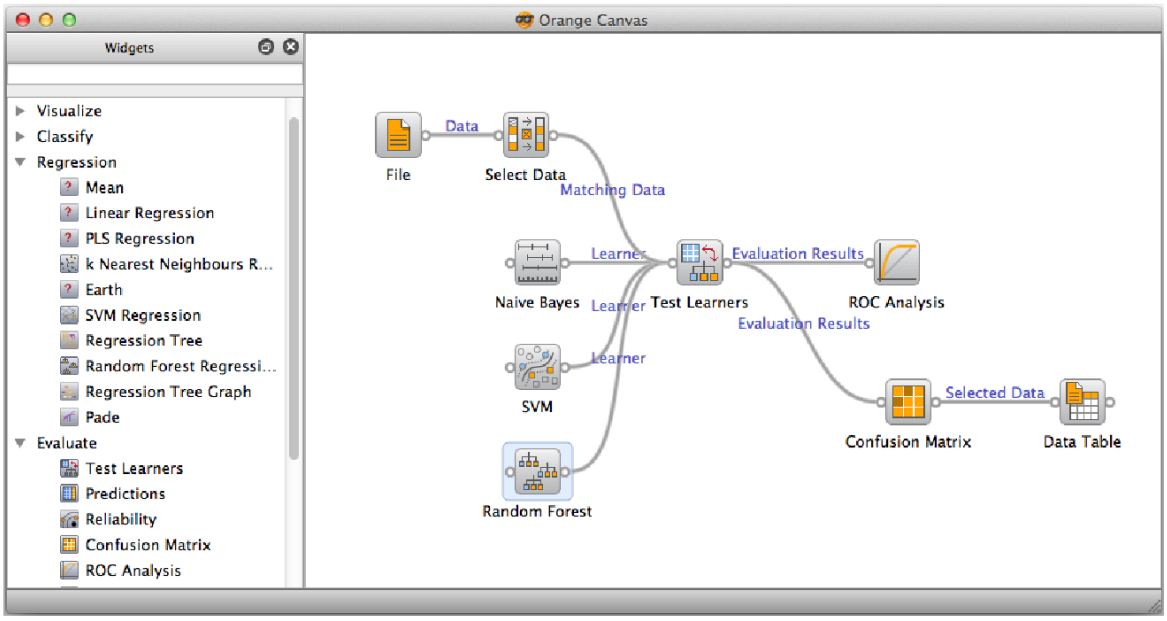

Figura 1. Entorno Orange [2].

Con el propósito de usar herramientas más flexibles y potentes, se decidió usar la **librería Scikit-learn**[5]. De esta manera, sería mucho más sencillo crear un programa que se ajuste a nuestras necesidades y al problema a tratar, por no mencionar el aumento de potencia que conlleva el uso de **Scikit** y **Python**.

[4] Programa informático creado para realizar minería de datos y análisis predictivos.

[5] Librería de software libre para el lenguaje de programación Python que proporciona algoritmos y modelos ya implementados orientados a la IA, sobre todo al campo del aprendizaje automático.

2.1 Tecnología usada

Python es un lenguaje de programación interpretado multiparadigma que soporta tanto la programación orientada a objetos[6] como la programación imperativa[7] y la funcional[8]. Pero si por algo este lenguaje es extremadamente popular es por ser de código abierto, por su legibilidad, su modularidad y su facilidad de uso. Al ser un lenguaje tan legible y sencillo de aprender, se usa en una infinidad de campos, es un lenguaje muy potente y accesible. Aunque se trate de un problema físico, matemático, financiero, informático, este lenguaje proporciona una cantidad inmensa de librerías sólidas, robustas y bien documentadas al alcance de cualquiera.

En este proyecto, se ha utilizado la librería Scikit-learn, pensada y diseñada especialmente para problemas de Data Science. Scikit-learn incluye varios algoritmos de clasificación tanto supervisada como no supervisada, algoritmos de regresión y análisis de grupos, entre muchos otros. No sólo facilita el uso de todo esto, sino que además proporciona una documentación excelente, artículos donde se estudia la comparativa entre un método u otro en el caso en que el programador no se decida a la hora de usar un modelo para su programa… En pocas palabras, facilita muchísimo el uso de sus funcionalidades gracias al tiempo dedicado a la documentación.

Dada la naturaleza de este trabajo, necesitamos un modelo que sea capaz no sólo de realizar las predicciones deseadas, sino también que sea capaz de justificar sus predicciones, al igual que un médico puede justificar por qué su paciente tiene o no una enfermedad. Por lo tanto, el primer modelo que hemos usado ha sido el **árbol de decisión**.

Respecto a la plataforma usada para el desarrollo de este proyecto, esta es **Jupyter Notebook**, una plataforma interactiva ejecutada en un entorno web. Dicha plataforma permite desarrollar software de código abierto y ofrece servicios para la computación interactiva en docenas de lenguajes de programación. Pero su punto más fuerte es el de poder mezclar lenguajes en un solo fichero. En un Jupyter notebook, se pueden escribir ficheros de formato .py, donde escribir nuestro código en Python, y ficheros con formato .ipynb. Este formato nos permite escribir tanto en lenguajes de marcado[9] como lenguajes como Python. Así pues, nos permite escribir, por ejemplo, en HTML, para redactar y explicar los pasos dados en cada procedimiento, y en Python, para ejecutar código de este lenguaje y poder ver los resultados en el mismo fichero. Se puede observar un fragmento de fichero .ipynb en la *Figura 2.*

[6] Paradigma de programación caracterizado por ofrecer la posibilidad de organizar el código en unidades llamadas clases, a partir de las cuales se pueden crear objetos que funcionan según lo dictan las clases a las que pertenecen.

[7] Paradigma de programación que se basan en una secuencia clara de instrucciones para un ordenador.

[8] Paradigma de programación basado en el uso de funciones, permitiendo mayor legibilidad del código y haciendo un código más modular y reducido.

[9] Un lenguaje de marcado es una forma de codificar un documento que incorpora etiquetes o marcas que contienen información addicional acerca de la estructura del texto.

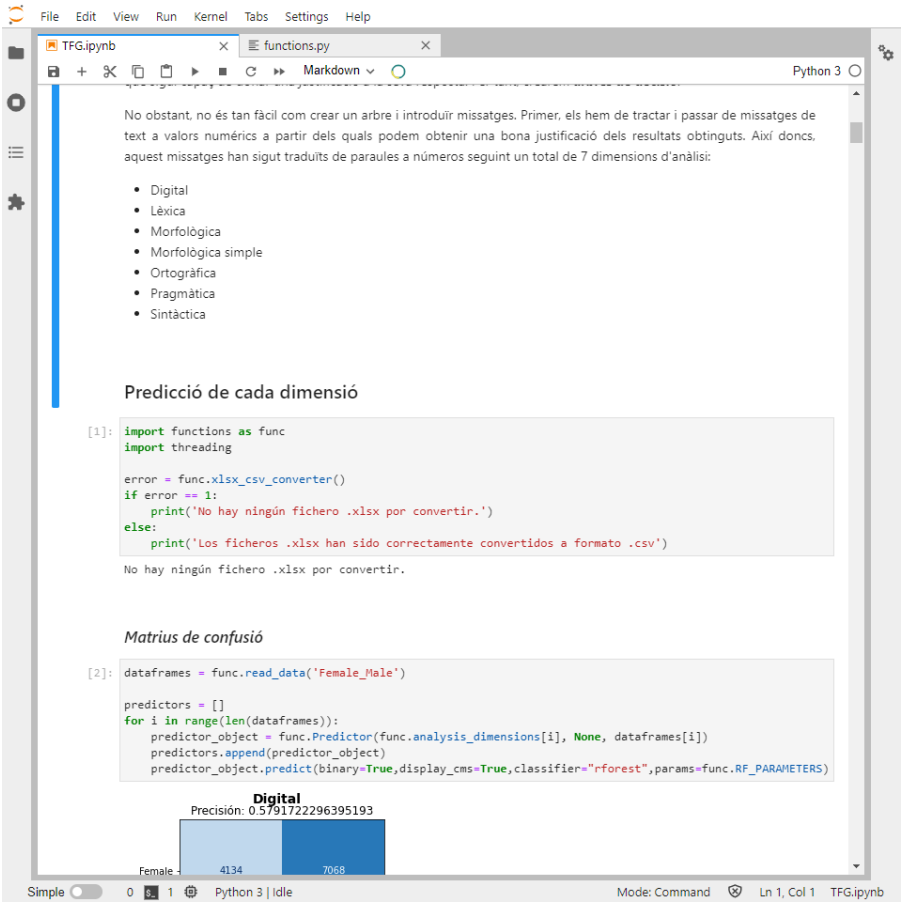

Figura 2. Fragmento del fichero TFG.ipynb.

2.2 Árboles de decisión

Un árbol de decisión (del inglés Decision Tree, DT) es un modelo o método dentro del campo de la IA que se caracteriza por ser un modelo de aprendizaje supervisado usado para la **clasificación** de datos.

2.2.1 Fundamentos teóricos

Este modelo tiene la estructura de un árbol, como su propio nombre indica. Todo árbol sigue una estructura jerárquica y tiene los mismos componentes: una **raíz**, **nodos**, **ramas** y **hojas**, tal y como sucede en la vida real.

La raíz es el primer elemento del árbol, el elemento a partir del cual se genera el resto de los elementos. Pero la raíz, además de ser el primer elemento, es un nodo, que se caracteriza por ser el inicial. En los árboles, un nodo es una estructura que puede contener información. Así pues, la raíz del árbol puede tener nodos hijos, denominados también nodos internos. El camino que lleva de la raíz a cualquiera de sus nodos hijos o, en general, el camino que lleva de un nodo padre a un nodo hijo se denomina rama. Finalmente, los nodos que se encuentran en la base de la jerarquía y que no tienen hijos reciben el nombre de nodos hoja.

Del mismo modo, todo árbol de decisión dispone de estos elementos, pero la clave se encuentra en la información que contienen los nodos y las ramas. Para clasificar un objeto, el modelo recorrerá el árbol desde la raíz hasta un nodo hoja según el valor almacenado en los atributos del objeto. Cada nodo representa un atributo, y cada rama representa un posible

valor de ese atributo. Entonces, en función del valor que tenga el objeto en el atributo indicado por el nodo, el modelo recorrerá el árbol por una rama o por otra. Si el modelo repite este proceso en cada uno de los nodos que encuentra, acabará en un nodo hoja y finalizará su recorrido por el árbol. Pero estos nodos son especiales: en lugar de almacenar un atributo, los nodos hoja indican a qué grupo pertenece el objeto introducido en el árbol. En otras palabras, los nodos hoja son los que clasifican definitivamente a un objeto.

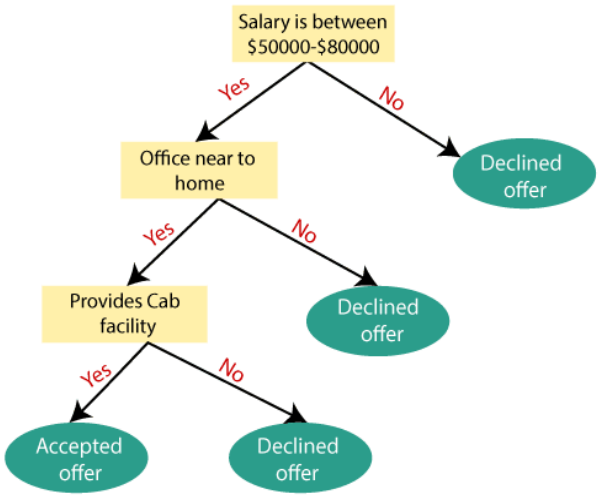

Figura 3. Ejemplo de árbol de decisión [3]

Debido a la estructura de este modelo y a la manera en que se recorre el árbol, los árboles de decisión son excelentes a la hora de justificar la clasificación de un objeto. Por ejemplo, en la *Figura 3*, el árbol de decisión indica si deberíamos aceptar o no una oferta de trabajo. Si usásemos este árbol de decisión para tomar una decisión, lo más normal sería querer saber por qué la decisión ha sido tomada. Supongamos que recibimos la decisión de rechazar la oferta. Para poder darnos una explicación, el modelo necesita únicamente indicar los nodos y ramas que ha recorrido. Un ejemplo muy válido sería decir que ha recorrido el nodo raíz, la rama "Sí", ha pasado por el nodo "Oficina de casa" y ha escogido la rama "No". Este ejemplo nos estaría comunicando que ha decidido no aceptar la oferta porque, aunque el salario se encuentra entre $50.000 y $80.000, como la oficina no está cerca de casa, no es una oferta que nos interese.

En lo que respecta a la construcción del modelo, en todo modelo de aprendizaje supervisado se pasa por dos fases claramente definidas: una primera fase de entrenamiento, en la que el algoritmo genera el modelo a partir de los datos categorizados introducidos y una segunda fase de prueba, en la que se utiliza el modelo o la estructura obtenida.

De la misma forma, un árbol de decisión es generado gracias a un conjunto de datos ya clasificados y es posteriormente probado con datos no clasificados. Un esquema representativo de este proceso es el que se puede observar en la *Figura 4*. En nuestro caso, los datos usados como conjunto de entrenamiento son un subconjunto de los datos generados por el doctorando Morales y el conjunto de prueba es la parte restante de los datos.

A continuación, explicaremos en detalle cómo funciona el algoritmo encargado de construir el árbol de decisión a partir de los datos de entrenamiento. De esta manera, podremos entender mejor qué hace que un árbol de decisión sea bueno o malo, qué praxis seguir y qué principios debemos tener en cuenta a la hora de usarlos en este proyecto.

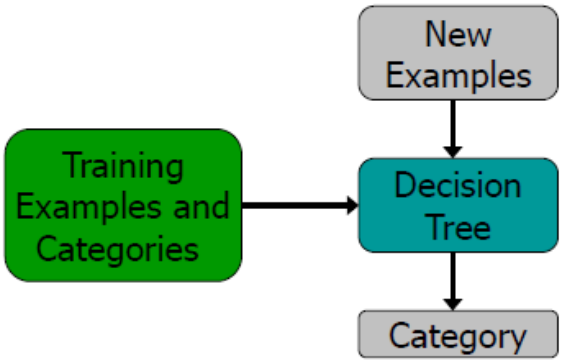

Figura 4. Esquema general de la generación de un árbol de decisión [4]

El algoritmo sigue cinco fases: en primer lugar, para crear un nuevo nodo, se busca el atributo que aporte más información al siguiente nodo; en segundo lugar, se asigna el atributo encontrado al nodo generado; en tercer lugar, se buscan los descendientes del nodo generado para cada valor posible del atributo asignado; en cuarto lugar, se usan los datos de entrenamiento para comprobar si todos son correctamente clasificados con el árbol generado hasta ahora; finalmente, si todos los datos de entrenamiento son debidamente clasificados, se considera que las hojas actuales son correctas. En caso contrario, se repite el proceso en busca de nuevos nodos hoja.

En cuanto a la manera en que el algoritmo busca el mejor atributo, se puede hacer de dos maneras: o bien mediante el cálculo de la **entropía** o bien mediante el **índice o coeficiente Gini**. Ambos permiten al algoritmo cuantificar cuánta información aporta un atributo dentro de un set de datos concreto. Con ello, es capaz de determinar qué atributo es el más importante, decidir el orden de los nodos internos y las hojas del árbol en la jerarquía, cuándo debe un nodo interno tener descendientes y medir la precisión de cada nodo [5]. En lugar de explicar en detalle cada uno de los métodos, daremos un breve repaso del concepto teórico de cada método además de mencionar las ventajas y desventajas de estos.

Por un lado, según Oxford Languages, en el campo de la física, la entropía es una magnitud termodinámica que indica el grado de desorden molecular de un sistema. Por otro lado, en el campo de la informática, la entropía no es más que la medida de la incertidumbre existente ante un conjunto de mensajes o datos [6]. Dicho de otra manera, la entropía permite cuantificar la impureza o la aleatoriedad de un dato dentro de su conjunto. Para calcular la entropía de *n* clases, se utiliza la fórmula (1), donde *S* es una colección de objetos, P_i es la probabilidad de los posibles valores e *i* son las posibles respuestas de los objetos.

$$\text{Entropía}(S) = \sum_{i=1}^{n} -p_i \, log_2 \, p_i \qquad (1)$$

Gracias a la entropía, se introduce el concepto de **ganancia de información**, que es una medida de discriminación, un indicador del siguiente atributo a ser seleccionado para continuar con el proceso de división. Esto es posible al discriminar el atributo seleccionado entre los demás atributos aún no clasificados, y se calcula utilizando la fórmula (2).

$$\text{Gan Inf}(S, A) = \text{Entropía}(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} \text{Entropía}(S_v) \quad (2)$$

Donde: *S* es una colección de objetos, *A* son los atributos de los objetos y *V(A)* es el conjunto de valores que *A* puede tomar.

Entonces, si se usa la entropía, cada atributo es seleccionado según la información que este proporciona en la decisión respecto del nodo actual. Dicho de otro modo, por cada iteración, el atributo asignado a un nuevo nodo aporta siempre una ganancia de información respecto al nodo actual.

Por otro lado, el índice Gini, también conocido como la impuridad de Gini, calcula la probabilidad de que un atributo seleccionado de manera aleatoria clasifique incorrectamente al objeto estudiado [5]. La fórmula es la (3), donde p_i denota la probabilidad de que un elemento sea clasificado por un atributo.

$$\text{Gini index} = 1 - \sum_{i=1}^{n} (p_i)^2 \qquad (3)$$

Con esta nueva fórmula, la base a partir de la cual se determina si un atributo gana o pierde información es distinta. Ahora, se considerará que un atributo gana información si es menos probable que clasifique incorrectamente a un objeto en comparación al nodo actual.

Método	Concepto	Ventajas	Desventajas
Entropía y ganancia de información	La entropía es la medida de la impureza o de la aleatoriedad de ciertos datos de un conjunto [5].	- Mayor precisión en el cálculo de ganancia de información.	- Coste temporal más elevado debido al cálculo de un logaritmo.
Índice o coeficiente Gini	Calcula la probabilidad de que un atributo escogido al azar clasifique incorrectamente a un objeto [5].	- Menor precisión en el cálculo de ganancia de información.	- Coste temporal reducido.

Tabla 1. Entropía V.S. índice Gini

Con ambos índices, la metodología a seguir es la misma. Para empezar, se calculan los coeficientes de cada atributo, ya sea con entropía o con Gini. Después, se selecciona el mejor atributo, es decir, el que mejor clasifica a un objeto en comparación al resto. Acto seguido, por cada iteración, el atributo seleccionado es aquel que aporte más ganancia de información respecto al nodo actual. De esta manera, el algoritmo va construyendo un árbol cada vez más informado sobre los datos que busca predecir.

En síntesis, los algoritmos de generación de árboles de decisión buscan ganar información en cada iteración al dividir los nodos internos. El coeficiente Gini y la entropía son los que calculan esta ganancia de información. Si comparamos ambos métodos, el uso de la entropía permite, de forma general, obtener resultados algo mejores a cambio de un coste temporal elevado, mientras que con el coeficiente Gini el cálculo a llevar a cabo es menos preciso, pero más sencillo, reduciendo así el coste temporal.

Todavía cabe señalar que, tal y como indica el artículo “How to tune a decision tree?” [7], según el paper “Theoretical comparison between the Gini index and Information Gain

criterio" [8], la frecuencia de acuerdo/desacuerdo entre el coeficiente Gini y la entropía es sólo del 2% en todos los casos, por lo que, de forma general, ambos métodos suelen ser perfectamente válidos en cualquier problema.

Hasta ahora, hemos visto qué es un árbol de decisión y cómo se genera. El siguiente punto que tratar es qué puede provocar que este árbol se genere de manera incorrecta y dé lugar a un árbol poco preciso y con muchos falsos positivos y negativos.

Por una parte, uno de los inconvenientes de los árboles de decisión es su **sensibilidad** a valores extremos. Supongamos que dividimos el total de nuestros datos en dos conjuntos, uno de entrenamiento y otro de prueba. Lo normal en un set de datos grande es que haya algún que otro dato extraviado en comparación al resto y el número de estos datos puede ser mayor o menor dependiendo del caso. Si dividimos todos los datos en los dos conjuntos mencionados de forma aleatoria, es posible que los datos extraviados estén, por pura aleatoriedad, en el conjunto de entrenamiento. Por esta razón, el árbol de decisión habrá sido entrenado en cierto modo para reconocer y clasificar estos datos extraviados cuando en realidad no son normales. Este hecho puede hacer que algunos datos normales del set de prueba sean clasificados incorrectamente, ya que el árbol ha sido entrenado incorrectamente.

Por otra parte, otro inconveniente de los árboles es el ***overfitting*** (*sobre entrenamiento* en castellano). Este fenómeno se produce cuando no se limita ningún aspecto del árbol como, por ejemplo, la profundidad máxima del mismo o el número máximo de nodos internos. Consecuentemente, el modelo es ajustado con demasiada precisión a los datos de entrenamiento. Dicho de otra manera, el modelo es entrenado para reconocer única y exclusivamente los datos de entrenamiento y no está preparado para clasificar los datos del conjunto de prueba, que es justo el objetivo de construir un árbol de decisión. Se puede observar este fenómeno en la *Figura 5*.

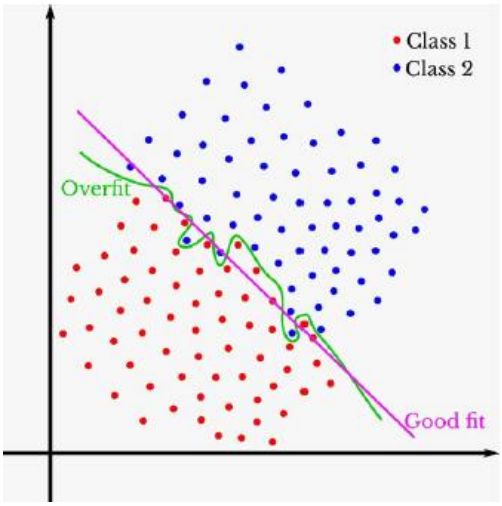

Figura 5. Gráfica representativa del overfitting [4].

Es imprescindible evitar este fenómeno para obtener un árbol de decisión de calidad. Para ello, existen diversas opciones: se puede establecer una **profundidad máxima**, un número máximo de nodos internos, un número máximo de hojas y, en última instancia, siempre se puede generar un árbol completo y podar posteriormente.

Para resumir, los árboles de decisión son modelos sencillos de entender y de implementar, además de ser rápidos y de ser capaces de justificar sus decisiones. Además,

según los desarrolladores de Scikit-Learn, el coste de los árboles de decisión es logarítmico, requieren de poca preparación de los datos y soporta tanto datos numéricos como categóricos [9]. Sin embargo, no son perfectos. Son muy sensibles al ruido y se tienen que parametrizar adecuadamente si queremos obtener un árbol de decisión funcional y sin overfitting.

2.2.2 Árboles de decisión en Scikit-learn: DecisionTreeClassifier

En la librería Scikit-learn, los modelos suelen estar ya implementados y se usan con el paradigma de programación orientada a objetos. Es decir, no hay que escribir ningún algoritmo ni nada por el estilo para hacer uso de sus modelos. Todos están listos para su uso y configuración. Sólo se necesita crear un objeto nuevo con el constructor correspondiente y ya dispondremos de métodos y atributos propios del modelo que se desea emplear.

En el caso de los árboles de decisión aplicados a casos de clasificación de datos, el objeto que proporciona la librería es el **DecisionTreeClassifier**. Este objeto dispone de los siguientes métodos básicos: **fit(X, y[, sample_weigth, check_input, …])**, **predict(X[, check_input])** y **score(X, y[, sample_weigth])**. Para más detalles sobre estos métodos, no duden en consultar la web [10]. El primero método es el encargado de construir el árbol de decisión a partir de los sets de entrenamiento (X, y) pasados por parámetro. El segundo tiene como objetivo usar el árbol construido para predecir los resultados del set de prueba X pasado por parámetro, o sea, clasificarlos. Finalmente, el método *score* devuelve la precisión media obtenida en la predicción llevada a cabo por *predict*.

Al mismo tiempo, se debe tener siempre en mente qué parámetros usar y cómo configurar el DecisionTreeClassifer. En un inicio, se utilizaron estos métodos para clasificar las dimensiones de análisis de manera individual sin haber configurado previamente el objeto DecisionTreeClassifier. En consecuencia, obtuvimos una precisión máxima del 63,8% (para la dimensión morfológica simplificada) y un árbol de nada más y nada menos que 21.335 nodos. El árbol se sobre entrenó y no fue capaz de realizar buenas predicciones.

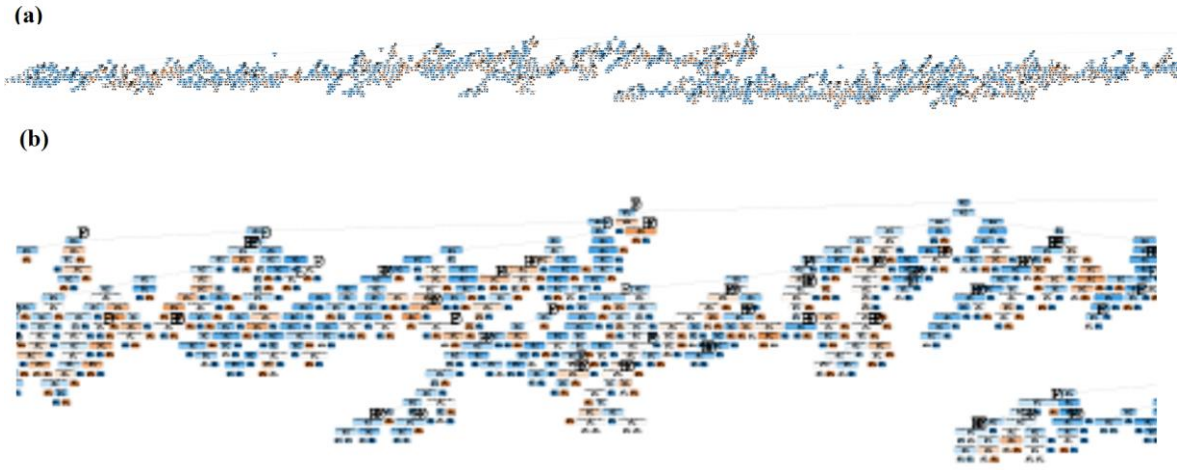

Figura 6. Árbol con overfitting. (a) Imagen de alrededor del 11% del árbol. (b) Zoom de ese 11%, apenas inteligible.

Así pues, estudiaremos a continuación los parámetros imprescindibles para la correcta configuración de un DecisionTreeClassifier, teniendo como base los fundamentos teóricos

vistos en el apartado anterior. Obviaremos los parámetros prescindibles para nuestro proyecto. Pero antes, hay que mencionar que estos parámetros reciben el nombre de **hiperparámetros**.

Primero, tenemos dos parámetros que determinan cómo escoger los atributos de cada nodo interno: el *criterion* y el *splitter*. El *criterion* es el que indica si usar el coeficiente Gini o la entropía para calcular la ganancia de información. Como hemos visto en el *Apartado 2.2.1*, a pesar de las diferencias entre ambos criterios, los resultados son prácticamente los mismos.

El parámetro *splitter* tiene dos opciones: "best" y "random". Según la implementación de Scikit de este parámetro [11], ambas opciones usan un algoritmo basado en Fisher-Yates para computar la permutación del array de atributos de los datos. No obstante, según el artículo de Mukesh Mithrakumar [7], conocer el funcionamiento de este algoritmo no es de vital importancia. La única diferencia entre ambas opciones es que, con la primera, se calculan todas las impuridades de los atributos y se escoge el siguiente nodo interno según ese cálculo, mientras que con la segunda esto se hace de manera aleatoria. Por ende, hay una gran probabilidad de escoger en algunas ocasiones atributos que no aportan apenas información, cosa que sería ideal para un árbol menos preciso y menos sobre entrenado. Aunque pueda parecer que la mejor opción es usar "best", "random" tiene una gran ventaja, y es que, como su selección es aleatoria, se pierde el overhead de tener que calcular todas las impurezas por cada iteración.

Después, hay parámetros que ajustan aspectos del árbol relacionados con su estructura. En primer lugar, está el *max_depth*, que indica la profundidad máxima del DecisionTreeClassifier. De manera general, mientras más profundo sea un árbol, más complejo es, dado que habrá más nodos internos y capturará más información sobre los datos de entrenamiento. Como se puede observar en la *Figura 6*, si no se especifica un valor, el árbol es sobre entrenado. Ahora bien, tampoco es bueno tener árboles de poca profundidad, puesto que el modelo podría estar poco entrenado.

En segundo lugar, están los parámetros *min_samples_split* y *min_samples_leaf* con definiciones similares a simple vista pero que tienen implicaciones muy distintas. Por un lado, *min_samples_split* almacena el número mínimo de muestras necesarias para crear descendencia o, dicho de otra manera, el número mínimo de muestras necesarias para generar nodos internos. Pongamos por caso que el algoritmo quiere crear un nuevo nodo interno para el árbol con un determinado atributo. Si por ese nodo pasan un total de 34 muestras en su recorrido, pero el mínimo especificado es de 35 muestras, el nodo no será generado. Esto permite tener un árbol más pequeño, con menos nodos y, por lo tanto, con menos riesgo de sobre entrenamiento. Tal y como se indica en el paper "An empirical study on hyperparameter tuning of decision trees" [12], el valor ideal de este parámetro oscila entre los valores 1 y 40. De nuevo, si hay valores muy altos, podemos encontrarnos ante casos de poco entrenamiento, mientras que valores muy bajos pueden provocar overfitting. Por otro lado, *min_samples_leaf* tiene como definición "El número mínimo de muestras requerido para convertirse en un nodo hoja. Un punto de split en cualquier profundidad será considerado únicamente si hay al menos *min_samples_leaf* muestras de entrenamiento en cada una de las ramas hijas de izquierda y derecha." [10]. De este modo, en lugar de limitarse la generación de nodos internos, lo que se limita es la generación de nodos hoja, pudiendo controlar así el nivel de entrenamiento del árbol. Según el paper [12], el valor idóneo para este parámetro se encuentra entre 1 y 20. Este paper también indica que los dos parámetros

comentados en este párrafo son los máximos responsables del rendimiento final de los árboles de decisión de Scikit.

Finalmente, Scikit permite configurar cuatro parámetros más que resultan interesantes para la disposición de nuestros datos y que afecta directamente a cómo coge los datos el modelo: *max_features*, *random_state*, *min_impurity_decrease* y *presort*. El primero de ellos determina el número de atributos a considerar cuando se quiere generar un nuevo nodo. El valor puede ser un entero, un número de coma flotante o un string. De ser un string, el parámetro admite únicamente los strings "sqrt" (*max_features=sqrt(n_features)*), "auto" (es idéntico a "sqrt"), "log2" (*max_features=log2(n_features)*) o *None*, donde *max_features=n_features*, siendo *n_features* el número de atributos presentes en nuestros datos. Si hay muchos atributos y muchos objetos, puede ser computacionalmente costoso calcular la entropía o el coeficiente Gini de cada uno de ellos en cada una de las iteraciones. Por lo tanto, hacer uso de esta opción permite aligerar la carga computacional. Otro uso del parámetro *max_features* es el de limitar el overfitting, ya que al reducir el número máximo de atributos se puede incrementar la estabilidad del árbol y reducir la varianza [7].

El siguiente parámetro de la lista es *random_state*, que recibe valores de dos tipos, tanto enteros como instancias de RandomState. Si se pasa por parámetro un entero, *random_state* usa como semilla de generador de números aleatorios ese entero. Según la documentación de Scikit [10], esta opción controla la aleatoriedad del estimador. El problema es que ya existe aleatoriedad, y más si se usan opciones como la opción "random" en el splitter. Si hay demasiada aleatoriedad, un árbol de precisión puede ser muy bueno por haber escogido unos buenos atributos o puede ser muy malo por el motivo contrario. En síntesis, es mejor dejar este parámetro por defecto, no tocarlo.

Después, el *min_impurity_decrease* hace que un nodo sea generado si induce un decrecimiento de impuridad mayor o igual al valor especificado. Entonces, nos ayuda a controlar la profundidad de nuestro árbol basándonos esta vez en la impuridad de los atributos.

Finalmente, disponemos del parámetro *presort*, que es un booleano que indica si se deben reordenar los datos antes de iniciar la fase de entrenamiento. Según la documentación oficial [10], activar este parámetro con sets de datos grandes puede ralentizar bastante el proceso. No obstante, si el set de datos es pequeño o de profundidad restringida, este parámetro puede acelerar la creación del árbol. A su vez, según el artículo de cómo parametrizar adecuadamente un DecisionTreeClassifier [7], en el caso de tener sets de datos pequeños o de querer limitar la profundidad de nuestro árbol, usar el *presort* nos puede ayudar.

Es necesario recalcar que, mientras que *max_depth*, *min_samples_split* y *min_samples_leaf* son criterios que limitan el crecimiento del árbol, *min_impurity_decrease* es un método de poda. Si el estimador tiene demasiados límites de crecimiento, este puede no estar bien entrenado. En el caso contrario, puede haber algo de sobre entrenamiento, y por ese mismo motivo existen métodos que permiten podar el árbol y terminar de perfeccionarlo. Lo ideal es tener parámetros que limiten el crecimiento del árbol y parámetros que lo perfeccionen al terminar.

Estas son las opciones que ofrece Scikit-Learn para crear un árbol de decisión. A pesar de no tener que teclear nosotros mismos el algoritmo, no es tarea sencilla obtener un árbol óptimo ajustado a nuestras necesidades. Requiere un buen conocimiento de los datos que tenemos, un buen conocimiento del funcionamiento de un árbol de decisión y un buen entendimiento de las opciones disponibles.

2.3 Random Forests

Los árboles de decisión son eficaces e increíblemente populares en el campo de la inteligencia artificial. No obstante, como hemos podido observar, tiene algunas carencias significativas que pueden dar lugar a malos resultados. Las dos carencias principales de estos son la sensibilidad al ruido de las muestras y el posible sobre entrenamiento que se puede producir. Ambas carencias pueden desaparecer con los **Random Forests (RF)**.

2.3.1 Fundamentos teóricos

Un Random Forest es literalmente un bosque aleatorio, en castellano. "Bosque" porque está compuesto por varios árboles de decisión, normalmente entre cien y doscientos. "Aleatorio" porque cada uno de esos árboles de decisión explora y clasifica los objetos de una forma distinta: es obligatorio que los árboles tengan diversidad y variabilidad entre ellos, y que sus predicciones sean lo más independientes posibles. Entonces, para cada árbol del bosque, se coge un set de entrenamiento aleatorio dentro de los datos iniciales. Normalmente, los datos de entrenamiento representan dos tercios de los datos totales. Después, durante la generación de cada nodo, no se analizarán todos los atributos disponibles. En su lugar, sólo se analizará un subconjunto aleatorio de estos. Normalmente, este subconjunto de atributos incluye de uno a log2(num_atributos) atributos. Por ese mismo motivo se denomina bosque aleatorio.

Cuando se quiere clasificar un nuevo objeto, este es clasificado por todos y cada uno de los árboles del bosque. Acto seguido, dependiendo de si se busca una predicción numérica o una clasificación, se calculará la media aritmética o se decidirá la clasificación por voto por mayoría respectivamente.

Por una parte, al disponer de varios árboles de decisión menos entrenados y con predicciones totalmente independientes, se evita cualquier caso de overfitting. Por otra, como por cada árbol, los atributos seleccionados son totalmente aleatorios, a menudo se evita coger atributos que incluyen valores extremos, eliminando así la sensibilidad al ruido en las muestras.

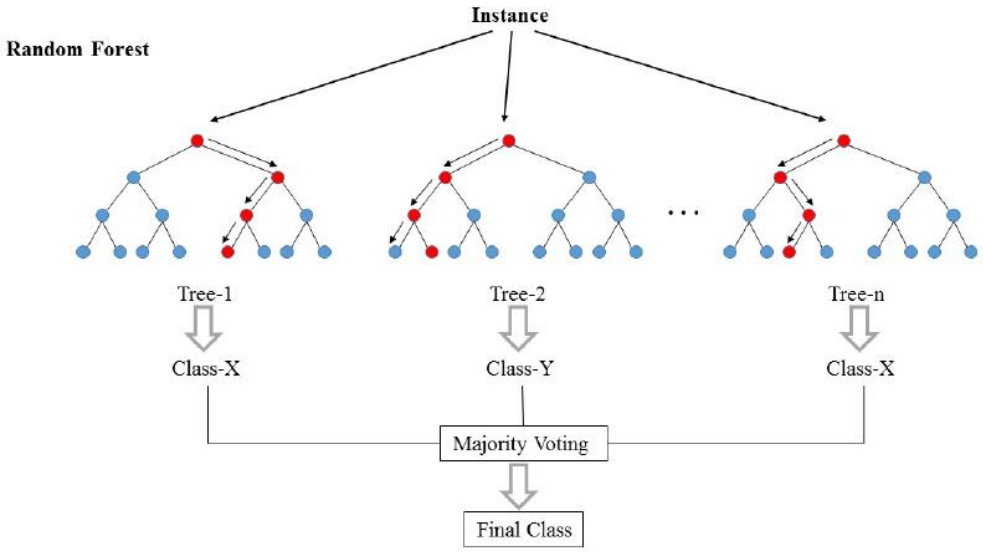

Figura 7. Esquema de clasificación de un objeto mediante Random Forests [13].

Haciendo alusión a la comparación hecha por el profesor de teoría de IA de cuarto de GEI y mi tutor de TFG Antonio Moreno Ribas, se puede pensar en un bosque aleatorio como si se consultase a expertos sobre un tema en la vida real. Normalmente, si alguien quiere tener una opinión experta lo más objetiva posible, puede hacer dos cosas: puede recurrir al mejor experto del mundo en ese ámbito o puede recurrir a varios expertos corrientes y sacar una respuesta más clara y con distintos puntos de vista. En el primer caso, se trata del mejor experto, sí, pero eso no significa que su opinión sea perfecta ni válida en muchos casos, sigue siendo humano y puede equivocarse. Sin embargo, en el segundo caso, al disponer de muchas opiniones con matices y puntos de vista distintos, se puede obtener una respuesta más completa y precisa. En general, así se adquiere más conocimiento.

Aunque las desventajas de un árbol de decisión individual desaparecen con los random forests, esto no significa que sean perfectos. El coste computacional de crear tantos árboles de decisión y utilizarlos en el proceso de clasificación es más elevado. Además, hay más parámetros a considerar y puede resultar más costoso obtener una configuración de parámetros óptima. Finalmente, debemos tener en cuenta que un factor de vital importancia para nuestro proyecto desaparece: un bosque aleatorio es incapaz de dar una justificación válida a sus decisiones. Al funcionar por voto por mayoría, lo único que sabemos al obtener una clasificación es que se ha hecho porque, por mayoría, esa ha sido la mejor decisión. No podemos saber nada más.

2.3.2 Random Forests en Scikit-Learn: RandomForestClassifier

Tal y como sucede con los árboles de decisión, la librería de IA ofrece bosques aleatorios ya implementados en forma de instancias que podemos crear y configurar. De esta manera, no debemos preocuparnos por la implementación del Random Forest, simplemente debemos importarlo a nuestro proyecto, crear un nuevo objeto con los parámetros que veamos adecuados y estará listo para clasificar objetos.

Pero, como hemos visto en los fundamentos teóricos de este modelo, se necesitan más parámetros para configurarlo. Requiere de un estudio del proyecto más profundo y de un buen conocimiento del comportamiento y efectos de cada hiperparámetro.

A continuación, repasaremos cada uno de los parámetros disponibles en un RandomForestClassifier. Para más información, no dude el lector a dirigirse a la documentación oficial [14].

En primer lugar, se dispone de algunos de los parámetros propios de un árbol de decisión: *criterion*, *max_depth*, *min_samples_split*, *min_samples_leaf*, *max_features*, *max_leaf_nodes*, *min_impurity_decrease*, etcétera. Estos hiperparámetros configuran las características de los árboles de decisión presentes en los random forests. No obstante, conviene hablar más en profundidad sobre algunos de ellos.

Al aumentar el valor de *max_depth*, se aumenta la probabilidad de combinar distintos atributos y valores. Es decir, mientras más profundidad, más splits habrá y más información se tendrá en cuenta a la hora de realizar una predicción. Conviene subrayar que ya no estamos ante un único árbol, por la cual cosa no se producirá overfit, al menos no en el modelo general. Ahora bien, debemos tener en mente el coste computacional. Si el RandomForestClassifier debe generar, por ejemplo, 300 árboles de decisión, cada uno de ellos siendo profundo, el coste computacional se elevará enormemente. Así pues, se necesita encontrar un equilibrio.

Otro parámetro propio de los DecisionTreeClassifier's de vital importancia es el *max_features*. De hecho, según un artículo de la propia biblioteca, es uno de los parámetros principales a configurar [15]. Como vimos en apartados anteriores, este parámetro indica el número de atributos que tendrá en cuenta cada árbol en la generación de un nuevo nodo. Por un lado, según el último artículo, los mejores valores empíricos son obtenidos con la opción "sqrt" para tareas de clasificación. Pero esto puede tener un coste computacional y temporal más elevado. Por otro lado, una alternativa podría ser tener "None" en *max_features* y *min_samples_split=2*, pero seguiría consumiendo mucha RAM al generar más nodos en cada árbol y los resultados podrían no ser los óptimos.

A su vez, tanto en el artículo de Scikit [15] como en un artículo web de Towards Data Science [16], la mejor forma de encontrar una buena configuración en este caso es mediante **validación cruzada**. De esto se hablará en profundad más adelante en este capítulo.

En segundo lugar, un RandomForestClassifier tiene hiperparámetros propios: *n_estimators*, *bootstrap*, *n_jobs* y *warm_start*. Como su propio nombre indica, *n_estimators* indica el número de estimadores que generará el bosque. Ya sabemos que el número común varía entre 100 y 200. Pero vayamos más allá. Por un lado, debido a la aleatoriedad en la construcción del random forest, si se dispone de muchas features en una base de datos e introducimos un número pequeño de estimadores, algunas de esas features con un potencial de predicción elevado podrían ser obviadas o ser poco usadas. Por lo tanto, no conviene usar un número pequeño de estimadores. Por otro lado, debido al coste computacional de crear cada uno de los árboles, tampoco es recomendable usar un número elevado.

Si *bootstrap* recibe el booleano "False", todo el set de datos de entrenamiento será usado para construir los árboles. En general, este hiperparámetro recibe floats que indican el porcentaje de los datos de entrenamiento que serán usados para entrenar cada árbol. Aunque el tamaño del bootstrap sea el mismo que el del set de entrenamiento, los datos usados por cada árbol serán distintos. Por lo tanto, tal y como se indica en el artículo [16], usar este parámetro no afecta al algoritmo en la gran mayoría de casos. Puede obviarse y dejarse el valor por defecto, que es "False".

El siguiente hiperparámetro es el primero centrado puramente en el paralelismo del algoritmo: *n_jobs* indica el número de tareas a ejecutar en paralelo. Los RandomForestClassifier's están preparados para ser ejecutados con threads debido a su coste computacional y temporal elevados. Por defecto, el parámetro recibe el valor "None", es decir, *n_jobs=1*. Sin embargo, este número puede aumentar para especificar el número de cores que queremos usar para ejecutar el algoritmo. Si *n_jobs=-1*, todos los cores de nuestra máquina serán usados.

Finalmente, tenemos la opción de configurar el parámetro *warm_start*. Se trata de un booleano que determina si, al ejecutar dos o más veces la función *fit* propia de un estimador, los estimadores generados se añaden al bosque creado en el/los anterior/es *fit*'s ejecutados. Por defecto, está en "False", es decir, cada vez que la función *fit* es ejecutada, se genera un bosque totalmente nuevo. En nuestro caso, conviene dejar el valor por defecto.

2.4 Optimización de parámetros

Como se ha podido observar en los apartados anteriores, son muchísimas las posibles combinaciones existentes a la hora de configurar nuestro árbol o bosque. Dar con la configuración óptima puede ser costoso. En el caso de los árboles, por cada vez que se dividen los datos en sets de entrenamiento y de prueba, no sólo los datos son distintos, sino

que el árbol tiene ciertos componentes con aleatoriedad que pueden afectar notoriamente al resultado de las predicciones. En el caso de los bosques, hay muchos parámetros que debemos combinar adecuadamente, parámetros como el número de árboles, la medida del set de entrenamiento de estos, cuántos atributos deberían escoger al generar sus nodos… En definitiva, cada vez que se genera un estimador, lo ideal sería reconfigurar los parámetros de forma óptima.

Por suerte, el equipo detrás de la librería Scikit ha pensado en esta problemática y ofrece diversas soluciones basadas en la **validación cruzada** (en inglés, cross-validation, abreviado como CV).

2.4.1 Cross-validation

La validación cruzada o cross-validation, a veces denominada estimación de rotación o out-of-sample testing, es un conjunto de técnicas de validación de modelos para evaluar cómo los resultados de un análisis estadístico pueden generalizar a un conjunto de datos independientes [17]. Si reformulamos esta definición, podemos decir que la validación cruzada es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y de prueba [18]. Se utiliza principalmente en entornos donde el objetivo es la predicción y se desea estimar la precisión con la que un modelo predictivo funcionará en la práctica.

Según el artículo “Cross-validation: evaluating estimator performance” [19], el flujo de acciones que se suele seguir al hacer uso de cualquier estimador es el que se puede observar en el diagrama de flujo de la *Figura 8*.

Mientras que en el flujo normal de todo estimador se dividen los datos en dos sets, uno de entrenamiento y otro de prueba, la CV sigue un procedimiento denominado **k-fold CV**. En este procedimiento, el set de entrenamiento es dividido en un número *k* de sets más pequeños. Acto seguido, los siguientes pasos son llevados a cabo por cada uno de los *k* subconjuntos:

- Un estimador es entrenado usando $k - 1$ sets como si se tratasen de los datos de entrenamiento.
- El estimador resultante es validado con los datos restantes (es decir, es usado como test para calcular medidas de rendimiento como la precisión conseguida con los parámetros establecidos).

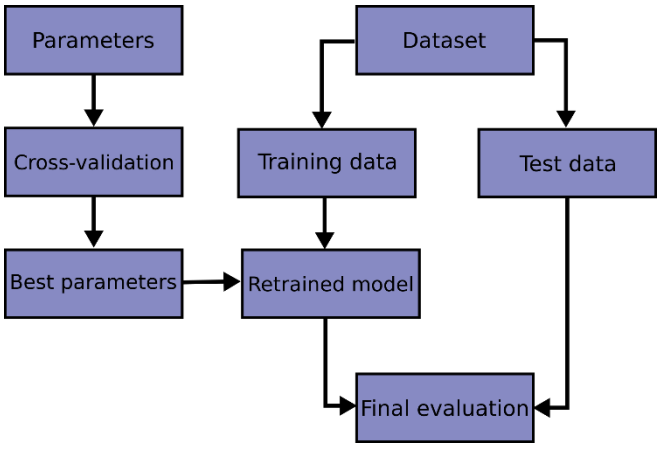

Figura 8. Diagrama de flujo de un estimador [19].

El rendimiento obtenido por los *k* sets de validación cruzada se calcula entonces con la media de los valores obtenidos en el bucle descrito anteriormente. Este acercamiento puede resultar computacionalmente costoso, pero no requiere de una gran cantidad de datos, cosa que supone un avance notorio en problemas con sets de datos pequeños.

En la *Figura 9*, se puede observar visualmente el proceso de CV de un clasificador o estimador.

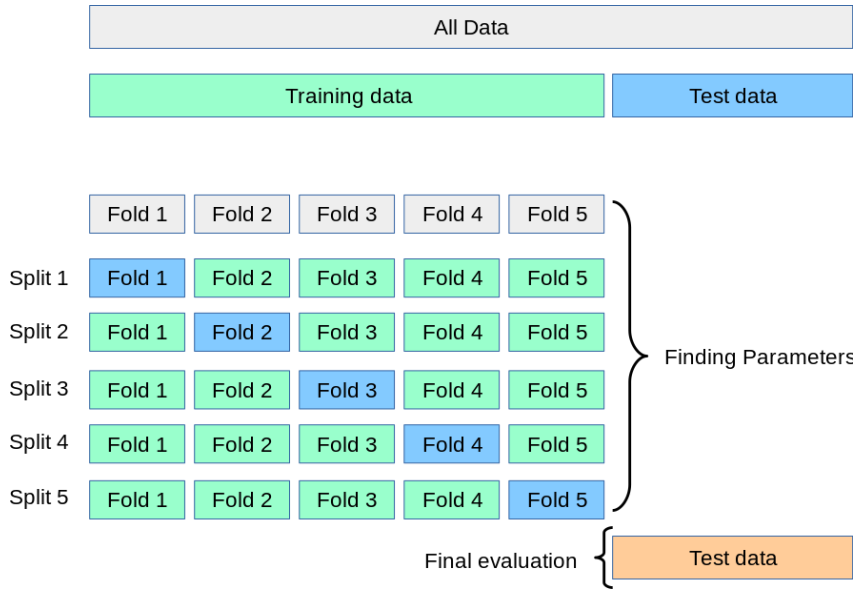

Figura 9. Split de datos en la CV [19].

Ahora bien, la CV permite validar, pero no devuelve los parámetros óptimos, sino que mide cuán óptimos son. Permiten calcular la calidad de un estimador ya parametrizado. Entonces, mediante esta técnica, es posible crear algoritmos capaces de encontrar la combinación óptima de los parámetros que deseemos con los estimadores que deseemos.

Como ya hemos visto, los hiperparámetros son introducidos con argumentos para el constructor de la clase de un estimador. Debemos crear un método de búsqueda de la combinación óptima de estos hiperparámetros.

Por lo general, según el artículo [20], una búsqueda de esta índole se basa en:

- Un estimador.
- Un espacio de parámetros.
- Un método de búsqueda de posibles candidatos.
- Un esquema de CV.
- Una función de puntuación.

En la búsqueda de la combinación óptima de hiperparámetros, hay dos acercamientos usuales proveídos por Scikit: **GridSearchCV**, que busca exhaustivamente todas las combinaciones de parámetros, y **RandomizedSearchCV**, capaz de crear un número determinado de candidatos a partir del espacio de parámetros posibles pasados al validador. Ambas herramientas tienen sus contrapartidas **HalvingGridSearchCV** y

HalvingRandomSearchCV, que pueden ser mucho más rápidas encontrando buenas combinaciones.

2.4.2 GridSearchCV y RandomizedSearchCV

Por una parte, el GridSearchCV genera de manera exhaustiva candidatos a partir de un diccionario de parámetros, donde las claves son los parámetros para analizar y sus correspondientes valores son los posibles candidatos a valores definitivos. La combinación que obtiene mejor puntuación es guardada en el atributo *best_params_*. Un ejemplo de diccionario de parámetros podría ser este:

```
params = {
        "criterion": ["gini","entropy"],
        "splitter": ["best","random"],
        "max_depth": [2,4,6,8,10,12,14,16],
        "min_samples_split": [10,15,20,25,30,35,40],
        "min_samples_leaf": [5,10,15,20],
        "max_features": [None,"sqrt","log2"],
        "min_impurity_decrease": [0.0,0.05,0.1,0.15,0.2]
}
```

Código 1. Ejemplo de espacio de hiperparámetros en un algoritmo de búsqueda.

Al fin y al cabo, el validador Grid Search se basa en una cuadrícula de parámetros (un "grid" de parámetros, en inglés).

Por otra parte, aunque el Grid Search sea el método más popular, el Randomized Search implementa una búsqueda aleatoria sobre parámetros con beneficios especiales. Según el artículo [20], dos de estos beneficios son los siguientes:

- Añadir parámetros que no influencian en el rendimiento no empeora la eficiencia.
- Se pueden obtener resultados independientemente del número de parámetros y sus posibles valores.

Los parámetros se especifican de igual manera que en el caso del Grid Search, con diccionarios. Para cada parámetro, se puede especificar o bien una distribución de posibles valores o una lista discreta de elecciones que serán probadas uniformemente.

Con respecto a la comparativa entre ambos algoritmos de búsqueda, en el artículo "Comparing randomized search and grid search for hyperparameter estimation" [21] se muestra una comparación real de los resultados obtenidos con estos, además del coste temporal de cada uno. Ambas búsquedas exploran exactamente el mismo set de parámetros. Los resultados en la configuración final de parámetros son bastante similares, mientras que el coste temporal cambia radicalmente: el coste temporal en el Randomized Search es drásticamente menor. Sin embargo, debemos tener en cuenta que el rendimiento es ligeramente peor en el caso de la búsqueda aleatoria, y es debido a un efecto de ruido creado durante el testeo.

En este ejemplo, se usa una base de datos de 10 clases, con alrededor 180 muestras por clase y un total de 1797 muestras [22], mucho menos que en nuestro proyecto. Esta base de

datos se estimará con tres estimadores distintos. En la búsqueda aleatoria, la optimización de parámetros tiene un coste temporal de 32,09 segundos para 20 candidatos a posible combinación de parámetros, obteniendo una media de puntuación de validación del 92% en el primer y el segundo estimador y una media del 91,8% en el tercero. En la Grid Search, el coste temporal es de 192,81 segundos para 100 candidatos a configuración de parámetros, obteniendo una media de puntuación de validación del 93,1%, 92,8% y 92,7% en los estimadores uno, dos y tres respectivamente.

En definitiva, el Grid Search es más costoso temporalmente, pero obtiene una puntuación de validación mayor que la obtenida con el Randomized Search.

2.4.3 Halving sucesivo

El **Halving sucesivo** o successive Halving (SH) es un algoritmo de búsqueda creado e implementado recientemente en la librería. Su funcionamiento es equiparable a un torneo entre las distintas combinaciones de parámetros. Se trata de un proceso iterativo de selección en el que todos los candidatos, es decir, las combinaciones de parámetros, son evaluadas con una pequeña cantidad de recursos en la primera iteración. Sólo una pequeña muestra de candidatos será seleccionada para la siguiente iteración, que dispondrá ahora de más recursos. De forma general, un recurso común suele ser el número de muestras de entrenamiento, pero también puede ser un parámetro numérico arbitrario, como por ejemplo el hiperparámetro *max_depth* en el caso de un DecisionTreeClassifier.

Este proceso se repite hasta dar con una combinación ganadora. Como se puede observar en la *Figura 10*, sólo unos pocos candidatos sobreviven en el torneo hasta la última iteración. El criterio que sigue el algoritmo para determinar qué candidato es mejor que el resto es la puntuación de CV.

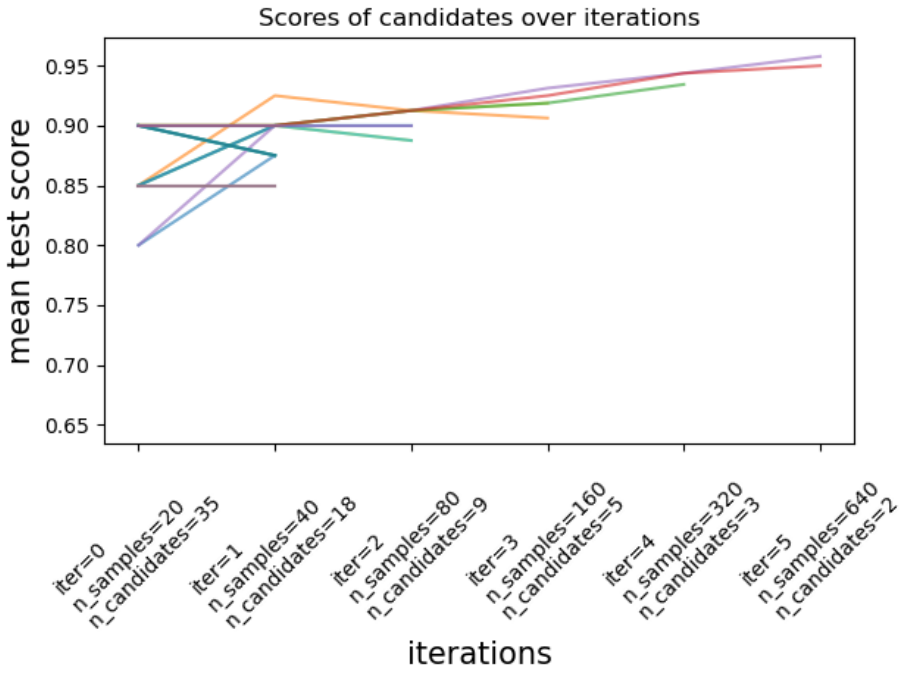

Figura 10. Candidatos en cada iteración de un SH [23].

El ritmo al que los recursos aumentan y los candidatos disminuyen en cada iteración no es fijo. Es más, se rige por un factor numérico configurable. Para calcular el número de recursos y de candidatos en cada iteración, se usan las fórmulas (4) y (5), donde: *i* es el número de la iteración; *factor* es el factor numérico configurable; *min_recursos* es el número de recursos en la primera iteración y *n_candidatos* es el número de candidatos en la primera iteración.

$$\text{n\_recursos\_i} = factor^{i} \times \text{min\_recursos} \quad (4)$$

$$\text{n\_candidatos\_i} = \frac{n\_candidatos}{factor^{i}} \quad (5)$$

En lo que toca a la implementación, está claro que es más compleja y requiere de más análisis que el Grid Search o la búsqueda aleatoria. Tal y como se indica en la documentación oficial [24], hay muchos parámetros que ajustar, únicos para el SH: *factor*, que controla la ratio con el que los recursos crecen por cada iteración y con la que los candidatos disminuyen; *min_resources* y *n_candidates*, cuyo significado hemos visto para las fórmulas (4) y (5); *resource*, que especifica el recurso que queremos usar en el torneo; *aggressive_elimination*, que es relevante en el caso en que, en las últimas iteraciones, el número de candidatos supervivientes sea excesivo y queramos obtener únicamente un candidato ganador, y muchos más.

Es conveniente destacar que, a diferencia de los métodos de búsqueda tradicionales, las implementaciones de SH están en fase experimental. Por lo tanto, ¿merece realmente la pena usar estos algoritmos? Pueden ser más rápidos que los métodos tradicionales, efectivamente, pero ¿Son igual de efectivos? ¿Merece la pena el esfuerzo de configurarlos adecuadamente?

2.4.4 SH V.S. Búsqueda tradicional

Para realizar una comparación lo más objetiva posible, se estudiarán tres ejemplos: un ejemplo documentado y publicado en la página oficial de Scikit-Learn [25], un ejemplo usado como estudio para determinar qué método es más eficiente [26] y un ejemplo aplicado a nuestro proyecto.

Como hemos visto anteriormente, el método tradicional con el que se obtiene mejores resultados y el más popular es el Grid Search. Por esto, dado que nuestro objetivo es encontrar el mejor resultado posible en nuestras predicciones, compararemos los resultados obtenidos con HalvingGridSearchCV y GridSearchCV. En el primer ejemplo mencionado, las pruebas se realizan con un set de 1000 muestras. Los resultados se pueden ver en la *Figura 11*. Con el método más popular, se consigue una puntuación de CV cercana a 1 en 13,964 segundos. En oposición, haciendo uso de SH, se consigue la misma puntuación de CV en un total de 2,731 segundos.

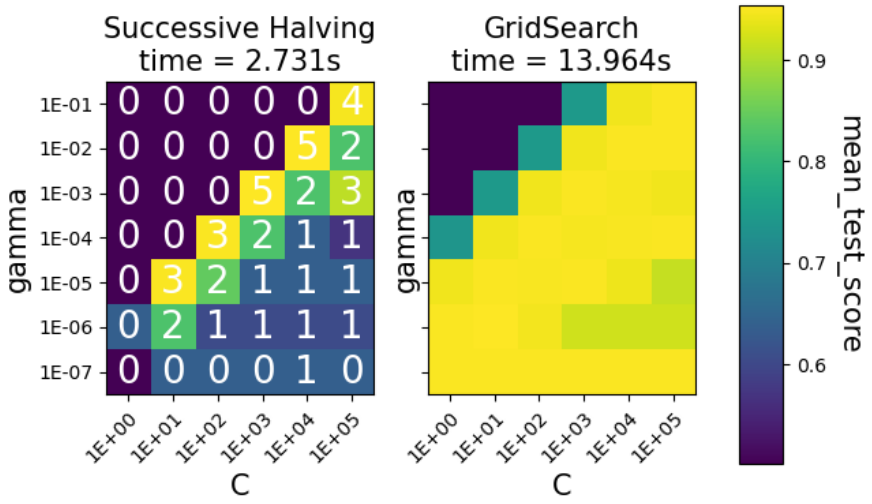

Figura 11. Comparativa HalvingGridSearchCV V.S. GridSearchCV [25].

En el segundo ejemplo, la base de datos con la que se experimenta es *Rain in Australia*, una base de datos que contiene observaciones meteorológicas hechas en Australia durante 10 años. Esta vez, hay un total de 145.460 entradas. Se obtienen aproximadamente los mismos resultados con ambos métodos, pero el SH resulta ir nada más y nada menos que once veces más rápido que el método tradicional.

En el tercer ejemplo, el experimento llevado a cabo ha sido ejecutar un bucle donde se calculaban los parámetros óptimos para un DecisionTreeClassifier. En cada iteración, se usaba un set de datos distinto correspondiente a cada una de las dimensiones de análisis de nuestro proyecto, obviando la dimensión morfológica (usando en su lugar la morfológica simplificada). Si se ejecuta este bucle haciendo uso del GridSearchCV, obtenemos un CV score de 0,70 en una hora, treinta y un minutos y diez segundos. De media, este popular método ha tardado cinco minutos en calcular todas las configuraciones óptimas exceptuando el caso de la dimensión morfológica simplificada, debido al número elevado de atributos que contiene. En este último caso, la instancia de GridSearchCV ha necesitado cuarenta y un minutos. En cambio, usando SH, cada iteración dura menos de un minuto exceptuando de nuevo la dimensión morfológica simplificada, que necesita dos minutos para terminar su ejecución. Y el CV score y las predicciones obtenidas con ambos métodos apenas varían.

En conclusión, como se ha podido observar en estos tres ejemplos reales, en todos los casos el coste temporal mejora notablemente al hacer uso del HalvingGridSearch. No sólo mejora el coste temporal, sino que además los resultados apenas varían. Así pues, a pesar de ser un método en fase experimental y que requiere más esfuerzo por parte del programador, la recompensa merece claramente la pena. El HalvingGridSearchCV ha sido el método usado para configurar de manera óptima las instancias de DecisionTreeClassifier y RandomForestClassifier.

2.5 Síntesis

Scikit-learn es una librería de aprendizaje automático pensada para Python. No sólo está brillantemente documentada, sino que además facilita muchísimo la implementación de algoritmos de data science con los modelos de aprendizaje supervisado e IA que ofrece. Nosotros, como programadores, no debemos implementar ningún modelo en sí, simplemente debemos crear objetos correspondientes a los modelos que queremos usar, parametrizarlos adecuadamente y usarlos.

No obstante, se debe hacer con cabeza. Para ello, es importante conocer cómo funcionan los modelos que usaremos en este trabajo. Por un lado, utilizaremos árboles de decisión. Estos son uno de los modelos de aprendizaje supervisado más usados a nivel mundial debido a su rapidez y a su sencillez. A partir de un set de datos de entrenamiento, el algoritmo genera una estructura jerárquica en forma de árbol basándose en la información que aporta cada atributo. Una vez el árbol es generado, este es recorrido por los objetos de prueba desde la raíz hasta las hojas para ser clasificados. Consecuentemente, este modelo es capaz de justificar sus decisiones. Basta con mostrar el recorrido que se ha hecho por el árbol de decisión. No obstante, es muy sensible al ruido en los datos y es propenso a ser sobre entrenado o sub entrenado.

Por otro lado, utilizaremos bosques aleatorios. Estos tratan de eliminar los inconvenientes de los árboles de decisión. Ahora, disponemos de varios árboles de decisión de menor tamaño. Cada uno de estos árboles será construido por un conjunto de datos y atributos aleatorios. De este modo, al disponer de más de un árbol de decisión y al proporcionar aleatoriedad a los datos seleccionados, se elimina el riesgo de sobre entrenamiento y la sensibilidad al ruido, respectivamente. No obstante, dado que ahora la decisión se toma mediante voto por mayoría o según la media aritmética de las decisiones, perdemos la capacidad de obtener una justificación de la decisión tomada. Aun así, en este documento, veremos si la teoría se cumple, es decir, si los random forests son más precisos que los árboles aleatorios en sus predicciones.

Ambos modelos requieren de una buena parametrización, sobre todo los RF. Si queremos obtener la mejor precisión posible en nuestras predicciones, no basta con parametrizar experimentalmente con prueba y error. Debemos ir más allá y buscar una parametrización óptima. Esto es posible gracias a algunos modelos de validación cruzada ofrecidos por la librería Scikit-learn: GridSearchCV, RandomizedSearchCV y modelos de Halving sucesivo todavía en fase de desarrollo. El más popular sin lugar a duda es el Grid Search, pero, a pesar de no haber una versión definitiva, los SH son notablemente más rápidos y proporcionan los mismos resultados. Así pues, usaremos SH para optimizar los clasificadores DT y RF.

Una vez interiorizados los conceptos teóricos y cómo usar las instancias de Scikit-learn, veremos los resultados obtenidos con esta librería. Además, podremos comprobar de primera mano si los fundamentos teóricos vistos hasta ahora se cumplen.

3 Implementación

Gracias a las bases teóricas explicadas a lo largo de este documento, podemos ahora estudiar los resultados obtenidos en la implementación del proyecto y sacar conclusiones objetivas. Las predicciones se han hecho de manera progresiva, es decir, no se han tenido en cuenta todos los factores y todas las dimensiones de análisis de golpe.

A lo largo de este apartado, veremos qué estudios se han hecho en el trascurso del proyecto, qué resultados hemos obtenido y qué métodos son los mejores a la hora de realizar predicciones en el caso que nos concierne.

3.1 Estructura del código

El proyecto consta de dos ficheros principales: el principal es un fichero de formato .ipynb, es decir, un Jupyter Notebook, en el cual se encuentran los estudios llevados a cabo y sus resultados. El secundario se trata de un fichero de formato .py en el que se encuentran todos los objetos y funciones que hacen que todo funcione. Se pueden encontrar tanto funciones auxiliares para facilitar la modularidad y legibilidad del código como objetos y métodos imprescindibles para la lectura de datos y las predicciones de sexo.

El fichero secundario consta de cinco apartados. En el primero, se definen las variables globales. Estas variables permiten configurar el fichero donde se guardan los mensajes de logging del programa. Además, almacenan el número total de dimensiones tratadas en el programa, los tipos de datos y dimensiones que nos podemos encontrar y dos diccionarios de parámetros que se usan por defecto en los estimadores DecisionTreeClassifier y RandomForestClassifier.

El segundo define el objeto Predictor, un objeto que se encarga de llevar a cabo las predicciones con una cantidad determinada de dimensiones. Almacena tanto las dimensiones con las que lleva a cabo las predicciones como sus datos correspondientes, la edad de los hombres y mujeres estudiados (si es que la edad se tiene en cuenta en ese Predictor) y el árbol de decisión generado. Dispone además de setters y getters, métodos auxiliares y métodos para llevar a cabo las predicciones, exportar los dataframes, los árboles de decisión que almacenan y las **matrices de confusión**[10] que genere y finalmente un método que permite hacer el análisis del árbol de decisión que tiene. El método más relevante para nuestro proyecto es el método *predict*, cuya implementación se puede leer en el *Código 2*. Los parámetros que recibe la función son los siguientes:

- *binary*: especifica si se tiene en cuenta o no la edad. Si no se tiene en cuenta, solamente se tendrá en cuenta si el autor es hombre o mujer, por lo tanto, es una opción binaria. Por defecto, esta opción está en falso.
- *display_cms*: determina si se debe mostrar una matriz de confusión o no al realizar la predicción. Por defecto, las matrices de confusión se muestran después de cada predicción.

[10] Una matriz de confusión es una matriz que permite visualizar los resultados de un algoritmo propio del aprendizaje supervisado. En estas matrices, se puede leer el número de falsos positivos y negativos en las predicciones, además del número de aciertos positivos y negativos.

- *classifier*: dicta qué estimador se utiliza para la predicción. Si la opción es "dtree", se usará un árbol de decisión. Si la opción es "rforest", se usará un random forest.
- *params*: recibe los parámetros que configurarán al clasificador del Predictor. De esta manera, se pueden pasar por parámetro los diccionarios del código que tienen los parámetros predeterminados configurados de forma manual, o se puede optar por hacer uso de los SH para obtener parámetros óptimos y usarlos.
- *debug*: si la opción es activada, la función loggeara la información pertinente en el archivo de logging del que dispone el código.

```
    def predict(self, binary = False, display_cms = True, classifier = 
"dtree", params = DT_PARAMETERS, debug=False):

      if debug is True:
            log_gender_count(self.get_df(), self.get_dim())

      X_train, X_test, y_train, y_test = split_x_y(self.get_df(), binary)

      if debug is True:
            log_splitted_gender_count(y_train, y_test, binary, 
self.get_dim())

      if classifier == "dtree":
            clf = tree.DecisionTreeClassifier(
            criterion = params["criterion"],
            splitter = params["splitter"],
            max_depth = params["max_depth"],
            min_samples_split = params["min_samples_split"],
            min_samples_leaf = params["min_samples_leaf"],
            max_features = params["max_features"],
            min_impurity_decrease = params["min_impurity_decrease"])

      if classifier = "rforest":
            y_train = y_train.values.ravel()
            clf = RandomForestClassifier(
            n_estimators = params["n_estimators"],
            criterion = params["criterion"],
            max_depth = params["max_depth"],
            min_samples_split = params["min_samples_spit"],
            min_samples_leaf = params["min_samples_leaf"],
            max_features = params["max_features"], n_jobs = -1)

      try:
            clf.fit(X_train, y_train)
            self.set_clf(clf)

            # Predice los resultados de X_test
            Prediction = clf.predict(X_test)

            # Calculamos la precision de la predicción y almacenamos el 
resultado
            accuracy = accuracy_score(y_test, prediction)
            logging.info("Precision en dim "+self.dim+": "+str(accuracy))
            self.set_accuracy(accuracy)

            if(display_cms == True):
```

```
                # Crea la matriz de confusión
                if self.age ¡= None:
                      title = str(self.dim) + " " + str(self.age* + "s"
                else:
                      title = self.dim
                cmatrix = display_confusion_matrix(clf, X_test, y_test,
                                                   accuracy, title)
                self.set_cmatrix(cmatrix)

      except ValueError:
            log_ValueError(X_train, X_test, self.get_dim())
            self.set_clf(clf)
```

Código 2. Implementación del método *predict*.

El tercer apartado contiene funciones orientadas totalmente a usos de la librería Scikit. Por ejemplo, se puede encontrar el método *optimal_config*, un método creado para encontrar la configuración óptima del estimador pasado por parámetro, ya sea haciendo uso del método tradicional GridSearchCV o usando HalvingGridSearchCV.

En el cuarto apartado se encuentran las funciones auxiliares que permiten facilitar la legibilidad del código, además de reducir la repetición innecesaria de sentencias en otros métodos, reduciendo así las líneas totales de código. Además, contiene funciones que se encargan de leer y tratar la base de datos.

Finalmente, el último apartado consta únicamente de funciones de logging que permiten escribir mensajes en el archivo de logging o informar de errores.

Con relación a la forma en que se han hecho los estudios, el procedimiento a seguir en todos ellos ha sido el siguiente: en primer lugar, se ha procedido a leer las dimensiones de análisis requeridas para el estudio. En segundo lugar, los datos obtenidos han sido tratados. Por ejemplo, si el estudio se hiciese con la combinación de dos dimensiones, primero tendrían que leerse ambas y posteriormente combinarse en un único set de datos. En tercer y último lugar, se ejecuta un bucle encargado de hacer las predicciones por cada uno de los sets de datos obtenidos en el paso anterior. Dentro de cada iteración, se crea un objeto Predictor con las dimensiones de análisis y los sets de datos que correspondan. Acto seguido, se obtiene la configuración óptima para el estimador que se utilizará a la hora de realizar las predicciones del objeto Predictor. Y, finalmente, se ejecuta la predicción deseada con el método *predict*.

3.2 Predicciones y resultados

El proyecto se ha desarrollado poco a poco, aumentando en cada estudio la cantidad de datos a tratar y las dimensiones a considerar. En todas las pruebas, se han dividido los datos usados en dos sets: un set de entrenamiento, que representa el 70% del total de estos, y un set de prueba que representa el 30% restante. Asimismo, en cada prueba realizada, ha habido dos variantes principales: una primera variante ejecutada con los datos más recientes y un estimador DT, y una segunda variante ejecutada con datos recientes y un estimador RF.

Referente a las pruebas con DT's, como hemos visto en el apartado de fundamentos teóricos de estos, una de las grandes ventajas de su uso no es solo su velocidad, sino que es uno de los pocos que están capacitados para justificar sus decisiones. Dada la naturaleza de este proyecto, es un modelo ideal y necesario: mediante la justificación de un árbol de

decisión, podríamos dar con la respuesta a la incógnita de si hay algún modo de saber si un texto ha sido escrito por un hombre o una mujer y, si lo hay, qué modo es. Una vez el árbol es generado y entrenado, se procede a predecir el 30% de datos de prueba y se obtienen las matrices de confusión de cada una de las dimensiones, además de la precisión obtenida en las predicciones.

Ahora bien, teniendo en cuenta esto, ¿por qué usar un RF? Aunque no nos permita conocer la línea de "pensamiento" que ha seguido el modelo para dar con una decisión, sabemos por lo descrito en el *Apartado 3.3* que, a la hora de lograr una buena predicción, es teóricamente mucho mejor que un DT. En los siguientes apartados, veremos si la teoría es cierta y se cumple en este trabajo.

Para acabar con esta introducción al desarrollo y resultados del proyecto, y concerniente al origen de los datos más recientes, el doctorando Damián Morales observó durante estos estudios que era posible que los datos originales tuviesen muestras que trampeaban los resultados. En otras palabras, que daban resultados erróneos por pura casualidad en la disposición de los datos. Por ese mismo motivo, se han realizado estudios con los datos originales y con datos modificados que subsanan este error. Es importante mencionarlo para algunos de los casos estudiados. No obstante, dado que los datos modificados son los correctos, de manera general, estudiaremos los resultados obtenidos con estos.

3.2.1 Una dimensión de análisis y sin distinción de edades

En este primer caso estudiado, por una parte, las predicciones se han hecho considerando únicamente cada dimensión de manera individual. Por otra parte, se han tenido en cuenta hombres y mujeres de todas las edades. Por lo tanto, se ha llevado a cabo con los ficheros de datos totalmente inclusivos, sin distinción de sexo ni edad. En otras palabras, las primeras pruebas se han realizado con Predictor's de una única dimensión y sin distinción de edad.

Conviene destacar los resultados obtenidos por primera vez en este caso. En la primera ejecución, al no configurar en absoluto el DecisionTreeClassifier, la predicción no fue tan buena como cabría esperar. La base de datos usada fue la base de datos original con fallas en la dimensión morfológica simplificada y pragmática, y la máxima precisión obtenida fue del 63,8% en la dimensión morfológica simplificada. Pero, como se ha podido observar en el *Apartado 3.2.2*, en la *Figura 6*, el árbol obtenido era monstruoso, totalmente sobre entrenado. Pero, después de realizar pruebas de manera exhaustiva, se consiguió dar con una combinación de parámetros que optimizaba el estimador. Ahora, el DT obtenido constaba de 99 nodos (nada comparado con los 21.335 nodos que había en el primer árbol generado), profundidad máxima de 8 y 50 hojas. Con este árbol, se ha obtenido un 70,07% de precisión. En otras palabras, se ha mejorado la precisión un 6,27%. Podemos entonces dar por demostrada la importancia de no sobre entrenar los árboles de decisión.

Pero se puede mejorar todavía más: al optimizar los parámetros con SH, se consigue en esa misma dimensión una precisión del 70,40%. Y sí, la mejora es mínima, pero lo realmente bueno y sorprendente de esta optimización es el árbol de decisión generado, que consta únicamente de cinco nodos, tres hojas y profundidad máxima de dos. En comparación, el último árbol consta de 94 nodos menos y con ambos se consigue prácticamente el mismo porcentaje de aciertos, por no hablar de la comparación que se podría hacer entre la *Figura 12* y el árbol de la *Figura 6*.

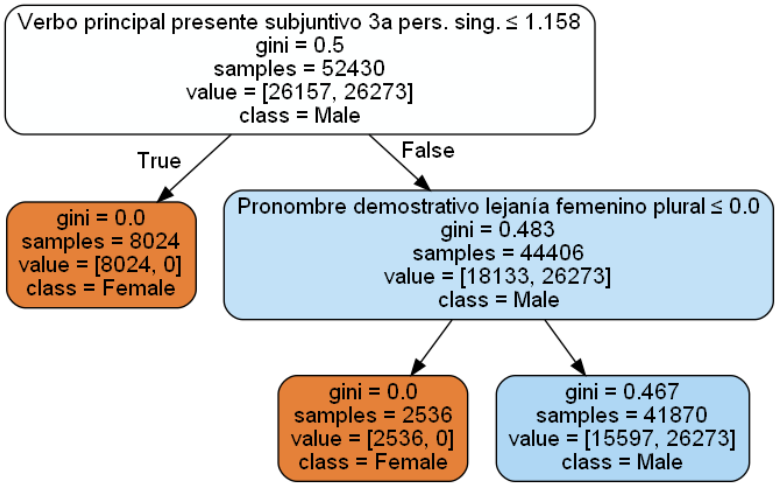

Figura 12. Árbol de decisión optimizado de la dimensión morfológica simplificada.

Al utilizar los datos modificados, los resultados han sido muy distintos. La que era antes la mejor dimensión ahora tiene una precisión del 54,18%. Entonces, el relevo ha sido tomado por la dimensión digital, con una precisión del 57,70%. No obstante, si obviamos las dimensiones modificadas, el resto han obtenido las mismas precisiones que van desde un 50% hasta un 55% aproximadamente. En la *Figura 13*, se pueden observar los dos primeros niveles del árbol de decisión de la dimensión digital, que en realidad consta de un total de 4 niveles de profundidad, 31 nodos 16 hojas.

Los resultados óptimos obtenidos con los bosques aleatorios y los datos modificados son algo mejores que los obtenidos con los DT. La media aritmética de mejora en la precisión de las predicciones es de un 1,005%. Este cálculo ha sido realizado con las matrices de confusión presentes en la *Figura 14* y la *Figura 15*.

Respecto a la parametrización con el modelo Random Forest, este caso merece una mención especial. Como hemos podido comprobar de buena mano en el *Apartado 2.4.4*, el Halving Sucesivo es el método más rápido con diferencia. Pero este método se comprobó con DT's. El caso de los RF's es muy distinto.

Tal y como se ha hecho con los DT's, para los RF's se ha buscado la configuración óptima con tal de obtener los mejores resultados posibles. Y así es como se hizo con los bosques aleatorios… Al menos en este primer caso. Teniendo en cuenta la cantidad de parámetros que tiene este modelo y los valores que puede tomar, la instancia HalvingGridSearchCV necesitó más de dos horas para dar con las combinaciones óptimas de parámetros. Dichos parámetros se pueden encontrar en el *Código 3*. Esto confirma la teoría de que los bosques aleatorios son realmente complicados de configurar, hay demasiadas variables a tener en cuenta. Así pues, después de ver cuánto tardaba el método más rápido, decidí configurar manualmente los parámetros de los RF's basándome en las combinaciones encontradas mediante SH. Acto seguido, comparé los resultados que obtenía con estos parámetros manuales versus los resultados óptimos y resultó que, de media, apenas había diferencia en la precisión de las predicciones. Esto significa que, en las siguientes pruebas, todos los random forest usarán los mismos parámetros con tal de obtener buenos resultados sin tener que dedicar tanto tiempo a la ejecución del algoritmo.

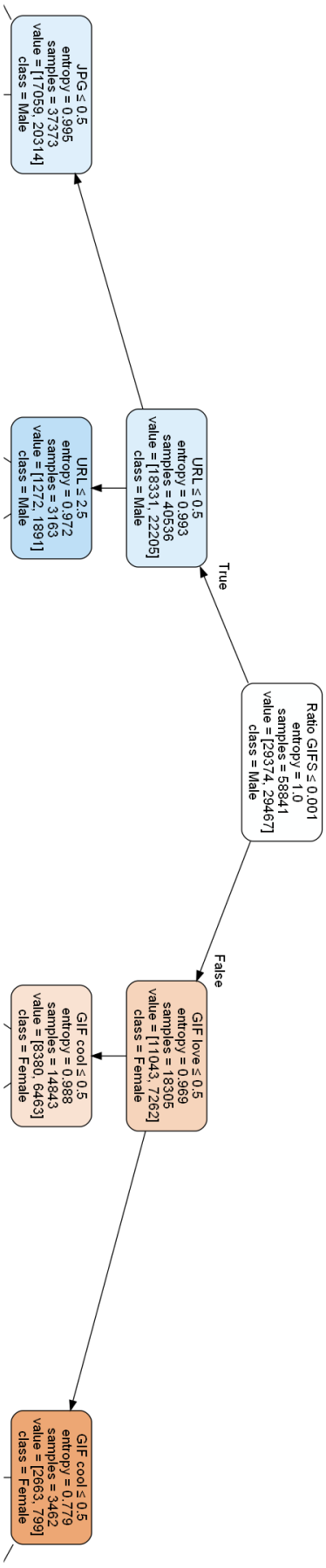

Figura 13. Dos niveles del árbol de decisión optimizado de la dimensión morfológica simplificada.

Figura 14. Precisión y matrices de confusión con DT's en la prueba 3.2.1.

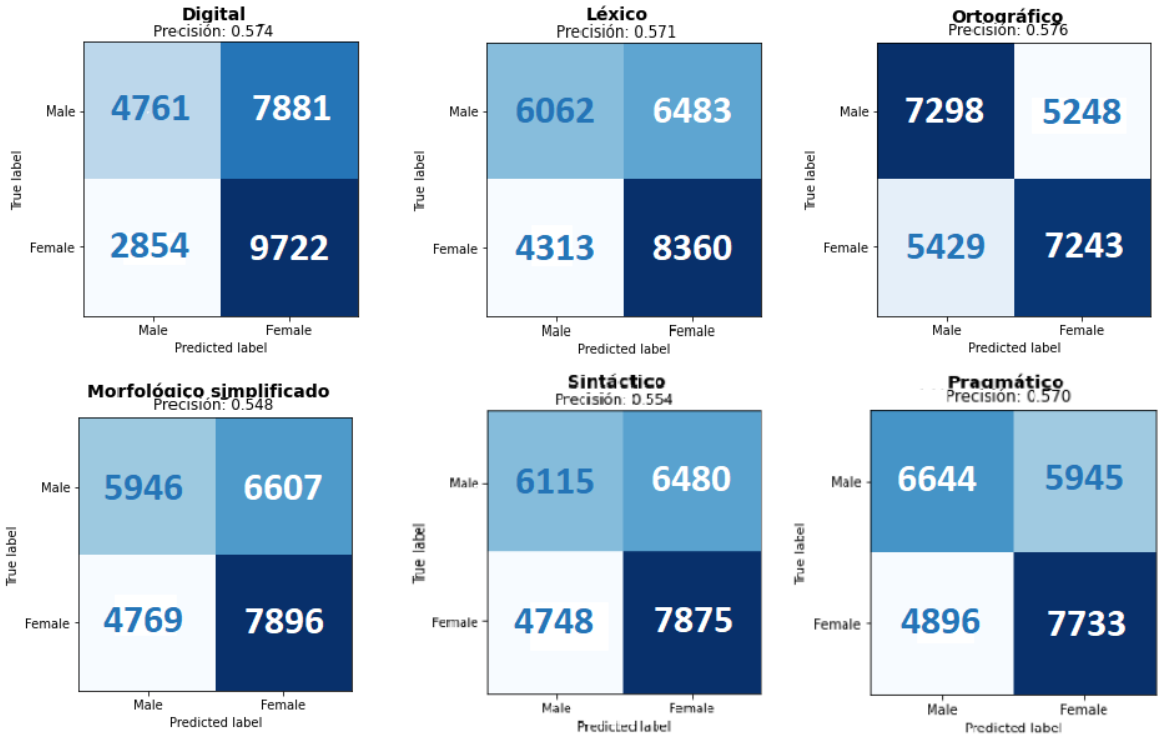

Figura 15. Precisión y matrices de confusión con RF's óptimos en la prueba 3.2.1.

```
DEBUG:log_HalvingGridSearchCV:804:Digital:
      CANDIDATOS:             [6048, 2016, 672, 224, 75, 25, 9, 3]
      MEJORES PARAMETROS:     {'criterion': 'entropy', 'max_depth': 8,
'max_features': 'sqrt', 'min_samples_leaf': 5, 'min_samples_split': 20,
'n_estimators': 100}
      MEJOR PUNTUACION:       0.5769327958606582

INFO:predict:165:Precision en dim Digital: 0.574311993655326
DEBUG:log_HalvingGridSearchCV:804:Léxico:
      CANDIDATOS:             [6048, 2016, 672, 224, 25, 9, 3]
      MEJORES PARAMETROS:     {'criterion': 'gini', 'max_depth': 12,
'max_features': 'sqrt', 'min_samples_leaf': 5, 'min_samples_split': 20,
'n_estimators': 250}
      MEJOR PUNTUACION:       0.560808615606724

INFO:predict:165:Precision en dim Léxico: 0.5718930922357046
DEBUG:log_HalvingGridSearchCV:804:Morfológico simplificado:
      CANDIDATOS:             [6048, 2016, 672, 224, 75, 25, 9, 3]
      MEJORES PARAMETROS:     {'criterion': 'gini', 'max_depth': 8,
'max_features': 'log2', 'min_samples_leaf': 15, 'min_samples_split': 30,
'n_estimators': 225}
      MEJOR PUNTUACION:       0.5535888334035256

INFO:predict:165:Precision en dim Morfológico simplificado:
0.548893647396718
DEBUG:log_HalvingGridSearchCV:804:Ortográfico:
      CANDIDATOS:             [6048, 2016, 672, 224, 75, 25, 9, 3]
      MEJORES PARAMETROS:     {'criterion': 'gini', 'max_depth': 10,
'max_features': 'log2', 'min_samples_leaf': 5, 'min_samples_split': 35,
'n_estimators': 250}
      MEJOR PUNTUACION:       0.572961915648878

INFO:predict:165:Precision en dim Ortográfico: 0.5766119438496312
DEBUG:log_HalvingGridSearchCV:804:Pragmático:
      CANDIDATOS:             [6048, 2016, 672, 224, 75, 25, 9, 3]
      MEJORES PARAMETROS:     {'criterion': 'gini', 'max_depth': 8,
'max_features': None, 'min_samples_leaf': 20, 'min_samples_split': 30,
'n_estimators': 175}
      MEJOR PUNTUACION:       0.5683171890981289

INFO:predict:165:Precision en dim Pragmático: 0.5701086525497661
DEBUG:log_HalvingGridSearchCV:804:Sintáctico:
      CANDIDATOS:             [6048, 2016, 672, 224, 75, 25, 9, 3]
      MEJORES PARAMETROS:     {'criterion': 'gini', 'max_depth': 10,
'max_features': 'log2', 'min_samples_leaf': 5, 'min_samples_split': 35,
'n_estimators': 200}
      MEJOR PUNTUACION:       0.5545394380602852

INFO:predict:165:Precision en dim Digital: 0.554762471250684
```

Código 3. Logging de combinaciones óptimas de un RF para cada dimensión sin distinción de edad.

Otro rasgo que merece la pena mencionar es el patrón que se puede observar en casi todas las matrices de confusión obtenidas. Las matrices de confusión suelen mostrar los valores TP/TN/FP/FN (True Positive, True Negative, False Positive, False Negative). En este caso, simplemente usaremos TP y FP, ya que ni Hombre ni Mujer son atributos que podríamos calificar como positivo o negativo. El caso es que en prácticamente todos los casos en la *Figura 14*, el número de FP de los hombres es superior al de las mujeres, mientras

que para las matrices de confusión de los random forest, sucede lo contrario. De momento, dejaremos constancia de este hecho y veremos si el patrón se repite.

Para acabar, volvamos a la *Figura 13*. El atributo presente en la raíz del árbol es el atributo "Ratio GIFS". Así pues, por el momento, parece ser el atributo que permite diferenciar con más facilidad el sexo del autor del mensaje en la dimensión digital. Ahora bien, dependiendo de qué camino siga el objeto por el árbol, el objeto será clasificado como hombre o como mujer. En estos primeros niveles, podemos ver que los atributos que más ganancia de información consiguen son "URL", "JPG", "GIF love" y "GIF cool". Es más, si los nodos a los que llega el objeto no clasificado son aquellos que representan los atributos "URL" y "JPG", el objeto será un hombre, al menos en estos primeros niveles del árbol. En caso contrario, el objeto será clasificado como una mujer. Esto puede indicar que, según el árbol obtenido, los hombres sin distinción de edad son más propensos a enviar links de páginas web e imágenes. Asimismo, la cantidad de GIF's de amor y de GIF's chulos parecen indicar que lo más probable es que el mensaje sea de una mujer.

3.2.2 Dimensiones tratadas de dos en dos

Para esta prueba, tratamos de ampliar nuestra base de datos para lograr encontrar una mejor precisión. Los resultados más prometedores han sido devueltos por la combinación de las dimensiones digital y léxico, que incluye la mejor dimensión encontrada en la prueba del apartado anterior.

Tal y como sucedía en la primera prueba, no hay un vencedor que destaque en lo que a precisión se refiere. No obstante, sí que se cumple un patrón: siempre que se usa la dimensión digital, la precisión aumenta. Toda dimensión que se combine con la digital obtiene una precisión mínima de 57,09%, cosa que no puede hacer ninguna otra. La precisión media es de 56,66%. La precisión mínima es de 54,46% (morfológico simplificado + sintáctico) y la máxima es de 59,96% (digital + léxico).

De nuevo, cuando se usan random forests, los resultados mejoran. Esta vez, las combinaciones con la dimensión digital no bajan del 58,72%, por lo que el patrón se confirma. La precisión media es de 58,19%, obteniendo así un 1,53% más de precisión media. La precisión mínima es del 56,37% (morfológico simplificado + sintáctico) y la máxima es del 60,38% (digital + léxico).

En la *Figura 16* se encuentra una parte del árbol de decisión de la combinación con mejores resultados. El árbol en su totalidad consta de cuatro niveles de profundidad, 30 nodos y 16 hojas. Y en la 17 y la 18 se pueden leer las matrices de confusión de esta prueba.

En este nuevo árbol obtenido, obtenemos información interesante. Por un lado, tanto en la raíz como en la parte derecha del árbol, los atributos de los nodos son exactamente los mismos que encontramos en el apartado anterior. Dicho de otra manera, se confirma que, hasta el momento, el atributo más importante a la hora de clasificar un objeto es la ratio de GIF's que envía el autor. Además, en estos primeros niveles, los atributos que tienen más probabilidad de clasificar al autor anónimo como femenino son "GIF love" y "GIF cool" de nuevo. Por otro lado, los atributos que parecen indicar que el emisor del mensaje es un hombre ahora son distintos, y son de la dimensión léxica. Estos son "Léxico apreciativo sufijado" (palabras acabadas en -ato, -azo, -in/-ino/-iño, -illo…), "Ratio letras/palabras" y "TTR Lema". El "TTR Lema" es un atributo que permite medir la riqueza lingüística en función de la variación léxica, es decir, la cantidad de palabras diferentes que contiene un texto.

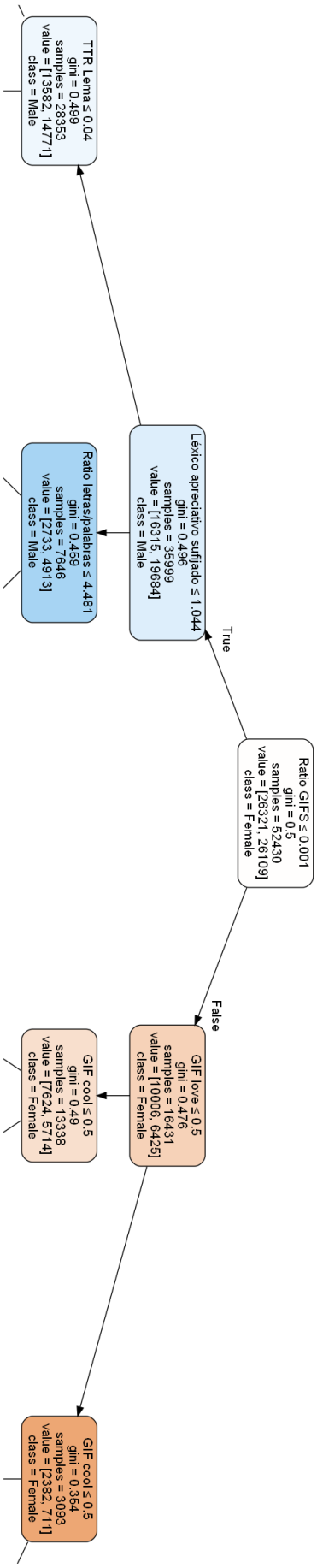

Figura 16. Dos niveles del árbol de decisión optimizado para la combinación Digital + Léxico.

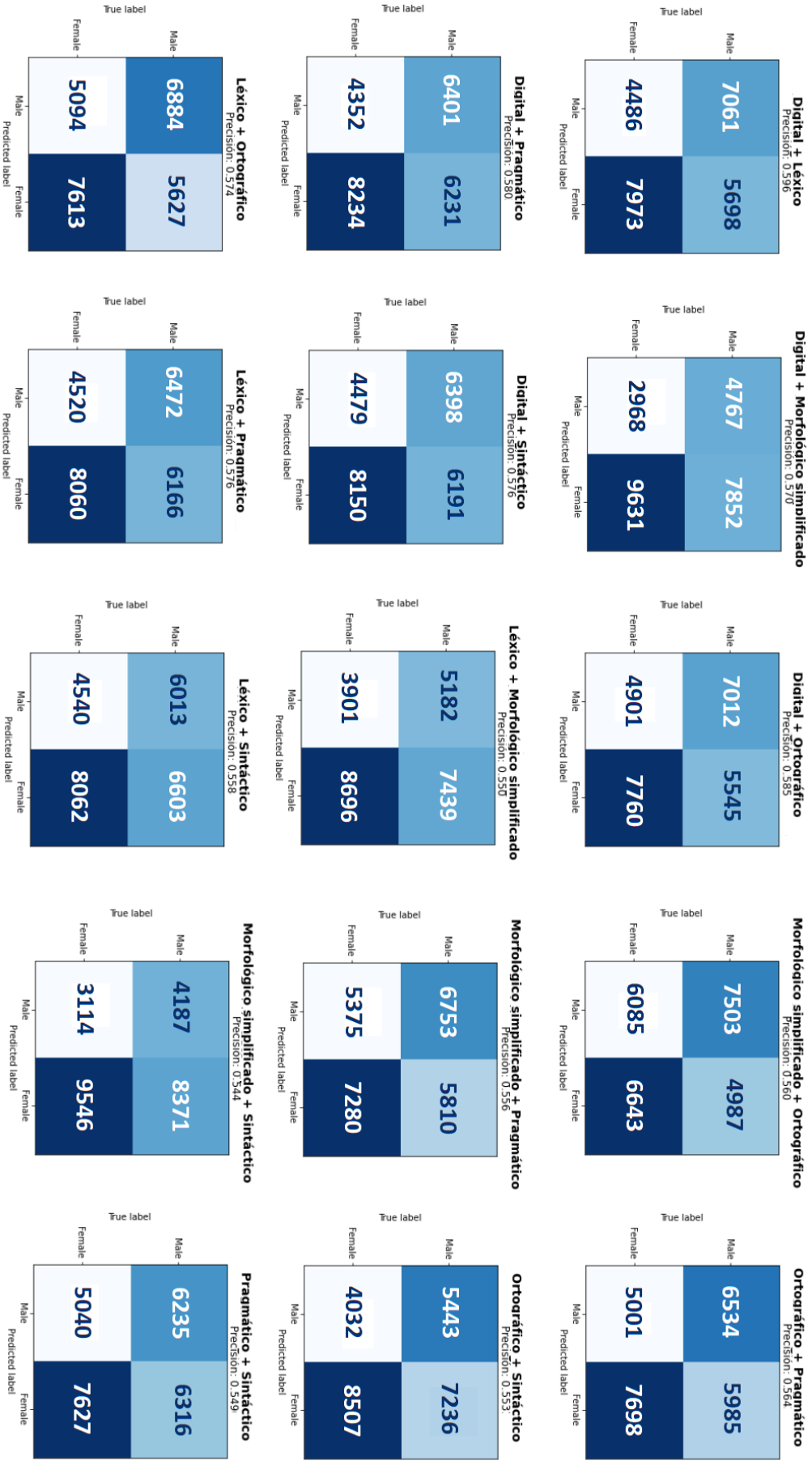

Figura 17. Precisión y matrices de confusión con DT's óptimos en la prueba 3.2.2.

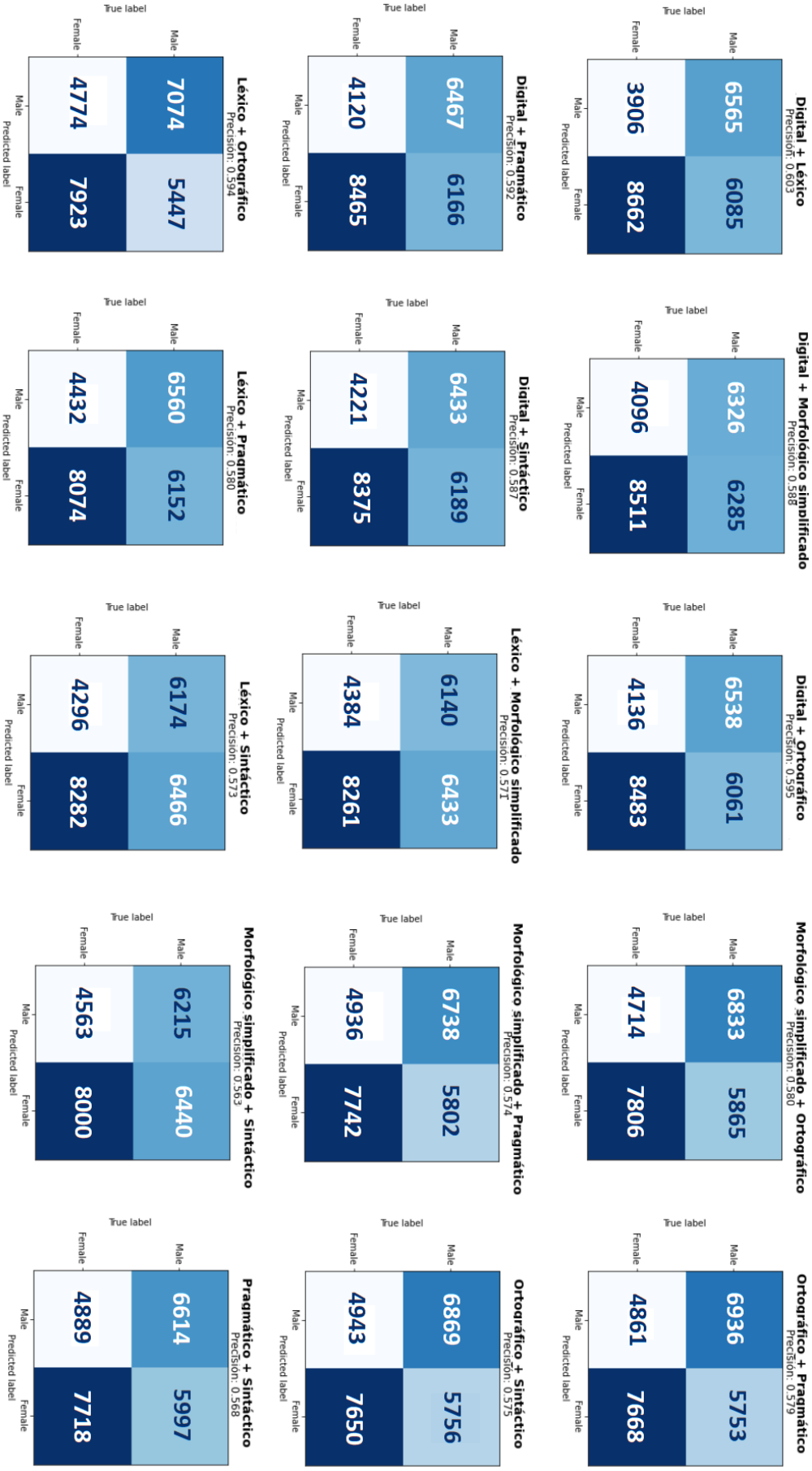

Figura 18. Precisión y matrices de confusión con RF's en la prueba 3.2.2.

Esto puede significar que, por lo general y sin tener en cuenta la franja de edad, los hombres tienden a usar un léxico más rico en sus comunicaciones vía Internet. Podemos sacar esta conclusión por los atributos “Ratio letras/palabras” y “TTR Lema”. Ahora bien, esto no significa que el contenido sea mejor ni más importante ni nada por el estilo, simplemente que, a la hora de escribir, parece ser que existe cierta tendencia a usar un vocabulario más amplio y variado.

Por último, en esta prueba sucede lo mismo que en la anterior: el número de FP de hombres es superior al de las mujeres para los DT, mientras que sucede justo lo contrario al usar RF.

3.2.3 Combinación Digital + Léxico + X

Vistos los resultados de las pruebas anteriores, si queremos conseguir todavía más precisión, un paso lógico a seguir es añadir más dimensiones a la combinación más prometedora. De este modo, se aumenta la cantidad de información a la que tiene acceso el árbol de decisión y la posibilidad de acertar puede aumentar.

Contrariamente a lo que deseábamos, los resultados no han sido tan buenos como se esperaba. Conviene subrayar que, en la prueba del *Apartado 3.2.*2, usando estimadores DT, se acertó el 59,62% de las veces. Ahora, la precisión mínima obtenida es igual al 59,03% de aciertos y la máxima es igual al 59,79% de aciertos en el caso de la combinación Digital + Léxico + Ortográfico. Asimismo, se ha acertado de media en el 59,4125% de las predicciones. Si comparamos estos resultados con la prueba anterior, podemos constatar que, aunque la precisión máxima apenas ha subido, la precisión media ha aumentado un 2,7525%.

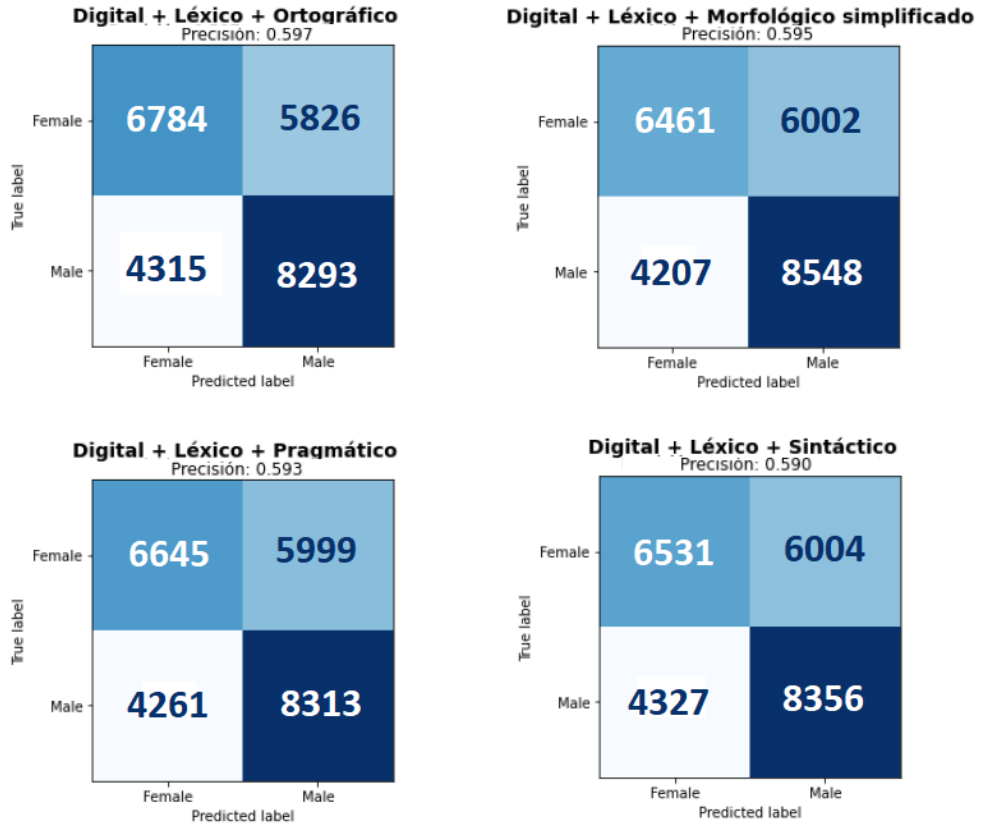

Figura 19. Precisión y matrices de confusión con DT’s óptimos en la prueba 3.2.3.

Con los RF, sucede algo similar. Apenas hay mejora respecto de la precisión obtenida en el *Apartado 3.2.2*. Consecuentemente, esto nos indica que, con los bosques aleatorios, seguimos obteniendo una mejora de alrededor del 1% respecto de las predicciones hechas con un DecisionTreeClassifier. En concreto, la media de aciertos es del 60,415%, que supone una mejora del 1,0025%.

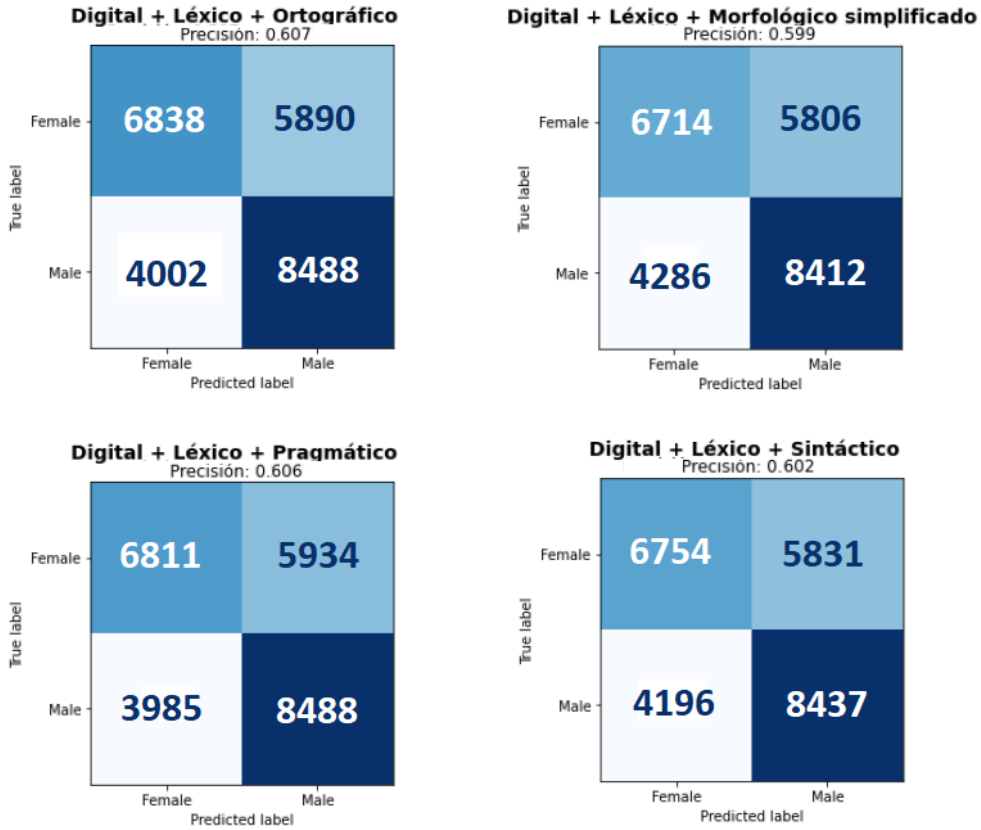

Figura 20. Precisión y matrices de confusión con RF's en la prueba 3.2.3.

En esta ocasión, la dimensión del árbol es de un tamaño considerable. Por ese motivo, y no aportar una gran mejora, no será incluido en esta documentación. Concretamente, dispone de 311 nodos repartidos en ocho niveles de profundidad, con 156 hojas. Al considerar tres dimensiones de análisis y, por lo tanto, muchísimos atributos y datos, es inevitable obtener árboles de gran tamaño.

3.2.4 Una dimensión de análisis con distinción de edad

Como ya no somos capaces de conseguir ninguna mejora notable, debemos encontrar alguna forma alternativa que nos permita hacerlo. Una de las vías probadas ha sido hacer uso de los datos que hacen referencia a la edad de los autores de mensajes de texto.

En la primera prueba, las precisiones obtenidas vuelven a porcentajes más moderados. Esta vez, el porcentaje mínimo es 53,56% y el máximo es obtenido con la dimensión léxica en la franja de los treinta, con un porcentaje de aciertos del 62,40%. Para poder comparar estas predicciones con las del bosque aleatorio, debemos tener en mente que, de media, se ha acertado en las predicciones el 56,11% de las ocasiones. El árbol obtenido es un árbol pequeño, con cuatro niveles de profundidad, 31 nodos y 16 hojas.

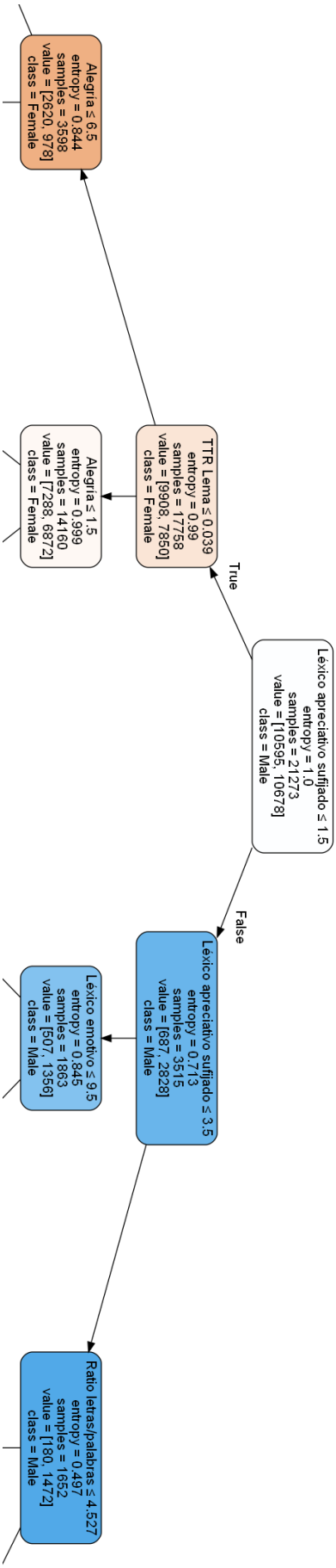

Figura 21. Dos niveles del árbol de decisión optimizado para la dimensión léxica.

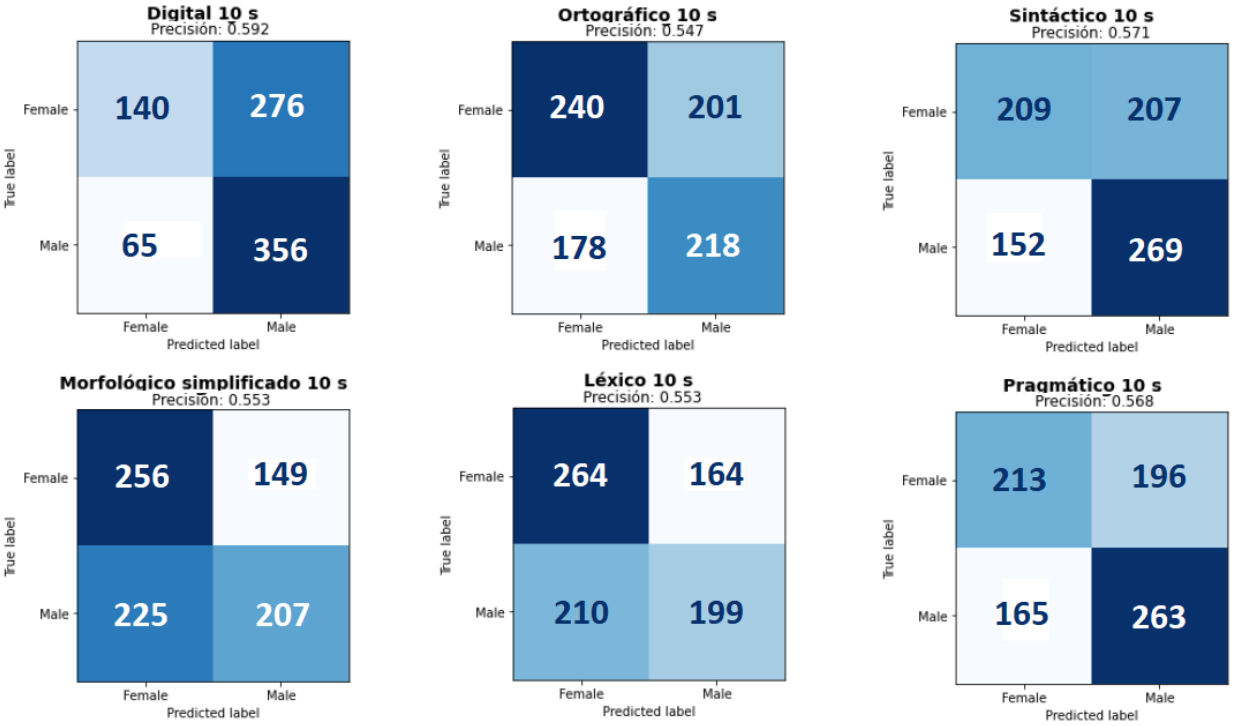

Figura 22. Precisión y matrices de confusión con DT's óptimos en la prueba 3.2.4 para la franja "10s".

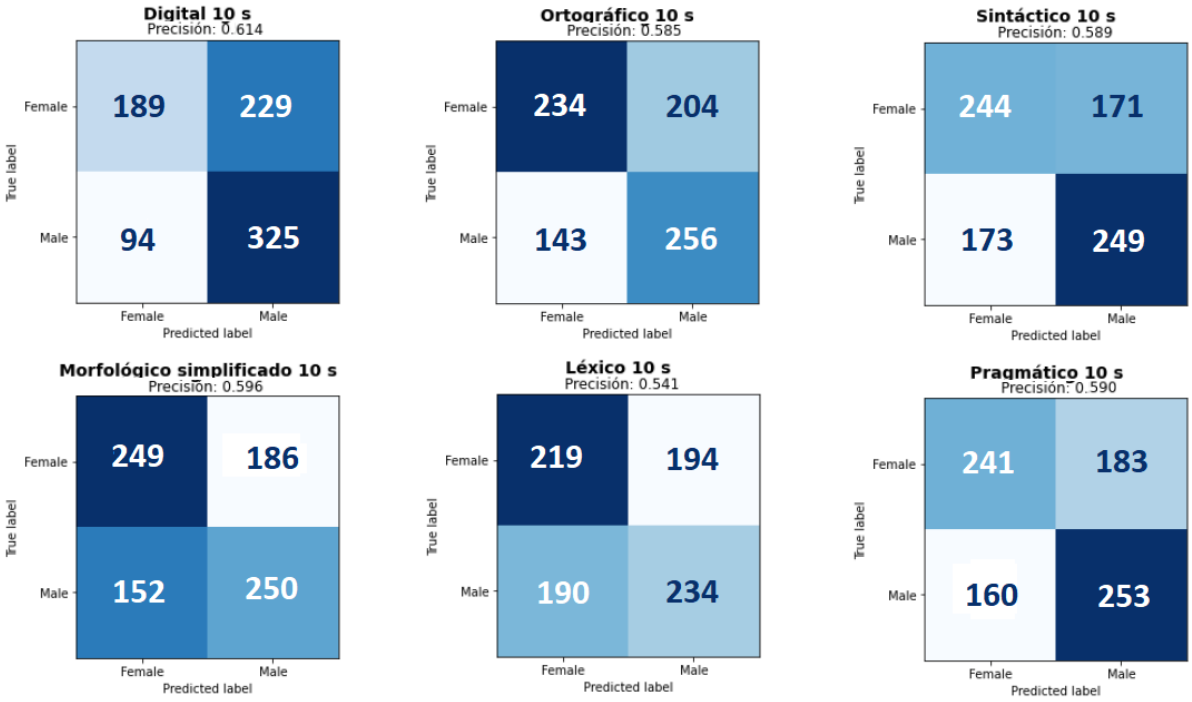

Figura 23. Precisión y matrices de confusión con RF's en la prueba 3.2.4 para la franja "10s".

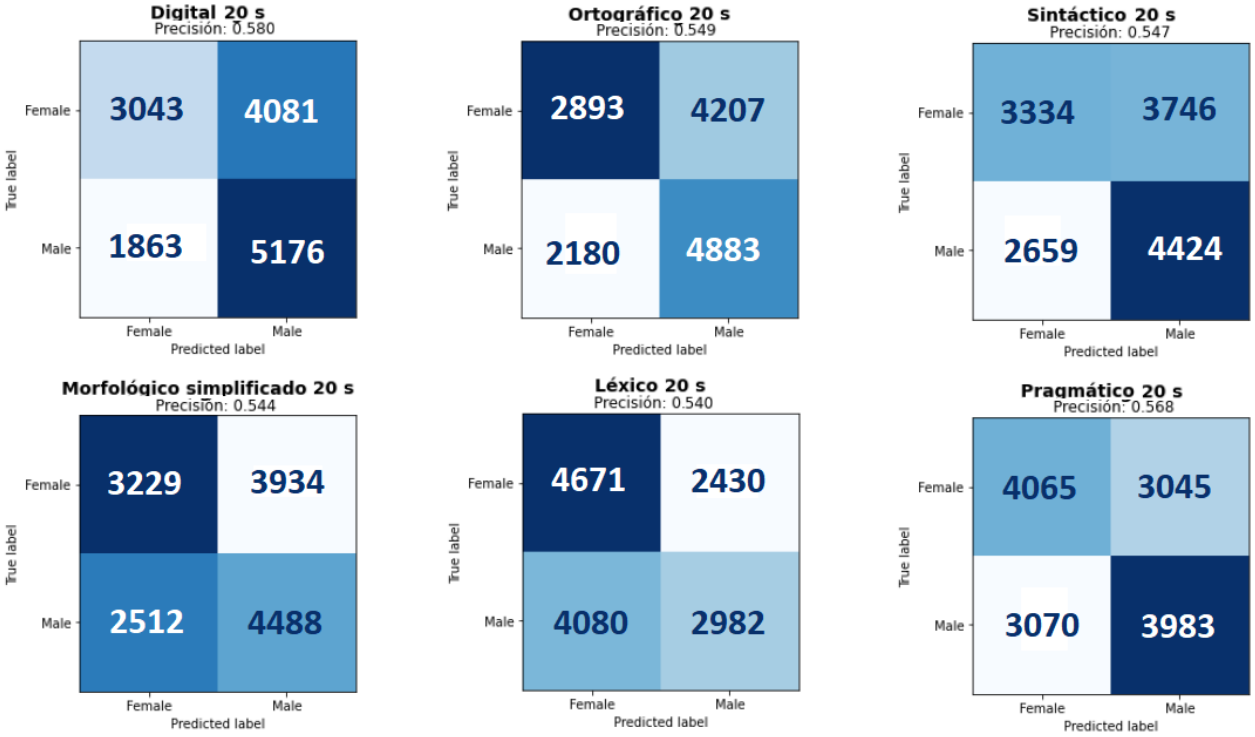

Figura 24. Precisión y matrices de confusión con DT's óptimos en la prueba 3.2.4 para la franja "20s".

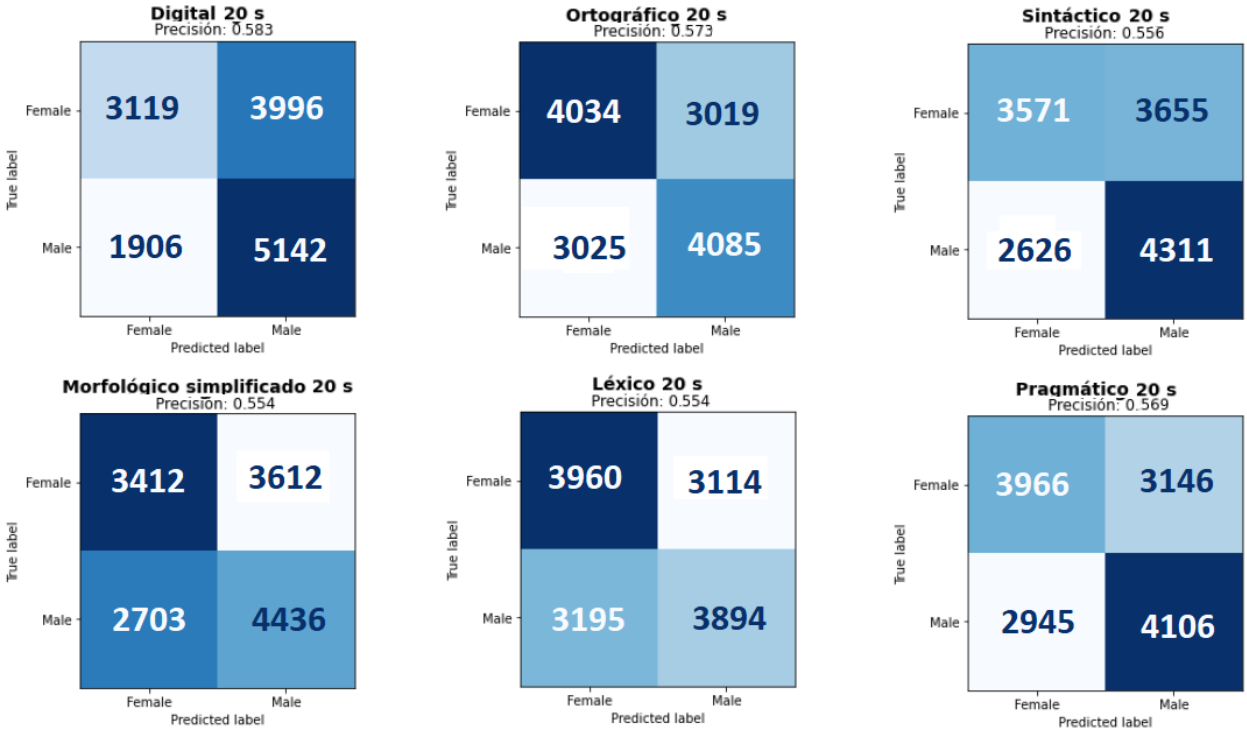

Figura 25. Precisión y matrices de confusión con RF's en la prueba 3.2.4 para la franja "20s".

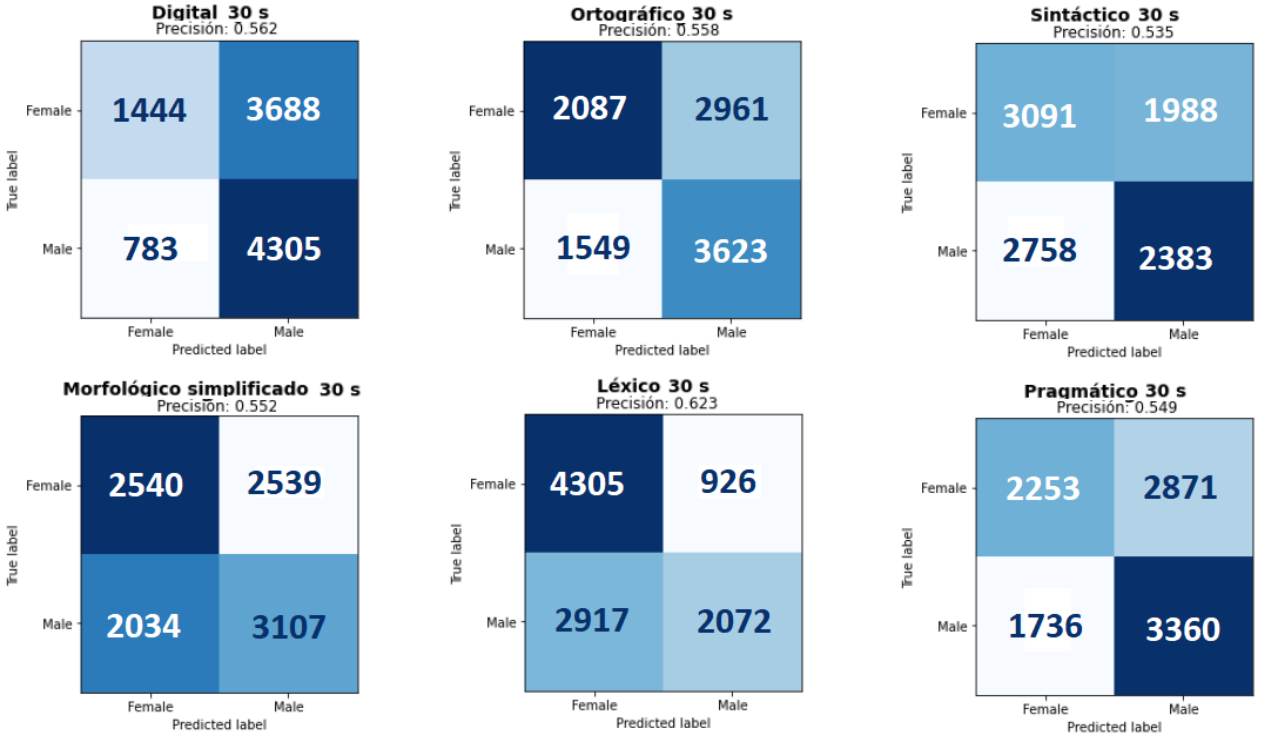

Figura 26. Precisión y matrices de confusión con DT's óptimos en la prueba 3.2.4 para la franja "30s".

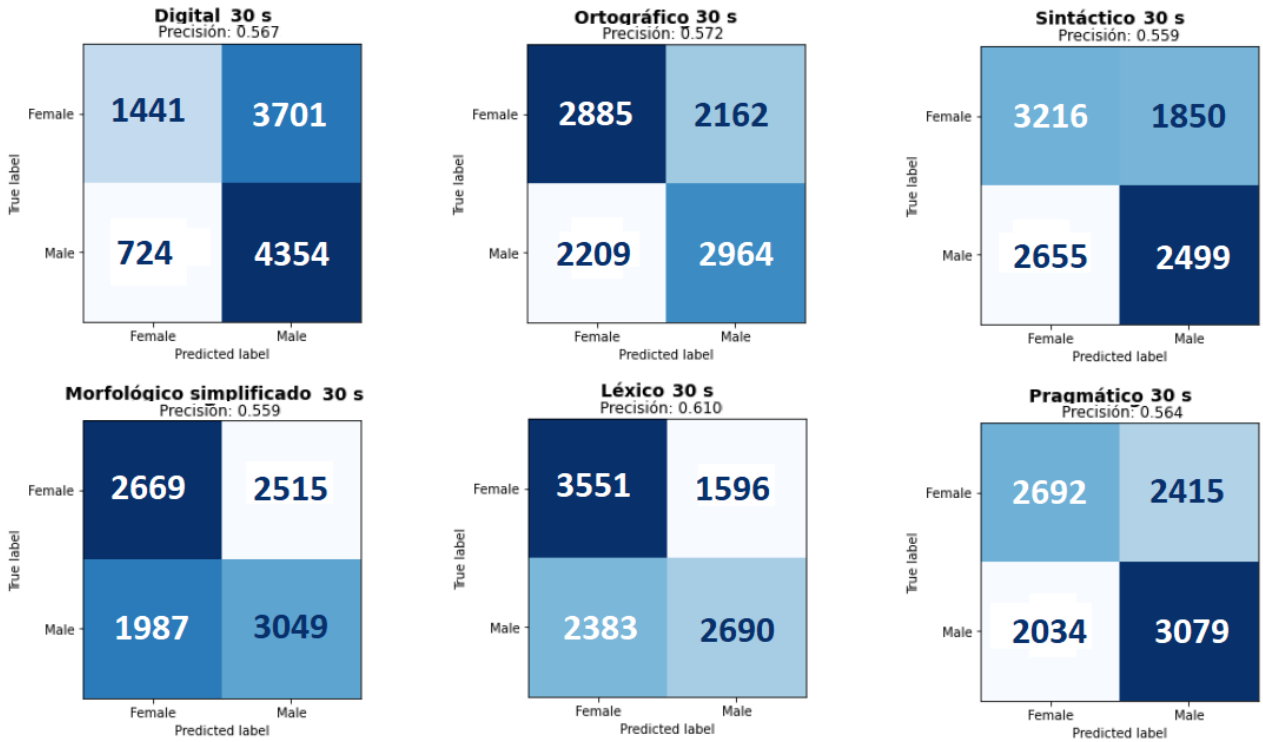

Figura 27. Precisión y matrices de confusión con RF's en la prueba 3.2.4 para la franja "30s".

Como siempre, en la segunda prueba con los RandomForestClassifier, los resultados son algo mejores. La predicción es acertada el 57,45% de las veces. O sea, se ha mejorado la precisión en un 1,34%.

En esta última prueba, podemos constatar dos hechos de interés. El primero es que, a pesar de usar únicamente una dimensión de análisis, los resultados son mucho mejores que en la primera prueba estudiada. Hasta el momento, la precisión más elevada ha sido obtenida con este análisis, es decir, usando una única dimensión y distinción de edad. Esto puede resultar incoherente, dado que ahora usamos una cantidad de datos reducida al estudiar únicamente una franja de edad, por lo que el árbol debería estar menos informado. Pero, aun así, conseguimos superar las precisiones de las pruebas anteriores. Además, ¿Por qué sucede esto en la franja de los treinta, y no en el resto?

El segundo es que ahora las tornas han cambiado. El patrón visto hasta ahora en las matrices de confusión es que, al usar árboles de decisión, el número de falsos positivos en hombres era superior al de las mujeres, mientras que para los bosques aleatorios sucedía lo contrario. Sin embargo, ahora ha pasado justo lo contrario, el patrón se ha dejado de cumplir.

En lo que toca al árbol de decisión, ahora disponemos de un árbol únicamente léxico. No como en los casos anteriores, donde teníamos árboles con atributos de la dimensión digital y léxica. Ahora, el atributo en el nodo raíz es el léxico apreciativo subjetivo, un atributo que ya vimos que era importante en esta dimensión. De nuevo, sigue siendo un nodo con clase “Hombre”.

En los niveles inferiores, en la parte izquierda, los nodos con clase “Mujer” almacenan los atributos “TTR Lema” y “Alegría”. Así pues, la teoría propuesta en el *Apartado 3.2.2* respecto al atributo TTR Lema queda desmontada. No obstante, podemos confirmar que se trata de un atributo importante a la hora de clasificar un objeto con la dimensión léxica, ya que, hasta el momento, siempre ha estado en los primeros niveles de los árboles de decisión. En la parte derecha se encuentran los nodos con clase “Hombre”. Estos almacenan los atributos “Léxico apreciativo sufijado”, “Léxico emotivo” y “Ratio letras/palabras”.

Debido al tamaño y a la disposición de los árboles de decisión de la dimensión léxica para la franja “10s” y “20s”, es imposible recortar los primeros niveles de estos sin que sean totalmente ininteligibles en este documento. No obstante, puedo describirlos. En el caso de la dimensión léxica para las personas de entre diez y veinte años, los atributos en los niveles superiores son “Ratio letras/palabras”, “Palabras de 4 a 6 caracteres”, “Palabras de 6 o más caracteres” y “Palabras de 1 a 3 caracteres”. Cabe destacar que hay un número reducido de muestras en esta franja de edad, por no decir que seguramente muchos de los mensajes son de autores que aún no tienen una capacidad de escritura o de habla propias de un adulto. Consecuentemente, es lógico pensar que en la dimensión léxica el modelo DecisionTreeClassifier no pueda encontrar patrones léxicos claros. Sin embargo, en esta franja vuelve a aparecer el atributo “Ratio letras/palabras” como en los árboles anteriores.

En la franja de los veinte, los atributos en los niveles superiores son “Formas no personales”, “Palabras de 6 o más caracteres”, “Riqueza léxica”, “Léxico apreciativo derivado”, “Léxico apreciativo sufijado” y “Condicional”. Al ser mensajes de personas de más edad y, por lo tanto, más inteligentes y con más experiencia en la escritura y el habla, los atributos dan más juego. Lo que me resulta realmente interesante en esta franja es que, una vez más, el léxico apreciativo es un atributo importante.

3.2.5 Dimensiones tratadas de dos en dos con distinción de edad

Se ejecutó una última prueba que combina todas las dimensiones de análisis de dos en dos, pero considerando únicamente la franja de los treinta, que es la franja en la que se han obtenido mejores precisiones.

Tal y como sucedía al añadir cada vez más dimensiones en pruebas anteriores, las precisiones aumentan. Obtenemos una precisión media de 57,56%, por lo que apenas hay mejoría, pero al menos la hay. En esta ocasión, el porcentaje máximo de veces que se ha acertado en la predicción es igual al 63,10% en el caso de la combinación Digital + Léxico. Entonces, se ha aumentado un 0,7% la precisión máxima. Conviene subrayar que es la misma combinación ganadora que en el *Apartado 3.2.*2.

Respecto a las pruebas de los random forests, el patrón visto hasta ahora se mantiene: las predicciones vuelven a mejorar. La media de aciertos es del 59,46%, lo que implica una mejora del 1,9% con exactamente los mismos datos y siendo la misma prueba. Al mismo tiempo, la precisión máxima obtenida alcanza el 64,27% en las dimensiones digital y léxica.

Para concluir, si comparamos la *Figura 28* con la *Figura* 16, podemos constatar que los atributos son prácticamente los mismos en ambos casos. Tanto en la franja de los treinta como para todas las franjas, los atributos que clasifican con más facilidad a los objetos no clasificados son los mismos. Podemos confirmar entonces que, para la dimensión digital, los atributos más importantes y que aportan mejores resultados son “Ratio GIF” y “GIF love”. Para la dimensión léxica, los que siempre aparecen en un árbol u otro en los niveles superiores de los mejores árboles son “Ratio letras/palabras” y “Léxico apreciativo sufijado”.

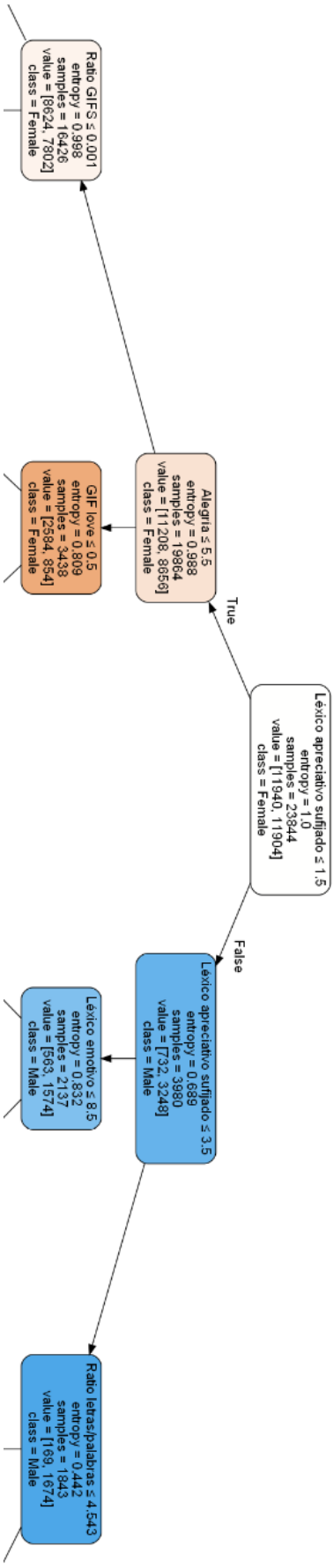

Figura 28. Dos niveles del árbol de decisión optimizado para las dimensiones digital y léxica en la franja “30s”.

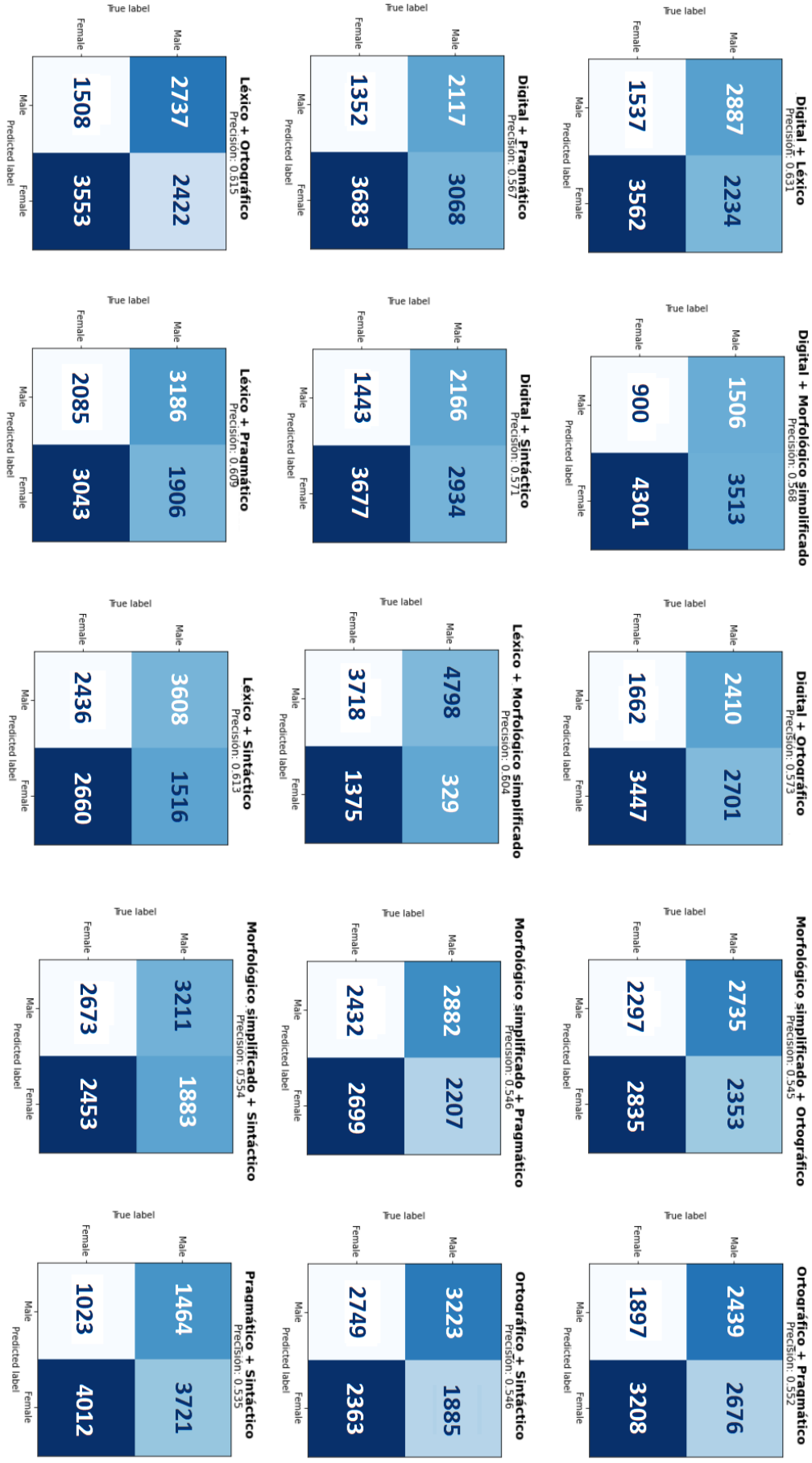

Figura 29. Precisión y matrices de confusión con DT's en la prueba 3.2.5 para la franja "30s".

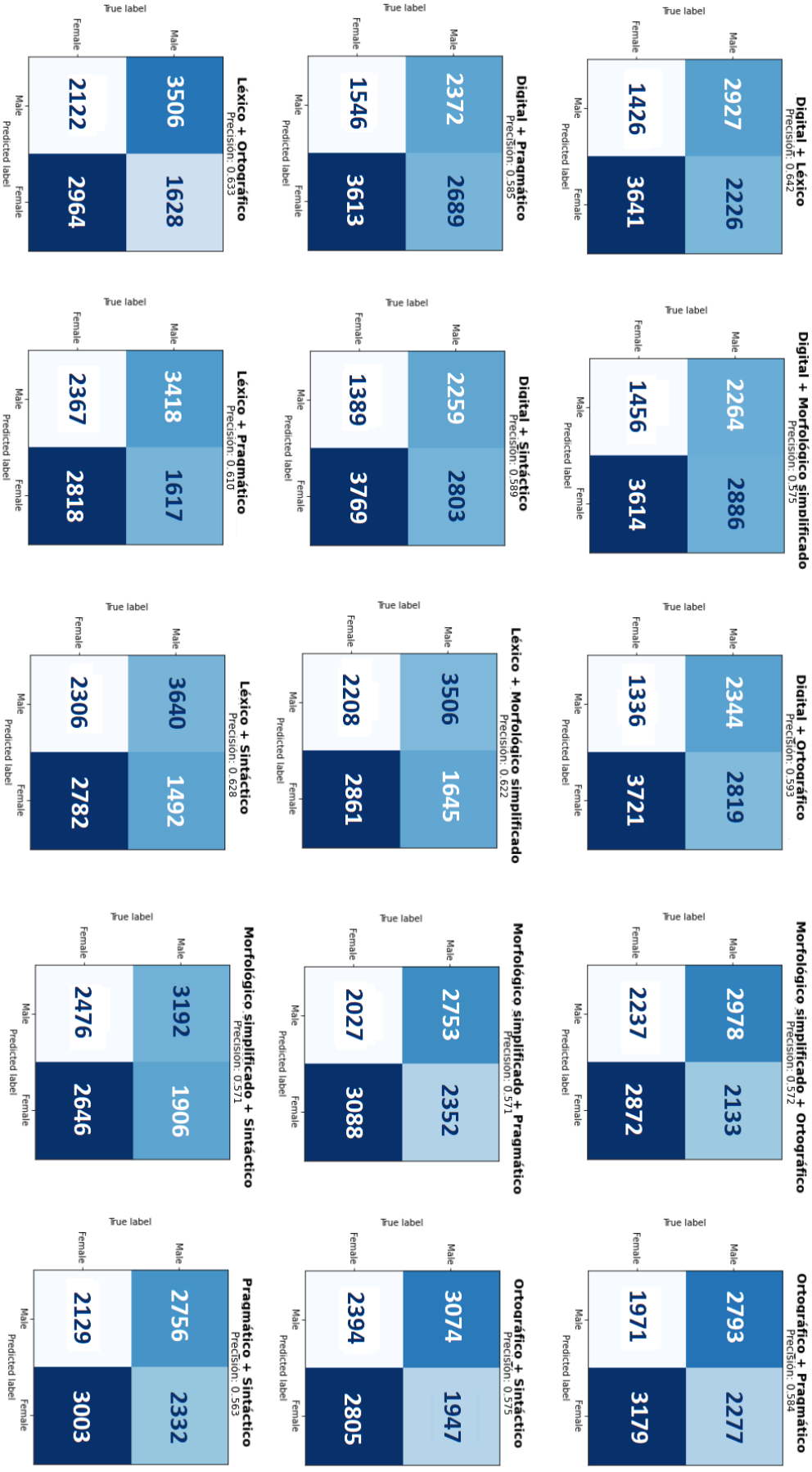

Figura 30. Precisión y matrices de confusión con RF's en la prueba 3.2.5 para la franja "30s".

3.3 Análisis de los resultados

Al haber terminado estas pruebas, se pueden observar diversos patrones. Y no, no me refiero al patrón que se busca en el proyecto del doctorando, sino a los patrones que buscamos en este TFG: patrones que dan indicios de cómo se puede mejorar el análisis de datos mediante IA y cómo obtener buenos resultados usando aprendizaje supervisado.

Estos patrones se pueden dividir en dos grupos. El primero de ellos describe qué podemos hacer con nuestra base de datos para obtener mejores predicciones. El segundo expone los patrones encontrados al comparar los DecisionTreeClassifier y los RandomForestClassifier, la importancia de su parametrización y la importancia de su optimización.

Después, me gustaría dedicar un último apartado a los resultados obtenidos al tener en cuenta las franjas de edad.

3.3.1 La importancia de la disposición de los datos

En un principio, los mejores resultados se han obtenido con la dimensión morfológica sin distinción de edad y con la dimensión pragmática en hombres y mujeres de más de treinta años. Pero la clarísima victoria de estas ante el resto hizo dudar al doctorando Morales. En consecuencia, decidió revisar los datos y enviar unos nuevos, ya divididos en sets de entrenamiento y de prueba.

Los resultados que se obtuvieron con estos nuevos datos no son los correspondientes a los resultados obtenidos en las pruebas expuestas. Fueron peores. Para tratar de darle un sentido a los nuevos resultados, se plantearon tres hipótesis.

En primer lugar, podría ser que el problema fuese causado por una mala partición de los datos en sets de entrenamiento y de prueba. Así pues, decidí consultar el número de hombres y mujeres de cada set, y observé que la división ahora no era como la que se hacía desde un principio. Por una parte, tanto con estos datos como con los originales, para todas las dimensiones, el número de hombres y mujeres en ambos sets era el mismo. En otras palabras, en los sets de entrenamiento, el número de hombres y mujeres es el mismo para todas las dimensiones. Y en los sets de test de cada dimensión sucede exactamente lo mismo. Se descartó consecuentemente la posibilidad de que se tratase de una falta de homogeneidad en las muestras. Por otra parte, la división realizada en el total de los datos no se hizo en la misma proporción. En lugar de dividir la base de datos en un conjunto de entrenamiento con un 70% y un conjunto de prueba con el 30% restante, se partió en sets de 90%-10%. Esto podría provocar que, aunque el árbol sea muy pequeño, este esté demasiado atado al conjunto de entrenamiento.

En segundo lugar, pensé que había algún tipo de pérdida de información en los nuevos datos. La mejor dimensión y la que daba mejores resultados era la morfológica simplificada y esto ya no era así. Resulta que, justo en las mejores dimensiones, se habían eliminado algunos atributos. Más concretamente, “Pronombres exclamativos”, de la dimensión morfológica y “Determinantes exclamativos” y “Oraciones enunciativas” en la dimensión pragmática. Por lo tanto, un razonamiento lógico sería pensar que esos atributos eran los que nos permitían obtener tales porcentajes.

Y, efectivamente, así fue. Damián eliminó a propósito estos atributos porque observó que, por pura casualidad, la disposición de los datos en estos atributos daba lugar a predicciones muy altas cuando no debería ser así. No sólo eso, sino que, al realizar la

partición de datos adecuadamente (en la proporción de siempre, 70%-30%), los resultados mejoraban también.

Una última observación que se puede realizar es que, mientras más dimensiones de análisis se tienen en cuenta, mayor precisión se logra. Es decir, mientras más grande sea la base de datos y mientras más se pueda entrenar un árbol de decisión (siempre teniendo en mente que la proporción de entrenamiento-prueba sea la adecuada), mejor será la precisión de sus predicciones.

En oposición, me gustaría retomar la pregunta que dejé en el aire al final del *Apartado 3.2.4*. ¿Cómo es posible que, usando muchos menos datos, consigamos la mejor precisión con diferencia? Resulta que no sólo importa la cantidad y la disposición, sino la calidad y qué significan esos datos. Es fácilmente constatable que usando una dimensión de análisis con la franja de edad "30s", obtenemos mejor precisión que con la combinación de tres dimensiones de análisis. Es decir, generamos mejores árboles con 34.066 datos que con 74.901 y una infinidad de atributos. En mi opinión, esto es debido a lo que representan estos datos, y es un resultado más representativo de la parte lingüística y filóloga del proyecto que de la parte informática. Estas últimas décadas, sobre todo a partir de los años noventa, la sociedad ha cambiado enormemente. Hoy en día, a raíz de miles de cambios históricos y socioculturales que han sucedido en tan poco tiempo (sobre todo teniendo en cuenta el auge tecnológico que hemos vivido), hay un amplio abanico de personalidades, géneros, gustos y motivaciones. El mundo es un lugar muy diversificado. Podríamos deducir que la victoria de esta franja de edad nos puede dar indicios de que esa diversidad es menos notable en adultos de treinta años o más, por la cual cosa encontrar un patrón que nos permita diferenciar mensajes de texto de hombres y mujeres es más sencillo.

Debemos tener en cuenta además que, en las otras dos franjas de edad, las precisiones son prácticamente idénticas. No hay una franja de edad que destaque claramente sobre la otra. Por un lado, esto puede deberse a que, para la franja "10s", el número de muestras es bastante inferior al número de muestras de la franja "20s". Esto podría ser la causa de la generación de árboles de decisión sub entrenados o poco informados. En otras palabras, es posible que, si dispusiésemos de más datos, la franja "10s" fuese mejor. Pero nunca lo sabremos. Por otro lado, tal y como he comentado antes, actualmente hay mucha variedad de gustos, géneros y opiniones entre los hombres y las mujeres en la franja de los 20. Se obtienen entonces resultados peores al no encontrar patrones claros en las muestras.

En conclusión, podemos destacar cuatro aspectos de la importancia de la base de datos en los algoritmos de aprendizaje supervisado. Primero, es de vital importancia revisar la disposición de los datos para evitar errores que puedan dar demasiados buenos resultados o resultados peores. Segundo, conviene mantener la misma partición de la base de datos durante el análisis de estos. Tercero, mientras más grande sea la base de datos, más información tendrá el algoritmo a su alcance, pudiendo devolver así resultados con un mayor conocimiento del problema. Y cuarto, el significado de los datos también se debe tener en consideración.

3.3.2 Decision Trees, Random Forests y optimización de parámetros

Gracias a las pruebas realizadas, hemos podido comprobar de primera mano que la parte teórica se cumple sin excepción. Para empezar, hemos visto lo importante que es parametrizar un estimador. Sin parametrización alguna, los estimadores se sobre entrenan sin control y se obtienen resultados notablemente peores. No sólo es importante parametrizar, sino que es imprescindible parametrizar de forma óptima. En mi opinión, es

fascinante ver como un árbol con 99 nodos y un árbol de 5 pueden tener precisiones prácticamente idénticas. En síntesis, hemos notado los daños de los inconvenientes de un árbol de decisión (su sensibilidad a datos erróneos y la posibilidad de overfitting) y hemos sabido corregirlos a posteriori.

De la misma forma, otro hecho teórico comprobado es que los bosques aleatorios son mejores que los árboles de decisión. En todos y cada uno de los casos estudiados, ha habido una mejora en la precisión. Pequeña, sí, pero eso es debido al tipo de proyecto en el que nos vemos involucrados. Si obviamos cuánto se ha mejorado, podemos concluir que los random forests son mejores que los árboles de decisión en lo que a predicciones se refiere. Recordemos que esto es debido a que un random forest no es tan sensible a valores extremos en las bases de datos y a que no padece tanto las consecuencias del overfitting. Aun así, sigue teniendo un gran inconveniente, y es que no podemos conseguir justificación alguna del motivo de sus elecciones.

Para terminar, me gustaría comparar los resultados obtenidos con los resultados de las competiciones mencionadas en la introducción. En este TFG, la mejor precisión obtenida ha sido 0,6427 y la precisión media 0,5946. Podemos decir entonces que hemos superado al mejor equipo del año 2013, que obtuvo un máximo de aciertos del 38,94%. Hemos superado también al mejor equipo del 2016, con una precisión media de 0,5258. Así pues, teniendo en cuenta que nuestro trabajo no tiene fines competitivos y que no he dispuesto de máquinas computacionalmente potentes que pudiesen computar optimizaciones de hiperparámetros en un menor tiempo, estoy contento con el resultado obtenido. Pienso que, si hubiese utilizado otros modelos o si hubiese tenido más poder de cálculo, habría podido obtener mejores resultados. Por un lado, los modelos utilizados no han sido escogidos con fines competitivos, sino con fines académicos. Es decir, nuestro interés principal no era sólo conseguir una buena precisión, sino una buena justificación. Por otro lado, en la mayoría de los casos no he podido optimizar la parametrización de los random forest debido al tiempo que esto conllevaba. Con más cores y más potencia de cálculo, habría sido capaz de probar más combinaciones de hiperparámetros con SH para dar con la mejor combinación posible en cada uno de los casos y conseguir así una mejor puntuación.

3.4 Síntesis

Al implementar las pruebas, se han creado dos ficheros principales: un fichero en el que leer los resultados obtenidos y un fichero en Python cuya única función consiste en proporcionar las funciones necesarias para leer, tratar y analizar los datos. En este último fichero, se ha creado un objeto denominado Predictor, encargado de usar estimadores DT y RF para predecir el sexo del autor o autora de un mensaje de texto. Existe también una función que devuelve los parámetros óptimos para un Predictor. Esta función puede obtener dichos parámetros mediante Grid Search o Halving sucesivo. Respecto al resto de métodos y funciones (el código consta de un total de 808 líneas de código), su propósito es aumentar la legibilidad del código y facilitar el análisis de datos.

Se han realizado un total de cinco pruebas principales. Su descripción y los resultados más importantes están en la *Tabla 2*.

Datos	Mejor dimensión	Precisión media (DT)	Precisión máxima (DT)	Precisión media (RF)	Precisión máxima (RF)
Una dimensión de análisis	Digital	55,61%	57,70%	56,62%	57,66%
Dos dimensiones de análisis	Digital + Léxico	56,66%	59,96%	58,19%	60,38%
Combinación Digital + Léxico + X	Digital + Léxico + Ortográfico	59,4125%	59,79%	60,415%	60,77%
Una dimensión y distinción de edad	Léxico (30s)	56,11%	62,40%	57,45%	61,07%
Dos dimensiones y franja de edad “30s”	Digital + Léxico	57,56%	63,10%	59,46%	64,27%

Tabla 2. Resumen de las pruebas ejecutadas

Al observar estos resultados, podemos extraer diversas conclusiones. Estas conclusiones se pueden agrupar en dos categorías: conclusiones que afectan a la base de datos y su relación con los resultados obtenidos y conclusiones en referencia a los clasificadores usados. Según lo visto en la primera categoría, tener una buena base de datos es fundamental para obtener buenos resultados. Si queremos sacarle el máximo partido, debemos asegurarnos de que la disposición sea la correcta. Además, es conveniente mantener una proporción constante en la partición de los datos durante el desarrollo del estudio. En nuestro caso, esta proporción ha sido 70%-30%. También hemos comprobado que, mientras más datos se encuentren al alcance del árbol, mejor podrá encontrar atributos informativos y obtendremos mejores resultados. No obstante, la cantidad no lo es todo. Si nuestros datos tienen un patrón claro, un árbol de decisión sabrá detectarlo aun teniendo menos datos.

Según lo visto en la segunda categoría, se ha demostrado que los conceptos teóricos se cumplen en todos los casos. Hemos visto casos de sobre entrenamiento, de árboles perjudicados por ruido en las muestras y hemos comprobado que en todos y cada uno de los casos los random forests son mejores predictores que los árboles de decisión. Además, se ha demostrado la importancia de una buena parametrización para ambos clasificadores.

4 Conclusiones

La tecnología nos permite tratar cada vez más datos con más facilidad y velocidad. Gracias al poder de la computación, el campo de la ciencia de datos ha ido cobrando más y más importancia en nuestro mundo. Al tener la capacidad de almacenar cualquier tipo de información como queramos y según nos convenga, podemos analizarlos y estudiarlos con mucha más facilidad. Esto nos permite avanzar como sociedad al poder realizar análisis de datos masivos en campos como la física, la biología, la medicina, entre muchos otros.

En este proyecto, hemos unido este campo con el campo de la lengua castellana. Al disponer de decenas de miles de mensajes convertidos a vectores numéricos, hemos sido capaces de estudiarlos con herramientas de IA. Más concretamente, hemos tratado de predecir si estos mensajes eran de hombres o de mujeres con árboles de decisión y random forests. Después de entender ambas herramientas a la perfección, con sus pros y sus contras, hemos usado esos conocimientos para aplicarlos en un caso real. Y los resultados han sido más que fructíferos, en mi opinión.

No sólo hemos obtenido buenas precisiones en las predicciones, sino que hemos podido sacar conclusiones pertinentes y hemos aportado nuestro granito de arena al proyecto del doctorando Damián Morales. Además, ha sido una oportunidad para aprender más sobre el mundo de la IA y el análisis de datos.

Personalmente, me ha gustado mucho hacer este trabajo de fin de grado. Para empezar, uno de los campos de la informática por el que me siento más atraído es por la IA. Pienso que es el presente y el futuro de la investigación. Nos puede ayudar muchísimo a avanzar y a desarrollar nuevas tecnologías, a descubrir nuevos tratamientos para enfermedades sin cura… En definitiva, opino que es una de las herramientas que más nos puede ayudar a ser mejores. Seguidamente, este proyecto ha resultado ser un desafío que he aceptado con gusto y que me ha hecho mejor ingeniero. Me ha permitido mejorar mis habilidades en Python, he entendido a la perfección cómo funcionan los modelos de aprendizaje supervisado usados y he aprendido conceptos y técnicas nuevas que desconocía totalmente como la validación cruzada. Por lo que a mí respecta, estoy más que satisfecho con este trabajo, con los resultados obtenidos y con los conocimientos que me ha aportado.

5 Referencias

[1] Página web: PAN, https://pan.webis.de/shared-tasks.html .

[2] Paper: J. Demsar, B. Zupan, "Orange: Data Mining Fruitful and Fun – A Historical Perspective", https://www.semanticscholar.org/paper/Orange%3A-Data-Mining-Fruitful-and-Fun-A-Historical-Demsar-Zupan/2a52478be9b4055aaae729090846e8dc318f7672 . 2013.

[3] Página web: https://www.mygreatlearning.com/blog/decision-tree-algorithm/ . 13 de febrero del 2020.

[4] PDF de la asignatura *Inteligencia artificial* de cuarto de GEI: Anónimo, "Decision trees: ID3 Algorithm".

[5] Página web: https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8 . 24 de marzo del 2020.

[6] Definición de Oxford Languages: https://www.google.com/search?q=qu%C3%A9+es+la+entrop%C3%ADa&oq=qu%C3%A9+es+la+entrop%C3%ADa&aqs=chrome.0.0l10.3978j1j7&sourceid=chrome&ie=UTF-8

[7] Artículo web: Mukesh Mithrakumar, "How to tune a Decision Tree?", Towards data science, https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680. 11 de noviembre del 2019.

[8] Paper: Laura Elena Raileanu and Kilian Stoffel, "Theoretical comparison between the Gini Index and Information Gain criteria" Annals of Mathematics and Artificial Intelligence 41: 77-93, 2004.

[9] Artículo web: Scikit-Learn developers, "1.10. Decision Trees", https://scikit-learn.org/stable/modules/tree.html

[10] Documentación Scikit-Learn: Scikit-Learn developers, "sklearn.tree.DecisionTreeClassifier", https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[11] Repositorio Github: https://github.com/scikit-learn/scikit-learn/blob/1495f69242646d239d89a5713982946b8ffcf9d9/sklearn/tree/_splitter.pyx

[12] Paper: Rafael Gomes Mantovani, Tomás Horváth, Ricardo Cerri, Sylvio Barbon Junior, Joaquin Vanschoren, André Carlos Ponce de León Ferreira de Carvalho, "An empirical study on hyperparameter tuning of decision trees", arXiv:1812.02207.

[13] Foro: Usuario Casimir, "Illustrating the random forest algorithm in TikZ", https://tex.stackexchange.com/questions/503883/illustrating-the-random-forest-algorithm-in-tikz . Septiembre del 2020.

[14] Documentación Scikit-Learn: Scikit-Learn developers, "sklearn.ensemble.RandomForestClassifier", https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html .

[15] Artículo web: Scikit-Learn developers, "Ensemble methods", https://scikit-learn.org/stable/modules/ensemble.html#parameters .

[16] Artículo web: Usuario z_ai, "Random Forest: Hyperparameters and how to fine-tune them", https://towardsdatascience.com/random-forest-hyperparameters-and-how-to-fine-tune-them-17aee785ee0d . 15 de octubre del 2020.

[17] Artículo web: Anónimo, "Cross-validation (statistics)", https://en.wikipedia.org/wiki/Cross-validation_(statistics). 4 de mayo del 2021.

[18] Artículo web: Anónimo, "Validación cruzada", https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada . 5 de marzo del 2020.

[19] Artículo web: Scikit-Learn developers, "3.1. Cross-validation: evaluating estimator performance", https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation.

[20] Artículo web: Scikit-Learn developers, "3.2 Tuning the hyper-parameters of an estimator", https://scikit-learn.org/stable/modules/grid_search.html#grid-search .

[21] Artículo web: Scikit-Learn developers, "Comparing randomized search and grid search for hyperparameter estimation", https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html#sphx-glr-auto-examples-model-selection-plot-randomized-search-py .

[22] Documentación Scikit-Learn: Scikit-Learn developers, "sklearn.datasets.load_digits", https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html#sklearn.datasets.load_digits .

[23] Documentación Scikit-Learn: Scikit-Learn developers, "Successive Halving Iterations", https://scikit-learn.org/stable/auto_examples/model_selection/plot_successive_halving_iterations.html .

[24] Documentación Scikit-Learn: Scikit-Learn developers, "sklearn.model_selection.HalvingGridSearchCV", https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.HalvingGridSearchCV.html#sklearn.model_selection.HalvingGridSearchCV .

[25] Artículo web: Scikit-Learn developers, "Comparison between grid search and successive halving", https://scikit-learn.org/stable/auto_examples/model_selection/plot_successive_halving_heatmap.html#sphx-glr-auto-examples-model-selection-plot-successive-halving-heatmap-py .

[26] Artículo web: Bex T., “11 Times Faster Hyperparameter Tuning with HalvingGridSearch”, https://towardsdatascience.com/11-times-faster-hyperparameter-tuning-with-halvinggridsearch-232ed0160155 . 9 de abril del 2020.

6 Anexo

INDICADORES DE ANÁLISIS COMBINADOS	
PRAGMÁTICO-DISCURSIVOS (39)	
Marcadores discursivos estructuradores de la información	
Marcadores discursivos conectores	
Marcadores discursivos reformuladores	
Marcadores discursivos argumentativos	
Frecuencia de actos de habla asertivos	
Frecuencia de actos de habla directivos	
Frecuencia de actos de habla compromisivos	
Frecuencia de actos de habla expresivos	
Frecuencia de actos de habla declarativos	
Frecuencia de deíxis personal	
Frecuencia de deíxis espacial	
Frecuencia de deíxis temporal	
Frecuencia de complementos focalizados (CD, CI…?)	
Frecuencia de apéndices interrogativos	
Fórmulas de tratamiento	
Frecuencia de oraciones enunciativas	
Frecuencia de oraciones interrogativas	
Frecuencia de oraciones interrogativas directas	
Frecuencia de oraciones interrogativas indirectas	
Frecuencia de oraciones interrogativas parciales	

Frecuencia de oraciones interrogativas totales	
Frecuencia de oraciones exclamativas	
Frecuencia de oraciones exhortativas	
Frecuencia de oraciones dubitativas	
Fórmulas de saludo (7)	buenos días, buenas tardes, buenas noches, buenas, hola, holi, holita
Fórmulas de despedida (7)	adiós, hasta pronto, hasta otra, hasta siempre, hasta luego, nos vemos, hasta la próxima
Fórmulas de agradecimiento y gratitud (2)	gracias, agradecer
Fórmulas de disculpas (4)	perdón, lo siento, disculpar, arrepentirse
Otras fórmulas de cortesía (2)	por favor, si me permites
Fórmulas gráficas de la risa (10)	jaja, jeje, jiji, jojo, juju, haha, hehe, hihi, hoho, huhu
Longitud fórmulas gráficas de la risa (en caracteres)	
Longitud promedio de las publicaciones (en palabras)	
Número total de líneas	
Número total de párrafos	
Número promedio de oraciones por párrafo	
Número promedio de palabras por párrafo	
Número promedio de caracteres por párrafo	
Número promedio de palabras por oración	
Número de líneas en blanco / total número de líneas	
DIGITALES (5)	
Número de emoticonos / palabras totales	
Número de emoticonos diferentes / emoticonos	
Emoticonos más frecuentes	
Número de enlaces URL	
Número de enlaces URL con el elemento *http*	

SEMÁNTICOS (5)	
Frecuencia de tópicos conversacionales LDA (Latent Dirichlet Allocation)	
Frecuencia de relaciones sinonímicas **https://babelscape.com/wordatlas#api**	
Frecuencia de relaciones antonímicas	
Frecuencia de relaciones hiperonímicas	
Frecuencia de relaciones hiponímicas	
LÉXICOS (10)	
Frecuencia de neologismos y voces de la jerga digital	
Frecuencia de términos emotivos	Spanish Emotion Lexicon
Frecuencia de insultos	
Frecuencia de atenuantes retóricos (41)	probablemente, posiblemente, aproximadamente, usualmente, ocasionalmente, raramente, generalmente, seguramente, especialmente, prácticamente, literalmente, relativamente, básicamente, realmente, particularmente, personalmente, casi.
	pensar (+ que), creer (+ que), considerar (+ que), poder (+ infinitivo), es posible (+ que), deber de (+ infinitivo), parecer (+ que)
	en verdad, si soy sincero/a, en mi opinión, desde mi punto de vista
	probable, posible
	un tipo de, un poco, en general, una parte de, más o menos, algo así como, de alguna manera, tirando a, en su mayoría, la mayor parte, algo, por cierto…
Frecuencia de non-dictionary words	
Frecuencia de apelativos afectivos (14)	tío, tía, chaval/a, macho, mi reina/rey, mi vida, mi tesoro, mi alma, cariño, cari, gordi, querida/o, amigos/as, estimados/as.
Longitud media por palabra (en caracteres)	
Riqueza léxica (total palabras diferentes/palabras)	

Palabras de más de 6 caracteres / palabras totales	
Palabras breves (de 1 a 3 caracteres) / palabras totales	
SINTÁCTICOS (46)	
Extensión oracional en palabras	
Reiteración del mismo término	*muy muy*, *lindo*, *lindo*…
Reiteración mediante coordinación	He esperado *horas y horas*
Frecuencia de oraciones con sujeto expreso	
Frecuencia de oraciones con sujeto tácito	
Frecuencia de oraciones personales	
Frecuencia de oraciones impersonales	
Frecuencia de oraciones impersonales unipersonales	
Frecuencia de oraciones impersonales gramaticalizadas	
Frecuencia de oraciones impersonales eventuales	
Frecuencia de oraciones impersonales reflejas	
Frecuencia de oraciones impersonales construcciones específicas	
Frecuencia de oraciones copulativas	
Frecuencia de oraciones predicativas	
Frecuencia de oraciones predicativas activas	
Frecuencia de oraciones predicativas pasivas perifrásticas	
Frecuencia de oraciones predicativas pasivas reflejas	
Frecuencia de oraciones predicativas transitivas	
Frecuencia de oraciones predicativas intransitivas	
Frecuencia de oraciones predicativas transitivas reflexivas	

Frecuencia de oraciones predicativas transitivas recíprocas	
Frecuencia de oraciones simples	
Frecuencia de oraciones compuestas	
Frecuencia de oraciones compuestas copulativas	
Frecuencia de oraciones compuestas disyuntivas	
Frecuencia de oraciones compuestas adversativas	
Frecuencia de oraciones compuestas distributivas	
Frecuencia de oraciones compuestas explicativas	
Frecuencia de oraciones complejas sustantivas	
Frecuencia de oraciones complejas sustantivas completivas	
Frecuencia de oraciones complejas sustantivas interrogativas	
Frecuencia de oraciones complejas relativas	
Frecuencia de oraciones complejas relativas especificativas	
Frecuencia de oraciones complejas relativas explicativas	
Frecuencia de oraciones complejas adverbiales	
Frecuencia de oraciones complejas adverbiales locativas	
Frecuencia de oraciones complejas adverbiales modales	
Frecuencia de oraciones complejas adverbiales temporales	
Frecuencia de oraciones complejas condicionales	
Frecuencia de oraciones complejas causales	
Frecuencia de oraciones complejas finales	

Frecuencia de oraciones complejas comparativas	
Frecuencia de oraciones complejas concesivas	
Frecuencia de oraciones complejas consecutivas	
Frecuencia de oraciones complejas ilativas	
Frecuencia de estructuras SVO	
MORFOLÓGICOS (165)	
Frecuencia de artículos determinados	el, la, los, las
Frecuencia de artículos determinados masculinos	el, los
Frecuencia de artículos determinados femeninos	la, las
Frecuencia de artículos determinados singulares	el, la
Frecuencia de artículos determinados plurales	los, las
Frecuencia de artículos indeterminados	un, una, unos, unas
Frecuencia de artículos indeterminados masculinos	un, unos
Frecuencia de artículos indeterminados femeninos	una, unas
Frecuencia de artículos indeterminados singulares	un, una
Frecuencia de artículos indeterminados plurales	unos, unas
Frecuencia de determinantes demostrativos	este, esta, estos, estas, ese, esa, esos, esas, aquel, aquella, aquellos, aquellas
Frecuencia de determinantes demostrativos cercanía	este, esta, estos, estas
Frecuencia de determinantes demostrativos distancia media	ese, esa, esos, esas
Frecuencia de determinantes demostrativos lejanía	aquel, aquella, aquellos, aquellas
Frecuencia de determinantes demostrativos masculinos	este, ese, aquel

Frecuencia de determinantes demostrativos femeninos	esta, esa, aquella
Frecuencia de determinantes demostrativos singulares	este, esta, ese, esa, aquel, aquella
Frecuencia de determinantes demostrativos plurales	estos, estas, esos, esas, aquellos, aquellas
Frecuencia de pronombres demostrativos	este, esta, esto, estos, estas, ese, esa, eso, esos, esas, aquel, aquella, aquello, aquellos, aquellas
Frecuencia de pronombres demostrativos cercanía	este, esta, esto, estos, estas
Frecuencia de pronombres demostrativos distancia media	ese, esa, eso, esos, esas
Frecuencia de pronombres demostrativos de lejanía	aquel, aquella, aquello, aquellos, aquellas
Frecuencia de pronombres demostrativos masculinos	este, estos, ese, esos, aquel, aquellos
Frecuencia de pronombres demostrativos femeninos	esta, estas, esa, esas, aquella, aquellas
Frecuencia de pronombres demostrativos neutros	esto, eso, aquello
Frecuencia de pronombres demostrativos singulares	este, esta, esto, ese, esa, eso, aquel, aquella, aquello
Frecuencia de pronombres demostrativos plurales	estos, estas, esos, esas, aquellos, aquellas
Frecuencia de determinantes posesivos	mi, tu, su, nuestro,
Frecuencia de determinantes posesivos de primera persona	mi, mis, nuestro, nuestra, nuestros, nuestras
Frecuencia de determinantes posesivos de segunda persona	tu, tus, vuestro, vuestra, vuestros, vuestras
Frecuencia de determinantes posesivos de tercera persona	su, sus.
Frecuencia de determinantes posesivos masculinos	
Frecuencia de determinantes posesivos femeninos	
Frecuencia de determinantes posesivos singulares	
Frecuencia de determinantes posesivos plurales	

Frecuencia de determinantes posesivos de un solo poseedor	
Frecuencia de determinantes posesivos de varios poseedores	
Frecuencia de determinantes posesivos sin distinción en el número de poseedores	
Frecuencia de pronombres posesivos de primera persona	mío, mía, míos, mías
Frecuencia de pronombres posesivos de segunda persona	tuyo, tuya, tuyos, tuyas
Frecuencia de pronombres posesivos de tercera persona	suyo, suya, suyos, suyas
Frecuencia de pronombres posesivos masculinos	mío, míos, tuyo, tuyos, nuestro, nuestros, vuestro, vuestros, suyo, suyos
Frecuencia de pronombres posesivos femeninos	mía, mías, tuya, tuyos, nuestra, nuestras, vuestra, vuestras, suya, suyas
Frecuencia de pronombres posesivos singulares	
Frecuencia de pronombres posesivos plurales	
Frecuencia de pronombres posesivos de un solo poseedor	
Frecuencia de pronombres posesivos de varios poseedores	
Frecuencia de pronombres posesivos sin distinción en el número de poseedores	
Frecuencia de determinantes indefinidos	
Frecuencia de pronombres indefinidos	
Frecuencia de determinantes numerales cardinales	
Frecuencia de determinantes numerales ordinales	
Frecuencia de determinantes numerales multiplicativos	
Frecuencia de determinantes numerales fraccionarios	
Frecuencia de pronombres numerales cardinales	
Frecuencia de pronombres numerales ordinales	

Frecuencia de pronombres numerales multiplicativos	
Frecuencia de pronombres numerales fraccionarios	
Frecuencia de determinantes interrogativos	
Frecuencia de determinantes exclamativos	
Frecuencia de pronombres interrogativos	
Frecuencia de pronombres exclamativos	
Frecuencia de determinantes relativos	
Frecuencia de pronombres relativos	
Frecuencia de pronombres personales	
Frecuencia de pronombres personales singulares	
Frecuencia de pronombres personales plurales	
Frecuencia de pronombres personales de primera persona	
Frecuencia de pronombres personales de segunda persona	
Frecuencia de pronombres personales de tercera persona	
Frecuencia de pronombres personales de primera persona singular	
Frecuencia de pronombres personales de segunda persona singular	
Frecuencia de pronombres personales de tercera persona singular	
Frecuencia de pronombres personales de primera persona plural	
Frecuencia de pronombres personales de segunda persona plural	
Frecuencia de pronombres personales de tercera persona plural	
Frecuencia de conjunciones	
Frecuencia de conjunciones coordinantes	
Frecuencia de conjunciones copulativas	
Frecuencia de conjunciones disyuntivas	
Frecuencia de conjunciones adversativas	

Frecuencia de conjunciones adversativas restrictivas	
Frecuencia de conjunciones adversativas exclusivas	
Frecuencia de conjunciones distributivas	
Frecuencia de conjunciones explicativas	
Frecuencia de conjunciones subordinantes	
Frecuencia de conjunciones completivas	
Frecuencia de conjunciones condicionales	
Frecuencia de conjunciones causales	
Frecuencia de conjunciones concesivas	
Frecuencia de conjunciones ilativas	
Frecuencia de conjunciones temporales	
Frecuencia de conjunciones consecutivas	
Frecuencia de conjunciones comparativas	
Frecuencia de conjunciones exceptivas	
Frecuencia de interjecciones	
Frecuencia de nombres	
Frecuencia de nombres propios	
Frecuencia de formas verbales	
Frecuencia de formas verbales personales	
Frecuencia de formas no personales	
Frecuencia de formas verbales en infinitivo	
Frecuencia de formas verbales en gerundio	
Frecuencia de formas verbales en participio	
Frecuencia de formas verbales en modo indicativo	
Frecuencia de formas verbales en presente (IND)	
Frecuencia de formas verbales en pretérito perfecto compuesto (IND)	
Frecuencia de formas verbales en pretérito imperfecto (IND)	
Frecuencia de formas verbales en pretérito pluscuamperfecto (IND)	
Frecuencia de formas verbales en pretérito perfecto simple (IND)	

Frecuencia de formas verbales en pretérito anterior (IND)	
Frecuencia de formas verbales en futuro imperfecto (IND)	
Frecuencia de formas verbales en futuro perfecto (IND)	
Frecuencia de formas verbales en condicional simple (IND)	
Frecuencia de formas verbales en condicional compuesto (IND)	
Frecuencia de formas verbales en modo subjuntivo	
Frecuencia de formas verbales en presente (SUB)	
Frecuencia de formas verbales en pretérito perfecto compuesto (SUB)	
Frecuencia de formas verbales en pretérito imperfecto (SUB)	
Frecuencia de formas verbales en pretérito pluscuamperfecto	
Frecuencia de formas verbales en futuro imperfecto (SUB)	
Frecuencia de formas verbales en futuro perfecto (SUB)	
Frecuencia de formas verbales en modo imperativo	
Frecuencia de formas verbales en primera persona	
Frecuencia de formas verbales en segunda persona	
Frecuencia de formas verbales en tercera persona	
Frecuencia de formas verbales en singular	
Frecuencia de formas verbales en plural	
Frecuencia de formas verbales en primera persona del singular	
Frecuencia de formas verbales en segunda persona del singular	
Frecuencia de formas verbales en tercera persona del singular	

Frecuencia de formas verbales en primera persona del plural	
Frecuencia de formas verbales en segunda persona del plural	
Frecuencia de formas verbales en tercera persona del plural	
Frecuencia de perífrasis modales	
Frecuencia de perífrasis modales de obligación	
Frecuencia de perífrasis modales de posibilidad o probabilidad	
Frecuencia de perífrasis modales de capacidad	
Frecuencia de perífrasis aspectuales	
Frecuencia de perífrasis aspectuales ingresivas	
Frecuencia de perífrasis aspectuales incoativas	
Frecuencia de perífrasis aspectuales terminativas	
Frecuencia de perífrasis aspectuales frecuentativas	
Frecuencia de perífrasis aspectuales resultativas	
Frecuencia de perífrasis aspectuales durativas	
Frecuencia de adjetivos	
Frecuencia de adjetivos relacionales	
Frecuencia de adjetivos calificativos	
Frecuencia de adjetivos calificativos especificativos	
Frecuencia de adjetivos calificativos explicativos	
Frecuencia de adjetivos calificativos en grado positivo	
Frecuencia de adjetivos calificativos en grado comparativo	
Frecuencia de adjetivos calificativos en grado superlativo	

Frecuencia de preposiciones	
Frecuencia de adverbios	
Frecuencia de adverbios de lugar	
Frecuencia de adverbios de modo	
Frecuencia de adverbios de cantidad	
Frecuencia de adverbios de negación	
Frecuencia de adverbios de afirmación	
Frecuencia de adverbios de duda	
Frecuencia de adverbiales relativos	
Frecuencia de acortamientos	
Frecuencia de siglación y acronimia	
Frecuencia de derivación por prefijación	
Frecuencia de derivación por sufijación	
ORTOGRÁFICOS (24)	
Número de comillas simples (‘) / caracteres	
Número de comillas dobles (“) / caracteres	
Número de comillas angulares / caracteres	
Número de comas (,) / caracteres	
Número de puntos (.) / caracteres	
Número de dos puntos (:) / caracteres	
Número de punto y coma (;) / caracteres	
Número de signos de interrogación (¿ ?) / caracteres	
Número de signos de exclamación (¡!) / caracteres	
Número de paréntesis (( )) / caracteres	
Número de guiones (-) / caracteres	
Número de aposiopesis (…) / caracteres	
Número de múltiples signos de interrogación (???) / caracteres	
Número de múltiples signos de exclamación (!!!) / caracteres	
Número de combinaciones de signos de interrogación y exclamación (¿!, ?!, ?¡) / caracteres	
Número de caracteres vocálicos repetidos (*muuuuy*) / caracteres	
Número de caracteres consonánticos repetidos (gritarrrr) / caracteres	
Número total de caracteres en las palabras / caracteres	

Número total de caracteres en mayúscula / caracteres
Número total de letras (a-z) / caracteres
Número total de números / caracteres
Número de palabras escritas completamente en mayúscula / palabras
Número total de espacios en blanco / caracteres
Número total de espacios de tabulación / caracteres