Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Informática

Sistema de identificación de prestaciones sociales para personas en riesgo de exclusión social

Trabajo fin de estudio presentado por:	Joaquín Gaspar Medina Arco
Línea de investigación:	Inteligencia Artificial
Director/a:	Rafael Alejandro Rodríguez Gómez
Fecha:	15 de septiembre de 2021
Código fuente:	https://github.com/jgasparmedina/SubventionsSimulator

Resumen

Según el último indicador AROPE (*At-Risk-Of Poverty and Exclusion*) de 2019, la tasa de población en España que se encuentra en riesgo de pobreza o de exclusión social asciende al 25,3%, que se traduce en cerca de 12 millones de personas. Tanto el Estado español como las diferentes Comunidades Autónomas proporcionan prestaciones para este colectivo de manera que puedan garantizarle un acceso a recursos que les permita salir de esta situación de vulnerabilidad. Por otra parte, la *European Anti-Poverty Network* (EAPN) es una red de asociaciones y organizaciones sin ánimo de lucro a nivel europeo cuyo fin es el de luchar por el fin de la pobreza y la exclusión social, teniendo en Madrid una fuerte presencia a través de su delegación EAPN Madrid con más de 70 asociaciones vinculadas. Entre sus actividades se encuentra el asesoramiento y orientación a personas en riesgo de pobreza y exclusión social para poder acceder a las ayudas y prestaciones que las administraciones públicas disponen. En el presente trabajo se lleva a cabo un estudio, análisis y comparación de métodos y algoritmos de clasificación para acabar diseñando y desarrollando una aplicación web basada en los principios de los árboles de decisión que permita a los técnicos de todas las asociaciones vinculadas a EAPN Madrid poder identificar si los diferentes usuarios pueden optar a alguna de las prestaciones disponibles mediante un proceso guiado y optimizado de preguntas. Este desarrollo es una continuación del trabajo realizado por la Clínica Jurídica de UNIR para EAPN Madrid donde se analizaron todas las prestaciones disponibles y sus requisitos para acceder a ellas. La aplicación proporciona mecanismos de ampliación, actualización y mantenimiento de las prestaciones disponibles. Por último, se aborda una aproximación para resolver el problema aplicando principios de aprendizaje automático e inteligencia artificial, comparándolo con la solución implementada.

Palabras clave: Inteligencia Artificial, Aprendizaje Automático, Algoritmos de clasificación, Árboles de decisión, Exclusión Social, Riesgo de Pobreza, Ayudas y subvenciones, EAPN, Grado en Ingeniería Informática

Abstract

According to the latest AROPE indicator (At-Risk-Of Poverty and Exclusion) about 2019, the population rate in Spain at risk of poverty or social exclusion amounts to 25.3%, which is just almost 12 million people. Both the Spanish State and the different Autonomous Communities provide subventions to this group assuring they can access to resources that allow them get out of this situation of vulnerability. On the other hand, European Anti-Poverty Network (EAPN) is a network of associations and non-profit organizations at European level whose goal is to fight against the poverty and social exclusion, having a strong presence in Madrid through its EAPN Madrid delegation which has more than 70 linked associations. Among its activities is to counsel and guide people at risk of poverty and social exclusion to access to the subventions and aids those public administrations have available. In this work, a study, analysis and comparation of different classification methods and algorithms is carried out with the final goal of design and develop a web application which allows to EAPN Madrid associated organizations workers to be able to identify if one person is eligible or not for some of these subventions following a guided and optimized process based on questions. This development is a continuation of the work done by UNIR's Legal Clinic for EAPN Madrid where all the available subventions and its requirements were analyzed. The application provides subventions extension, update and maintenance capabilities. Finally, an approach to solve the problem using principles of machine learning and artificial intelligence is addressed, comparing it with the implemented solution.

Keywords: Artificial Intelligence, Machine learning, Classification Algorithm, Decision Tree, Social Exclusion, Risk of Poverty, Grants and subsidies, EAPN, Computer Engineering Degree

"La paz no es solamente la ausencia de la guerra; mientras haya pobreza, racismo, discriminación y exclusión difícilmente podremos alcanzar un mundo de paz." (Rigoberta Menchú, Premio Nobel de la Paz 1992)

"Si una sociedad libre no puede ayudar a sus muchos pobres, tampoco podrá salvar a sus pocos ricos". (John F. Kennedy, presidente de los Estados Unidos de América)

"Erradicar la pobreza no es un acto de caridad, es un acto de justicia". (Nelson Mandela, presidente de Sudáfrica).

"Lo que todos tenemos que hacer es asegurarnos de que estamos usando la IA de una manera que sea en beneficio de la humanidad, no en detrimento de la humanidad". (Tim Cook, CEO de Apple).

"La ley de Zeroth: un robot no puede dañar a la humanidad o, por inacción, permitir que la humanidad sufra daño." (Isaac Asimov, escritor y creador de las tres leyes de la robótica).

"Siempre hay algo que puedo hacer." (Darth Vader, personaje de StarWars).

Agradecimientos

Este trabajo no sería posible sin el incondicional apoyo de mi familia que ha sacrificado algo tan valioso como el tiempo juntos para que pudiera dedicarme al estudio y desarrollo del mismo, además de insuflarme energía y ánimos constantes para llevar a buen puerto esta tarea.

Gracias a mi mujer y compañera Guadalupe por animarme a emprender este camino, por tus correcciones de experimentada lectora, por tu esfuerzo para adentrarte en el mundo de la tecnología y entender su complejidad para aportar tu crítica siempre constructiva, así como tu infinita paciencia para no cansarte de que este haya sido nuestro monotema durante unos meses.

Gracias a mi hija Victoria por enviarme a estudiar y recibirme siempre con una sonrisa a pesar de no estar todo el tiempo juntos que nos gustaría, eres toda una motivación para mí.

Gracias al Dr. Rafael Alejandro Rodríguez Gómez por elegirme para llevar a cabo este trabajo tan ilusionante, su tutela durante todo el trabajo, sus sugerencias y correcciones siempre acertadas y por sus aportaciones enriquecedoras.

Gracias a UNIR y EAPN Madrid por brindarme la posibilidad de que el Trabajo Fin de Grado tenga una utilidad social desinteresada y que pueda ayudar a mejorar la vida de personas en situaciones excepcionales.

Por último, gracias a todas las personas que colaboran desinteresadamente en combatir la pobreza, la exclusión social y la desigualdad porque demuestran que siempre hay esperanza.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

Se define el indicador AROPE, *At Risk Of Poverty and/or Exclusion*, como aquel que mide el porcentaje de población que está en riesgo de pobreza y/o exclusión social. Dicho indicador, clasificado como de tipología social, se definió en 2010 con el objetivo de medir la pobreza relativa en Europa, de manera que está incluido dentro del catálogo de todos aquellos indicadores que cada estado miembro de la Unión Europea debe medir. En términos prácticos, según su definición, una persona se considera incluida dentro de este indicador si cumple alguno de las siguientes condiciones:

- Riesgo de pobreza: si el ingreso medio en el hogar es inferior al 60% del ingreso medio en el país[1]. (European Commission, 2021)
- Carencia material severa[2]: cuando no puede hacer frente a, al menos, 4 de los 9 conceptos definidos como esenciales.
- Hogar sin trabajo o con baja intensidad laboral: cuando las personas que viven en él y que, estando en edad laboral, no trabajaron más del 20% del total posible.

Aterrizando este indicador al Estado español, según el Instituto Nacional de Estadística (INE), en 2019, siendo este el último periodo observado del que hay datos consolidados, el porcentaje de personas que se incluyen dentro de este indicador asciende al 25,3% (Instituto Nacional de Estadística, 21), lo que supone un total de casi 12 millones de personas[3], que, aunque mejora con respecto al año anterior (ver Figura 1), sigue siendo alarmante que una de cada cuatro personas se encuentre en esta situación de riesgo.

Como medidas paliativas, las diferentes administraciones públicas, tanto locales como autonómicas y estatales, ponen a disposición de las personas que están dentro de este colectivo diferentes prestaciones y subvenciones para que puedan tener acceso a servicios esenciales o disponer de oportunidades que dignifiquen sus vidas y les permitan salir de esta situación de riesgo. Sin embargo, el conocimiento de este catálogo y de los requisitos a

[1] Página oficial de la Unión Europea donde se describe este indicador y su metodología de medición: https://ec.europa.eu/social/main.jsp?catId=818&langId=en&id=8
[2] Página oficial del INE donde se describen los conceptos esenciales que miden este indicador: https://www.ine.es/ss/Satellite?L=es_ES&c=INESeccion_C&cid=1259925456180&p=1254735110672&pagename=ProductosYServicios/PYSLayout
[3] Informe del INE emitido el 21 de julio de 2020: https://www.ine.es/prensa/ecv_2019.pdf

satisfacer para poder ser considerado beneficiario de alguna de estas ayudas requiere, en ocasiones, realizar un análisis profundo tanto de las diferentes legislaciones como de la situación específica de cada uno, y no todas las personas tienen ni los medios ni las capacidades para llevarlo a cabo.

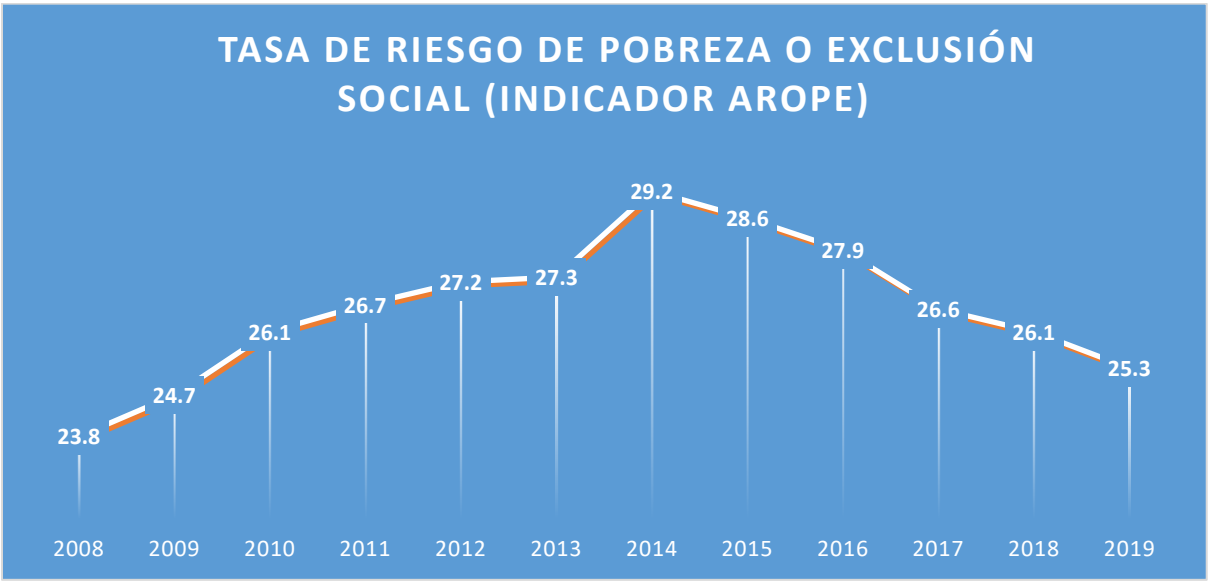


Figura 1. Gráfico de evolución del indicador AROPE en España. (Elaboración propia a partir datos del INE)

Estas limitaciones o complicaciones para acceder a las diferentes subvenciones impiden que muchas personas o familias vivan en condiciones precarias o no tengan oportunidades para mejorar su calidad de vida, por lo que parece necesario profundizar en la búsqueda y diseño de soluciones o mecanismos que permitan aumentar el conocimiento del universo de prestaciones sociales, así como de sus requerimientos, de manera que pueda estar al alcance del mayor número de personas posible.

1.1.Justificación del tema elegido

El presente trabajo se enmarca en un proyecto de colaboración interdisciplinar en la Universidad Internacional de la Rioja, entre la Facultad de Derecho, a través de su clínica jurídica[4], y la Escuela Superior de Ingeniería y Tecnología para poder aportar una solución en el ámbito del Legal Tech a EAPN Madrid.

[4] Enlace de la Clínica Jurídica de UNIR: https://www.unir.net/facultades/facultad-derecho/clinica-juridica/

La *European Anti Poverty Network*, EAPN, nacida en 1990, se define como una coalición de ONG y otros grupos independientes cuyo principal objetivo es el de la lucha contra la pobreza y la exclusión social en los estados miembros de la UE, y que tiene presencia en dichos estados. En España esta red tiene desde 1991 su representación mediante la EAPN-ES y está compuesta por 19 EAPN autonómicas y 18 de nivel estatal, todas con el mismo objetivo que su matriz europea.[5] (EAPN-ES, 2021)

EAPN Madrid, en colaboración con la clínica jurídica de UNIR y con el objetivo de facilitar su labor en la lucha contra la pobreza en el Estado español, ha realizado un análisis de las diferentes ayudas que existen y son de ámbito nacional, según la legislación vigente en España y la Comunidad de Madrid, así como del perfil de personas que pueden optar a cada una de ellas con el fin de poder establecer una guía para, en función de las características de una persona, deducir a cuál o cuáles de todas las ayudas disponibles puede tener acceso.

Como continuación de esta línea de análisis elaborada entre EAPN Madrid y UNIR, surge la idea de este TFG, cuya motivación es la de proporcionar una solución informática que permita hacer accesible este catálogo de ayudas a través de una aplicación web para que los técnicos de EAPN Madrid puedan guiar y asesorar a las personas que puedan encontrarse en riesgo de pobreza y/o exclusión social.

Se trata, por tanto, de aplicar tecnología al servicio del derecho o de aplicación jurídica. Esta tecnología jurídica es lo que se conoce como Legal Tech, de manera que, a través del uso de determinados sistemas informáticos, se obtienen servicios legales y jurídicos, con mayor o menor nivel de automatización. Las aplicaciones del Legal Tech son muy diversas y abarcan desde un asesoramiento legal automatizado como puede ser el objetivo de este trabajo, hasta procesos de automatización documental o análisis predictivo de un caso legal haciendo uso de inteligencia artificial (Barrio Andrés, 2019).

Relacionar conceptos y soluciones de Legal Tech con asociaciones sin ánimo de lucro cuyo objetivo es el de proporcionar ayudas a personas en riesgo de exclusión social es, sin duda,

[5] Información oficial del site de EAPN-ES: https://www.eapn.es/que-es-eapn.php

una motivación adicional para llevar a cabo el presente trabajo, ya que, en condiciones normales, este tipo de asociaciones no tienen acceso a ese tipo de soluciones tecnológicas.

1.2. Problema y finalidad del trabajo

Como resultado al estudio pormenorizado de las legislaciones de las diferentes administraciones públicas del Estado español realizado por EAPN Madrid junto con UNIR, se obtiene un catálogo de hasta 21 leyes, decretos u órdenes que regulan hasta un total de 32 tipos distintos de posibles ayudas aplicables en la Comunidad de Madrid.

Cada una de estas potenciales subvenciones está destinada a un determinado colectivo, de manera que son múltiples las características a tener en cuenta, tales como edad, nacionalidad, residencia, ingresos, unidad familiar, estudios, etc. De igual forma, el objeto de cada una de estas subvenciones es de diferente índole, ya que pueden ser prestaciones asociadas al trabajo (desempleo, jubilación, incapacidad, etc.), a los estudios (becas de estudios, de transporte, etc.), a la salud y dependencia, o, incluso, al asilo.

La combinación del universo de rasgos o condiciones a satisfacer y el de las diferentes prestaciones sociales conforman una matriz de decisión que permite identificar si un determinado individuo puede, o no, tener acceso a una determinada subvención en función de sus características personales. Sin embargo, esta matriz no es una solución explotable o accesible por el usuario medio que requiere acceso a este tipo de información.

La finalidad del presente TFG es la de convertir dicha matriz de decisión en una aplicación web que, a través de una sucesión de preguntas, pueda perfilar al usuario que acude al soporte de los técnicos de EAPN y poder identificar, siempre de manera orientativa, las diferentes ayudas o subvenciones a las que podría tener acceso dicho usuario en base a la información facilitada.

El proceso de recabar dicha información debería ser inteligente y selectivo, de manera que las preguntas se adapten a las respuestas proporcionadas anteriormente, con el objetivo de evitar redundancias y contradicciones, a fin de minimizar el número de preguntas para llegar a la propuesta final.

Y finalmente, dado que la normativa es susceptible de ser modificada con relativa frecuencia, el sistema debería ser flexible para poder adaptarse con facilidad y prontitud a estos cambios, sin que ello suponga una reingeniería de la solución.

1.3. Objetivos del TFG

A través del desarrollo del presente TFG se pretenden conseguir los siguientes objetivos:

1.3.1. Objetivo general

Desarrollar una aplicación web que permita a cualquier persona poder consultar si cumple o no las condiciones necesarias para poder acceder a alguna de las diferentes ayudas que ofrecen tanto el Estado Español como la Comunidad de Madrid. El proceso se llevará a cabo a través de un cuestionario adaptativo de preguntas que permita a la aplicación recopilar información suficiente para identificar las potenciales prestaciones para las que se satisfacen sus condiciones.

1.3.2. Objetivos específicos

1) Analizar los diferentes algoritmos de clasificación aplicables a este caso de uso para seleccionar el más adecuado.
2) Analizar y seleccionar la tecnología con la que el desarrollo será llevado a cabo, así como diseñar la arquitectura de la solución.
3) Aplicación de conocimientos adquiridos durante el Curso de Adaptación a Grado en lo referente a Ingeniería del Software.
4) Captación de requerimientos funcionales y no funcionales del sistema a desarrollar.
5) Prototipado de la solución y revisión con el usuario final para desarrollar la experiencia de usuario idónea.
6) Desarrollo de la solución acorde a los requerimientos funcionales y no funcionales identificados.
7) Validación y verificación de la solución desarrollada.
8) Implantación de la solución en infraestructura productiva de EAPN Madrid.
9) Uso de herramientas necesarias para la completitud del resto de objetivos, tales como repositorio de software, entornos de desarrollo, etc.

2. Marco teórico

En el presente apartado se va a profundizar en la base teórica sobre la que se sustentará el desarrollo de la aplicación objeto de este trabajo de manera que, a través de este análisis, se pueda formular un diseño apropiado al caso de uso. El esquema del contenido de este capítulo queda reflejado en la Figura 2.

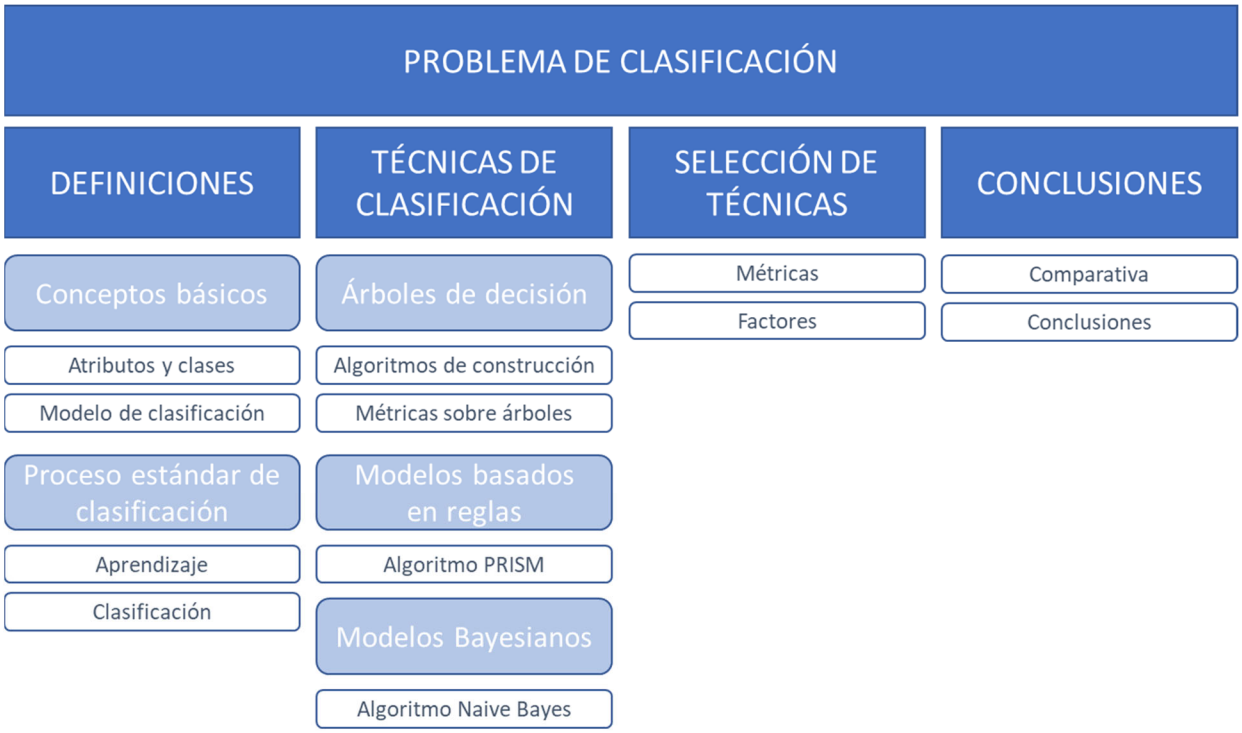


Figura 2. Esquema del capítulo

2.1. Concepto de clasificación

Según la Real Academia de la Lengua, podemos definir el concepto de clasificación como la *acción de disponer u ordenar algo en conjuntos con caracteres comunes*.[6] En la vida cotidiana hay innumerables ejemplos en los que se usan criterios de clasificación que van desde ejemplos tan simples como el cómo ordenamos las aplicaciones de nuestros móviles hasta conceptos más metódicos y universales como la clasificación de los elementos en la tabla periódica, por lo que se podría llegar a afirmar que tenemos una habilidad innata para ordenar o clasificar las cosas en categorías (Tan, Steinbach, Karpatne, & Kumar, 2006).

[6] Definiciones de clasificación, clasificar y clase según la RAE: https://dle.rae.es/

Esta tarea, que inicialmente puede ser sencilla, puede alcanzar grados de complejidad altos cuando los factores a tener en cuenta para acabar asignando un elemento a una determinada clase son muchos y, además, son de dominios diferentes no agregables entre sí. Dicho de otra forma, clasificar a las personas por su género o edad es una tarea sencilla ya que podría ser, incluso evidente, sin embargo, la tarea se vuelve mucho más complicada cuando los atributos por los que se tienen que discernir la clase a la que un individuo pertenece son tan variopintos como, por ejemplo, la edad, el nivel de estudios, el grupo sanguíneo, los ingresos medios del año, la ciudad de residencia y el número de cohabitantes en el domicilio principal. Para este último ejemplo, parece evidente que es necesario algo más que un proceso mental o, incluso, un papel y un lápiz.

Cuando el volumen de datos y de atributos crece es cuando un proceso automático, bien definido y con capacidad de adaptación o aprendizaje a nuevos valores es requerido para poder llevar a cabo esta tarea de clasificación. Es por esto que, desde un punto de vista computacional, el estudio y la aplicación de métodos y procesos de clasificación son unas piezas fundamentales dentro de las disciplinas de *Data Mining*[7] y *Machine Learning*[8].

2.2. Algunos conceptos básicos

Antes de profundizar en cómo funciona el proceso de clasificación y cuáles son sus diferentes variantes, es necesario definir algunos conceptos.

2.2.1. Atributo

Un atributo es una propiedad o característica que un elemento tiene[9]. Los atributos sirven para describir un elemento dentro de un grupo enumerando el valor de cada uno de ellos. Los atributos pueden ser de cualquier índole, de manera que pueden ser medibles o no, es decir, pueden ser cuantitativos o cualitativos. Por ejemplo, son atributos de una persona su edad, nacionalidad, formación, etc.

[7] Proceso por el cual métodos inteligentes se aplican para extraer patrones (Tan, Steinbach, Karpatne, & Kumar, 2006)
[8] Técnica o tecnología por la cual las aplicaciones pueden aprender de manera automática a reconocer patrones complejos y tomar decisiones inteligentes en función de los datos (Tan, Steinbach, Karpatne, & Kumar, 2006)
[9] Definición de atributo según la RAE: https://dle.rae.es/atributo

2.2.2. Clase

Una clase es un conjunto de elementos que tienen caracteres comunes[10]. Las clases sirven para agrupar elementos con atributos similares y, de manera inversa, poder predecir los atributos que tienen los elementos que la componen. A diferencia de los atributos, las clases tienen que ser categóricas. Por ejemplo, una clase de personas puede ser la que está compuesta por personas mayores de 65 años, solteras y residentes en Madrid.

2.2.3. Modelo de clasificación

Un modelo de clasificación es aquel que correlaciona atributos con clases de manera que, a partir de un conjunto de valores para los atributos, determina cuál es la clase a la que pertenece.

Tomando como datos de ejemplo para un problema de clasificación los indicados en la Tabla 1, se puede definir el modelo de clasificación de vehículos como aquel que, basándose en los valores de los atributos asociados a si tiene o no motor, el número de ruedas, la cilindrada, si puede volar, si puede navegar y si puede circular por raíles, determina la tipología del vehículo.

Tabla 1. Ejemplo de datos para clasificación de vehículos

Vehículo	Motor	Ruedas	CC	Vuela	Mar	Raíl	Tipo vehículo
Seat Panda	Sí	4	900	No	No	No	Automóvil
BMW 1200	Sí	2	1200	No	No	No	Motocicleta
Piaggo MP3 500 LT	Sí	3	500	No	No	No	Motocicleta
Orbea Gain D30	No	2	0	No	No	No	Bicicleta
Honda Shadow	Sí	2	45	No	No	No	Ciclomotor
Talgo 350	Sí	0	0	No	No	Sí	Tren
Airbus 310	Sí	8	0	Sí	No	No	Avión

[10] Definición de clase según la RAE: https://dle.rae.es/clase

Velero de crucero	Sí	0	2000	No	Sí	No	Barco

Los modelos de clasificación se pueden usar en dos contextos diferentes o cumplen dos funciones bien diferenciadas (Tan, Steinbach, Karpatne, & Kumar, 2006):

- Como **modelo predictivo** que permita clasificar elementos no identificados anteriormente. Según el ejemplo anterior, el modelo debería predecir qué tipo de vehículo sería un Mercedes Benz Clase A a partir del valor de sus atributos (ver Tabla 2)
- Como **modelo descriptivo** que permite identificar las características que diferencian dos clases. En este caso, el modelo debería conocer las diferencias que existen entre los atributos de lo que es un tren y un barco.

Por último, es importante recalcar que no todos los atributos son igual de relevantes en un modelo de clasificación, ya que pueden no ser suficientes para determinar la clase a la que corresponde el elemento, aunque sí necesarios (Tan, Steinbach, Karpatne, & Kumar, 2006).

Tabla 2. Datos para vehículo Mercedes Benz Clase A de vehículos

Vehículo	Motor	Ruedas	CC	Vuela	Mar	Raíl	Tipo vehículo
Mercedes Benz Clase A	Sí	4	1800	No	No	No	¿?

Siguiendo el ejemplo anterior, el atributo que define si un vehículo tiene alas o no sí que es suficiente para identificar la clase del vehículo (avión) y, sin embargo, el número de ruedas no es suficiente ya que debe combinarse con otros atributos para poder llegar a la clasificación correcta.

2.3. Proceso estándar de clasificación

Entendiendo entonces que el proceso de clasificación es aquel por el que se asignan etiquetas o clases a elementos no etiquetados o clasificados previamente (Tan, Steinbach, Karpatne, & Kumar, 2006), este se puede estructurar como un proceso de dos pasos: **aprendizaje** y **clasificación** (Han, Pei, & Kamber, 2011).

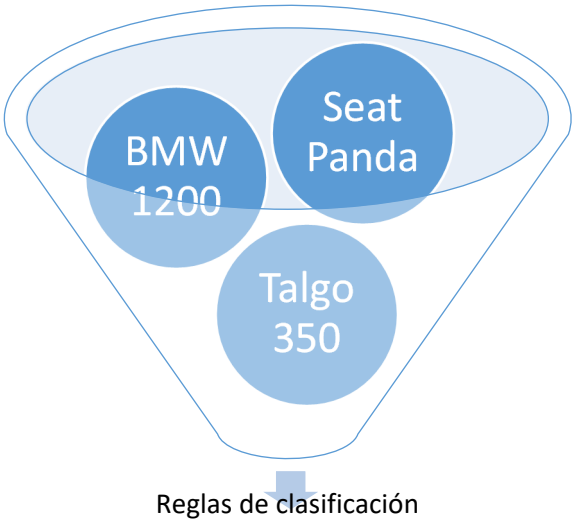
2.3.1. Aprendizaje

La primera fase del proceso de clasificación es aquella por la que se construye el modelo de clasificación. Para ello se parte de un conjunto de datos iniciales acerca de diferentes elementos, sus atributos y su clase asociada, que sirven de referencia para que un algoritmo pueda aprender y generar un modelo que le permita clasificar elementos futuros, fuera del universo del conjunto de datos inicial. La Tabla 1 de datos se puede considerar como ese set de datos iniciales de entrenamiento o aprendizaje para un algoritmo de clasificación de vehículos.

En la Figura 3 se muestra una representación visual de este proceso. Considerando que el embudo es el algoritmo de clasificación y que por él pasan los distintos elementos del set de datos iniciales de manera que, como salida del algoritmo (embudo), se van definiendo las reglas que satisfacen el conjunto de set inicial y servirán para clasificar nuevos elementos.

En función del conjunto de datos inicial que se utiliza para la fase de aprendizaje en los procesos de clasificación, estas pueden ser (Han, Pei, & Kamber, 2011):

- **Supervisados**: cuando para cada elemento del set inicial se conoce la clase a la que pertenece, como se indicaba en el ejemplo de los vehículos.
- **No supervisados**: al contrario que en los supervisados, la clase a la que pertenece cada elemento del set inicial no es conocida y, además, tampoco se conoce el total de clases posibles.



Reglas de clasificación

Si ruedas en (2, 3) y tiene motor y cilindrada es <=29 y no tiene alas y no circula por railes y no circula por mar entonces tipo vehículo es Ciclomotor
Si tiene alas entonces tipo vehículo es Avión
Si circula por el mar entonces tipo vehículo es Barco
etc.

Figura 3. Fase de aprendizaje en clasificación.

2.3.2. Clasificación

La segunda fase del proceso de clasificación es aquella a la que, a partir del modelo obtenido en la fase de aprendizaje, se le aplica un conjunto de datos de prueba para que el algoritmo de clasificación pueda predecir a qué clase pertenece y medir así su nivel de exactitud. Si el nivel conseguido es aceptable se puede entonces aplicar el algoritmo a datos nuevos reales (Han, Pei, & Kamber, 2011).

Siguiendo con el ejemplo de los vehículos, en la fase de clasificación, el set de reglas obtenidas en la fase de aprendizaje se aplicaría sobre un set de datos de test como el indicado en la Tabla 3 con el objetivo de revisar cómo se comportan dichas reglas y su completitud.

Tabla 3. Ejemplo de datos de prueba para fase de clasificación de vehículos

Vehículo	Motor	Ruedas	CC	Vuela	Mar	Raíl	Tipo vehículo
Volkswagen Passat	Sí	4	1900	No	No	No	¿?
CAF 800	Sí	0	0	No	No	Sí	¿?
Boing 747	Sí	8	0	Sí	No	No	¿?
Honda X-ADV	Sí	2	125	No	No	No	¿?
Buque Evergreen	Sí	0	0	No	Sí	No	¿?

El set de pruebas debe ser diferente al set inicial con el que se han generado las reglas de clasificación para asegurar que el algoritmo puede predecir la clase a la que un nuevo elemento pertenece (Tan, Steinbach, Karpatne, & Kumar, 2006).

Para medir la calidad del modelo de clasificación creado se tienen en cuenta las siguientes métricas, que son inversamente proporcionales:

- La efectividad o exactitud: el porcentaje de aciertos que ha tenido sobre el set de pruebas (Han, Pei, & Kamber, 2011).
- La tasa de error: el porcentaje de errores que ha obtenido sobre el set de pruebas (Tan, Steinbach, Karpatne, & Kumar, 2006).

El objetivo, por tanto, es el de maximizar la efectividad o, lo que es lo mismo, minimizar la tasa de error.

2.4. Técnicas de clasificación

Una vez entendidos los conceptos de clasificación y los problemas que se intentan resolver, queda analizar qué técnicas hay para aplicar estos conceptos y poder, por tanto, construir soluciones que resuelvan cada problema, adecuadas a las características de este. Aunque el universo de posibles soluciones es amplio, en este análisis se profundizará en tres estrategias con enfoques diferentes: árboles de decisión, basados en estrategia "divide y vencerás"; modelos basados en reglas; y modelos Bayesianos, basados en probabilidad.

2.4.1. Árboles de decisión

Tomando como referencia el ejemplo de los vehículos, ante un nuevo vehículo al que se le quiere averiguar cuál es su tipología, el procedimiento podría consistir en formular preguntas para conocer el valor de cada atributo y, en función de la respuesta obtenida, formular una nueva pregunta, y repetir este proceso sucesivamente hasta llegar a identificar de manera unívoca el tipo de vehículo que es. Este procedimiento o flujo de trabajo se puede representar en forma de árbol, motivo por el que recibe el nombre de árbol de decisión.

Figura 4. Árbol de decisión para tipo de vehículos

En la Figura 4 está representado el que podría ser un árbol de decisión para resolver la clasificación del ejemplo de tipología de vehículos. En él se aprecian diferentes elementos:

- **Nodo raíz**: aquel que da inicio al proceso, es el único que no tiene líneas entrantes. Corresponde a la primera pregunta por la que se inicia el proceso de clasificación, o lo que es lo mismo, al primer atributo a validar para un elemento.
- **Nodo intermedio**: representa las preguntas intermedias, es decir, revisa el valor de un atributo determinado.
- **Nodo hoja o terminal**: representa las clases del modelo. Al llegar a uno de estos nodos se ha completado el proceso y se ha obtenido la clasificación buscada.
- **Rama o arco**: cada línea que conecta dos nodos entre sí representa un valor determinado para un atributo.

Para el caso de la Honda Shadow, recogidos sus datos en la Tabla 1, la ruta seguida por el árbol de decisión sería la indicada en la Figura 5, donde cada rama roja representa el valor de cada uno de sus atributos (nodos intermedios) hasta llegar al nodo hoja que representa su clase.

Figura 5. Ejemplo de aplicación de árbol de decisión para tipo de vehículos

2.4.1.1. Algoritmos de construcción de árboles de decisión

Para un mismo set de datos iniciales se pueden construir múltiples árboles de decisión, pero todos no serán iguales en cuanto a rendimiento y eficacia y, además, encontrar el óptimo puede tener un coste computacional muy alto. Sin embargo, con el paso de los años, se han desarrollado algunos algoritmos que encuentran un árbol de decisión con una tasa de efectividad alta en un tiempo razonable. (Tan, Steinbach, Karpatne, & Kumar, 2006)

Los algoritmos más conocidos son:

- **ID3 – *Iterative Dichotomiser*:** desarrollado por John Ross Quinlan y centrado en los árboles de decisión binarios. (Ross Quinlan, Induction of decision trees, 1986)
- **C4.5**: desarrollado también por John Ross Quinlan y que es una evolución del ID3. (Ross Quinlan, C4.5: Programs for Machine Learning, 1992)
- **CART**: desarrollado por Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone, también centrado en los árboles de decisión binarios, pero de manera paralela al desarrollo del ID3. (Breiman, Friedman, Olshen, & Stone, 1984)

Todos ellos utilizan una misma estrategia basada en el uso de algoritmos voraces o *greedy*, de manera que, recursivamente, el árbol de decisión se forma de arriba a abajo utilizando técnicas de "divide y vencerás" (Tan, Steinbach, Karpatne, & Kumar, 2006).

El proceso de construcción sigue los siguientes pasos:

- Partir de un nodo inicial que contiene todos los elementos del set de aprendizaje.
- Si los elementos del nodo son todos de la misma clase, se marca el nodo como hoja y se convierte en el nodo de esa clase.
- Si los elementos del nodo no son de la misma clase, se elige el atributo por el que se debe segregar el conjunto de elementos y se le asigna al nodo.
- Para cada valor del atributo se genera una rama que representa el valor del atributo elegido para segregar y que va a un nodo que contiene los elementos que satisfacen ese valor. En función de la tipología del atributo, las ramas que nacen pueden ser dispares (Han, Pei, & Kamber, 2011):
    - Atributo binario: se generarán dos ramas. En el ejemplo de la Figura 4, el nodo "¿Tiene motor?" es un ejemplo de este tipo de atributo.

- Atributo discreto: aquel que su valor está entre una lista de valores posibles, bien ordenados o sin ordenar. Dado que el número de valores posibles para este tipo de atributos puede ser dispar, las ramas que nacen de este nodo pueden ser una por cada valor posible o se puede seguir un esquema binario dividiendo el conjunto de valores posibles en dos. En el ejemplo de la Figura 4, el nodo raíz es un ejemplo de atributo discreto del que salen tantas ramas como valores posibles.
- Atributo continuo: aquel que es de tipo cuantitativo. En este caso las ramas que surgen de este atributo son dos de manera que se establece un punto de corte en los valores del atributo para segregar los datos. En el ejemplo de la Figura 4, el nodo "Cilindrada" es un ejemplo de atributo continuo donde se divide utilizando como punto de corte el valor "49" para indicar los centímetros cúbicos por los que se diferencian un ciclomotor de una motocicleta.

- Se repite el proceso recursivamente hasta completar el árbol de decisión.

La diferencia entre ID3, C4.5 y CART está en el método que usan para seleccionar, en cada punto del proceso, qué atributo se elige para hacer la división, ya que, como se indicaba anteriormente, las opciones son tantas como atributos haya. Estas elecciones se basan en medidas heurísticas que ayudan a elegir la mejor opción en función de un criterio, de manera que intentan que los nodos hijos que nacen de la segregación sean lo más homogéneos posibles, es decir, que sean de la misma clase.

2.4.1.2. Ganancia de Información

Se define **entropía** como la información necesaria esperada para poder clasificar un elemento *E* (Han, Pei, & Kamber, 2011):

$$Entropía\ (E) = -\sum_{i=0}^{c-1} p_i \log_2 p_i(t)$$

Donde p_i es la probabilidad de que un elemento cualquiera pertenezca a la clase C_i. Esta probabilidad se obtiene del universo de datos de aprendizaje al dividir el número total de elementos que pertenecen a la clase C_i entre el total de elementos de la muestra.

Sea X un atributo que se usa como elemento de segregación en el árbol, entonces se crearán varias ramas en función del tipo de atributo (binario, discreto o continuo), y se subdividirá el

grupo de elementos entre dichas ramas. Una vez realizada esta selección de atributo de segregación, se puede calcular la información que todavía es necesaria para clasificar correctamente los elementos I_X (Han, Pei, & Kamber, 2011):

$$I_X(E) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \cdot Entropía(D_j)$$

Donde *v* es el número de valores posibles que tiene el atributo *X*, *D* es el número total de elementos del set de datos y D_j el número de elementos que tienen para el atributo X el valor *j-ésimo*.

Por último, podemos definir la ganancia de información de un atributo X como la diferencia entre la información que se necesita para clasificar un elemento y la que se necesita una vez elegido un atributo *X* (Han, Pei, & Kamber, 2011):

$$Ganancia(X) = Entropía(E) - I_X(E)$$

Tabla 4. Datos de entrenamiento para árbol de decisión

ID	Edad	Ocupación	Propietario vivienda	Conocimiento mercados	Cliente potencial
1	18-35	Estudiante	Sí	No	No
2	35-65	Desempleado	No	No	Sí
3	35-65	Empleado	Sí	Sí	Sí
4	>65	Jubilado	Sí	No	No
5	>65	Jubilado	No	Sí	Sí
6	18-35	Empleado	No	Sí	No
7	18-35	Desempleado	Sí	Sí	Sí
8	35-65	Empleado	No	No	Sí
9	35-65	Empleado	No	Sí	No

10	35-65	Desempleado	Sí	Sí	No
11	35-65	Desempleado	Sí	No	Sí
12	18-35	Estudiante	No	Sí	Sí
13	18-35	Estudiante	Sí	Sí	Sí
14	>65	Jubilado	Sí	Sí	No
15	>65	Jubilado	No	No	Sí

La Tabla 4 contiene datos de ejemplo para un proceso que define a una persona como potencial cliente de un producto en función de su edad, actividad, si es propietario de una vivienda y si tiene conocimiento de mercados financieros. Las clases posibles para este ejemplo son dos: Sí o No. Suponiendo que se inicia el proceso para generar el árbol de decisión se pueden realizar los cálculos anteriormente comentados para identificar cuál debería ser el atributo que ocupa el nodo raíz.

Por tanto, se puede calcular la entropía:

$$Entropía\ (E) = -\frac{6}{15}\log_2\left(\frac{6}{15}\right) - \frac{9}{15}\log_2\left(\frac{9}{15}\right) = 0.971$$

Ahora se puede calcular la información necesaria si se utiliza el atributo Edad para segregar, sabiendo que hay 5 elementos entre 18 y 55 años (de los cuales 2 son Sí y 3 No), 6 entre 35 y 65 (de los cuales 4 son Sí y 2 No) y 4 mayores de 65 (de los cuales 2 son Sí y 2 No):

$$I_{edad}(E) = \frac{5}{15}\left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) + \frac{6}{15}\left(-\frac{4}{6}\log_2\frac{5}{6} - \frac{2}{6}\log_2\frac{2}{6}\right) + \frac{4}{15}\left(-\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4}\right) = 0.958$$

Por último, ya se puede calcular la ganancia de información para el atributo Edad:

$$Ganancia\ (Edad) = 0.971 - 0.958 = 0.013$$

De forma análoga se aplica el proceso al resto de atributos de manera que:

$$Ganancia\ (Ocupación) = 0.971 - 0.933 = 0.038$$

$$Ganancia\ (Propietario\ Vivienda) = 0.971 - 0.936 = 0.035$$

$$Ganancia\ (Conocimiento\ Mercados) = 0.971 - 0.962 = 0.009$$

Con esto, el atributo Ocupación es el que mayor ganancia de información tiene por lo que sería el elegido para asignarlo al nodo raíz, se crearán cuatro ramas (Estudiante, Empleado, Desempleado y Jubilado) y para cada una de ellas se repite el proceso para elegir el siguiente atributo, teniendo en cuenta que ahora el universo de elementos que aplica a cada rama es el subconjunto que satisface el valor del atributo Ocupación al del indicado en la rama.

Esta es la métrica que ID3 utiliza para elegir el atributo por el que hará la segregación en ramas siguientes, de manera que utilice siempre el atributo de mayor ganancia de información.

2.4.1.3. Ratio de Ganancia

El uso de la ganancia de información provoca que el árbol se ramifique utilizando los atributos que tienen valores más diferentes, como se ha podido ver en el ejemplo anterior. Llevado a un extremo, si se hiciera el cálculo de ganancia de información sobre le atributo ID del ejemplo anterior, se llegaría a tener un árbol donde cada hoja sería un elemento, lo cual no es una solución válida.

Para evitar este comportamiento se usa el concepto de ratio de ganancia, que intenta normalizar la ganancia de información a partir de la entropía de la partición resultante (Berzal Galiano, 2021). Esta entropía de la partición se define, para un conjunto de elementos E y un atributo A por el que se quiere segregar, como:

$$Entropía\ Partición_A(E) = -\sum_{j=1}^{v} \frac{|E_j|}{|E|} \cdot \log_2\left(\frac{|E_j|}{|E|}\right)$$

De esta manera, esta entropía representa la potencial información generada al dividir el set de datos *E* en los *v* posibles valores del atributo *A*.

Con esto se define la ratio de ganancia de un set de datos E y para un atributo A como (Han, Pei, & Kamber, 2011):

$$Ratio\ de\ Ganancia(A) = \frac{Ganancia\ de\ Información\ (A)}{Entropía\ Partición_A(E)}$$

Si se aplica este criterio para el atributo Edad del ejemplo anterior, se obtiene:

$$Entropía\ Partición_{Edad}(E) = -\frac{4}{15}\log_2\left(\frac{4}{15}\right) - \frac{5}{15}\log_2\left(\frac{5}{15}\right) - \frac{6}{15}\log_2\left(\frac{6}{15}\right) = 1.566$$

Y tomando la ganancia de información para Edad calculada anteriormente (0.013), se obtiene la ratio de ganancia:

$$Ratio\ de\ Ganancia\ (Edad) = \frac{0.013}{1.566} = 0.008$$

Aplicando para el resto de atributos se obtiene:

$$Ratio\ de\ Ganancia\ (Ocupación) = \frac{0.038}{1.99} = 0.019$$

$$Ratio\ de\ Ganancia\ (Propietario\ Vivienda) = \frac{0.035}{0.997} = 0.035$$

$$Ratio\ de\ Ganancia\ (Conocimiento\ Mercados) = \frac{0.009}{0.971} = 0.009$$

En este caso, la mayor ratio de ganancia lo da el atributo Propietario Vivienda por lo que se escogería como criterio de segregación.

Esta es la métrica que utiliza C4.5 para elegir el atributo por el que hará la segregación en ramas siguientes, utilizando siempre el atributo de mayor ratio de ganancia.

2.4.1.4. Índice Gini

El índice Gini, a diferencia de las dos métricas anteriores, en lugar de buscar la ganancia de información, se centra en medir la impureza del set de datos *E* de la siguiente forma (Han, Pei, & Kamber, 2011):

$$Gini(E) = 1 - \sum_{i=1}^{c} p_i^2$$

Donde p_i es la probabilidad de que un elemento cualquiera pertenezca a la clase C_i. Esta probabilidad se obtiene del universo de datos de aprendizaje al dividir el número total de elementos que pertenecen a la clase C_i entre el total de elementos de la muestra. El sumatorio aplica al número de clases que hay (*c).*

Adicionalmente, otra característica fundamental del índice Gini es que considera que las segregaciones en los árboles son siempre binarias, por lo que se presenta un problema adicional para elegir el punto de corte dentro de un atributo cuyos valores son discretos o continuos:

- En el caso de los valores discretos se deben formular todas las combinaciones posibles que dividan el universo de valores en dos grupos y calcular el índice Gini para cada combinación.
- En el caso de los valores continuos, de igual forma, se deben formular todas las combinaciones posibles, pero, en este caso, considerando el punto medio entre dos valores consecutivos como punto de partición.

El índice Gini de una partición realizada sobre un conjunto de datos *E* a través de un atributo *A* se obtiene a partir de una suma ponderada de los subconjuntos E_1 y E_2 de la siguiente forma:

$$Gini_A(E) = \frac{|E_1|}{|E|} Gini_A(E_1) + \frac{|E_2|}{|E|} Gini_A(E_2)$$

Una vez calculado el índice Gini para todas las posibles particiones para un atributo *A*, se escoge la que menor índice Gini tenga. De manera análoga, se puede definir la reducción de impureza como (Han, Pei, & Kamber, 2011):

$$\Delta Gini(A) = Gini(E) - Gini_A(E)$$

Por lo que, la partición que tiene el menor índice Gini es, a su vez, la partición que más reduce la impureza.

Tomando de nuevo el ejemplo de datos usado anteriormente se puede calcular el índice Gini para el grupo inicial de la siguiente forma, sabiendo que hay 6 elementos que tienen *Sí* y 9 que tienen *No* como valor para la clase Cliente Potencial:

$$Gini(E) = 1 - \left(\frac{6}{15}\right)^2 - \left(\frac{9}{15}\right)^2 = 0.48$$

Para decidir el atributo por el que subdividir el conjunto es necesario calcular el índice Gini para cada atributo:

- **Edad**: tiene tres valores posibles por lo que hay tres posibles combinaciones de segregación (18-35 por un lado y 35-65 y >65 por otro (a); 18-35 y 35-65 por un lado y >65 por otro (b); y, por último, 18-35 y >65 por un lado y 35-65 por otro (c). Por tanto, hay que calcular el valor del índice Gini para estas tres combinaciones:

$$Gini_a(E) = \frac{5}{15}Gini(E_1) + \frac{10}{15}Gini(E_2)$$
$$= \frac{5}{15}\left(1 - \left(\frac{2}{5}\right)^2 - \left(\frac{3}{5}\right)^2\right) + \frac{10}{15}\left(1 - \left(\frac{4}{10}\right)^2 - \left(\frac{6}{10}\right)^2\right) = 0.48$$

$$Gini_b(E) = \frac{11}{15}Gini(E_1) + \frac{4}{15}Gini(E_2)$$
$$= \frac{11}{15}\left(1 - \left(\frac{4}{11}\right)^2 - \left(\frac{7}{11}\right)^2\right) + \frac{4}{15}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) = 0.473$$

$$Gini_c(E) = \frac{9}{15}Gini(E_1) + \frac{6}{15}Gini(E_2)$$
$$= \frac{9}{15}\left(1 - \left(\frac{4}{9}\right)^2 - \left(\frac{5}{9}\right)^2\right) + \frac{6}{15}\left(1 - \left(\frac{2}{6}\right)^2 - \left(\frac{4}{6}\right)^2\right) = 0.474$$

En este caso, la combinación que menor índice Gini tiene es la (b) con 0.**473**.

- **Ocupación:** este atributo tiene ocho posibles valores por lo que las combinaciones posibles ascienden a siete. Aplicando el mismo proceso anterior se llega a la conclusión de que el que menor índice Gini es la segmentación basada en los grupos conformados por Estudiante y Desempleado por un lado y Empleado y Jubilado por otra, con un valor de **0.457**.
- **Propietario vivienda:** en este caso, al ser un atributo binario, solo hay una opción de segmentación, y el índice Gini asociado es **0.457**.
- **Conocimiento mercados:** también se trata de un atributo binario por lo que solo existe una opción de división. El índice Gini es **0.474.**

Teniendo en cuenta estos resultados, el atributo por el que se debe segregar primero podría ser Ocupación (con la agrupación óptima) o Propietario vivienda, ya que ambos obtienen el mismo índice Gini, de manera que ambos consiguen la máxima reducción de impureza (0.48 - 0.457 = 0.023).

Esta es la métrica que utiliza CART para elegir el atributo por el que hará la segregación en ramas siguientes, utilizando siempre el atributo y combinación de menor índice Gini.

2.4.2. Modelos basados en reglas

Se puede considerar que una regla es una secuencia de condicionantes que, en caso de satisfacerse, permiten determinar un hecho o, en el caso de problemas de clasificación, identificar la clase a la que pertenece un determinado elemento.

Un ejemplo de regla sencilla puede ser: *Si el resto de dividir X entre 2 es 0 entonces X es par*; de manera que se puede deducir si X es par o no en función de si satisface la condición de que sea divisible entre dos. La condición a satisfacer puede ser compuesta por una consecución de condiciones simples, como, por ejemplo: *Si el resto de dividir X entre 2 es 0 y el resto de dividir X entre 3 es cero entonces X es divisible entre 6*.

Los modelos de clasificación basados en reglas utilizan un conjunto de diferentes reglas ajustadas a los diferentes atributos del universo de elementos a clasificar para conseguir su clasificación (Han, Pei, & Kamber, 2011).

2.4.2.1. Relación con árboles de decisión

De un árbol de decisión dado, se puede derivar fácilmente un modelo de clasificación basado en reglas, de manera que satisface la clasificación que el árbol de decisión contiene. Para ello basta con recorrer cada posible camino que va desde el nodo raíz a cada una de las hojas (Han, Pei, & Kamber, 2011), de manera que cada rama recorrida es una condición a satisfacer en la regla generada.

Retomando el árbol de decisión del ejemplo de clasificación de vehículos (Figura 3), se puede aplicar este procedimiento para generar las siguientes reglas:

Si el vehículo tiene alas entonces es un avión.

Si el vehículo navega por el mar entonces es un barco.

Si el vehículo circula por raíles entonces es un tren.

Si el vehículo circula por carretera y no tiene motor entonces es una bicicleta.

Si el vehículo circula por carretera y tiene motor y tiene cuatro ruedas entonces es un coche.

Si el vehículo circula por carretera y tiene motor y tiene menos de cuatro ruedas y tiene una cilindrada menor de 49 cc entonces es un ciclomotor.

Si el vehículo circula por carretera y tiene motor y tiene menos de cuatro ruedas y tiene una cilindrada mayor de 49 cc entonces es una motocicleta.

De este modelo de reglas generado se pueden observar varias características:

- Para cada combinación posible de valores para los diferentes atributos hay una regla por lo que, a un universo de atributos alto, el volumen de reglas puede ser inmanejable y difícil de entender.
- No hay conflictos entre reglas, es decir, para un elemento dado, solo puede pasar por una regla.

2.4.2.2. Algoritmo PRISM

El algoritmo PRISM (Cendrowska, 1987), basado en ID3, define un método por el cual se generan reglas que satisfacen un set de datos de entrenamiento o inicial.

Se trata de un proceso iterativo e incremental, de manera que, a cada iteración, se busca maximizar el número de elementos que pertenecen a una clase y minimizar los que no pertenecen a dicha clase, por lo que se van añadiendo condiciones a la regla hasta que el resultado de aplicarla devuelva elementos todos de una misma clase.

Retomando el ejemplo de la Tabla 4, el objetivo es formar reglas para las distintas clases posibles: *Sí es cliente potencial* y *No es cliente potencial*. Comenzando con la primera, el punto de partida es: *Si X entonces Sí es Cliente Potencial.* El objetivo es componer y encontrar los condicionantes para completar la *X* de la regla.

Para elegir la primera condición de la regla, se elige aquel atributo y valor que maximice el número de elementos de la clase que lo satisfacen (Witten, Frank, Hall, & Pal, 2016). Para ello se estima la ratio de cobertura de cada atributo y valor. Ejemplos:

- Edad es igual a 18-35: hay 5 elementos con este valor, de los cuales 3 son clientes potenciales, por lo que la ratio es 3/5.
- Edad es igual a 35-65: hay 6 elementos con este valor, de los cuales 4 son clientes potenciales, por lo que la ratio es 4/6.
- Edad es igual a >65: hay 4 elementos de este valor, de los cuales 2 son clientes potenciales, por lo que la ratio es 2/4.

Este proceso se realiza para todos los atributos y valores posibles y se elige aquel que maximice dicha ratio. En este caso el mayor es cuando Ocupación es Desempleado, donde 3 elementos de 4 son clientes potenciales.

Con este paso se construye la primera condición: *Si Ocupación es Desempleado entonces Sí es Cliente Potencial.* Sin embargo, esta regla aún no es completa ya que su cobertura es de 3 de 4 posibles, por tanto, se busca el siguiente condicionante de la regla para aumentar su exactitud.

Fijando el valor de Ocupación a Desempleado, se repite el cálculo de la ratio de cobertura para cada atributo y valor restantes. Ejemplos:

- Edad es igual a 18-35: solo hay un elemento que sea desempleado y además es cliente potencial por lo que su ratio es 1.
- Edad es igual a 35-85: hay 3 elementos que sean desempleados, de los cuales 2 son clientes potenciales por lo que su ratio es 2/3.

Sucesivamente se computa la ratio de todas las posibilidades y se obtiene que para los criterios de Edad igual a 18-35, Propietario vivienda igual a No y Conocimiento mercados igual a No se obtiene una ratio de 1, es decir, todos sus elementos son clientes potenciales. En aras de maximizar el número de elementos que caigan dentro de la regla, se elige aquel cuya aportación en número de elementos que la satisfacen sea mayor. En este caso, el atributo elegido es Conocimiento mercados igual a No ya que aporta 2 elementos.

Con esto se llega a la regla: *Si Ocupación es Desempleado y No tiene Conocimientos de mercados entonces Sí es Cliente Potencial*.

Esta regla es completa, ya que se eligen a todos los posibles elementos que satisfacen ambas condiciones por lo que, del set de datos se eliminan esos registros y se vuelve a empezar todo el proceso.

El algoritmo PRISM repite este proceso hasta que el universo de set de datos queda vacío, o, lo que es lo mismo, se han encontrado todas las reglas.

2.4.3. Modelo Bayesiano

Thomas Bayes, a finales del siglo XVIII, estableció un método de cálculo de probabilidades condicionadas de manera inversa al que es el procedimiento estándar. Generalmente, la

probabilidad condicionada permite calcular la probabilidad de que ocurra un evento Y a partir de los resultados de un evento X, es decir, calcula las probabilidades de que ocurra Y dado X. El teorema de Bayes estipula justo el mecanismo contrario, para poder calcular la probabilidad de que ocurra X sabiendo Y.

Para revisarlo en profundidad es necesario definir algunos conceptos (Mitchell, 1997):

- **Probabilidad a priori**: es la probabilidad básica o pura de que un evento ocurra, sin tener en cuenta más factores y antes de observar los datos de la muestra. Se representa como *P(X)* para un supuesto *X*.
- **Probabilidad a posteriori:** es la probabilidad de que un evento Y ocurra sabiendo que ha ocurrido X. Se representa como *P(Y|X).*

El teorema de Bayes, estipula que la probabilidad a posteriori de *P(Y|X)* se calcula de la siguiente forma:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Por ejemplo, considerando que, tras una encuesta realizada sobre los gustos sobre la cerveza, se obtienen los siguientes resultados: al 40% de los encuestados le gusta la cerveza, de los cuales el 75% son hombres y el 25% mujeres, y del 60% restante que no le gusta la cerveza, el 40% eran hombres y el 60% mujeres:

1. La probabilidad de que el encuestado sea mujer, sería una probabilidad pura a priori, ya que no se tiene en cuenta ningún otro factor, y esta sería:

$$P(Mujer) = P(Cerveza) * P(Mujer|Cerveza) + P(No\ Cerveza) * P(Mujer|No\ cerveza) = 0.4 \cdot 0.25 + 0.6 \cdot 0.6 = 0.46$$

2. Por otro lado, sabiendo que el encuestado ha sido mujer, la probabilidad que hay de que le guste la cerveza, sería una probabilidad a posteriori ya que se conoce que es mujer y lo que se debe inducir es con qué probabilidad le gustará la cerveza. Para este caso se puede aplicar el Teorema de Bayes, de manera que:

$$P(Cerveza\ |Mujer) = \frac{P(Cerveza\ |Mujer) * P(Cerveza)}{P(Mujer)} = \frac{0.25 \cdot 0.4}{0.46} = 0.217$$

Los métodos de clasificación Bayesianos se basan en este teorema de manera que su principio es el de predecir la probabilidad de que un determinado elemento de un set de datos pertenezca a una clase determinada (Han, Pei, & Kamber, 2011).

Los métodos basados en Bayes son populares entre los procesos de identificación de textos y, más concretamente, en los procesos de filtrado de correo electrónico o spam. Algunos ejemplos detallados se encuentran en Mitchell (1997) y algunos artículos como el de Sahami, Dumais, Heckerman, & Horvitz de la Universidad de Stanford o el de Metsis, Androutsopoulos, & Paliouras (2006) que aportan más detalles sobre esta aplicación de uso.

2.4.3.1. Algoritmo Naive Bayes

Los algoritmos Naive[11] Bayes se basan en la premisa de que los atributos de un elemento no están relacionados entre sí, es decir, que el hecho de que un elemento presente un valor determinado para un atributo no está relacionado con que otro atributo tenga otro valor específico.

El método que estos algoritmos siguen es, a partir de un set de datos inicial o de entrenamiento con elementos, atributos y sus clases asignadas, asignar a un nuevo elemento la clase cuya probabilidad a posteriori es mayor utilizando el teorema de Bayes, de la siguiente forma (Han, Pei, & Kamber, 2011):

- Para un elemento E, el objetivo es buscar la clase C_i cuya probabilidad a posteriori es máxima, o lo que es lo mismo, maximizar la función:

$$P(C_i|E) = \frac{P(E|C_i)P(C_i)}{P(E)}$$

- La probabilidad pura de E (*P(E)*) es constante para todas las clases, es decir, un elemento E tiene la misma probabilidad de pertenecer a cualquier clase, por lo que no impacta en la búsqueda del máximo de la función, o, lo que es lo mismo, solo hay que maximizar el numerador.
- La probabilidad pura de C_i se obtiene como el número de elementos del set de datos inicial que pertenecen a esa clase dividido entre el total de elementos del set (su porcentaje).

[11] Ingenuo o inocente, en castellano.

- Para calcular la probabilidad $P(E|C_i)$, considerando la premisa de que los atributos son independientes entre sí, se aplica:

$$P(E|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

Donde cada x_k es el valor para el atributo k-ésimo del elemento E. Es decir, es el producto de las probabilidades que se tienen de pertenecer a la clase i-ésima teniendo para atributo k-ésimo el valor que tiene el elemento E para dicho atributo.

Tabla 5. Datos de elemento a clasificar según modelo bayesiano

ID	Edad	Ocupación	Propietario vivienda	Conocimiento mercados	Cliente potencial
16	18-35	Estudiante	Sí	Sí	¿?

Retomando el ejemplo indicado en la Tabla 4, suponiendo que se quiere clasificar el elemento ID16 de la Tabla 5, el proceso sería el siguiente:

- Se calculan las probabilidades para cada clase, a partir de los datos del set inicial:
  - $P(Cliente\ Potencial) = 9/15 = 0.4$
  - $P(No\ Cliente\ Potencial) = 6/15 = 0.4$
- Se calculan las probabilidades de pertenecer a cada clase según el valor de cada atributo:
  - Edad:
    - $P(18-35\ |Cliente\ Potencial) = 3/9 = 0.33$
    - $P(18-35\ |No\ Cliente\ Potencial) = 2/6 = 0.33$
    - $P(35-65\ |Cliente\ Potencial) = 4/9 = 0.44$
    - $P(35-65\ |No\ Cliente\ Potencial) = 2/6 = 0.33$
    - $P(>65\ |Cliente\ Potencial) = 2/9 = 0.22$
    - $P(>65\ |No\ Cliente\ Potencial) = 2/6 = 0.33$
  - Ocupación:
    - $P(Estudiante\ |Cliente\ Potencial) = 2/9 = 0.22$

- $P(Estudiante\ |No\ Cliente\ Potencial) = 1/6 = 0.17$
- $P(Empleado\ |Cliente\ Potencial) = 2/9 = 0.22$
- $P(Empleado\ |No\ Cliente\ Potencial) = 2/6 = 0.33$
- $P(Desempleado\ |Cliente\ Potencial) = 3/9 = 0.33$
- $P(Desempleado\ |No\ Cliente\ Potencial) = 1/6 = 0.17$
- $P(Jubilado\ |Cliente\ Potencial) = 2/9 = 0.22$
- $P(Jubilado\ |No\ Cliente\ Potencial) = 2/6 = 0.33$

o Propietario vivienda:

- $P(Propietario\ |Cliente\ Potencial) = 4/9 = 0.44$
- $P(Propietario\ |No\ Cliente\ Potencial) = 4/6 = 0.67$
- $P(No\ Propietario\ |Cliente\ Potencial) = 5/9 = 0.56$
- $P(No\ Propietario\ |No\ Cliente\ Potencial) = 2/6 = 0.33$

o Conocimientos financieros:

- $P(Conocimientos\ |Cliente\ Potencial) = 5/9 = 0.56$
- $P(Conocimientos\ |No\ Cliente\ Potencial) = 4/6 = 0.67$
- $P(No\ Conocimientos\ |Cliente\ Potencial) = 4/9 = 0.44$
- $P(No\ Conocimientos\ |No\ Cliente\ Potencial) = 2/6 = 0.33$

- Ya se pueden calcular las probabilidades para el elemento nuevo:

o Probabilidad para que sí sea un cliente potencial:

$$P(\text{ID16}\ |Cliente\ Potencial) = P(18-35\ |Cliente\ Potencial) * P(Estudiante\ |Cliente\ Potencial) * P(Propietario\ |Cliente\ Potencial) * P(Conocimientos\ |Cliente\ Potencial) = 0.33 * 0.22 * 0.44 * 0.56 = 0.018$$

o Probabilidad para que no sea un cliente potencial:

$$P(\text{ID16}\ |No\ Cliente\ Potencial) = P(18-35\ |No\ Cliente\ Potencial) * P(Estudiante\ |No\ Cliente\ Potencial) * P(Propietario\ |No\ Cliente\ Potencial) * P(Conocimientos\ |No\ Cliente\ Potencial) = 0.33 * 0.17 * 0.67 * 0.67 = 0.025$$

- El algoritmo Naive Bayes predice que el sujeto ID15 pertenece a la clase de No es Cliente potencial.

2.5.Selección de técnicas de clasificación

Tal y como se ha descrito en el apartado anterior, existen distintas técnicas para resolver un mismo problema de clasificación, por lo que se hace necesario saber identificar cuál de estas técnicas se ajusta más a un caso de uso determinado.

Si bien las distintas técnicas no tienen una casuística objetivo a la que le aplican con mejor grado de eficiencia o acierto, sí que cada una puede tener un comportamiento diferente en cada caso, teniendo un mejor o peor nivel de acierto en la clasificación que otras técnicas para ese mismo supuesto.

Para medir este nivel de calidad o exactitud que puede presentar un modelo de clasificación ante un problema dado existen diferentes métricas que permiten comparar de manera cuantitativa la eficacia de los diferentes métodos. Estas métricas se basan, principalmente, en las ratios de acierto o error que un método proporciona para un set de datos de pruebas que no debe ser coincidente con el set de aprendizaje o construcción del modelo (Han, Pei, & Kamber, 2011).

No forma parte de este trabajo describir en detalle todas y cada una de estas métricas, pero sí que se enuncian algunas para su conocimiento:

- Ratio de acierto: proporción de elementos clasificados correctamente con respecto al total de elementos de la muestra.
- Ratio de error: proporción de elementos no clasificados correctamente con respecto al total de elementos de la muestra.
- Sensibilidad: proporción de elementos correctamente clasificados como de la clase mayoritaria o principal con respecto al total de elementos que son de esa clase.
- Especificidad: proporción de elementos correctamente clasificados como del resto de clases que no son la mayoritaria o principal con respecto al total de elementos que son de esas clases.
- Precisión: proporción de aciertos de clasificación de una clase con respecto a la suma del total de aciertos y del total de fallos de clasificación para esa clase.

Adicionalmente existen otros factores a tener en cuenta en la selección del modelo o técnica de clasificación tales como (Han, Pei, & Kamber, 2011):

- Coste computacional.
- Robustez ante ausencia de datos.
- Escalabilidad ante set de datos muy grandes.
- Interpretabilidad o inteligibilidad, es decir, la capacidad de entender o seguir el modelo una vez construido.

2.6. Comparativa y conclusiones

2.6.1. Comparativa

Según todo lo expuesto hasta el momento, ante un problema determinado de clasificación, existen diferentes métodos para afrontarlos, utilizando técnicas basadas en criterios "divide y vencerás" como los árboles de decisión o modelos basados en reglas, o criterios estadísticos o probabilísticos como los modelos bayesianos. Adicionalmente, no explicados en este trabajo, existen métodos más sofisticados tales como las redes neuronales, redes bayesianas, clasificadores basados en casos, etc. que también ofrecen soluciones a esta tipología de problemas.

Todas estas técnicas pueden ser comparables entre sí de una forma cuantitativa aplicando métricas que miden la eficacia y tasa de acierto del método aplicado, aunque, para llevar a cabo estas mediciones, es necesario construir el modelo. Es decir, a priori no es trivial identificar qué método tendrá un comportamiento mejor para un problema dado, ya que para calcular la tasa de acierto es necesario exponer cada método a un set de datos de pruebas sobre el que, de manera empírica, aplicará el algoritmo de clasificación construido a partir del set de datos de aprendizaje y arrojará unos resultados sobre los que obtener dichas métricas.

Sin embargo, entendidas cada una de las técnicas descritas en este apartado, se pueden argumentar algunas ventajas e inconvenientes de cada una de ellas que permita servir de referencia a la hora de tomar una decisión sobre el método más adecuado a priori, sin tener que implementar varios y comparar su rendimiento, tal y como se recoge en la Tabla 6 (Kerbs, 2001) (Jadhav & Channe, 2013).

En definitiva, y tras el análisis realizado de estas técnicas y de sus principales fortalezas y debilidades y entendiendo que los árboles de decisión y los modelos de reglas siguen un

mismo patrón "divide y vencerás" y que pueden ser, incluso, derivados los segundos de los primeros, se puede considerar que la alternativa a estos son los modelos bayesianos o estadísticos. Por tanto, en función del grado de cobertura y exactitud que se exija a la solución al problema en cuestión, se deberá optar por árboles de decisión para garantizar unas ratios de acierto máximas o por los modelos bayesianos si el objetivo está centrado en aproximaciones con un coste computacional mucho menor.

Tabla 6. Comparativa técnicas de clasificación

Técnica	Ventajas	Inconvenientes
Árboles de decisión	✓ Técnica muy extendida y estudiada. ✓ Alta precisión en la clasificación. ✓ Es un método rápido. ✓ Es entendible, es sencillo interpretarlo cuando su tamaño no es muy grande. ✓ Existen implementaciones muy eficaces y eficientes. ✓ Su representación proporciona una visión global de todo el conjunto de datos y clases. ✓ Se puede convertir en un modelo basado en reglas sin demasiado esfuerzo. ✓ Existen diversos criterios de selección de atributos en cada paso del árbol con métricas asociadas que justifican su elección (ID3, C4.5 y CART).	× Se vuelven muy complejos cuando los parámetros son muchos. × Solo permiten un parámetro de decisión cada vez (en cada nodo intermedio). × Tienden a ajustarse demasiado al set de datos de entrenamiento por lo que aparecen muchas ramas que deben ser podadas.

	✓ La incorporación de nuevos parámetros o clases es relativamente sencilla. ✓ No se ve afectado por datos no relevantes.	
Basados en reglas	✓ Son más inteligibles que los árboles de decisión complejos. ✓ Se pueden derivar de un árbol de decisión. ✓ Se pueden valorar varios parámetros a la vez.	× Requiere una clase por defecto ya que es posible que queden indeterminaciones o elementos que no satisfagan ninguna regla. × Es posible que se creen reglas que solo cubran solo un elemento del set de muestra por lo que el número de reglas puede ser excesivamente grande e inmanejable. × No se pueden representar gráficamente de manera sencilla para su comprensión. × Se pueden generar reglas contradictorias.
Modelos Bayesianos	✓ Son ligeros computacionalmente hablando y su rendimiento es bastante bueno. ✓ Las características irrelevantes no tienen peso específico en el modelo, incluso son desechadas. ✓ Si los atributos no están relacionados entre sí ofrece una	× Requiere una gran cantidad de elementos de aprendizaje de manera que las probabilidades calculadas permitan tener unos buenos resultados. × Al estar basado en probabilidad, su nivel de acierto es menor que el del resto de técnicas. × Presupone que los atributos no están relacionados entre sí y en la

eficacia más alta que otros métodos.	realidad es algo que pocas veces ocurre.

2.6.2. Conclusiones

Después de todo el análisis teórico realizado, así como de la comparativa entre las virtudes y defectos de cada uno de los métodos estudiados, y entendiendo que el caso de uso que se persigue con este trabajo tiene un universo finito y bien definido de clases, con un conjunto de atributos no muy extenso y donde la exactitud debe ser máxima o total, ya que el objetivo final es determinar con precisión si una persona puede optar o no a una subvención y, concretamente, a cuál de ellas, se considera que la mejor opción de atajar este problema es mediante la técnica de árboles de decisión.

Aparte de la necesidad de que la solución garantice la clasificación (posible subvención a la que puede acogerse) de un elemento nuevo (persona que accede al servicio), otros factores se han tenido en cuenta para elegir esta técnica como solución:

- Inteligibilidad: como se ha comentado anteriormente, los árboles de decisión se pueden representar gráficamente de una manera muy sencilla, aportando claridad a su uso y al caso de uso que solucionan, pudiendo servir incluso como alternativa para realizar el proceso de manera manual en caso de una contingencia tecnológica. En el caso de este trabajo, los técnicos de EAPN Madrid que serán usuarios de la solución aquí descrita, podrán tener una representación gráfica e inteligible de la lógica que este proceso sigue como apoyo a su aplicación.
- Ampliación futura: entendiendo que el alcance de este trabajo está centrado en ayudas o subvenciones para personas en riesgo de exclusión social o pobreza, se puede fácilmente pensar que podría ser extendido a otro tipo de ayudas o subvenciones relacionadas con otros aspectos, tales como de ámbito laboral o educativo. Considerando que los atributos por los que se pueden aplicar esas nuevas subvenciones pueden ser, en gran parte, comunes a los que aplican en el caso actual, entonces la extensión del árbol de decisión construido ahora es fácilmente ampliable añadiendo nuevas ramas y hojas que permitan reutilizar toda la lógica construida y el aprendizaje del modelo.

- Predicción guiada: uno de los objetivos principales del desarrollo a realizar para EAPN Madrid es que el proceso de predicción de las subvenciones a las que una persona pueda acceder sea guiado en base a una serie de preguntas, cuyas respuestas deben derivar en otras preguntas que vayan cerrando el abanico de opciones. Este proceso se puede trasponer fácilmente como el recorrido desde una pregunta inicial (nodo raíz), a las preguntas intermedias (nodos intermedios) asociadas a las respuestas anteriores (ramas) para llegar a una clasificación (nodo hoja). De esta manera, casi de forma natural, el propio árbol de decisión guía el proceso de clasificación y resuelve el requerimiento inicial.

3. Contextualización

Después del análisis conceptual y teórico de la tipología de problema que se pretende resolver con este TFG, es el momento de profundizar en el alcance del mismo, su origen y finalidad, de manera que sirvan como punto de partida para la captura de requerimientos y el diseño posterior de la solución.

3.1. Contexto del problema

La Red Madrileña de lucha contra la pobreza y la exclusión social, EAPN Madrid, es el resultado de la asociación de 63 entidades que tienen el objetivo común de luchar contra la pobreza y la exclusión social en el ámbito de la Comunidad de Madrid, y, a su vez, forma parte de EAPN España, que es la suma de las diferentes EAPNs autonómicas. Esta asociación pertenece a lo que se conoce como *El Tercer Sector de Acción Social*, que son aquellas entidades tales como asociaciones, fundaciones, etc., que no son ni públicas ni tienen ánimo de lucro cuyo objetivo es el de desarrollar proyectos de carácter social y/o representar a determinados colectivos de cualquier índole con la meta de defender sus intereses de la mejor manera posible, y todo esto lo realizan en base a la colaboración desinteresada de voluntarios y otras entidades, con un pequeño número de profesionales en plantilla (EAPN-ES, 2021).

Una de las misiones de EAPN Madrid es la de desarrollar programas de acompañamiento social y orientación laboral, orientados a todo tipo de personas cuyo riesgo de pobreza o exclusión social es alto, tales como mujeres víctimas de violencia de género, personas con discapacidad, minorías étnicas, jóvenes tutelados, extranjeros sin regularización, etc. En concreto, uno de estos programas es el de proporcionar asesoramiento a personas de esos colectivos acerca del catálogo de subvenciones que tanto el Estado español como la Comunidad de Madrid tienen para ayudar a personas en riesgo de exclusión social o pobreza (EAPN Madrid, 2021).

Para llevar a cabo estas tareas de asesoramiento se necesita un estudio y análisis previos y profundos para poder identificar todas las características de estas ayudas, su público objetivo, las condiciones de las mismas y su aplicabilidad en cada caso, requiriendo este proceso un conocimiento alto en el ámbito jurídico y legal, capacidades que no siempre están presentes en los profesionales de la asociación.

Por ello, EAPN Madrid, en colaboración con la Clínica Jurídica de la Facultad de Derecho de la UNIR, desarrollaron un trabajo de análisis en el que se identifica todo el universo de ayudas de carácter social disponibles por parte de ambas instituciones (Estado español y Comunidad Autónoma de Madrid) junto con las condiciones que una persona debe satisfacer para poder acceder a cada una de ellas. El resultado de este ejercicio es una matriz que cruza las diferentes subvenciones (leyes, órdenes, decretos, etc.) con las diferentes características que una persona debe cumplir para poder acceder a ella. Un ejemplo real es el descrito en la Tabla 7 que recoge las condiciones para poder acceder a las prestaciones de la Seguridad Social.

Tabla 7. Matriz de prestaciones de la Seguridad Social

Prestación	Periodo mínimo cotización – Alta	Periodo mínimo cotización – No Alta	<21 años	<65 años	>65 años
Jubilación ordinaria	15 años				Sí
Incapacidad permanente total	1800 días			Sí	
Incapacidad permanente parcial	1800 días			Sí	
Incapacidad permanente absoluta	1800 días	15 años		Sí	
Gran invalidez	1800 días	15 años		Sí	
Viudedad	500 días	15 años			
Orfandad	500 días	15 años	Sí		
Desempleo	365 días		Sí		

Una vez identificada y definida esta matriz, el siguiente paso del problema es el de sistematizar el uso y explotación de dicha información para que todas las asociaciones de EAPN Madrid puedan utilizarla en sus proyectos de soporte y ayuda a personas vulnerables. Para ello se hace necesario el desarrollo de una aplicación informática que recoja esta base jurídica y aumente su aprovechamiento por la mayor cantidad de personas posibles, a través de los servicios que EAPN Madrid ofrece, proporcionando un elemento centralizador de este servicio y permitiendo su adaptación y actualización futuras de manera transparente para las entidades que hagan uso de él.

Este concepto de Legal Tech que vincula los servicios o asesoramiento jurídicos con una aplicación informática es el principal objetivo del presente TFG, aplicado a este caso de uso y como continuación del trabajo de colaboración ya realizado entre EAPN Madrid y UNIR.

3.1.1. Universo cubierto

El resultado del análisis realizado por la Clínica Jurídica de UNIR y EAPN Madrid contempla la revisión detallada y descripción de las condiciones requeridas de un total de 21 regulaciones aplicables a la población residente en la Comunidad de Madrid, que van desde reales decretos del Gobierno español a órdenes o decretos redactados por el Gobierno de la Comunidad de Madrid.

Estas regulaciones analizadas se traducen en un total de 32 subvenciones diferentes aplicables en función de un conjunto de factores que se deben satisfacer. Dichas subvenciones se agrupan en los siguientes apartados:

- Rentas mínimas y de reinserción:
    - Renta Mínima Vital de ámbito nacional.
    - Renta Mínima de Inserción de la Comunidad de Madrid.
    - Renta Activa de Inserción para desempleados de larga duración.
- Minusvalía y Dependencia:
    - Ayudas diversas (transporte, asistencia sanitaria y farmacéutica, rehabilitación, etc.) de ámbito nacional para favorecer la integración de minusválidos.
    - Atención a personas con dependencia, de ámbito nacional.
- Educación:

    - Becas de diferente índole, tanto de carácter estatal como autonómico, para diferentes aspectos y etapas de la vida educativa.
    - Ayudas al transporte proporcionadas por la Comunidad de Madrid.
- Seguridad Social:
    - Prestaciones por distintos tipos de incapacidad laboral.
    - Jubilación.
    - Pensiones asociadas a condiciones familiares (viudedad, orfandad, etc.)
    - Desempleo.
- Asilo y Extranjería:
    - Integración de extranjeros en España.
    - Asilo y protección para refugiados.

3.1.2. Alcance potencial

Según el INE, los datos que arroja el índice AROPE en la Comunidad de Madrid establecen un porcentaje del 19% de la población de esa comunidad en riesgo de pobreza y de exclusión social, tal y como se refleja en la Figura 6.

Desglosado por las distintas características recogidas por dicho indicador, se obtiene el reparto indicado en la Figura 7, del que se puede apreciar que el grueso de las personas recogidas en el indicador AROPE se debe a tener ingresos por debajo del 60% de la media, con un 57% del total, que podrían ser posibles beneficiarios de las rentas mínimas y ayudas a la educación. Con un 26% le siguen los que viven en hogares con baja intensidad laboral que podrían ser posibles beneficiarios de las ayudas de la Seguridad Social. Y, por último, con un 17% están los que tienen carencia material severa que podrían ser posibles beneficiarios de todas las anteriores.

Figura 6. Evolución del índice AROPE en la Comunidad de Madrid. (Elaboración propia a partir de datos del INE)

Figura 7. Ratio de personas en riesgos de exclusión social y/o pobreza en la Comunidad de Madrid por tipología. (Elaboración propia a partir de datos del INE)

Transformando estos porcentajes a números reales, el potencial público que podría acceder a este servicio ofrecido por EAPN Madrid asciende a casi 1,3 millones de personas[12], por lo que la necesidad de este tipo de soluciones estaría más que justificada.

[12] Según el INE la población de la Comunidad de Madrid a 1 de enero de 2020 es de: 6.779.888 personas.

3.2. Estado del arte

Una vez entendido el alcance del problema, su ámbito de actuación y el contexto que lo engloba, es importante profundizar en cómo se debe abordar la solución al mismo, realizando primero un estudio de campo en busca de soluciones similares o que ya cubran este caso de uso, para identificar el estado del arte de la cuestión.

Para ello el patrón de búsqueda se ha centrado en la existencia de simuladores de subvenciones o ayudas que pudieran cubrir parcial o totalmente el alcance del problema planteado en este trabajo y siempre dentro del ámbito nacional, independientemente de su diseño o plataforma.

De dicho ejercicio se obtienen las siguientes referencias, ordenadas de mayor a menor cobertura o compatibilidad con el alcance requerido.

3.2.1. *Les meves ajudes* – Ayuntamiento de Barcelona

El Ayuntamiento de Barcelona dispone en su página web[13] de un servicio que podría asemejarse mucho al que se plantea en este trabajo ya que, según la descripción del mismo, pretende permitir a cualquier usuario conocer todas las ayudas sociales a las que tiene acceso, independientemente de cuál sea el ámbito de esas ayudas y la administración pública que las proporcione o regule.

Este servicio se llama *Les meves ajudes* (*Mis ayudas* en castellano)[14] y su apariencia es la que se puede ver en la Figura 8.

[13] Página web del Excmo. Ayto. de Barcelona: https://ajuntament.barcelona.cat/
[14] Página web del servicio Les meves ajudes: https://ajuntament.barcelona.cat/lesmevesajudes/es

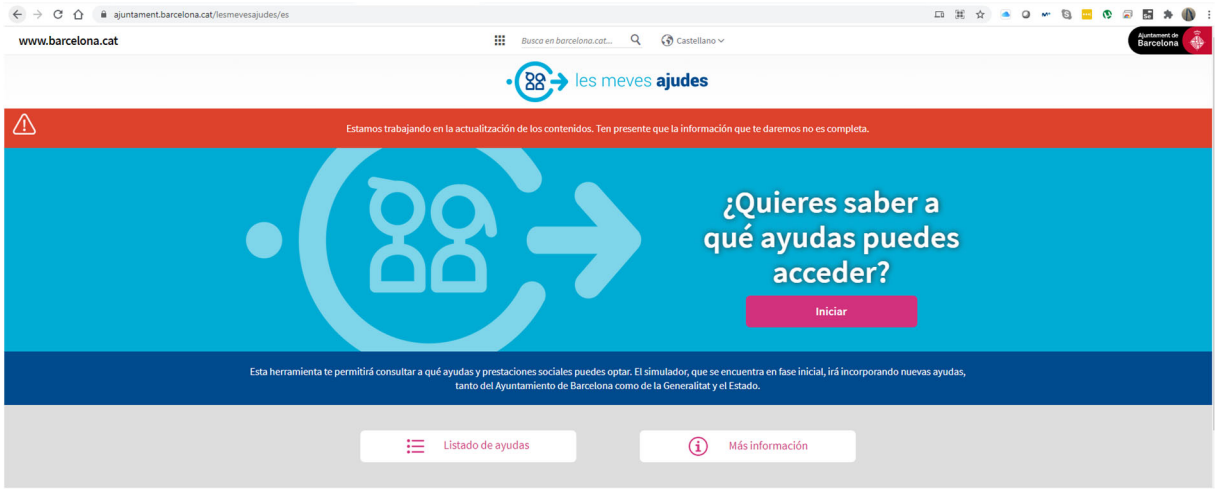


Figura 8. Captura de pantalla del servicio Les meves ajudes del Ayto. de Barcelona

Este portal tiene dos apartados relevantes:

1. Lista de ayudas: se listan y describen las ayudas que están incorporadas en el motor de simulación incluyendo las condiciones que se deben cumplir para poder acceder a cada una de ellas. A la fecha de análisis y redacción de este trabajo (abril 2021) este listado solo contempla prestaciones de aplicación en el ámbito de la ciudad de Barcelona, aunque cubre disparidad de temas, tales como la renta mínima, ayuda de comedor, desahucios, transporte o ayuda al alquiler[15].
2. Simulador: mediante un proceso guiado, se va completando información de las circunstancias laborales, familiares y sociales del individuo y de su unidad familiar con el objetivo final de listar las ayudas a las que se tiene acceso. Las Figuras 9, 10 y 11 reflejan un caso de prueba realizado siguiendo los pasos del simulador. Una vez completado todo el proceso de simulación la herramienta arroja como resultado el listado de ayudas a las que se tiene acceso.

[15] Listado de ayudas incorporadas en *Les meves ajudes*: https://ajuntament.barcelona.cat/lesmevesajudes/es/listado-de-ayudas

Figura 9. Captura de datos personales y familiares en Les meves ajudes.

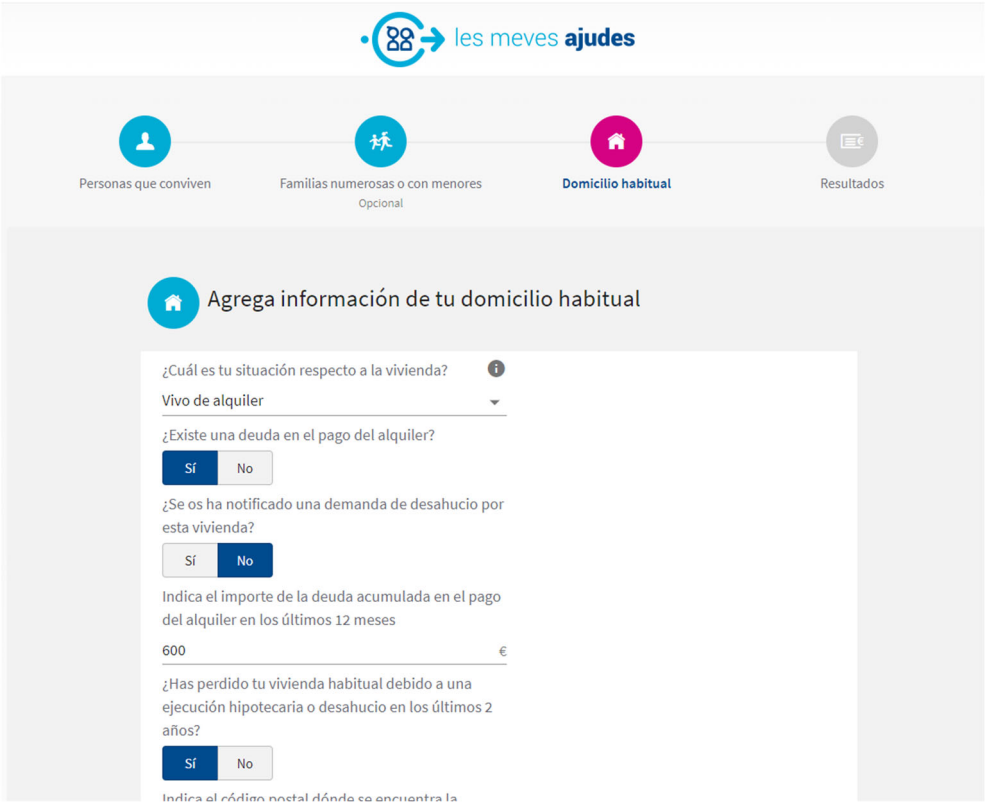


Figura 10. Captura de datos de residencia en Les meves ajudes.

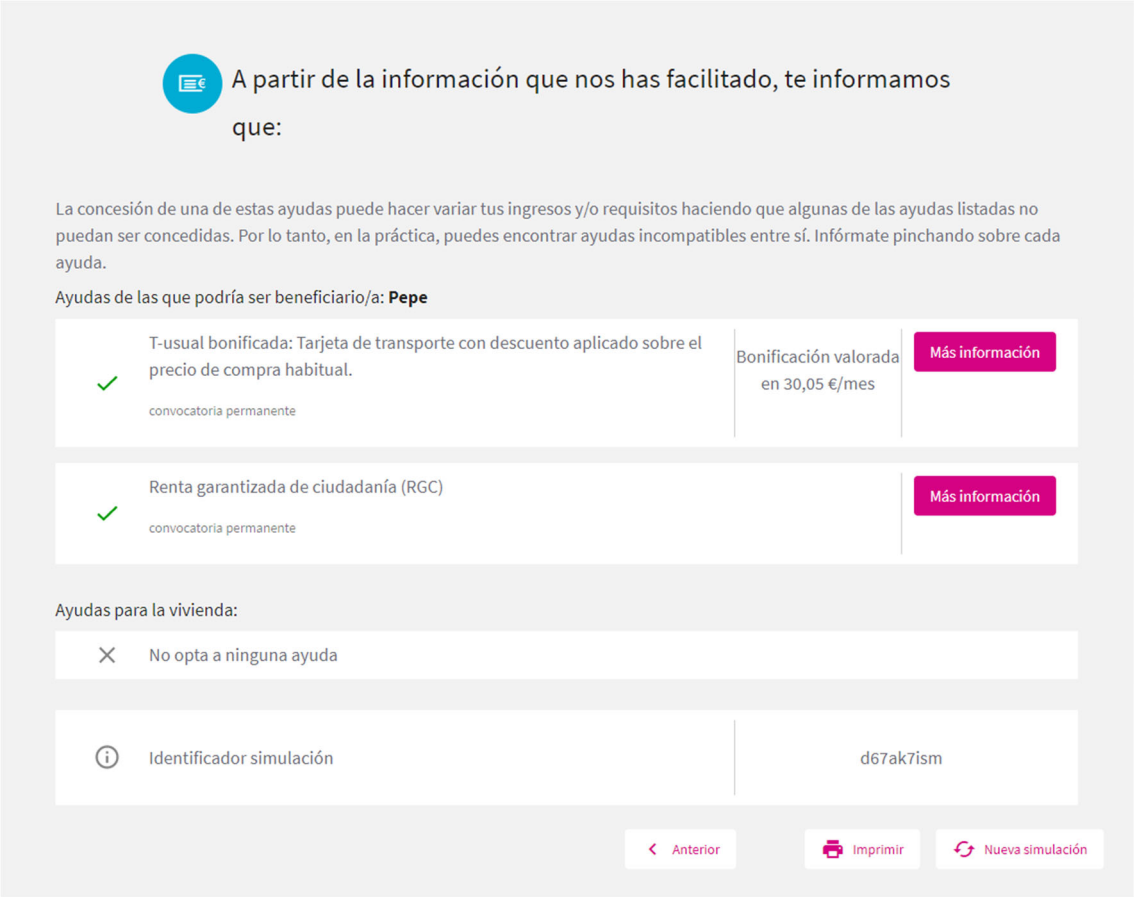


Figura 11. Resultado de la simulación en Les meves ajudes.

Esta utilidad ha sido desarrollada por la entidad Jamgo Cooperativa Tecnológica[16], y en la descripción que se hace de este desarrollo[17] se pueden deducir dos elementos de diseño interesantes:

- Utilizan un motor de clasificación basado en reglas *OpenSource* denominado OpenFisca[18] cuyo objetivo es el de servir de *framework* de desarrollo de reglas relacionadas con aspectos legales o fiscales, teniendo, incluso, desarrollados paquetes de algunos países o ciudades (entre los que se encuentra este caso de Barcelona).
- Este motor de reglas está desarrollado completamente en Python y ofrece un API de servicio consumible desde web para el catálogo de países o legislaciones que tienen disponibles.

[16] Página web de Jamgo: https://jamgo.coop/es/
[17] Descripción de la aplicación Les meves ajudes por parte del equipo desarrollador: https://jamgo.coop/es/project/simulador-ayudas-sociales/
[18] Openfisca: https://openfisca.org/en/

Esta entidad (Jamgo Cooperativa Tecnológica) fue contactada durante el desarrollo de este trabajo para que sirviera de referencia y anticipara los problemas o complejidades que se encontraron en el desarrollo de la aplicación, pero se remitieron al Ayuntamiento de Barcelona para contestar a las preguntas planteadas ya que la propiedad intelectual es de este último. Tras contactar con dicho Ayuntamiento, no se obtuvo respuesta.

En resumen, esta aplicación web, en su alcance propuesto, podría ser una solución para el caso de uso de EAPN Madrid, sin embargo, su desarrollo no está completo y no tiene cobertura de todo lo requerido para el problema planteado. Sí que aporta un punto de partida que puede servir de referencia para el desarrollo de la solución de este trabajo.

3.2.2. Simulador Ingreso Mínimo Vital – Ministerio de Inclusión, Seguridad Social y Migraciones

El Ministerio de Inclusión, Seguridad Social y Migraciones pone a disposición de los ciudadanos una aplicación web que sirve como simulador del Ingreso Mínimo Vital[19], a través del cual, mediante una especie de *chatbot*[20], el usuario interactúa en una conversación respondiendo a preguntas que se le plantean y cuyas respuestas se van teniendo en cuenta durante la conversación para acabar dirimiendo si el usuario es un potencial beneficiario del Ingreso Mínimo Vital y en qué cuantía. En las Figuras 12 y 13 se puede observar un ejemplo realizado.

Al completar la simulación y si, finalmente, se determina que el usuario puede ser beneficiario del Ingreso Mínimo Vital aparece un enlace que redirige a la solicitud online de la ayuda.

Aunque este simulador solo cubre una prestación concreta y no se conocen los detalles técnicos de su implementación y diseño, sí que se puede utilizar como referencia para el desarrollo de la aplicación de EAPN Madrid.

[19] Simulador del Ingreso Mínimo Vital del Ministerio de Inclusión, Seguridad Social y Migraciones del Gobierno de España: https://ingreso-minimo-vital.seg-social-innova.es/

[20] Un chatbot se puede definir como un asistente virtual que interactúa con el usuario a través de un chat o conversación emulando a una persona.

Figura 12. Inicio del Simulador del Ingreso Mínimo Vital

Figura 13. Resultado de simulación de Ingreso Mínimo Vital

3.2.3. Simulador de Renta Mínima de Inserción Laboral de la Junta de Andalucía

La Junta de Andalucía cuenta con una ayuda propia para personas o unidades familiares que viven en riesgo de pobreza o exclusión social, así como en urgencia o emergencia social. Esta ayuda, denominada Renta Mínima de Inserción Laboral, cuenta con un simulador en la página

web de la Junta de Andalucía[21] mediante el cual, introduciendo unos datos en un formulario, se puede validar si se tiene derecho o no a dicha prestación. Las Figuras 14 y 15 sirven para ilustrar esta funcionalidad.

Al igual que ocurría con el Simulador de la Renta Mínima, este simulador también está focalizado en una ayuda específica por lo que no cubre las necesidades planteadas en el presente trabajo.

De igual forma, aun sin conocer los detalles técnicos y de diseño de la implementación llevada a cabo, sirve como referencia para este TFG.

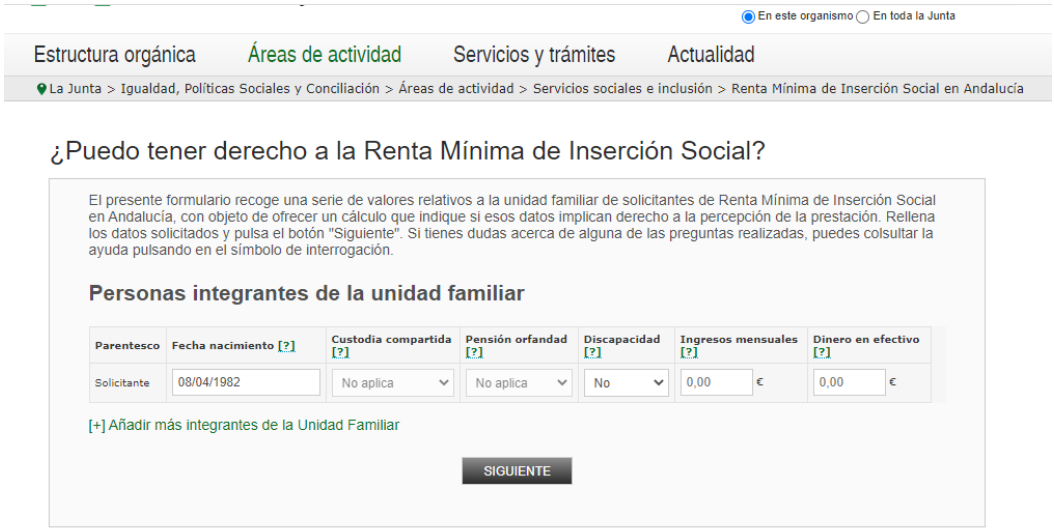


Figura 14. Simulador Renta Mínima de Inserción Laboral de la Junta de Andalucía

3.2.4. Conclusiones

Una vez analizadas las posibles soluciones que podrían ser aplicadas al caso de uso y tras detectar que no existe una aplicación o solución que, de manera nativa, pueda ser aplicada al problema planteado, se puede concluir que el desarrollo de esta aplicación es necesario y, a su vez, novedoso puesto que proporciona una solución inédita que da cobertura a un conjunto de necesidades concretas y hasta ahora insatisfechas.

[21] Simulador de Renta Mínima de Inserción Laboral de la Junta de Andalucía: https://www.juntadeandalucia.es/organismos/igualdadpoliticassocialesyconciliacion/areas/inclusion/rmi/simuladorRMISA.html

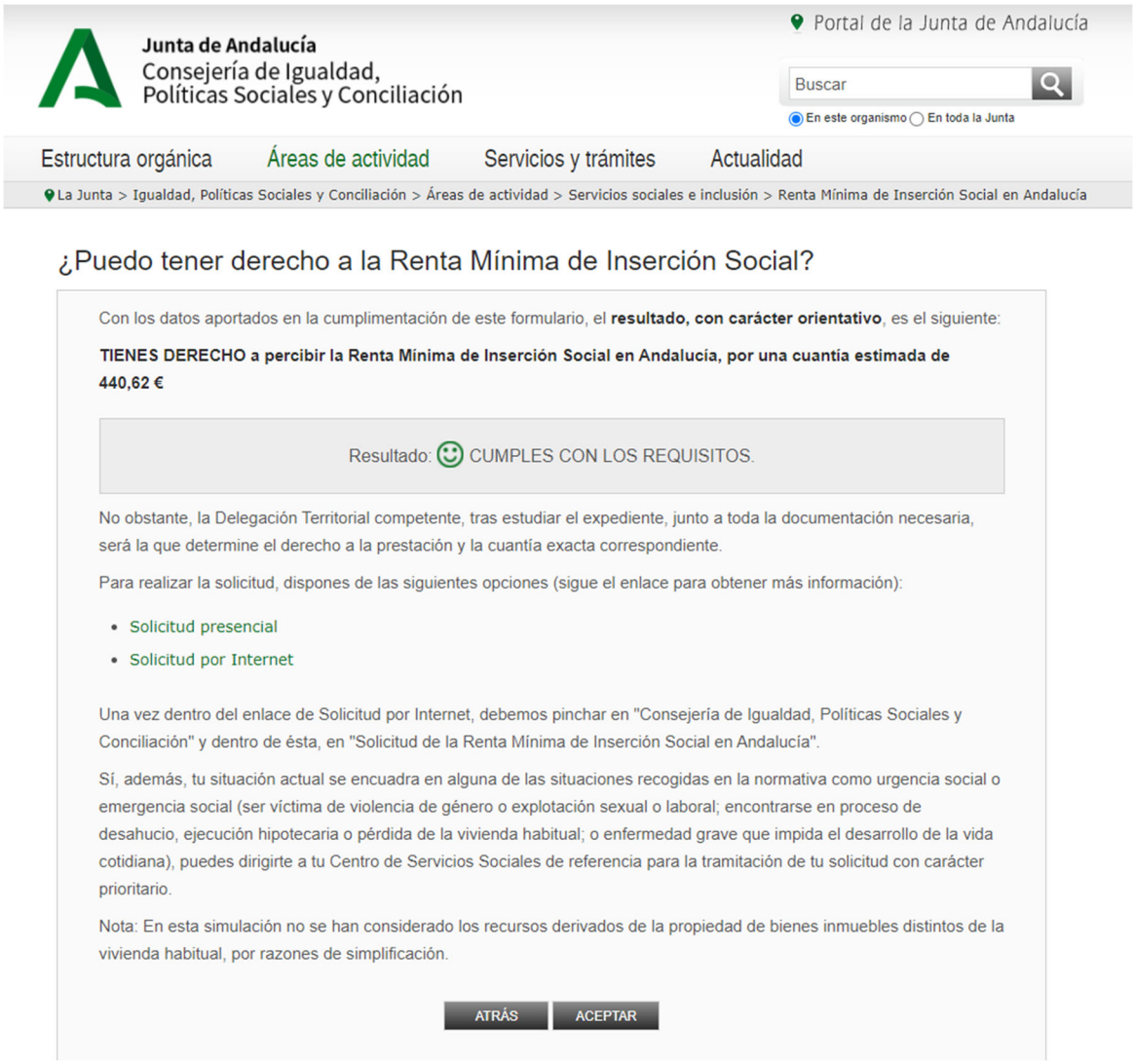


Figura 15. Resultado simulación Renta Mínima de Inserción Laboral de la Junta de Andalucía

3.3. Contexto de la solución

Justificada, por tanto, la construcción y desarrollo de una solución acorde a las necesidades de EAPN Madrid, solo queda describir el contexto de la solución en cuanto a la tecnología a aplicar, aunque en su arquitectura y diseño se profundizará más adelante.

En lo referente a la tecnología a aplicar en el desarrollo de esta solución, teniendo en cuenta que la técnica elegida para resolver el problema de clasificación es la basada en árboles de decisión y que se trata de una solución web, la elección debe basarse en criterios que permitan aunar ambos conceptos.

Dada la diversidad de lenguajes de programación que podrían combinar en una misma solución requerimientos web y de inteligencia artificial, es necesario recurrir a algún indicador

externo para identificar el ranking de lenguajes de programación más comunes o más usados. Para ello existe el indicador TIOBE[22] cuyo objetivo es el de establecer una clasificación de los lenguajes de programación en función de su popularidad a partir del número de páginas web que hay referenciadas a cada lenguaje de programación[23]. Aunque este indicador no habla de la calidad de un lenguaje de programación en concreto o del volumen de código escrito en él, sí que es relevante ya que representa la tendencia a nivel mundial de su uso.

Según este indicador, a fecha de abril de 2021, la clasificación es la indicada en la Figura 16, donde el podio lo ocupan C, Java y Python.

Teniendo en cuenta esta clasificación parece acertado elegir Python como lenguaje de programación ya que, aparte de ser un lenguaje consolidado y en pleno crecimiento en los últimos años (en 2018 y 2020 fue el lenguaje más usado según TIOBE), proporciona soporte para las dos grandes necesidades requeridas para el presente trabajo:

- Desarrollo web: existen entornos de desarrollo completos para desarrollo web basados en Python con una gran extensión, tales como Django[24] o Flask[25], con grandes casos de éxito como Instagram o Pinterest.
- Inteligencia artificial y *machine learning*: Python es el lenguaje más usado para los analistas de datos (un 87%), según la encuesta global realizada por Kaggle en 2020[26].

Adicionalmente, se trata de un lenguaje flexible, sencillo y ligero que no requiere grandes recursos hardware.

[22] Web oficial del índice TIOBE: https://www.tiobe.com/tiobe-index/
[23] Definición del índice TIOBE: https://www.tiobe.com/tiobe-index/programming-languages-definition/
[24] Framework para desarrollo web en Python Django: https://www.djangoproject.com/
[25] Framework para desarrollo web en Python Flask: https://flask.palletsprojects.com/en/1.1.x/
[26] State of Data Science and Machine Learning 2020: https://storage.googleapis.com/kaggle-media/surveys/Kaggle%20State%20of%20Machine%20Learning%20and%20Data%20Science%202020.pdf

Apr 2021	Apr 2020	Change	Programming Language	Ratings	Change
1	2	↑	C	14.32%	-2.40%
2	1	↓	Java	11.23%	-5.49%
3	3		Python	11.03%	+1.72%
4	4		C++	7.14%	+0.36%
5	5		C#	4.91%	+0.16%
6	6		Visual Basic	4.55%	-0.18%
7	7		JavaScript	2.44%	+0.06%
8	14	⇈	Assembly language	2.32%	+1.16%
9	8	↓	PHP	1.84%	-0.54%
10	9	↓	SQL	1.83%	-0.34%
11	19	⇈	Classic Visual Basic	1.54%	+0.71%
12	22	⇈	Delphi/Object Pascal	1.47%	+0.77%
13	13		Ruby	1.23%	-0.02%
14	12	↓	Go	1.22%	-0.13%
15	11	⇊	Swift	1.19%	-0.32%
16	10	⇊	R	1.12%	-0.42%
17	48	⇈	Groovy	1.04%	+0.86%
18	16	↓	Perl	0.99%	+0.03%
19	18	↓	MATLAB	0.99%	+0.06%
20	34	⇈	Fortran	0.91%	+0.58%

Figura 16. Clasificación de lenguajes de programación en abril 2021 según el índice TIOBE. (Fuente web TIOBE)

4. Objetivos y metodología

Una vez completadas las fases de análisis teórico y contextualización del problema, es el momento de poner en práctica los conceptos estudiados con el objetivo de diseñar e implementar una solución que permita identificar las prestaciones a las que se puede tener acceso en función de los datos que se vayan aportando relativos a las situaciones económicas, familiares, sociales, etc. de un individuo concreto que acuda a alguna de las oficinas de EAPN Madrid en busca de ayuda. En los próximos capítulos se dará cobertura a todo el proceso de análisis, diseño y construcción, detallando el procedimiento a seguir, con la intención de tener, al final, una solución completa para el problema planteado. Si bien el alcance puede ser demasiado grande, se intentará dar cobertura a la mayor parte del mismo con el objetivo de que la base construida sirva para su futura ampliación en trabajos posteriores.

En este primer capítulo se detallarán los objetivos concretos que se persiguen en el desarrollo de la aplicación, así como la metodología de trabajo a aplicar en el proyecto de diseño y construcción de la solución.

4.1. Objetivo general

Tal y como se explicaba en la introducción de este TFG, el principal objetivo es el de desarrollar una aplicación web que permita a los técnicos de EAPN Madrid evaluar para cualquier persona que acuda a su servicio de asistencia técnica si cumple o no las condiciones necesarias para poder acceder a alguna de las diferentes ayudas que ofrecen tanto el Estado Español como la Comunidad de Madrid. La aplicación guiará el proceso en base a preguntas o formularios que los técnicos de EAPN irán completando en base a los datos del usuario para perfilar al mismo y obtener así las posibles subvenciones a las que tendría acceso. Es importante aclarar que esta herramienta es una ayuda en el proceso por el que los técnicos de EAPN Madrid prestan asesoramiento a personas que acuden a sus centros, proporcionando un filtrado previo de las prestaciones a las que se podrían tener acceso teniendo en cuenta los datos proporcionados pero que no podrá considerarse como concluyente el resultado obtenido.

4.2. Objetivos específicos

Este objetivo generalizado se detalla y traduce en los siguientes objetivos específicos, que cubren el entregable mínimo:

1. Desarrollo web sencillo y adaptado a las necesidades de los técnicos de EAPN: el equipo de técnicos que conforman la red de asociaciones de EAPN Madrid está compuesto por personas con conocimientos dispares en el ámbito de la ofimática, informática y jurisdicción por lo que es necesario que la aplicación no requiera conocimientos profundos de ninguno de estos ámbitos y que, de forma sencilla, ayude a los técnicos a guiar a las personas que acuden a ellos para obtener ayuda.
2. El diseño debe ser compatible con cualquier dispositivo: puesto que los usuarios objetivos son miembros de asociaciones dispares y, generalmente, con pocos recursos, los dispositivos informáticos que tienen para poder realizar su trabajo pueden ser de diferentes tipologías, tales como tabletas, portátiles, móviles u ordenadores personales con pantallas de diferente índole.
3. Adaptabilidad a cambios de regulación e incorporación de nuevas subvenciones de manera sencilla: el universo de prestaciones está en continuo cambio ya que están sometidas a plazos, ajustes de criterios, incorporación de nuevas prestaciones, etc., por lo que la aplicación debe ser flexible y configurable para poder realizar estos ajustes de manera sencilla y sin requerir desarrollo informático adicional.
4. El sistema debe incluir, de base, el universo de subvenciones identificadas en el momento del desarrollo de este proyecto, como continuación del trabajo realizado por la Clínica Jurídica de UNIR con EAPN Madrid.

Se establecen, además, otros objetivos opcionales que, no siendo necesarios para el funcionamiento del sistema propuesto, sí que permitirán enriquecer el mismo y profundizar en los conceptos teóricos analizados:

1. Incorporación de aprendizaje en el sistema para que, a partir de un conjunto de datos de prueba, construya el árbol de decisión asociado al nuevo universo de subvenciones, en lugar de ser definido por los técnicos de EAPN y la clínica jurídica.
2. Motor de generación de conjuntos de datos de ejemplo para la validación y verificación por parte de los técnicos de EAPN que permita construir y consolidar el árbol de decisión asociado al universo de subvenciones que se pretenden cubrir.

4.3. Metodología de trabajo

La metodología por la que se lleva a cabo la ejecución de este proyecto está basada en el marco definido por Proceso Unificado de Desarrollo (PUD).

4.3.1. Proceso Unificado de Desarrollo

Proceso Unificado de Desarrollo Software define un marco metodológico aplicable al desarrollo software que está basado en las mejores prácticas asociadas a este ámbito y que es fruto de décadas de evolución retroalimentadas por el uso práctico de dichas técnicas (Jacobson, Booch, & Rumbaugh, 2000).

Este marco metodológico se centra principalmente en tres pilares:

1. **Está dirigido por Casos de Uso**: un caso de uso es una funcionalidad completa que debe satisfacer el sistema y que aporta valor para el usuario o actor que invoca al sistema. Todas las actividades y fases del proceso de desarrollo software están basadas y referenciadas a los casos de uso, por eso se dice que la metodología está dirigida por ellos.
2. **Está centrado en la arquitectura**: el concepto de arquitectura software incluye los aspectos estáticos y dinámicos del sistema. Se comienza con un borrador de la arquitectura que es independiente de los casos de uso (plataforma) aunque sin perder de vista los mismos para garantizar que tienen cabida. A partir de ahí, se moldea en base a un subconjunto de casos de uso en términos de subsistemas, clases y componentes y se va madurando iterativamente.
3. **Es un proceso iterativo e incremental**: el proceso de desarrollo de software se realiza en iteraciones donde se van abordando un conjunto de casos de uso de manera que se limita el alcance de cada iteración, siempre aportando nueva funcionalidad para el usuario, y a través de las iteraciones siguientes se van incorporando nuevos casos de uso. Esto permite tener, aparte de versiones del producto de manera frecuente con funcionalidad válida, retroalimentación de los usuarios sobre la funcionalidad ya entregada por lo que se puede incorporar a la siguiente iteración las mejoras o ajustes necesarios.

Proceso Unificado divide su metodología en ciclos en los que, al final de cada uno de ellos, se obtiene una versión del software funcional, validada y lista para su uso en entorno productivo. Cada ciclo se divide en 4 fases (Jacobson, Booch, & Rumbaugh, 2000):

- **Inicio**: fase en el que se describe el alcance del proyecto, con el universo de requisitos a satisfacer y mediante la cual se puede hacer una estimación/planificación del mismo. Conceptos como el modelo de dominio, el de negocio, el documento de requisitos o los riesgos forman parte de esta fase del proceso. Aproximadamente el 20% de los casos de uso son detallados o especificados en esta fase.
- **Elaboración**: fase en el que se completa el análisis (80% de los casos de uso restantes) y se define/diseña la arquitectura del sistema resultante, de manera que se puede hacer una planificación detallada de cuándo se atenderá la construcción de cada bloque de funcionalidad, cómo se faseará, etc.
- **Construcción o implementación**: es la fase en el que el diseño resultante de la fase anterior es llevado a cabo utilizando una combinación de desarrollo, pruebas unitarias, pruebas integradas y depuración de código. Como resultado de esta fase se obtiene la realización de los casos de uso que forman parte del ciclo o versión.
- **Transición**: es la fase que va desde el disponer del producto software o la versión en un entorno operativo real para iniciar la fase de pruebas beta con usuarios hasta su implantación en el entorno productivo y su posterior mantenimiento y soporte.

Cada una de estas fases tiene *n* iteraciones en las que cinco flujos de trabajo se van aplicando en mayor o menor medida. Estos flujos son:

- **Flujo de requisitos**: es el flujo en el que se identifican todas las necesidades o requerimientos que debe satisfacer el sistema a construir, entendiendo como requisito toda aquella funcionalidad de valor visible para el usuario. Este flujo tiene como objetivo acabar generando un listado de requerimientos en un lenguaje que entienda el usuario ya que se acabará convirtiendo en el compromiso entre el usuario y el equipo de desarrollo de lo que hay que construir. Es importante también identificar los requisitos no funcionales que son aquellos que tienen que ver con las restricciones/límites externos tales como seguridad, plataforma, etc.
- **Flujo de análisis**: el objetivo de este flujo es el de traducir los requisitos a un lenguaje más cercano al del equipo de desarrollo eliminando ambigüedades y esbozando

componentes del sistema, incluso con algún prototipo de interfaz. El resultado debe ser un conjunto rigurosamente detallado de funcionalidades con sus restricciones, limitaciones, etc.

- **Flujo de diseño**: a partir del análisis lo que se busca en este flujo es la contextualización de cómo será el diseño de la arquitectura resultante, definiendo los componentes, sus comportamientos, interacciones, interfaces, etc., de manera que sirvan como punto de entrada para la implementación. En este punto se han de elegir las tecnologías y el lenguaje de programación a aplicar, ya que, por ejemplo, el modelo de realización de casos de uso o de clases de diseño deberán ir especificadas ya con el lenguaje de programación.
- **Flujo de implementación**: es por el cual el equipo de desarrollo acaba generando el software y componentes en las tecnologías elegidas en la fase de diseño. Como resultado de este flujo se obtiene una serie de componentes desarrollados y estructurados en objetos tales como librerías, ejecutables, servicios, etc. que han sido validados de manera unitaria e integrada, de manera que están listas para someterse a las pruebas de usuario.
- **Flujo de pruebas**: este flujo lo que proporciona son los mecanismos por los que el software construido debe ser probado para validar y verificar que su desarrollo es correcto. Para ello se utilizan técnicas de pruebas de caja blanca y caja negra, así como pruebas de validación con usuarios en entornos de desarrollo (alfa) y en entornos reales (beta). El plan de pruebas, la definición de los casos de prueba y la ejecución de los mismos forma parte de este flujo de trabajo.

Las principales ventajas que aporta esta metodología y por las que se ha elegido para llevar a cabo este desarrollo son las siguientes:

- Se obtienen versiones funcionales del sistema al final de cada ciclo permitiendo la retroalimentación, verificación y validación.
- Es adaptable a cambios durante el ciclo de vida.
- Permite identificar y gestionar los riesgos asociados al proyecto de manera más efectiva ya que la unidad de gestión es cada ciclo de desarrollo.

Aunque la metodología es muy completa y extensa, tal y como se puede comprobar en Jacobson, Booch, & Rumbaugh (2000), esta es ajustable a cada proyecto por lo que en el caso

de este trabajo no aplicarán todos los artefactos descritos sino que será una versión reducida y proporcional al tamaño del problema a resolver.

De esta forma, las fases de inicio y elaboración, con los flujos de requisitos, análisis y diseño, se cubrirán en el capítulo 5 de esta memoria, desglosadas en las siguientes tareas:

- Definición de casos de uso.
- Identificación de requerimientos tanto funcionales como no funcionales.
- Diseño de la solución desde un punto de vista de arquitectura tecnológica así como funcional.

Las fases de construcción y transición, con los flujos de implementación y pruebas, estarán detallados en el capítulo 6 y están desglosadas en las siguientes tareas:

- Entorno de desarrollo por el cual se decidirá la tecnología propia a utilizar.
- Desarrollo e implementación de la solución diseñada.
- Evaluación de la solución desarrollada.
- Implantación de la solución.

4.3.2. Planificación

Para llevar a cabo el proyecto, una vez identificadas las tareas, sus relaciones y dependencias y cuantificado el esfuerzo necesario para llevar a cabo cada una de ellas, se realiza una planificación de dos meses de trabajo, teniendo en cuenta que solo hay un recurso para llevarla a cabo, cuyo detalle se puede apreciar en la Figura 17.

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1		**Fase de inicio**	**3 días**	**mar 01/06/21**	**jue 03/06/2**
2		Definición de casos de uso	1 día	mar 01/06/21	mar 01/06/2
3		Identificación de requerimientos funcionales	1 día	mié 02/06/21	mié 02/06/2
4		Identificación de requerimientos no funcionales	1 día	jue 03/06/21	jue 03/06/2
5		**Fase de elaboración**	**1 día**	**vie 04/06/21**	**vie 04/06/2**
6		Diseño funcional de la solución	0,5 días	vie 04/06/21	vie 04/06/2
7		Diseño técnico de la solución	0,5 días	vie 04/06/21	vie 04/06/2
8		**Fase de construcción**	**18 días**	**lun 07/06/21**	**mié 30/06/2**
9		Entorno de desarrollo	2 días	lun 07/06/21	mar 08/06/2
10		Desarrollo del motor de clasificación	6 días	mié 09/06/21	mié 16/06/2
11		Desarrollo del frontal web	6 días	jue 17/06/21	jue 24/06/2
12		Desarrollo motor de configuración	10 días	jue 17/06/21	mié 30/06/2
13		Configuración de las prestaciones	1 día	jue 17/06/21	jue 17/06/2
14		Pruebas unitarias	2 días	vie 18/06/21	lun 21/06/2
15		Despliegue entorno integración	0,5 días	mar 22/06/21	mar 22/06/2
16		**Fase de transición**	**21 días**	**jue 01/07/21**	**jue 29/07/2**
17		Ciclo 1 de pruebas de usuario	7 días	jue 01/07/21	vie 09/07/2
18		Revisión y corrección	5 días	lun 12/07/21	vie 16/07/2
19		Ciclo 2 de pruebas de usuario	7 días	lun 19/07/21	mar 27/07/2
20		Implantación	2 días	mié 28/07/21	jue 29/07/2

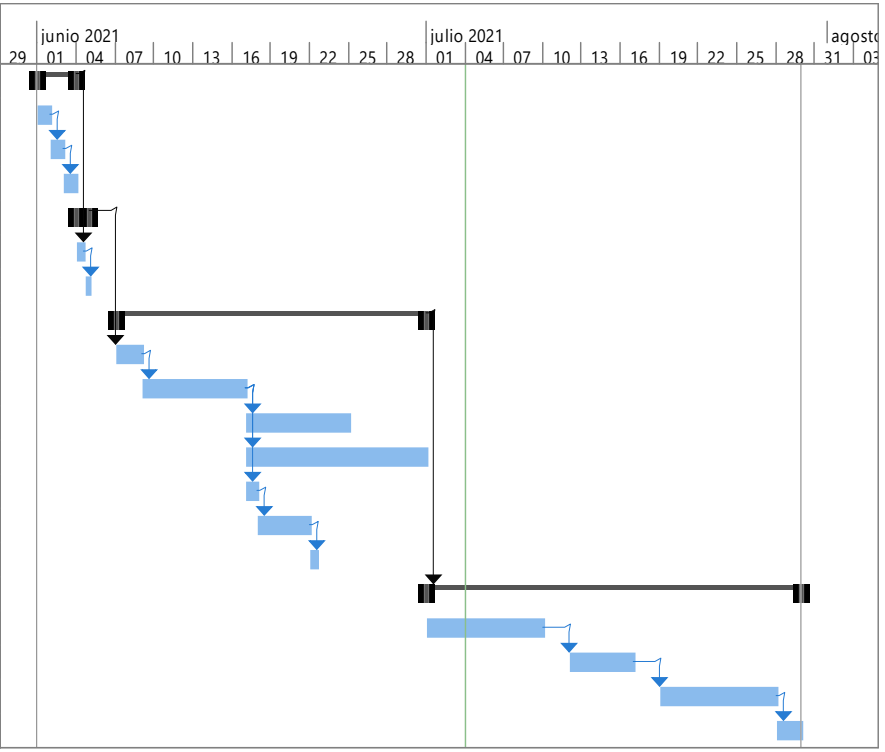


Figura 17. Diagrama de Gantt de la planificación del proyecto

5. Análisis y diseño de la solución

En este capítulo se cubrirán las fases de análisis y elaboración donde se detallarán los diferentes casos de uso que deben ser cubiertos por la aplicación, así como el desglose de esos casos de uso en requisitos funcionales y no funcionales, para acabar derivando toda esta información como punto de partida para poder construir el diseño de la solución.

5.1. Casos de uso y requerimientos

5.1.1. Casos de uso

Antes de definir los casos de uso es necesario identificar los actores relevantes que interactuarán con el sistema:

- Técnico de EAPN: serán los diferentes agentes y técnicos de las oficinas asociadas a EAPN Madrid que utilizarán la aplicación web en nombre de las personas que acudan en busca de ayuda.
- Administrador EAPN: serán las personas que tengan permisos para poder cambiar la configuración de las ayudas, así como incorporar nuevas.

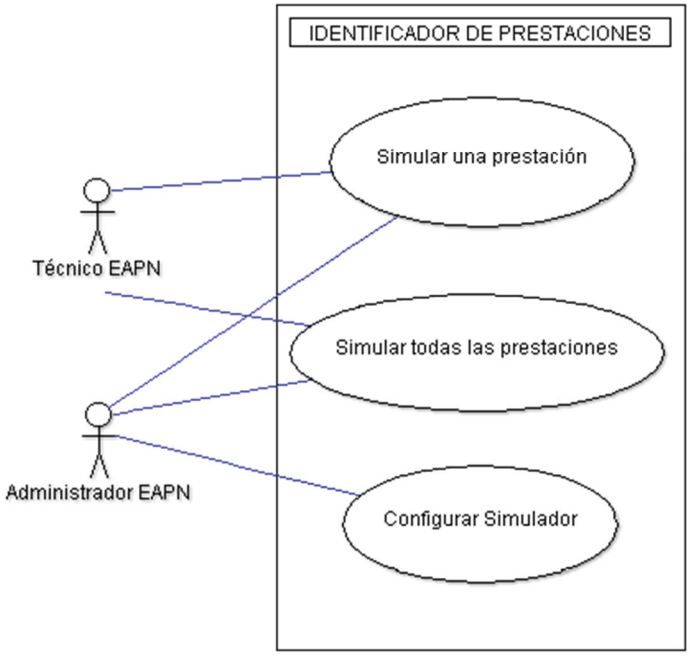


Figura 18. Diagrama de casos de uso del sistema (elaboración propia).

Como se puede apreciar en la Figura 18, en el modelo de casos de uso que aplican a la aplicación, se identifican tres casos de uso cuyo alcance se detalla en las Tablas 8, 9 y 10.

Para cada caso de uso se definen una serie de atributos que permiten contextualizar y explicar el alcance de los mismos:

- Caso de Uso: nombre asignado al caso de uso.
- Descripción: descripción algo detallada del caso de uso
- Actores: actores que participan o inician en el caso de uso.
- Precondiciones: condiciones necesarias que deben satisfacerse para que el caso de uso pueda llevarse a cabo.
- Postcondiciones: acciones o condiciones posteriores a la ejecución del caso de uso.
- Flujo de eventos: acciones que desarrollan el caso de uso. Generalmente solo se indicará el flujo de eventos principal y no todas las posibles alternativas.

Tabla 8. Caso de uso de Simulador de una ayuda

Nombre CU:	Simulador de una ayuda
Descripción:	Verificar si se tiene acceso o no a una determinada ayuda
Actores:	Técnico EAPN, Administrador EAPN
Precondiciones:	La ayuda está activa y configurada en el sistema
Flujo de eventos:	1. El usuario elige la opción Simular una ayuda. 2. El usuario elige una ayuda de las disponibles en el sistema. 3. El sistema inicia el proceso de preguntas asociadas a esa ayuda para evaluar si se cumplen las condiciones o no. Por cada pregunta: a. Si la opción elegida satisface la condición se continua con la siguiente. b. Si la opción elegida no satisface la condición se finaliza el proceso indicando que no se cumplen los requisitos para acceder a esa ayuda. 4. Una vez finalizado todo el proceso de preguntas y condiciones: a. Si todas las condiciones se cumplen se indica que se puede acceder a dicha ayuda y se muestran instrucciones para ello. b. Si no se cumplen todas las condiciones se indica que no se cumplen requisitos para acceder a la ayuda.

Postcondiciones:	Al finalizar se mostrará un resultado (positivo o negativo) en función de si se satisfacen todos los requerimientos asociados a esa ayuda o subvención.

Tabla 9. Caso de uso de Simulador de ayudas

Nombre CU:	Simulador de ayudas
Descripción:	Verificar si se tiene acceso o no a cualquier ayuda
Actores:	Técnico EAPN, Administrador EAPN
Precondiciones:	Hay ayudas configuradas y activas en el sistema
Flujo de eventos:	1. El usuario elige la opción Simular ayudas. 2. El sistema inicia el proceso de preguntas para evaluar si se cumplen las condiciones o no de alguna ayuda. 3. El proceso de preguntas se va ajustando en función de las respuestas anteriores, descartando aquellas que correspondan a ayudas de las que ya se sabe que no cumplen todos los requisitos. 4. Una vez finalizado todo el proceso de preguntas y condiciones se mostrarán: a. Las posibles ayudas para las que se cumplen requisitos. b. Si no se cumplen condiciones para ninguna se indica que no se pude acceder a ninguna.
Postcondiciones:	Al finalizar se mostrará un listado de ayudas para las que se cumplen requisitos junto con los enlaces a la información oficial de cada prestación y a la solicitud de cada una de ellas (si existen).

Tabla 10. Configuración Simulador

Nombre CU:	Configuración Simulador
Descripción:	Configurar el universo de ayudas y los requisitos asociados a cada una
Actores:	Administrador EAPN

Precondiciones:	Ninguno
Flujo de eventos:	1. El usuario accede al módulo de configuración 2. El usuario puede: a. Añadir, modificar, eliminar parámetros o atributos que forman parte de las condiciones que deben cumplirse para cada ayuda (por ejemplo, edad, nacionalidad, etc.). b. Añadir, modificar, eliminar ayudas configurando las condiciones que debe cumplir en función de los atributos definidos, así como la legislación que corresponde a la ayuda, enlaces a la documentación, nombre, etc.
Postcondiciones:	Al finalizar, el sistema aplicará la nueva configuración definida para siguientes simulaciones.

5.1.2. Requisitos funcionales

En este apartado se detallarán los requisitos funcionales asociados al proyecto, esto es, las necesidades que debe cubrir y para las cuales se desarrolla la aplicación. Para cada requisito se definen una serie de atributos a completar:

- Código: código o identificador asociado al requisito.
- Nombre: nombre del requisito.
- Descripción: descripción detallada del requisito.
- Criticidad: indica la criticidad del requisito, siendo alta para aquellos que son irrenunciables y baja para aquellos que pueden relegarse a fases siguientes.
- Prioridad: orden de prioridad del requisito, siendo 1 la más alta y 3 la más baja.
- Relacionado con: si el requisito está relacionado a otros requisitos.

Tabla 11. Requisito funcional 1

Identificador:	REQF01
Nombre:	Opciones del sistema
Descripción:	El sistema debe ofrecer dos opciones para interactuar con él: 1. *¿A qué ayudas puedo acceder?*: mediante esta opción se simulan todas las ayudas disponibles

	2. *¿Puedo acceder a una prestación concreta?*: mediante esta opción se elige una ayuda concreta y se simula si se cumplen sus requisitos
Criticidad:	Alta
Prioridad:	1
Relacionado con:	REQF02, REQG03

Tabla 12. Requisito funcional 2

Identificador:	REQF02
Nombre:	Simulador de ayudas
Descripción:	Mediante esta opción el sistema permite ir capturando datos del usuario para ir dirimiendo a qué ayudas puede acceder de todo el catálogo que tiene configurado. Al final del proceso se indicarán las ayudas para las que se cumplen los requisitos o un mensaje indicando que no se satisfacen los requerimientos para ninguna.
Criticidad:	Alta
Prioridad:	1
Relacionado con:	REQF01, REQF04

Tabla 13. Requisito funcional 3

Identificador:	REQF03
Nombre:	Simulador de una ayuda en concreto

Descripción:	Mediante esta opción el sistema ofrece al usuario la posibilidad de elegir una ayuda o subvención en concreto e iniciar el proceso de simulación, es decir, el sistema irá capturando datos del usuario para ir dirimiendo si cumple o no los requisitos de esta ayuda. Al final del proceso se indicará si se puede optar o no a esa ayuda.
Criticidad:	Alta
Prioridad:	1
Relacionado con:	REQF01, REQF04

Tabla 14. Requisito funcional 4

Identificador:	REQF04
Nombre:	Proceso de simulación
Descripción:	El proceso de simulación que debe realizar el sistema debe ser dinámico y adaptado a las ayudas para las que realiza la simulación, de manera que las preguntas que va realizando el sistema para captar la información del usuario deben tener en cuenta las respuestas a las preguntas anteriores de manera que solo solicite información que aporte valor añadido para la simulación, descartando en cada pregunta las ayudas para las que ya no cumple los requisitos.
Criticidad:	Alta
Prioridad:	1
Relacionado con:	REQF02, REQF03

Tabla 15. Requisito funcional 5

Identificador:	REQF05
Nombre:	Enlace a ayudas
Descripción:	Para cada ayuda para la que el sistema determina que se cumplen los requisitos, el sistema debe proporcionar una referencia o enlace a la documentación oficial de dicha ayuda, así como al documento o web donde se puede solicitar.
Criticidad:	Baja
Prioridad:	3
Relacionado con:	N/A

Tabla 16. Requisito funcional 6

Identificador:	REQF06
Nombre:	Configuración de criterios de una ayuda
Descripción:	El sistema debe proporcionar mecanismos para configurar los parámetros o criterios por los que se determina si se puede optar a una ayuda o subvención. Por ejemplo, límite de edad, nacionalidad, etc.
Criticidad:	Media
Prioridad:	2
Relacionado con:	N/A

Tabla 17. Requisito funcional 7

Identificador:	REQF07

Nombre:	Mantenimiento de ayudas
Descripción:	El sistema debe permitir mantener el catálogo de ayudas de forma que se puedan crear nuevas o modificar las existentes.
Criticidad:	Media
Prioridad:	2
Relacionado con:	N/A

Tabla 18. Requisito funcional 8

Identificador:	REQF08
Nombre:	Glosario de términos
Descripción:	El sistema debe ofrecer descripción de los términos que se muestran en la aplicación para mayor comprensibilidad del usuario.
Criticidad:	Media
Prioridad:	3
Relacionado con:	N/A

5.1.3. Requisitos no funcionales

Una vez descritos lo requisitos funcionales, es el turno de los que se denominan no funcionales, es decir, aquellos que no siendo solicitados por parte del usuario sí que limitan el sistema en función de criterios técnicos.

Tabla 19. Requisito no funcional 1

Identificador:	REQNF01

Nombre:	Aplicación Web
Descripción:	El sistema debe ser una aplicación web compatible con los distintos navegadores: Chrome, Edge y Firefox.
Criticidad:	Alta
Prioridad:	1
Relacionado con:	REQF01, REQF02, REQF03, REQF04, REQF05, REQF08

Tabla 20. Requisito no funcional 2

Identificador:	REQNF02
Nombre:	Web multidispositivo
Descripción:	La aplicación web debe adaptarse a cada dispositivo desde el que se accede a ella (*responsive*).
Criticidad:	Media
Prioridad:	1
Relacionado con:	REQNF01

Tabla 21. Requisito no funcional 3

Identificador:	REQNF03
Nombre:	Diseño sencillo
Descripción:	La aplicación web debe tener un diseño simple y sencillo para facilitar su uso.

Criticidad:	Media
Prioridad:	2
Relacionado con:	REQNF01

Tabla 22. Requisito no funcional 4

Identificador:	REQNF04
Nombre:	Verificación de datos
Descripción:	La aplicación debe verificar que los datos recogidos durante el proceso son correctos y cumplen el formato esperado.
Criticidad:	Baja
Prioridad:	3
Relacionado con:	REQNF01

Tabla 23. Requisito no funcional 5

Identificador:	REQNF05
Nombre:	Protección de datos
Descripción:	El sistema no debe recopilar ningún dato del usuario que pueda considerarse personal o que deba ser protegido.
Criticidad:	Alta
Prioridad:	3
Relacionado con:	REQNF01

5.2. Diseños funcional y técnico

5.2.1. Diseño funcional

Del análisis de los requerimientos se derivan tres elementos o subsistemas funcionales claramente diferenciados:

- **Motor de simulación o clasificación**: este componente se encargará de guiar el proceso de simulación o clasificación de un elemento a partir de la información que le proporcione este y del universo de prestaciones o ayudas que tenga configurado.
- **Motor de configuración**: módulo independiente que permite configurar el universo de prestaciones y cuya salida será consumida por el motor de clasificación. Esta decisión de diseño se justifica en dos criterios principales: por un lado, simplifica la aplicación web y su arquitectura al no requerir modelar permisos de administración ni autenticación, ni base de datos, aumentando la seguridad y protección ante posibles accesos indebidos de manipulación de la configuración; y por otro reduce el número de usuarios que pueden configurar el universo de prestaciones.
- **Frontal web**: aplicación web que será consumidora del motor de simulación y servirá de interfaz de usuario para proporcionar la funcionalidad esperada desde el punto de vista de los técnicos de EAPN Madrid.

5.2.1.1. Motor de simulación o clasificación

Este subsistema es el encargado de realizar la clasificación de un elemento en función de los valores que tome para cada atributo de manera que pueda determinar si satisface o no las condiciones de una o varias prestaciones.

Para llevar a cabo esta funcionalidad se identifican varias clases de diseño, cuyo diagrama de clases asociado se puede observar en la Figura 19:

- **`Attribute`:** esta clase representa un atributo o característica de un individuo en un determinado aspecto. Por ejemplo: la edad, las condiciones económicas, el centro de estudios, grado de minusvalía, etc. Los atributos y sus diferentes valores son los que establecen posteriormente los criterios por los que se definen las diferentes prestaciones. Un atributo estará definido por su nombre, los valores posibles que

puede tener, el tipo de atributo (si es numérico, lista, booleano, etc.), la pregunta asociada al atributo y su descripción para un entendimiento mejor.

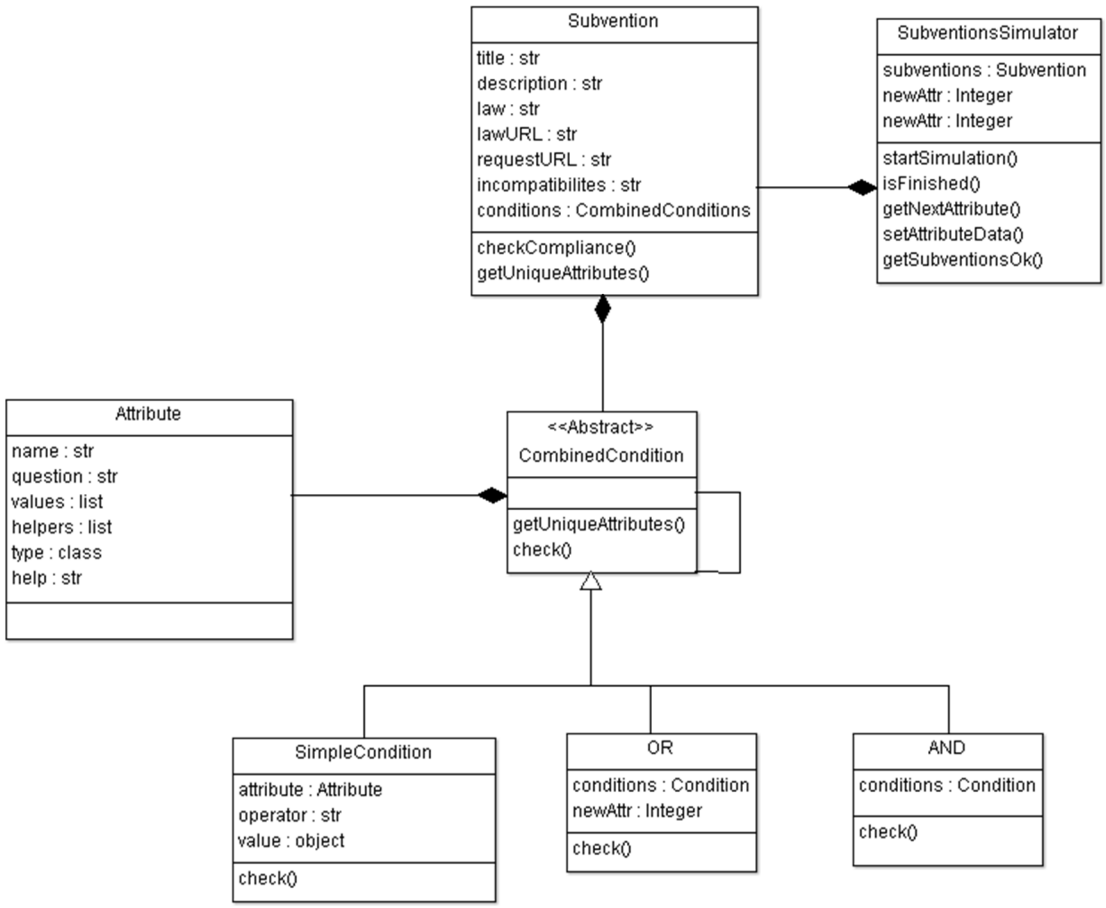


Figura 19. Diagrama de clases del subsistema de simulación

- **`CombinedCondition`:** representa un conjunto de condiciones donde cada condición puede, a su vez, ser también una condición combinada. Se trata de una clase abstracta.
- **`SimpleCondition`:** hereda de `CombinedCondition` y representa una condición sencilla. Esta condición está definida por un atributo, un operador y un valor a satisfacer. Los operadores pueden ser:
  - **`'='`:** para indicar que el valor del atributo debe ser igual al indicado.
  - **`'!='`**: para indicar que el valor del atributo debe ser diferente al indicado.
  - **`'>'`:** para indicar que el valor del atributo debe ser mayor que el indicado.
  - **`'>='`:** para indicar que el valor del atributo debe ser mayor o igual que el indicado.
  - **`'<'`:** para indicar que el valor del atributo deber ser menor que el indicado.

  - **'<=':** para indicar que el valor del atributo debe ser menor o igual que el indicado.
  - **'IN':** para indicar que el valor del atributo debe estar entre los proporcionados.
  - **'NOT IN':** para indicar que el valor del atributo no debe estar entre los proporcionados.
- **OR**: hereda de `CombinedCondition` y representa una condición compuesta por dos condiciones (simples o compuestas) que se evaluarán siguiendo un criterio de *o lógico*, es decir, bastará con que una de ellas sea cierta.
- **AND**: hereda de `CombinedCondition` y representa una condición compuesta por dos condiciones (simples o compuestas) que se evaluarán siguiendo un criterio de *y lógico*, es decir, las dos deben ser ciertas.
- **Subvention:** esta clase representa una prestación de manera que incorpora los datos asociados a la misma, tales como nombre, descripción, URL donde está ubicada la información oficial, URL donde se puede solicitar, las incompatibilidades que tiene y el conjunto de condiciones que debe satisfacer. Proporciona dos métodos relevantes:
  - **checkCompliance**: para evaluar si un conjunto de datos proporcionados a determinados atributos satisface o no las condiciones asociadas a la prestación.
  - **getUniqueAttributes**: para conocer cuáles son los atributos únicos que forman parte de las condiciones que conforman la prestación, ya que un mismo atributo puede formar parte de más de una condición de una prestación.
- **SubventionsSimulator**: esta clase engloba la lógica de clasificación y dirige el proceso de captura de información de forma óptima para resolver el problema de clasificación y detectar las prestaciones a las que se tienen acceso. Es una clase con estado ya que se interactúa con ella mientras el proceso de simulación esté en curso. Sus métodos:
  - **startSimulation**: inicia el proceso de simulación para todas las prestaciones o para un conjunto determinado.
  - **isFinished**: para consultar si la simulación está finalizada o si aún quedan atributos por recopilar información.
  - **getNextAttribute**: este método proporciona cuál es el siguiente atributo sobre el que recabar información.

- **`setAttributeData`**: este método permite asignar un valor a un atributo requerido.
- **`getSubventionsOk`**: en el caso que se haya completado la simulación, retornará el conjunto de prestaciones de las que se satisfacen los requerimientos.

5.2.1.2. Motor de configuración

Mediante este subsistema se ofrecen opciones de gestión de configuración de manera que se pueda cargar desde diferentes fuentes tales como fichero, base de datos, etc. Adicionalmente deberá proporcionar funcionalidad para configurar el universo de atributos, prestaciones y condiciones que estas deben cumplir.

En referencia a la carga de la configuración, el modelo de clases es el indicado en la Figura 20, donde se aprecia una clase abstracta denominada **`SubventionsDataLoader`** que representa el universo de atributos y prestaciones obtenidos a partir de un set de configuración. Esa configuración se puede obtener a partir de diferentes fuentes de manera que bastaría con extender la clase abstracta con una clase que obtenga la información de una fuente diferente (por ejemplo, Excel, json, diferentes bases de datos, etc.).

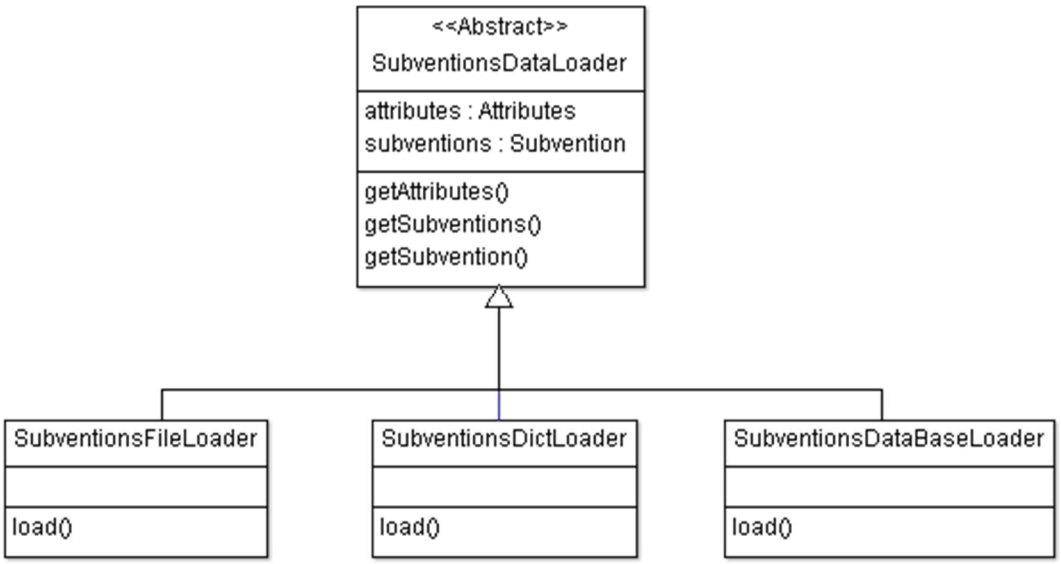


Figura 20. Diagrama de clases del subsistema de configuración

Para la gestión y definición de la configuración de atributos y prestaciones se diseña una solución de escritorio que permita añadir, modificar y eliminar prestaciones y que solo esté disponible para un conjunto delimitado de personas que puedan realizar dichas modificaciones. La decisión de realizarlo en una aplicación de escritorio se basa en mantener la sencillez de la aplicación web donde no hay requerimientos de base de datos ni autenticación y reduciendo a pocas personas la funcionalidad de configuración.

Esta herramienta de configuración dispondrá de dos apartados diferenciados:

1. **Configuración de atributos**: en esta sección se podrá consultar la configuración de cada uno de los atributos registrados en el sistema, así como dar de alta nuevos. Prototipos de diseño de esta sección se pueden ver en las Figuras 21 y 22. En ambas figuras se observa el prototipo de espacio de trabajo para configurar un atributo de manera que hay un menú con botones de creación, eliminación y edición de atributos, una lista a la izquierda donde están los atributos definidos ya en el sistema y en la derecha el detalle del atributo seleccionado. Este detalle está compuesto por los elementos que definen un atributo: nombre, la pregunta que se debe hacer para obtener la información asociada a dicho atributo, la ayuda que permite contextualizar el atributo y entender su significado, si es numérico (Figura 21), lista de opciones (Figura 22) o de Sí/No, y, por último, los posibles valores que puede tener en caso de ser de estos dos últimos tipos.
2. **Configuración de prestaciones**: en esta sección se podrá consultar la configuración de cada una de las prestaciones registradas en el sistema, así como dar de alta nuevas, asociadas siempre al universo de atributos definidos y permitiendo configurar las reglas o condiciones que deben satisfacer. Un prototipo de diseño de esta sección se puede ver en la Figura 23, en la que se aprecia, de nuevo, un menú con botones de creación, eliminación y edición de prestaciones seguido del listado de las prestaciones disponibles en el sistema a la izquierda y a la derecha los detalles de la prestación seleccionada. Este detalle está compuesto por los elementos que definen una prestación: el nombre, el título que aperecerá en la aplicación web, la ley asociada a dicha prestación, una descripción detallada de la especificación de la prestación, la dirección web donde está publicada dicha prestación, la dirección web donde se solicita y, finalmente, un panel inferior conteniendo las condiciones asociadas a los

atributos definidos que deben satisfacerse para poder ser beneficiario de esa prestación.

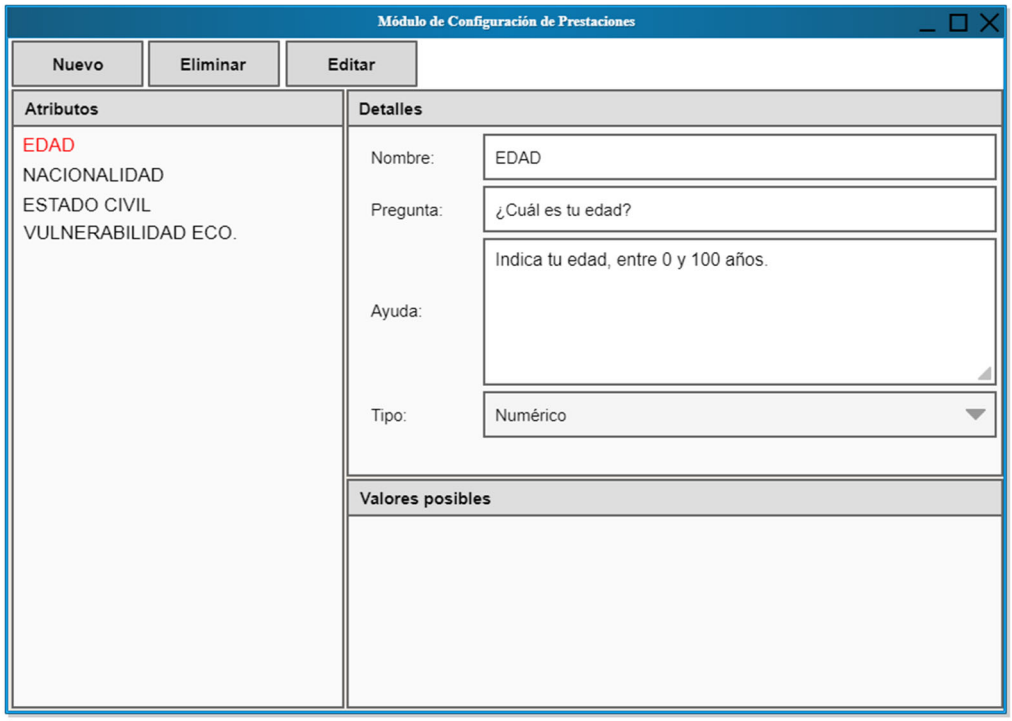


Figura 21. Prototipo de edición de un atributo de tipo numérico

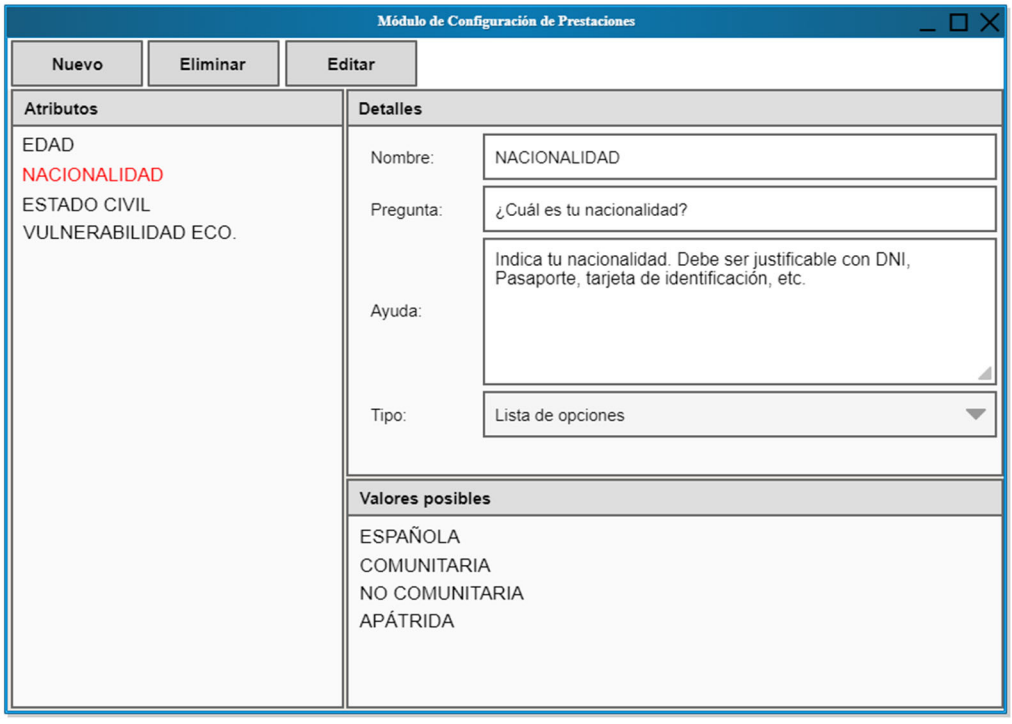


Figura 22. Prototipo de edición de un atributo con varios valores posibles

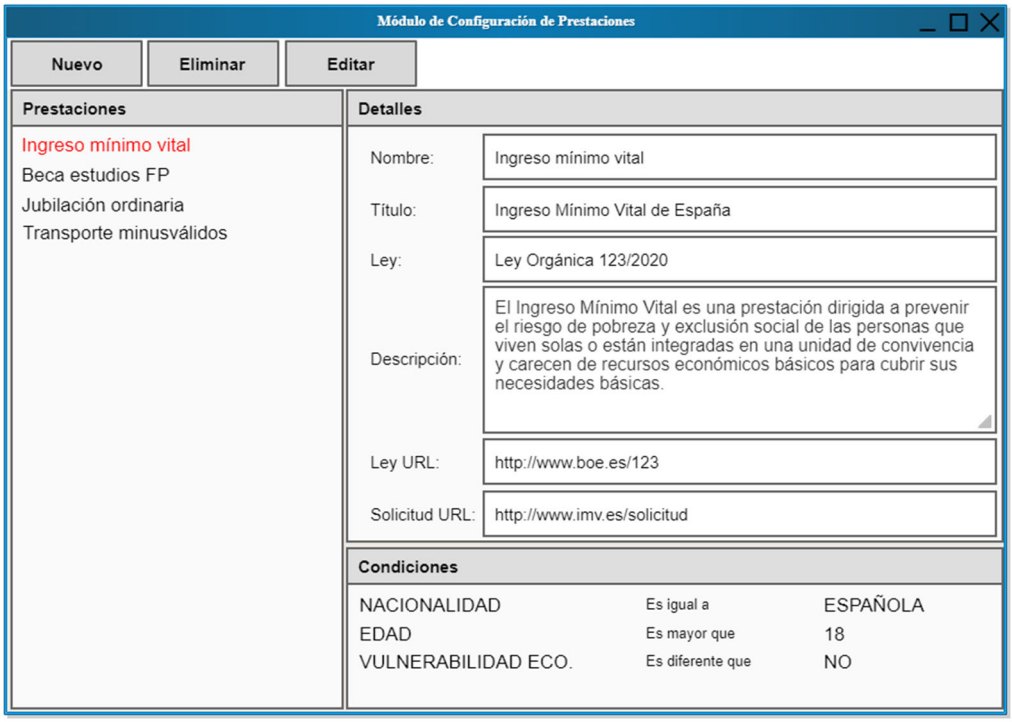


Figura 23. Prototipo de edición de una prestación

5.2.1.3. Frontal web

Este subsistema completa la solución proporcionando la capa de presentación de datos. El diseño de este subsistema se basa en un prototipo elaborado a partir del trabajo previo realizado por la clínica jurídica de UNIR junto con EAPN Madrid, definiendo un diseño web sencillo y minimalista, como se recoge en la Figura 24, que fue refrendado con personal de EAPN Madrid como guía de estilo a seguir en el desarrollo de la aplicación.

Como parte del diseño de este subsistema también se debe dar propuesta al mapa web que se debe desarrollar, estando este compuesto por las siguientes páginas:

- **Principal:** página de inicio donde se haga una descripción de la aplicación y en la que aparezcan las diferentes acciones u opciones de ejecutar la aplicación (simulación de cualquier ayuda concreta o simular todas ellas para identificar a cuál se tiene acceso).
- **Catálogo:** página donde se mostrará el catálogo de todas las prestaciones configuradas en el sistema con una pequeña descripción, así como opciones de simular dicha prestación o consultar enlaces de interés asociados.

- **Simulación**: página dinámica que mostrará para cada atributo a solicitar información las diferentes opciones que debe seleccionar el usuario.
- **Resultados**: página donde se mostrarán las prestaciones a las que, supuestamente, se podría tener acceso ya que se han satisfecho sus requerimientos.

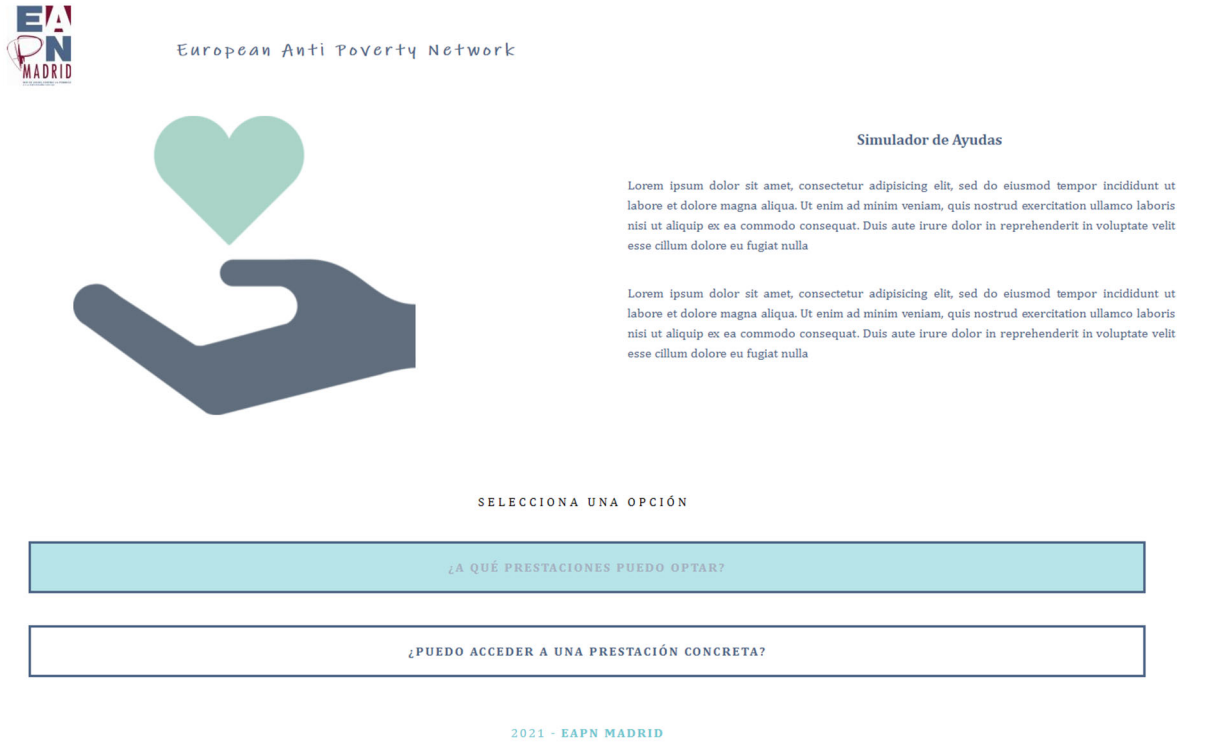


Figura 24. Prototipo de interfaz web para la aplicación

5.2.2. Diseño técnico

El diseño técnico de la solución se basa en una arquitectura Cliente-Servidor de cliente ligero de dos capas, según se aprecia en la Figura 25, basándose en los siguientes puntos:

- La arquitectura es Cliente-Servidor puesto que la aplicación se modela como un servicio prestado a través de un protocolo web, donde los clientes necesitan conocer dónde está alojada la aplicación (dirección url de la web) pero el servidor no conoce la ubicación de los clientes.
- Es de cliente ligero ya que toda la lógica de negocio se encuentra en el lado servidor sin necesidad de desplegar ningún software en el lado del cliente, más allá de un navegador web que será el que permita el acceso a la aplicación.
- Se trata de una arquitectura de dos capas ya que solo se distinguen capa de presentación, que se ejecuta en el lado del cliente a través del navegador, y la capa de

lógica de aplicación que discurre en el lado del servidor. No se requiere capa de almacenamiento o datos puesto que la aplicación en su conjunto se puede considerar estática en cuanto a su contenido en tiempo de ejecución, es decir, no captura datos que deban ser almacenados, ni gestionados.

Sin embargo, para el módulo o motor de configuración, la arquitectura propuesta es la de aplicación monolítica de escritorio que permita generar de manera ágil un fichero de configuración con los datos asociados a las prestaciones, que pueda ser alojado en el servidor de la aplicación web para su aplicación.

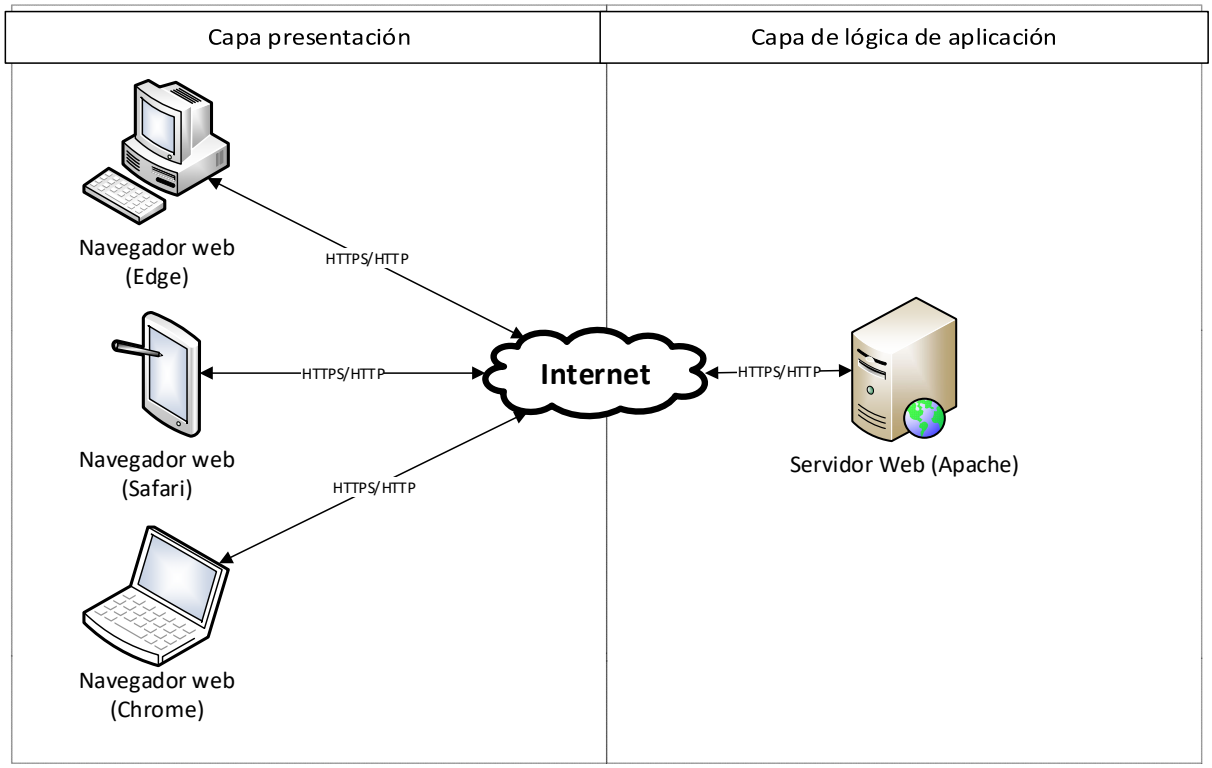


Figura 25. Arquitectura técnica de la solución

Por último, como parte del diseño técnico, es necesario definir las tecnologías a aplicar en cada uno de los subsistemas de forma que el conjunto elegido es el que se observa en la Tabla 24.

Tabla 24. Tecnologías para cada subsistema

Subsistema de simulación o clasificación:	Python
Subsistema de configuración:	Python y Qt

Subsistema de portal web:	HTML, CSS y Javascript

6. Desarrollo y validación de la solución

Una vez completadas las fases de inicio y elaboración, es el turno de trasladar todo el análisis y diseño a una implementación específica que cumpla todas las especificaciones realizadas hasta el momento. Una vez completada la fase de desarrollo, se completará el ciclo de desarrollo con la fase de pruebas y validación y la posterior implantación en un entorno productivo.

6.1. Entorno de desarrollo

Para llevar a cabo el desarrollo del proyecto se ha necesitado configurar un entorno de desarrollo basado en un conjunto de herramientas y módulos que se describen a continuación.

6.1.1. PyCharm 2021.1.2

PyCharm[27] es una solución denominada IDE (*Integrated Development Environment*) que permite el desarrollo de todo tipo de aplicaciones basadas en Python proporcionando utilidades y herramientas que facilitan el desarrollo tales como depuración en tiempo de ejecución, análisis y corrección de sintaxis en tiempo real del código desarrollado, autocompletado de código, etc. Además, tiene un abanico amplio de extensiones que permiten ampliar las funcionalidades de base del producto como, por ejemplo, integración con *Flask* y *GitHub*.

6.1.2. Flask

Flask[28] es una plataforma o *framework* de desarrollo de aplicaciones web en HTML y CSS pero realizando su desarrollo en código Python. De esta forma se puede desarrollar toda la lógica de aplicación en Python apoyándose en esta solución para resolver el problema de servir las páginas web en cada caso.

La elección de Flask respecto a otros *frameworks* que cumplen la misma función y lenguaje de programación, como por ejemplo Django[29], se debe principalmente a la funcionalidad que se

[27] Página web oficial PyCharm: https://www.jetbrains.com/es-es/pycharm/
[28] Página web oficial Flaskl: https://flask.palletsprojects.com/en/2.0.x/
[29] Página web oficial Django: https://www.djangoproject.com/

requiere para la aplicación web, que en este caso es bastante sencilla y sin grandes necesidades tecnológicas como pueden ser APIs, gestión de bases de datos, etc. Para este tipo de escenarios Flask es más ágil y fácil de aprender, tal y como se recoge en la comparativa realizada por Ghimire (2020), aparte de incorporar un servidor web propio que facilita toda la fase de desarrollo, depuración y pruebas sin necesidad de requerir un entorno de test y despliegues continuos, como sí ocurre con Django.

6.1.3. Bulma

Bulma[30] es un *framework* gratuito y de código abierto que proporciona componentes de interfaz web basados y desarrollados en CSS permitiendo el desarrollo de sitios web adaptables o *responsive*.

La inclusión de este *framework* ha sido de gran utilidad ya que ha permitido proporcionar un diseño y estilo sencillos y minimalistas, tal y como era requerido, sin necesidad de definir toda una hoja de estilos desde cero.

6.1.4. GitHub

GitHub[31] es un repositorio de código fuente y control de versiones basado en Git[32] alojado en la nube.

6.1.5. Apache

Apache[33] es un servidor web de código abierto multiplataforma sobre el que se pueden desplegar aplicaciones web de diferente índole.

6.1.6. Qt

Qt[34] es un *framework* multiplataforma para el desarrollo de interfaces de usuario que permite ser usado desde Python para crear aplicaciones de escritorio.

[30] Página web oficial Bulma: https://bulma.io/
[31] Página web oficial GitHub: https://github.com/
[32] Página web oficial Git: https://git-scm.com/
[33] Página web oficial Apache: https://httpd.apache.org/
[34] Página web oficial Qt: https://www.qt.io/

6.2. Implementación

Para llevar a cabo la implementación del proyecto, previamente analizado y diseñado en las fases anteriores de la metodología, se han definido los siguientes módulos Python, todos ellos disponibles bajo el mismo repositorio de GitHub:

- **`SubventionsSimulator`:** módulo que contiene el núcleo del motor de simulación, donde se encuentra la lógica de correlación entre atributos, prestaciones y el proceso de obtención de información y validación de resultados.
- **`SimulatorWeb`:** módulo que contiene la lógica de presentación web basada en Flask.
- **`SubventionsConfigTool`:** módulo con la implementación del cliente de escritorio que permite la configuración del universo de atributos y prestaciones.

6.2.1. SubventionsSimulator

En este módulo Python se implementan las clases especificadas en el diseño que son pertenecientes al subsistema de motor de simulación o clasificación, así como la parte del subsistema de configuración que tiene que ver con la carga de la configuración, es decir, leer el universo de atributos y prestaciones definido.

En lo referente a la implementación de la lógica de simulación, se han aplicado fundamentos similares a la ganancia de información comentada en el capítulo 2, de manera que ante un universo dado de prestaciones del que hay un número *n* total de atributos diferentes, la secuencia de orden por el que se solicitará información será siempre el que más riqueza de información proporcione, es decir, aquel que sea más común entre las diferentes prestaciones, ya que su capacidad de poda en el árbol será mayor que aquellos que solo estén presentes de manera residual. El código significativo se aprecia en la Figura 26, entendiendo que el conjunto `self._attributes` es un diccionario con clave el nombre del atributo y valor el número de prestaciones que lo tienen en alguna de sus condiciones.

A partir del dato proporcionado a cada atributo se lleva a cabo la tarea de poda consistente en identificar todas aquellas prestaciones que tienen a ese atributo como condición necesaria y cuyo valor invalida de manera que, si una prestación ya ha quedado anulada, todos sus atributos deben dejar de tenerse en cuenta reduciendo su ocurrencia en el total. Tras esta poda, los atributos pendientes vuelven a ser ordenados para obtener de nuevo el de mayor riqueza de información, tal y como se observa en la Figura 27.

```
def getNextAttribute_(self):
    """
    Returns next attribute to be asked for
    :return: attribute to be asked for
    """
    if self._attributes:
        self._attributesInOrder = sorted(self._attributes.items_(), key = operator.itemgetter(1), reverse = False)
        self._attributeExpected, count = self._attributesInOrder.pop()
        self._attributes.pop(self._attributeExpected)
        return self._attributeExpected
    else:
        raise NoMoreAttributes_("No more attributes!")
```

Figura 26. Código significativo de ordenado por ganancia de información

```
def _checkCompliance_(self):
    """
    Checks the status of the crossing the list of subventions with the value of the attributes already provided
    :return: updated instance
    """
    for subventionId in self._subventionsToSimulate:
        if subventionId in self._subventionsKo:
            continue
        try:
            subventionOk = self._subventions[subventionId].checkCompliance(**self._attributesData)
            if subventionOk and subventionId not in self._subventionsOk:
                self._subventionsOk.append(subventionId)
            if not subventionOk:
                if subventionId not in self._subventionsKo:
                    self._subventionsKo.append(subventionId)
                attributes = self._subventions[subventionId].getUniqueAttributes()
                for attribute in attributes:
                    try:
                        self._attributes[attribute] -= 1
                        if self._attributes[attribute] == 0:
                            self._attributes.pop(attribute)
                    except KeyError:
                        pass
        except IncompleteDataException:
            continue
    if not self._attributes:
        self._inProcess = False
    else:
        self._attributesInOrder = sorted(self._attributes.items(), key = operator.itemgetter(1), reverse = False)
```

Figura 27. Código significativo de la poda

6.2.1.1. Ejemplo concreto del proceso de poda

Para demostrar el proceso de poda se define un ejemplo concreto en el que se parte de un conjunto de dos prestaciones con un total de tres atributos diferentes y en el que, aplicando la poda, se pasa de tener que realizar tres preguntas inicialmente a completar el árbol de decisión solo con dos.

Dicho ejemplo está compuesto por los atributos:

- `Edad`: numérico.
- `Nacionalidad`: puede ser española, comunitaria y no comunitaria.

- `Estado civil`: puede ser soltero o casado.

```
>>> from SubventionsSimulator import Attribute, Condition, Subvention, Simulator
>>> edad = Attribute.Attribute ("Edad", "¿Cuál es tu edad?", None, None, float,
"")
>>> nacionalidad = Attribute.Attribute ("Nacionalidad", "¿Cuál es tu
nacionalidad?", ['Española', 'Comunitaria', 'No comunitaria'], None, list, "")
>>> estado_civil = Attribute.Attribute ("Estado civil", "¿Cuál es tu estado
civil?", ['Soltero', 'Casado'], None, list, "")
```

Basados en estos atributos se definen dos prestaciones con sus correspondientes condiciones:

- `Ayuda 1`: `Edad` debe ser mayor que 25 años y la `Nacionalidad` debe ser `Española`.
- `Ayuda 2`: `Edad` debe ser menor de 25 años y el `Estado civil` debe ser `Soltero`.

```
>>> #Creación de condiciones para Ayuda 1
>>> condicion1 = Condition.SimpleCondition (edad, ">", 25)
>>> condicion2 = Condition.SimpleCondition (nacionalidad, "=", 'Española')
>>> andAyuda1 = Condition.AND (condicion1, condicion2)
>>> #Creación de Ayuda 1
>>> ayuda1 = Subvention.Subvention ("Ayuda 1", "Ayuda 1", "Ley 1", conditions =
andAyuda1)

>>> #Creación de condiciones para Ayuda 2
>>> condicion3 = Condition.SimpleCondition (edad, "<", 25)
>>> condicion4 = Condition.SimpleCondition (estado_civil, "=", 'Soltero')
>>> andAyuda2 = Condition.AND (condicion3, condicion4)
>>> #Creación de Ayuda 1
>>> ayuda2 = Subvention.Subvention ("Ayuda 2", "Ayuda 2", "Ley 2", conditions =
andAyuda2)
```

Al iniciar el proceso de simulación para estas dos prestaciones se ordenan los atributos de manera que el más significativo, es decir, aquel que está presente en más prestaciones, se solicita primero. En el caso de ejemplo será el atributo `Edad` ya que está presente en las dos prestaciones, quedando pendiente de solicitar los otros dos.

```
>>> simulador = Simulator.Simulator ({1:ayuda1, 2:ayuda2})
>>> simulador.startSimulation ()
>>> simulador.getNextAttribute ()
>>> next = simulador.getNextAttribute()
>>> next.getName()
'Edad'
>>> simulador.getPendingAttributes()
2
```

Al proporcionarle valor a `Edad` de 30, por ejemplo, `Ayuda 2` incumple uno de las condiciones por lo que queda desechada y, con ella, todos los atributos que quedaran pendientes de solicitar que no estén presentes en otra prestación, de manera que el atributo `Estado civil` se descarta y solo quedaría pendiente solicitar el atributo `Nacionalidad`.

```
>>> simulador.setAttributeData('Edad', 30)
>>> simulador.getPendingAttributes()
1
>>> next = simulador.getNextAttribute()
>>> next.getName()
'Nacionalidad'
```

Si a este atributo se le proporciona el valor `Española`, se obtiene un resultado positivo para la `Ayuda 1`.

```
>>> simulador.setAttributeData('Nacionalidad', "Española")
>>> simulador.isFinished()
True
>>> simulador.getSubventionsOk()
[1]
```

6.2.2. SimulatorWeb

El subsistema de presentación está basado en el *framework* de Flask y se aprovecha de la funcionalidad de *templates* que proporciona, de manera que se definen unas plantillas en lenguaje HTML que en tiempo de ejecución son completadas y construidas con lógica escrita en Python.

Para la aplicación web se han definido 3 rutas que engloban toda la funcionalidad requerida:

- **Raíz o *Home***: sirve la página principal de la aplicación donde se exponen las diferentes opciones disponibles, esto es, navegar por el catálogo de prestaciones y simular una de ellas; y simular a qué prestaciones se puede tener acceso, tal y como se aprecia en la Figura 28.
- **Catálogo**: sirve la página con el catálogo de todas las prestaciones configuradas en el sistema y permite acceder a simular cada una de ellas, así como mostrar la información asociada y enlaces, si los tiene, tal y como se muestra en la Figura 29.
- **Simulación:** a través de este punto de entrada se inicia el proceso de simulación para todas las prestaciones disponibles, aplicando la lógica del simulador de prestaciones del módulo anterior (`SubventionsSimulator`). La simulación irá guiando el proceso recabando los diferentes atributos y realizando la poda correspondiente. Para ello, a través de una plantilla definida para tal efecto, se muestran las distintas opciones de cada atributo junto con información adicional para cada posible valor (si este está configurado). Tal y como se aprecia en la Figura 30, se incorpora una barra de avance donde se puede apreciar el número de atributos pendientes de recopilar información para acabar el proceso, de manera que este número va reduciéndose en función de las

podas realizadas. Al final del proceso se muestran las prestaciones a las que se tiene acceso con información adicional a tener en cuenta, tal como las incompatibilidades. En las Figuras 31 y 32 se pueden ver resultados de dos simulaciones, una con éxito y otra sin él, respectivamente.

Figura 28. Página principal de la aplicación web desarrollada

Figura 29. Página con el catálogo de prestaciones

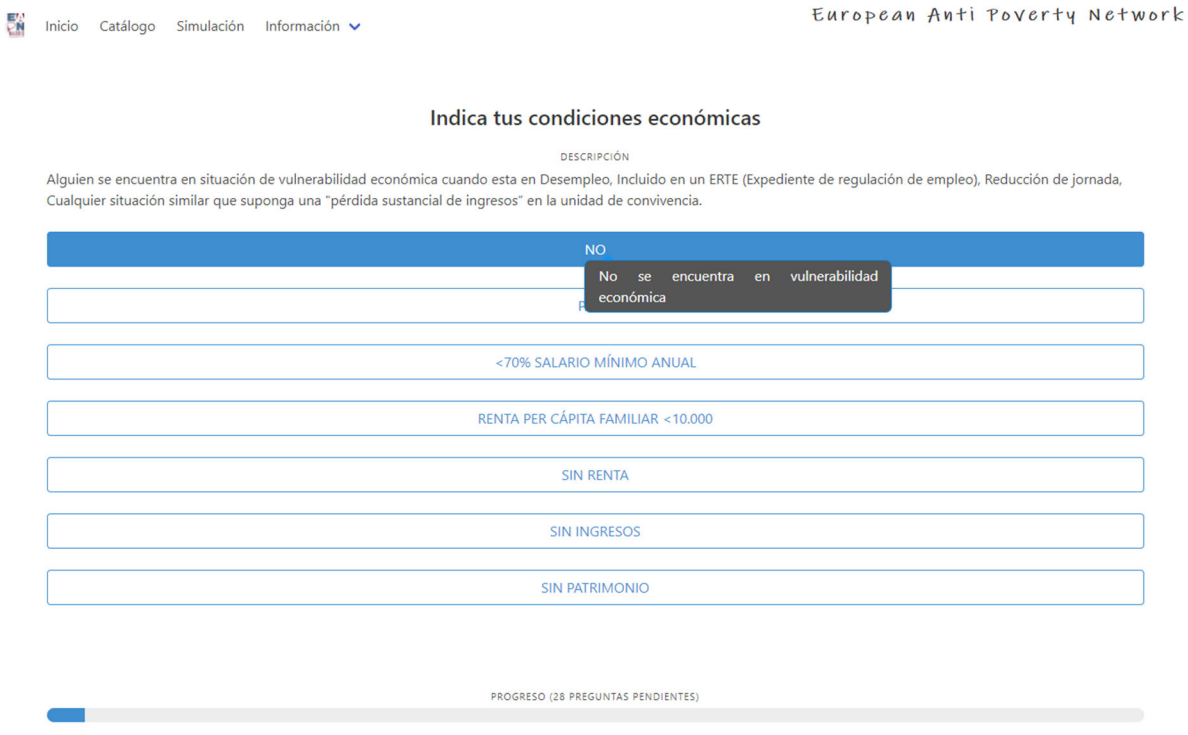


Figura 30. Página con el catálogo de prestaciones

Figura 31. Página con resultado positivo tras una simulación

Figura 32. Página con resultado negativo tras una simulación

6.2.3. SimulatorConfigTool

Por último, el módulo de edición de la configuración se ha desarrollado como aplicación de escritorio permitiendo la lectura y escritura de ficheros de configuración que, tras desplegarlos en el servidor web, se cargarán de forma automática sin necesidad de reiniciar la instancia del servidor.

La herramienta ofrece un interfaz de usuario para realizar las tareas de:

- **Gestión de ficheros de configuración** (abrir, guardar, crear uno nuevo).

- **Gestión de atributos**: alta, baja y modificación de atributos, permitiendo proporcionar toda la información asociada a un atributo y los posibles valores que puede tomar, como se muestra en la Figura 33.
- **Gestión de prestaciones**: alta, baja y modificación de prestaciones, permitiendo proporcionar toda la información asociada a una prestación y todas las reglas y condiciones que debe satisfacer. En lo referente a este último punto la lógica implementada es la de soportar *n* reglas que son condiciones necesarias todas ellas (reglas de tipo *Y*) donde cada una de ellas puede estar compuesta por *m* condiciones donde al menos se debe cumplir una (reglas del tipo *O*), como se aprecia en la Figura 34. En ese ejemplo la prestación tiene tres condiciones a satisfacer donde la última de ellas puede ser satisfecha por dos alternativas posibles.

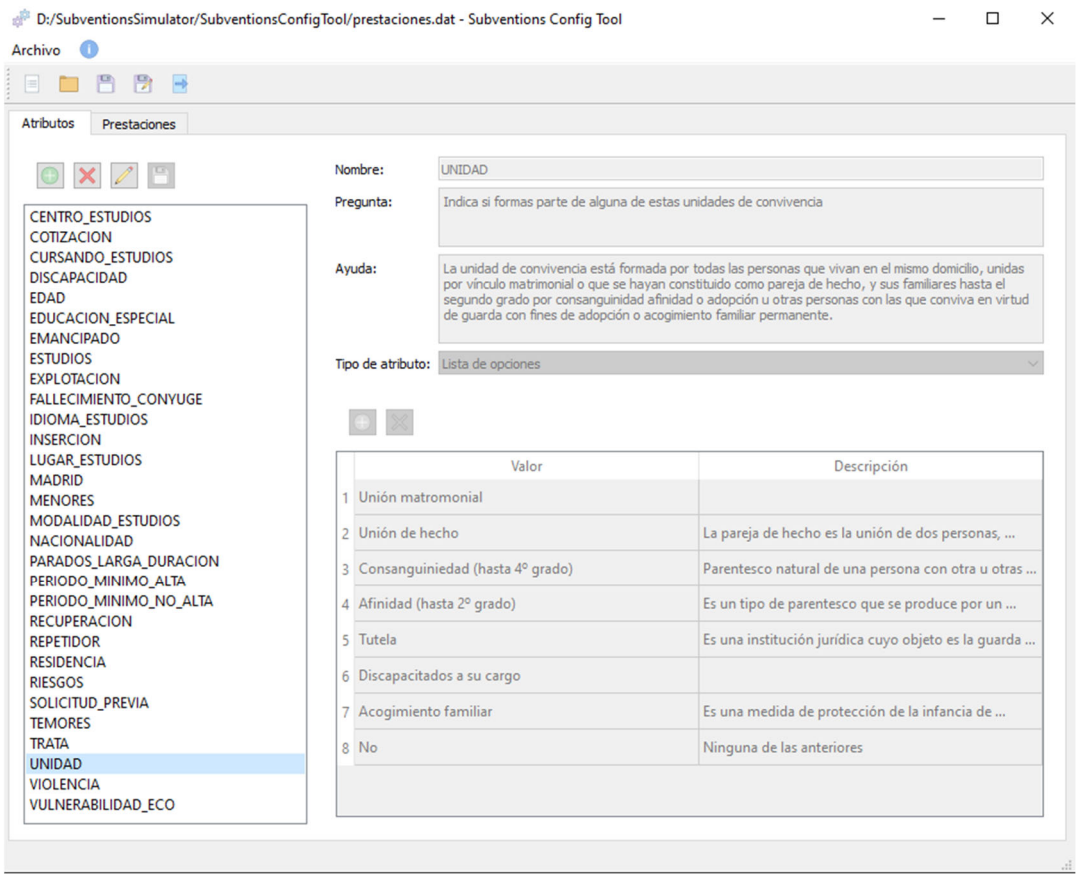


Figura 33. Aplicación de configuración en su apartado de Atributos

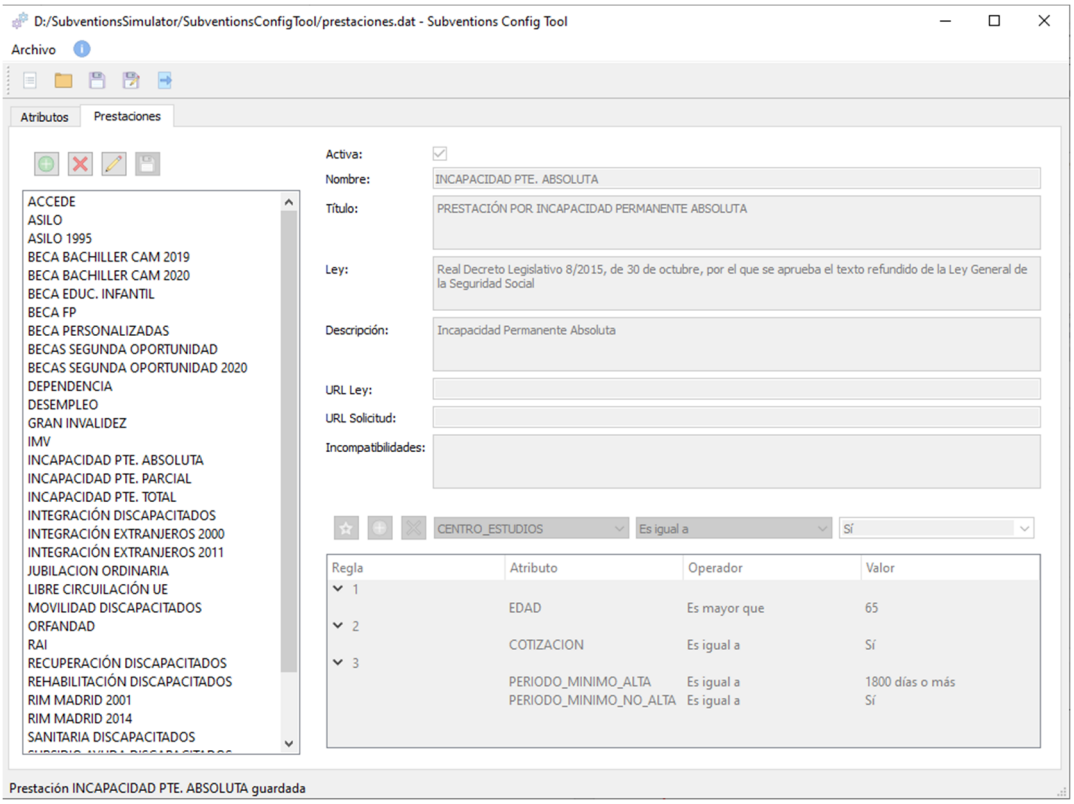


Figura 34. Aplicación de configuración en su apartado de Prestaciones

6.3. Plan de pruebas

La metodología de pruebas seguida se basa en un enfoque incremental de manera que se parte de unas pruebas unitarias o atómicas que validan las clases y sus métodos, continúan por las integradas donde se validan los elementos en su conjunto, y acaban con pruebas de usuario donde usuarios reales en un entorno real prueban y validan la solución desarrollada.

6.3.1. Pruebas unitarias

Las pruebas unitarias no solo se centran en el funcionamiento básico de las clases y sus métodos, sino que se deben validar los extremos del dominio, es decir, validar que ante entradas anormales el sistema se protege y gestiona la excepción.

Para llevarlas a cabo, se ha utilizado el módulo de pruebas de Python llamado *unittest*, el cual permite escribir casos de prueba indicando los datos de entrada y permitiendo comprobar que los datos de salida son los esperados e, incluso, validar que se gestionan las excepciones cuando se exceden los límites del sistema.

Para el caso de los subsistemas de motor de simulación (`SubventionsSimulator`) y de configuración (`SubventionsConfigTool`) se ha utilizado este mecanismo, definiendo un

total de 77 casos de pruebas disponibles en una carpeta denominada *tests* que se encuentra junto con el código fuente. En la Figura 35 se observa el resultado de ejecución de estas pruebas unitarias.

```
(venv) D:\SubventionsSimulator\tests>python -m unittest
.....................................................................................
----------------------------------------------------------------------
Ran 77 tests in 8.143s

OK

(venv) D:\SubventionsSimulator\tests>
```

Figura 35. Resultado de ejecución de pruebas basadas en unittest

6.3.1. Pruebas integradas

Las pruebas integradas son aquellas que validan que, técnicamente, los diferentes subsistemas se entiendan entre sí, trabajando en conjunto para proporcionar una funcionalidad mayor.

En el caso de este sistema, la prueba integrada se realiza sobre el subsistema de capa de presentación (`SimulatorWeb`) que es el que se apoya en los otros subsistemas para proporcionar la funcionalidad esperada.

Para llevar a cabo estas pruebas se ha utilizado Selenium[35], que es un software que permite automatizar pruebas sobre páginas web proporcionando mecanismos para validar elementos tales como que el contenido mostrado es el esperado, simular interacciones de usuario, etc. Una vez definidas estas pruebas se pueden lanzar de manera desatendida y masiva.

En total se han definido 8 casos de pruebas que persiguen validar las diferentes funcionalidades que ofrece el sistema en determinados escenarios, comprobando que el comportamiento es el esperado.

En la Figura 36 se puede observar ejemplo de definición de un caso de prueba y el resultado de la ejecución.

[35] Página web oficial Selenium: https://www.selenium.dev/

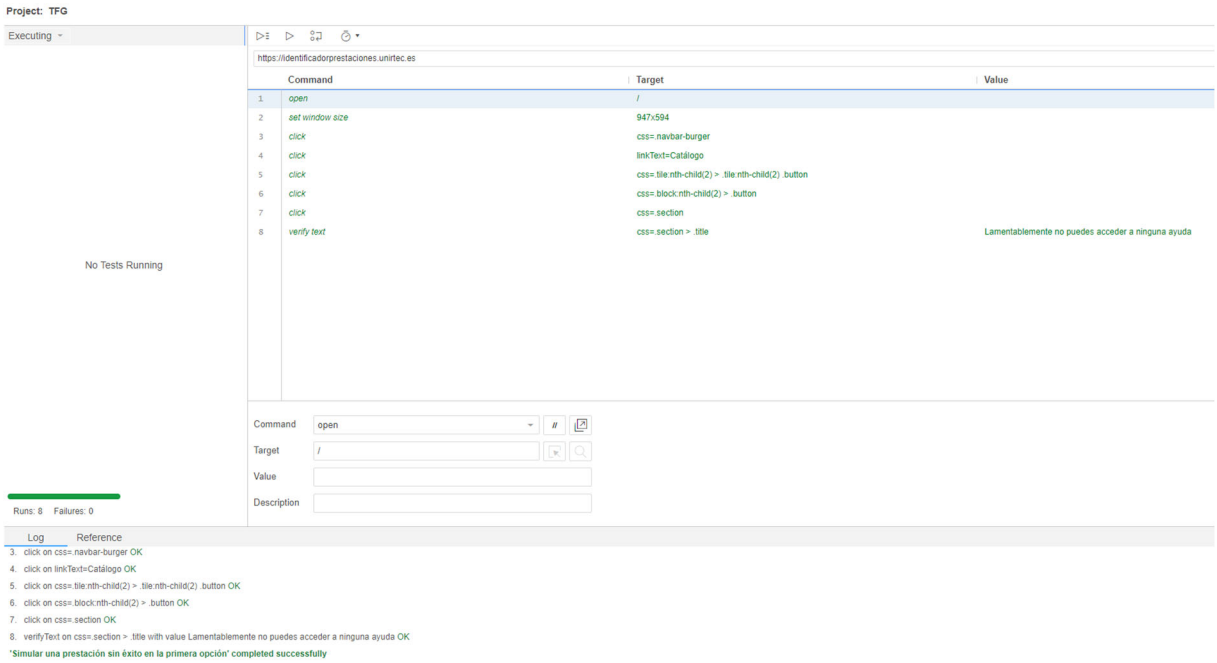


Figura 36. Resultado de ejecución de pruebas en Selenium

6.3.2. Pruebas de usuario

Las pruebas de usuario son las que, una vez desplegado el sistema en un entorno similar al real, usuarios finales del sistema lo usen para revisar si cumple con sus necesidades, validar su usabilidad y determinar si se considera válido para su puesta en producción.

Para llevar a cabo estas pruebas, el sistema ha sido desplegado en un servidor proporcionado por UNIR, con URL de acceso https://identificadorprestaciones.unirtec.es/.

Adicionalmente, para medir la satisfacción de los usuarios, se ha preparado un formulario en el que se pretende recopilar información referente a tres aspectos:

1. **Usabilidad**: medir la facilidad de uso de la aplicación utilizando el Sistema de Escalas de Usabilidad[36].
2. **Funcionalidad**: medir si la aplicación proporciona la funcionalidad esperada y si esta es de utilidad.
3. **Uso potencial y futuro**: identificar la aplicabilidad en las actividades diarias de los técnicos de EAPN y posibles aplicaciones en otros ámbitos.

[36] Definición y fundamentos: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

La encuesta se ha definido utilizando *Google Forms* y se puede consultar en la siguiente url: https://forms.gle/M9u95vJR83U5TU2b9.

Al cierre de esta memoria el equipo interlocutor de EAPN Madrid con el que se ha trabajado para el desarrollo de esta aplicación no ha podido dedicar tiempo a realizar una batería de pruebas de usuario exhaustiva ni han podido cumplimentar y distribuir la encuesta entre las diferentes oficinas de las asociaciones vinculadas a EAPN Madrid, por lo que no se puede incorporar aquí su retroalimentación que permitiera medir el grado de satisfacción y validez de la solución diseñada y desarrollada.

Sin embargo, en aras de obtener alguna retroalimentación, la aplicación ha sido puesta a prueba por algunas personas del entorno con perfiles diferentes y, de igual forma, han cumplimentado el formulario para poder recoger sus impresiones acerca del uso, usabilidad y utilidad (el detalle puede consultarse en el ANEXO A). De este ejercicio se han obtenido las siguientes conclusiones, aunque es importante recalcar que el público objetivo no son técnicos dedicados a las labores para las que se ha desarrollado esta aplicación:

- **Usabilidad:** según el resultado obtenido en el formulario, la calificación promedio obtenida siguiendo el Sistema de Escalas de Usabilidad es de un **90.5**, lo cual se corresponde con una calificación de tipo **A** o con buena usabilidad.
- **Funcionalidad:** en este aspecto, el punto negativo identificado es el relacionado con la terminología y el contexto de las preguntas, es decir, la información que proporciona la herramienta parece no ser suficiente. Este defecto es esperado ya que se necesita conocimiento específico legal y técnico del dominio asociado a las prestaciones para entender determinados conceptos y, además, la fidelidad de aplicación en lo referente a los textos asociados a las ayudas, conceptos y opciones no es definitiva, ya que debe ser proporcionada una versión final por el equipo de EAPN Madrid.

  De manera análoga se identifica un desorden en las preguntas que la aplicación va realizando para captar la información, de manera que no hay una secuencia lógica en cuanto a la temática de las mismas. Este comportamiento también es el esperado ya que el orden de las preguntas está definido por la riqueza de información que proporciona cada una en cada momento para maximizar la poda, en lugar de ser una secuencia fija basada en la tipología de la información. Para entender este

funcionamiento es necesaria una explicación introductoria que sí se haría a los usuarios finales de EAPN Madrid.

- **Uso potencial y futuro:** podría ser una herramienta de uso habitual y podría evolucionarse para incorporar prestaciones de otros ámbitos, así como proporcionar información asociada a los importes o cuantías correspondientes a cada prestación.

En cuanto a errores detectados en la aplicación, se identifican algunos errores ortográficos y comportamientos anómalos cuando se interrumpe una simulación. Estos errores son corregidos y se promociona una nueva versión de la aplicación al entorno anteriormente citado.

6.4. Implantación

Tras la fase de pruebas, y una vez corregidas las incidencias detectadas, el último punto del proceso y del proyecto es implantar la solución en el entorno productivo real definitivo. Sin embargo, a la finalización de este TFG, EAPN Madrid no disponía de infraestructura para desplegar la solución por lo que se acuerda considerar el entorno de pruebas proporcionado por UNIR como solución temporal de producción.

Para el módulo de configuración se ha realizado un Manual de Usuario, adjuntado como ANEXO B del presente trabajo.

7. Estudio de soluciones basadas en aprendizaje automático

Una vez completado el desarrollo del sistema de identificador de prestaciones, se puede usar parte de lo desarrollado para hacer un análisis e investigación sobre cómo podrían aplicarse soluciones basadas en aprendizaje automático supervisado y cuáles serían sus costes (en términos computacionales) y su efectividad. De esta forma cubriremos los dos objetivos adicionales planteados en el apartado 4.2.

El alcance de este estudio se centrará en los tres algoritmos de árboles de decisión analizados en el capítulo 2, esto es, ID3, C4.5 y CERT.

El código fuente creado para llevar a cabo este estudio se encuentra en el mismo repositorio donde está el resto del trabajo, bajo un módulo denominado **`DecisionTrees`**.

7.1. Motor de generación de conjunto de datos

La construcción automática de árboles de decisión se basa en disponer de un conjunto de elementos en el que para cada uno están presentes todos los atributos con sus respectivos valores y la clase a la que dicho elemento corresponde. A partir de ese conjunto de datos, los algoritmos construyen un árbol de decisión que satisface ese conjunto de datos de manera que pueda ser aplicado para clasificar nuevos elementos. Ese conjunto inicial de datos se conoce como conjunto de datos de entrenamiento o aprendizaje.

Una vez construido el árbol de decisión a partir del conjunto inicial, es necesario disponer de otro conjunto de datos diferente que servirá para poner a prueba el árbol construido. Este conjunto se conoce como conjunto de clasificación.

Por tanto, para ambos objetivos, se hace necesario el disponer de un mecanismo que permita la generación de conjuntos de datos. Para ello se ha desarrollado la clase **`DataGenerator`** con los siguientes aspectos:

- Recibe el conjunto de atributos y prestaciones definidos.

Proporciona una funcionalidad de generar datos para un número de elementos, indicado por parámetro, en formato diccionario Python (**`generateData`**). A cada elemento les asigna valor a todos los atributos.

- Proporciona una funcionalidad idéntica a la anterior, pero devolviendo los datos en un formato específico de la librería *pandas*[37] de Python (**`generateDataSet`**).
- Ambas funcionalidades de generación de datos disponen de dos opciones:
    - **Clasificar**: al generar los datos pueden devolver también la clase a la que pertenece cada elemento. Para ello hacen uso del motor de simulación desarrollado.
    - **Multiclase**: habilita o no que un elemento pueda tener acceso a más de una prestación o no. En el caso que esté habilitado, por cada elemento creado se generarán tantos registros como prestaciones tenga acceso, en caso contrario solo aparecerá la primera prestación a la que tiene acceso.

7.1.1. Proceso de generación

Este proceso de generación reemplaza el proceso de capturar datos del mundo real y que hayan sido clasificados por técnicos expertos en jurídica de manera que sirvieran de entrenamiento al motor. Sin embargo, en este estadio primario del análisis, se desconoce cuál sería el número ideal de elementos clasificados para obtener un rendimiento óptimo por lo que este proceso de generación proporciona dos grandes ventajas: permite tener volúmenes de datos ilimitados y servirá para determinar el número ideal de elementos que serán necesarios para tener un algoritmo de clasificación con ratios de éxito óptimos.

Para llevar a cabo la generación de los datos se utiliza, principalmente, la pseudo aleatoriedad que proporciona el módulo *random*[38] de Python. Sin embargo, para evitar la inverosimilitud que pueden tener los datos generados de manera aleatoria, como por ejemplo que se genere un elemento cuya edad sea 3 años e indique que ha cotizado más de 15 años a la seguridad social, se deben aplicar algunas restricciones.

Estas restricciones se definen a nivel de atributo, de manera que se genera una precondición para que ese atributo tenga un determinado valor a que otro atributo tenga o no un valor

[37] Es una extensión de Python específica para gestión y análisis de datos, que proporciona funcionalidades potentes de gestión de grandes volúmenes de datos, filtrados, selección de subconjuntos, etc. (https://pandas.pydata.org/)

[38] Módulo de Python que proporciona funciones para generar números pseudo aleatorios. En la aplicación realizada en este trabajo se considera la semilla inicial el reloj del sistema. (https://docs.python.org/3/library/random.html)

concreto. Por ejemplo, en el siguiente atributo se define una dependencia para que la edad sea inferior a 18 años:

```
'EMANCIPADO': {'NAME': 'Emancipado',
               'QUESTION': '¿Estás emancipado?',
               'VALUES': ['Sí', 'No'],
               'HELPERS': None,
               'TYPE': bool,
               'HELP': 'Si es menor de edad pero se ha independizado.',
               'DEPENDENCIES': [('EDAD', '=', '<18')]},
```

Con esto se aumenta la calidad de los elementos generados y su aproximación a casos reales.

En un escenario real, los datos de entrenamiento para generar el árbol de decisión se pueden proporcionar de dos formas:

- Capturando o generando elementos del mundo real, como comentando anteriormente, en un volumen de elementos suficiente que asegure ratios de acierto óptimos. Esto requerirá que técnicos expertos en la materia realicen el inventario y clasifiquen cada elemento (identifiquen las prestaciones a las que tiene acceso).
- Utilizando este motor de generación, pero analizando y proporcionando más restricciones al proceso para que los datos generados sean más próximos a elementos del mundo real. Esto requiere igualmente la colaboración de técnicos expertos que supervisen el universo de datos generados y permitan identificar esas restricciones.

7.1.2. Simplificación

Adicionalmente a la inclusión de dependencias entre atributos y con el objetivo de simplificar los árboles de decisión generados durante esta investigación, se ha discretizado el atributo EDAD para acoplarlo a un conjunto finito de opciones.

El atributo EDAD está definido inicialmente como un valor entero, lo que hace de él un atributo de tipo continuo, por lo que en la aplicación de árboles de decisión se podría dar el caso que hubiera una rama por cada valor posible (infinitos) o requeriría aplicar lógica para determinar cuáles deberían ser los puntos de corte o intervalos. Para reducir esta complejidad se ha optado por convertir el atributo EDAD en un valor discreto y categórico (texto) de varias opciones. Estas opciones se deducen del análisis previo de la Clínica Jurídica donde se identificaban las condiciones de las prestaciones en lo referente a la edad no en términos numéricos sino de umbrales, tales como: menor de 3 años, mayor de 18, mayor de 21, mayor

de 30, menor de 30, mayor de 65 y menor de 65. Usando ese criterio, el atributo se puede modelar como una lista de opciones con esos posibles valores.

En la Figura 37 se puede observar un conjunto de 10 elementos generados con algunos de sus atributos y la clase a la que corresponde cada uno.

	NACIONALIDAD	RESIDENCIA	MADRID	...	FALLECIMIENTO_CONYUGE	TEMORES	CLASS
0	Comunitaria	España	No	...	No	No	INTEGRACIÓN EXTRANJEROS 2011
1	Comunitaria	España	No	...	No	Raza	INTEGRACIÓN EXTRANJEROS 2011
2	No comunitario	Madrid	Sí	...	Sí	Género	ASILO 1995
3	Apátrida	Madrid	Sí	...	No	Género	ASILO 1995
4	Española	Madrid	No	...	No	Opiniones Políticas	DEPENDENCIA
5	Comunitaria	España	Sí	...	Sí	No	LIBRE CIRCUILACIÓN UE
6	Apátrida	Madrid	Sí	...	No	Opiniones Políticas	ASILO 1995
7	Española	Madrid	Sí	...	No	Orientación sexual	None
8	No comunitario	España	No	...	No	No	INTEGRACIÓN EXTRANJEROS 2011
9	Comunitaria	Madrid	Sí	...	No	Opiniones Políticas	LIBRE CIRCUILACIÓN UE

[10 rows x 31 columns]

Figura 37. Ejemplo de datos generados

7.2. Algoritmos de árboles de decisión

Una vez resuelto el paso de generar universos de datos para entrenamiento y clasificación, el siguiente paso es implementar los algoritmos de creación de árboles de decisión. Para ello se ha diseño la estructura de clases que se muestra en la Figura 38.

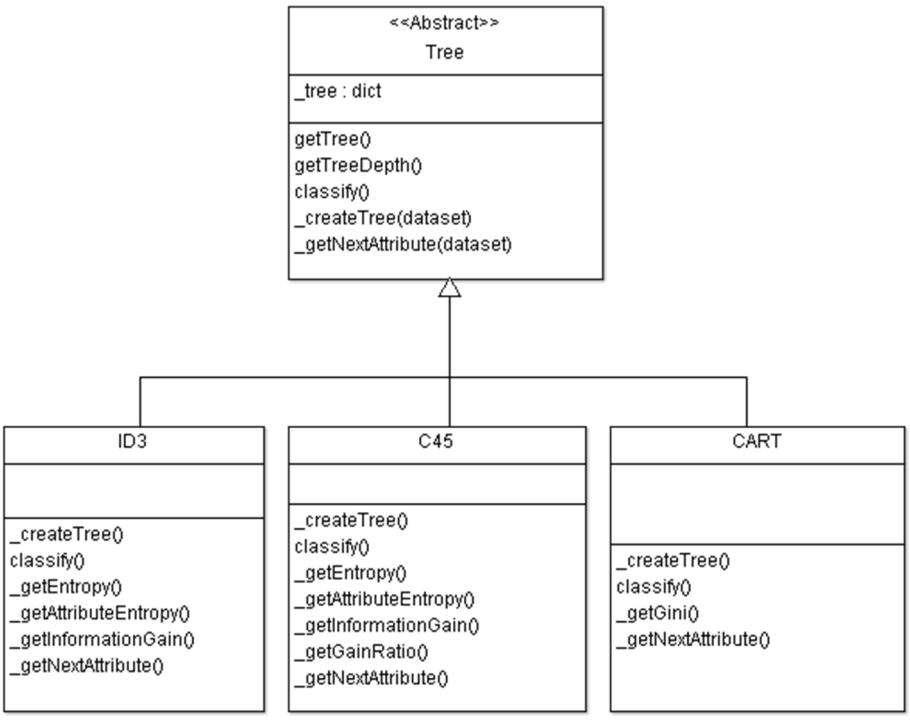


Figura 38. Diagrama de clases de árboles de decisión

La clase base `Tree` contiene las funciones comunes a todos los árboles de decisión, tales como obtener el árbol generado (`getTree`) y obtener la profundidad del árbol (`getTreeDepth`). El resto de métodos son interfaces que las clases hijas deben implementar ya que cada tipo de algoritmo genera el árbol con criterios diferentes y, en consonancia, la simulación o clasificación de un elemento se hace de forma diferente.

7.2.1. Consideraciones previas

Antes de entrar a evaluar cada uno de los algoritmos desarrollados es necesario establecer y conocer los criterios bajo los que se probará y medirá la efectividad de cada uno de ellos, de manera que se pueda interpretar correctamente cada uno de los resultados.

1. Los tres algoritmos se han entrenado con el mismo conjunto de elementos en cada iteración de manera que las ratios de efectividad pueden ser comparables entre sí.
2. Una vez entrenados, los algoritmos se han puesto a prueba con el mismo conjunto de datos formado por 5000 elementos y diferentes a los del conjunto de entrenamiento.
3. Las simulaciones realizadas van desde 50 elementos de entrenamiento hasta 50000, creciendo en 50 hasta los 1000, en 1000 hasta los 20000 y en 5000 hasta los 50000, teniendo un total de 46 observaciones para cada algoritmo.
4. Las métricas recogidas son:
    a. Tiempo de creación del árbol de decisión: servirá para indicar el coste computacional de cada técnica y poder derivar el coste/beneficio del algoritmo y el set de datos proporcionado.
    b. Profundidad del árbol de decisión generado: mide la complejidad del árbol de decisión y el camino máximo a seguir para clasificar un elemento. En los árboles ID3 y C4.5 este dato puede ser comparable ya que en cada nivel del árbol se considera un atributo como elemento de división por lo que la profundidad podría medir el número de atributos máximos que se tendrán en cuenta para clasificar un elemento. Sin embargo, en el caso de CART, al ser un árbol binario, no se divide por atributo sino por un valor concreto de un atributo por lo que es normal que la profundidad de los árboles resultantes sea mayor.
    c. Tiempo de clasificación de los 5000 elementos: permitirá medir la velocidad del proceso de clasificación una vez generado el árbol de decisión y, por ende, su coste computacional.

d. Número de aciertos en la clasificación: número de elementos correctamente clasificados en términos absolutos. Esta métrica será útil para validación cruzada más adelante.
e. Porcentaje de exactitud: ratio de elementos clasificados correctamente. Es la métrica a maximizar.

Todo esto se ha realizado con un script desarrollado en Python y disponible en el repositorio, denominado **`Benchmark.py`**, y que proporciona un conjunto de opciones para ejecutar y recopilar esta información de forma automática.

7.2.2. ID3

Según lo analizado en el apartado 2.4.1.2, el método de construcción de árboles de decisión ID3 se basa en el concepto de Ganancia de Información (Ross Quinlan, Induction of decision trees, 1986). El proceso de construcción del árbol de decisión es un proceso recursivo en el que en cada iteración busca el atributo que tenga una mayor ganancia de información o, lo que es lo mismo, reduzca más la entropía del conjunto restante (la cantidad de información necesaria para clasificar un elemento).

En el siguiente código se puede apreciar las funciones que seleccionan el atributo por el que ramificar el árbol y el proceso recursivo de creación del mismo.

```
def _getNextAttribute(self, dataset, attributeClass = 'CLASS'):
    """
    Returns the next attribute to split the data according to the greatest
Information Gain
    :param dataset: set of data to be analyzed
    :param attributeClass: name of the attribute which informs the classification
of elements.
    :return: the attribute with better information gain.
    """
    entropies = {}
    for attribute in dataset.keys():
        if attribute == attributeClass:
            continue
        entropies[attribute] = self._getInformationGain(dataset, attribute)
    return (sorted(entropies.items(), key = operator.itemgetter(1), reverse =
False)).pop()

def _createTree(self, dataset, attributeClass = 'CLASS', parent = None):
    """
    Creates the decission tree based on ID3 criteria
    :param dataset: set of data pending to analyze
    :param attributeClass: name of the attribute which informs the classification
of elements.
    :param parent: if the decision tree is part of another dictionary
    :return: the tree for the dataset provided
```

```
    """
    if not parent:
        parent = {}
    attribute, infoGain = self._getNextAttribute(dataset)
    if attribute not in parent.keys():
        parent[attribute] = {}
    attributeValues = np.unique(dataset[attribute])
    for attributeValue in attributeValues:
        subset = dataset[dataset[attribute] == attributeValue]
        classes, counters = np.unique(subset[attributeClass], return_counts =
True)
        if len(classes) <= 1:
            parent[attribute][attributeValue] = classes[0]
        else:
            parent[attribute][attributeValue] = self._createTree(subset)
    return parent
```

Los resultados de evaluación de este algoritmo se aprecian en la Figura 39. En el extremo de las simulaciones, la eficacia del algoritmo alcanza un 98,12% tras haber generado un árbol de decisión con una profundidad de 21 niveles e invertir un tiempo de cómputo total de casi 174 segundos. Es relevante destacar que con un conjunto pequeño de datos de entrenamiento (50) ya proporciona un porcentaje de acierto elevado (81,24%) por lo que podría ser una opción interesante para hacer predicciones o clasificaciones que se permitan ese porcentaje de acierto.

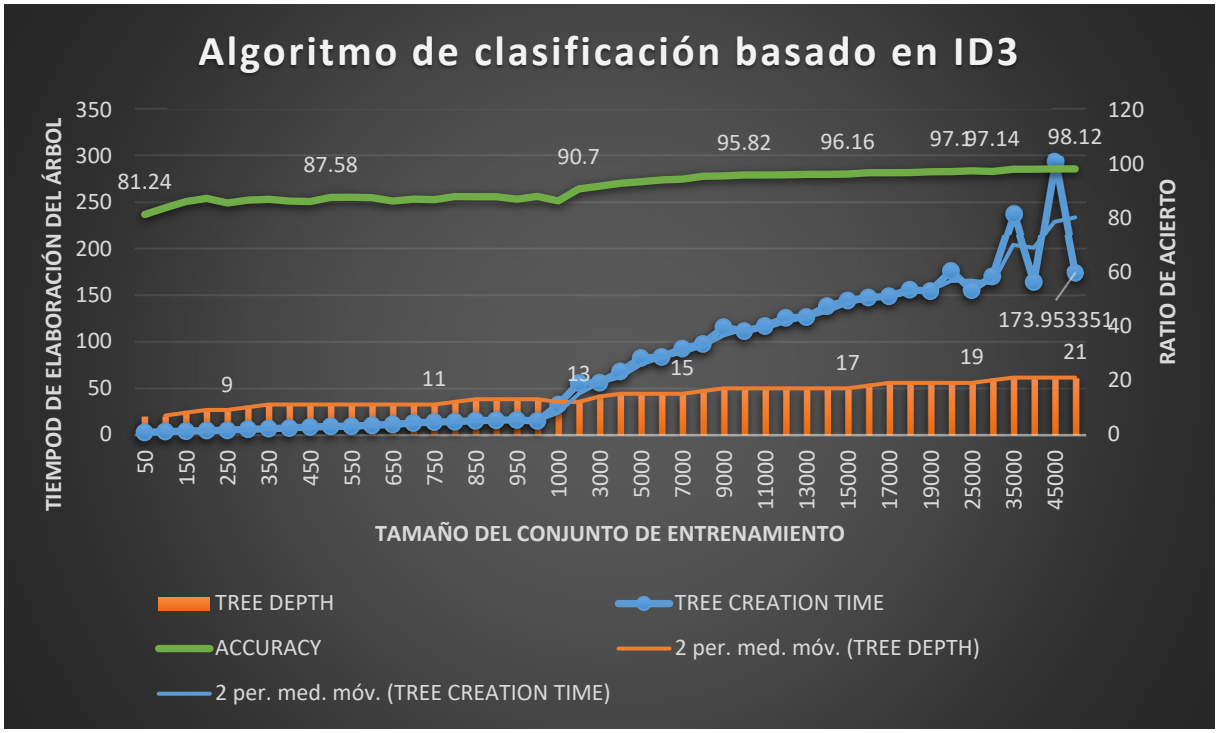


Figura 39. Resultados de simulación del algoritmo de clasificación basado en ID3

Los tiempos de clasificación son despreciables ya que en apenas 0,03 segundos se clasifican todos los elementos del conjunto.

La oscilación final en los tiempos de generación de las últimas simulaciones se debe a concurrencia de otros procesos por lo que se añade la línea de tendencia que normaliza la serie para apreciar la pendiente de la serie.

7.2.3. C4.5

El método de construcción de árboles de decisión C4.5 es una evolución de los ID3 en el que se intenta mejorar la selección de los nodos sobre los que pivotar el árbol incorporando el concepto de Ratio de Ganancia (Ross Quinlan, C4.5: Programs for Machine Learning, 1992), que se basa en la relación entre la entropía del atributo y la de la partición resultante si se eligiese dicho atributo.

El código asociado a este algoritmo es prácticamente el mismo salvo que el procedimiento para elegir el siguiente atributo para incorporar en el árbol de decisión se basa en el cálculo del Ratio de Ganancia, descrito en el siguiente método:

```
def _getGainRatio (self, dataset, attributeName):
    """
    Compute the Gain Ratio of an attribute in one dataset
    :param dataset: set of data to be analyzed
    :param attributeName: name of the attribute to compute Gain Ratio.
    :return: the Gain Ratio associated to the provided attribute
    """
    splitEntropy = self._getEntropy(dataset, attributeName = attributeName)
    infoGain = self._getInformationGain(dataset, attributeName = attributeName)
    gainRatio = infoGain / splitEntropy if splitEntropy != 0 else 0
    return gainRatio if not math.isnan(gainRatio) else 0
```

En la Figura 40 se encuentran los resultados de las simulaciones para la implementación del algoritmo basado en C4.5. La eficacia del algoritmo es ligeramente superior al ID3 llegando al 98,9% y con unos tiempos también mejores ya que en el último extremo ha invertido casi 150 segundos. Sin embargo, la profundidad del árbol es mayor que la del ID3, llegando este hasta los 31 niveles. A diferencia del algoritmo ID3, el C4.5 requiera un conjunto de entrenamiento mayor para proporcionar un nivel de éxito por encima del 85% y, sin embargo, alcanza ratios de éxito por encima del 90% con más rapidez que el ID3 ya que con 350 elementos de entrenamiento ya llega a esos porcentajes.

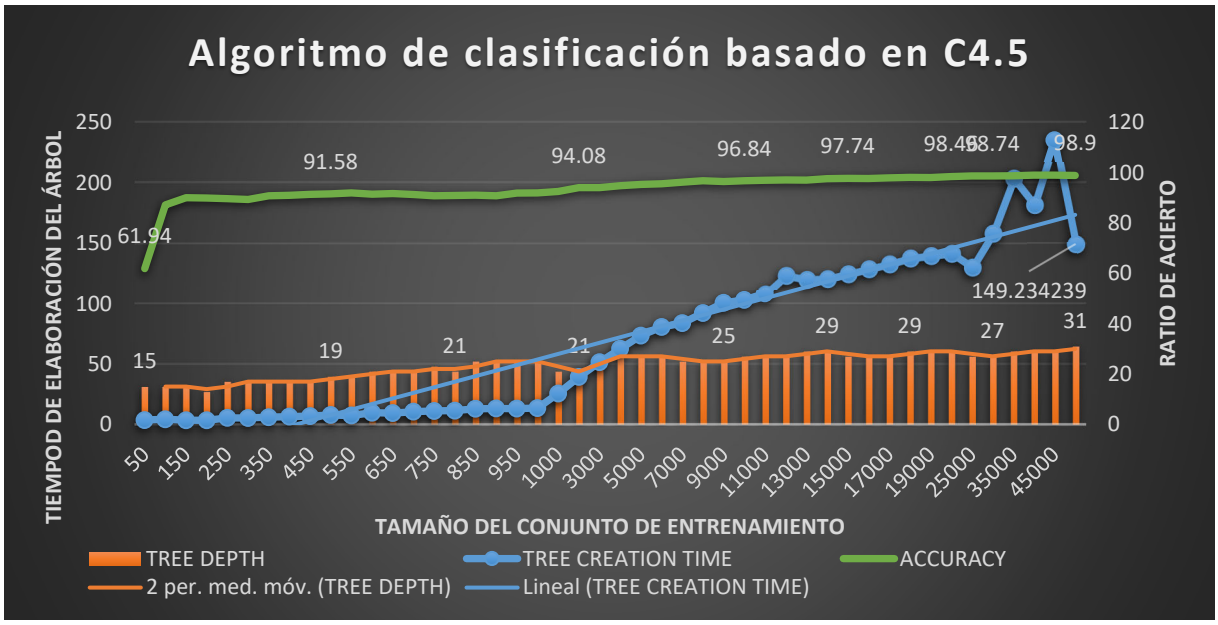


Figura 40. Resultados de simulación del algoritmo de clasificación basado en C4.5

Los tiempos de clasificación vuelven a ser despreciables y oscilan en los mismos términos que el algoritmo ID3 (0.03 segundos para 5000 elementos).

7.2.4. CART

El algoritmo CART es radicalmente diferente a los anteriormente analizados ya que este genera árboles estrictamente binarios por lo que para atributos discretos o categóricos de más de dos posibles valores acaba generando ramas en el que se evalúa si el atributo tiene un valor concreto o no, englobando en el grupo negativo las demás opciones que tendrán que ser evaluadas posteriormente. Para realizar esa selección de atributo y valor óptimos en cada momento utiliza el conocido como índice Gini, el cual se centra en la impureza de los datos y el atributo y valor que más reducen dicha impureza (Breiman, Friedman, Olshen, & Stone, 1984).

Por la peculiaridad de este mecanismo de selección, para cada atributo es necesario realizar todas las combinaciones posibles de elección de un valor de dicho atributo y calcular su índice Gini. Y para elegir un atributo es necesario repetir ese proceso para todos los atributos disponibles y seleccionar el que mayor impureza reduce. Por esto el coste computacional de este algoritmo es mucho más elevado que los anteriores.

A continuación, se puede ver el código desarrollado para seleccionar los atributos y valores que irán formando los nodos del árbol:

```
def _getNextAttribute(self, dataset, attributeClass = 'CLASS'):
    """
    Selects the new attribute and value to split the tree
    :param dataset: set of data to be analyzed
    :param attributeClass: name of the attribute which informs the classification
of elements.
    :return: name of the attribute and its value to split
    """
    ginis = {}
    best_ginis = {}
    for attribute in dataset.keys():
        if attribute == attributeClass:
            continue
        values, valueCounters = np.unique(dataset[attribute], return_counts =
True)
        if len (values) <= 1:
            continue
        if attribute not in ginis:
            ginis[attribute] = {}
        for value in values:
            subset_1 = dataset.where(dataset[attribute] == value).dropna()
            subset_2 = dataset.where(dataset[attribute] != value).dropna()
            gini_1 = self._getGini(subset_1)
            gini_2 = self._getGini(subset_2)
            total_elements = subset_1.count ()[0] + subset_2.count ()[0]
            gini = (subset_1.count()[0] / total_elements) * gini_1 +
(subset_2.count ()[0] / total_elements) * gini_2
            ginis[attribute][value] = gini
    for attribute in ginis:
        minorValue, minorGini = sorted (ginis[attribute].items(), key =
operator.itemgetter(1), reverse = True).pop()
        best_ginis[attribute] = {minorValue : minorGini}
    attributeToSplit = None
    valueToSplit = None
    minorGini = math.inf
    for attribute, value in best_ginis.items():
        attributeValue, gini = list(value.items())[0]
        if gini < minorGini:
            attributeToSplit = attribute
            valueToSplit = attributeValue
            minorGini = gini
    return attributeToSplit, valueToSplit

def _createTree(self, dataset, attributeClass = 'CLASS', parent = None):
    """
    Creates the decision tree associated to a provided dataset
    :param dataset: set of training data
    :param attributeClass: name of the attribute which informs the classification
of elements.
    :param parent: if the decision tree is part of another dictionary
    :return: the tree for the dataset provided
    """
    if not parent:
        parent = {}
    if dataset.count ()[0] == 1:
        classes = np.unique (dataset[attributeClass])
        return classes[0]
    attributeToSplit, valueToSplit = self._getNextAttribute(dataset)
    subset_Yes = dataset.where(dataset[attributeToSplit] == valueToSplit).dropna()
```

```
    subset_No = dataset.where(dataset[attributeToSplit] != valueToSplit).dropna()
    classes_Yes = np.unique(subset_Yes[attributeClass])
    classes_No = np.unique(subset_No[attributeClass])
    if (len (classes_Yes) == 1 and len (classes_No) == 1 and classes_No ==
classes_Yes):
        return classes_Yes[0]
    parent[(attributeToSplit, valueToSplit)] = {'YES' :
self._createTree(subset_Yes),
                                                'NO' :
self._createTree(subset_No)}
    return parent
```

Los resultados asociados a las simulaciones realizadas con el algoritmo basado en CART se encuentra en la Figura 41. En ella se aprecia que la efectividad del algoritmo CART es la mayor de los tres vistos llegando al 99,6% de acierto en el extremo superior de la simulación. Al igual que el C4.5, la ratio de éxito es muy baja con conjuntos de datos de entrenamiento pequeños, pero es el que más pronto alcanza tasas del 90% ya que solo necesita 150 elementos para llegar a esos porcentajes. A cambio, el coste computacional es mucho mayor que los otros dos algoritmos, con diferencia ya que el procesamiento se dispara a partir de conjuntos de datos superiores a 1000 elementos, llegando a tardar más de 12 minutos en elaborar el árbol de decisión asociado al último punto de la serie con 50000 elementos. Por último, en términos de profundidad del árbol, es el que más niveles tiene, aunque no muy lejos del C4.5, a pesar de ser un árbol binario.

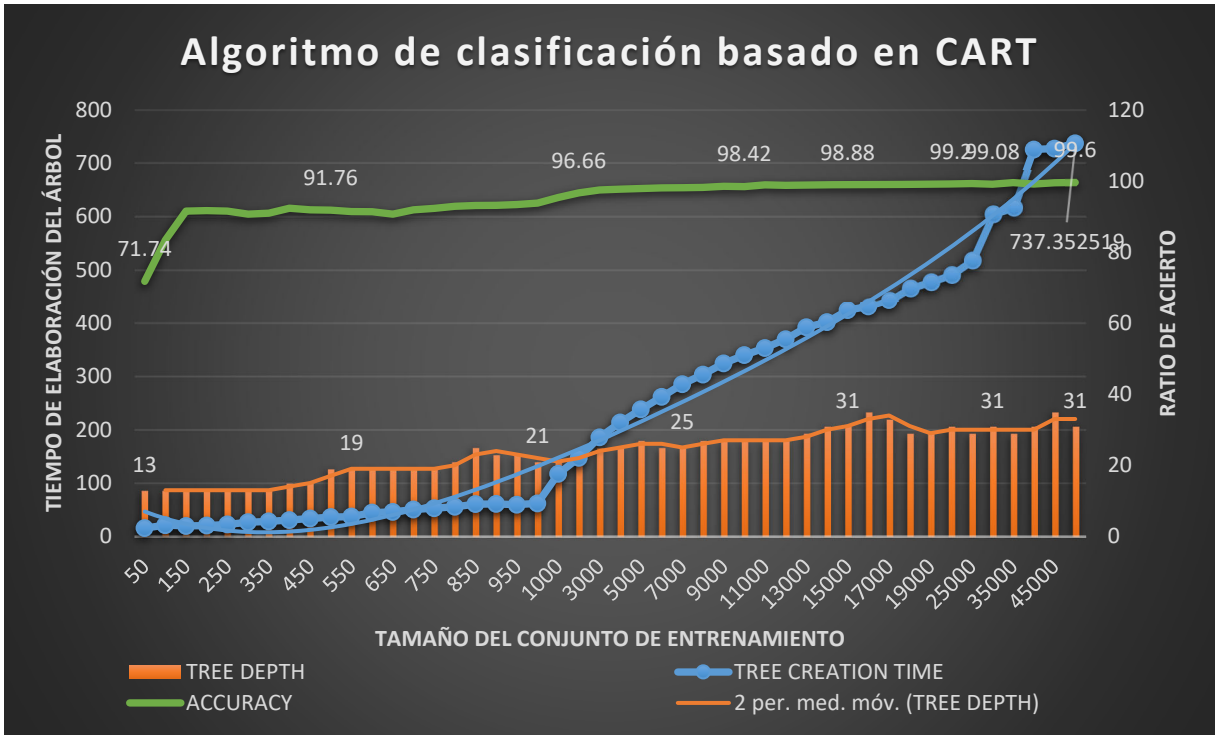


Figura 41. Resultados de simulación del algoritmo de clasificación basado en CART

Los tiempos de clasificación también son despreciables y oscilan en los mismos términos que los anteriores algoritmos (0.04 segundos para 5000 elementos).

7.3. Comparativa y conclusiones

En la Figura 42 se puede apreciar comparativa del porcentaje de acierto de los 3 algoritmos juntos, así como el coste computacional para elaborar los árboles de decisión. En esa comparativa se puede apreciar rápidamente que:

- A partir de conjuntos de entrenamiento superiores a 1000 elementos el algoritmo CART dispara su coste computacional, aunque el beneficio obtenido es muy pequeño.
- Los algoritmos ID3 y C4.5 necesitan hasta 15 veces más elementos para llegar a proporcionar ratios de éxito similares a los de CART ya que hasta los conjuntos superiores a 15000 elementos no alcanzan el 96%, aunque a un coste computacional mucho menor.
- Apenas hay diferencia entre los algoritmos ID3 y C4.5 ya que sus resultados son prácticamente idénticos a partir de conjuntos de datos superiores a 200 elementos.

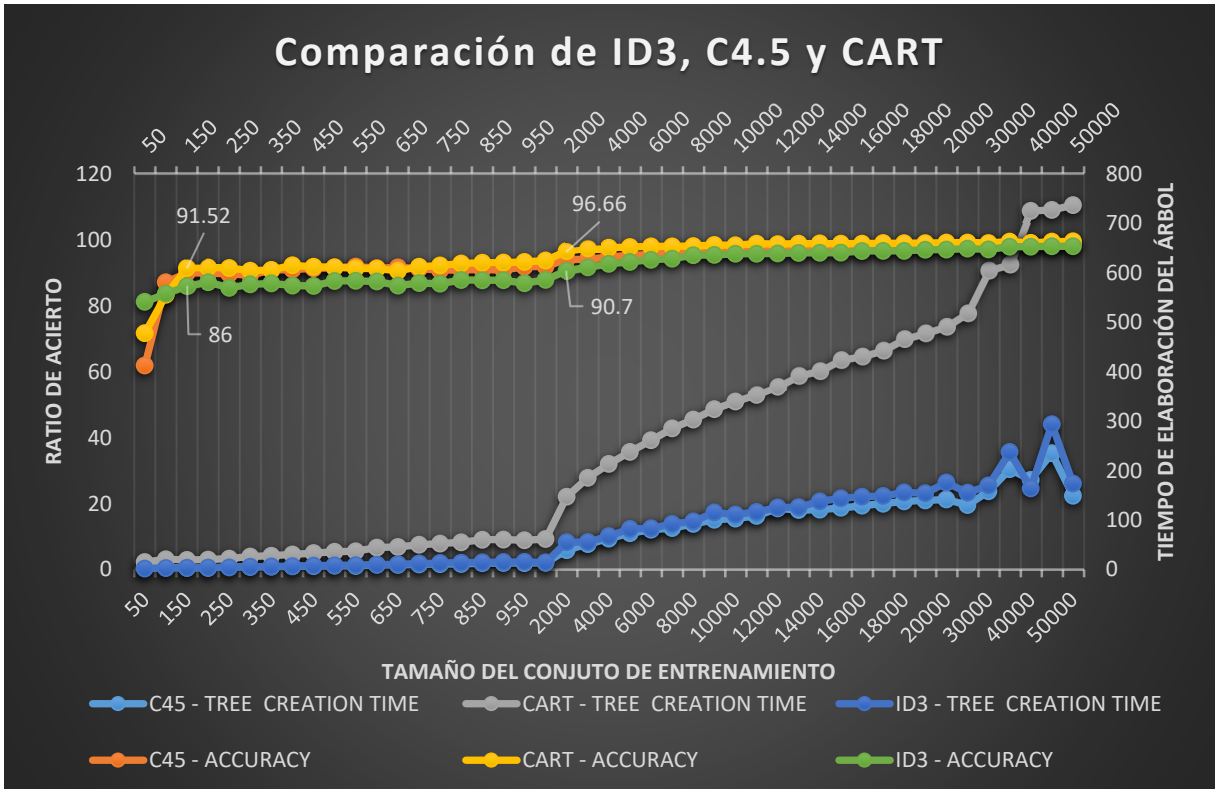


Figura 42. Comparativa de los tres algoritmos de clasificación desarrollados

7.3.1. Validación cruzada

Como último punto de análisis, antes de pasar a las conclusiones y una vez identificado el valor de entrenamiento razonable y óptimo (1000 elementos), se procede a una validación cruzada que permita independizar los porcentajes de éxito obtenidos del ejercicio realizado.

La validación cruzada es un método por el cual, a partir de un conjunto total de elementos, se divide en bloques del mismo tamaño de manera que se realizan tantas simulaciones como bloques haya para que, en cada simulación, uno de esos bloques sirva de conjunto de prueba y el resto como conjunto de entrenamiento del algoritmo. La efectividad resultante es la suma de todos los aciertos conseguidos en todas las iteraciones dividida entre la totalidad del tamaño del conjunto total de elementos. (Tan, Steinbach, Karpatne, & Kumar, 2006)

A través de esta validación se obtienen resultados que son independientes de la partición que se realice entre elementos de entrenamiento y de prueba o clasificación, de manera que son extrapolables a otros conjuntos.

En este caso, el mismo script **`Benchmark.py`** usado para las simulaciones inicial proporciona una opción que permite realizar esta validación cruzada indicando el número de particiones en las que se quiere dividir el conjunto de elementos. En la validación realizada se ha indicado un conjunto total de 1200 elementos a dividir en seis bloques de manera que se ejecutan seis simulaciones en las que en cada una de ellas hay 1000 elementos de entrenamiento y 200 de prueba.

En la Tabla 25 están los resultados obtenidos (número de elementos clasificados correctamente), así como la efectividad media resultante de la validación cruzada.

Tabla 25. Resultados validación cruzada

Método	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Iter. 6	Media
ID3	172	167	161	175	169	165	**84.083%**
C4.5	188	181	179	185	182	176	**90.917%**
CART	198	191	185	189	194	184	**95.083%**

7.3.2. Curvas ROC

Una forma de explicar los resultados obtenidos en la validación cruzada es a través de las denominadas curvas ROC, del acrónimo *Receiver Operating Characteristics* (Característica Operativa del Receptor), cuya principal utilidad es la de enfrentar dos ratios relevantes en los problemas de clasificación de manera que facilitan la selección del modelo o algoritmo óptimo: el de verdaderos positivos y el de falsos positivos (Concejero Cerezo, 2004).

La ratio de verdaderos positivos se define como el número de elementos clasificados correctamente por el algoritmo de clasificación entre el número total de positivos reales. De manera análoga, la ratio de falsos positivos es el número de elementos clasificados incorrectamente por el algoritmo de clasificación entre el número total de negativos reales.

Las curvas ROC requieren que el problema de clasificación sea binario, es decir, que el resultado de la clasificación sea positivo o negativo. Sin embargo, los árboles de decisión aplicados en este problema son multiclase por lo que se requiere una simplificación de manera que se considera un resultado positivo cuando un individuo tiene acceso a una prestación (sin importar cual sea) y negativo cuando no tiene acceso a ninguna.

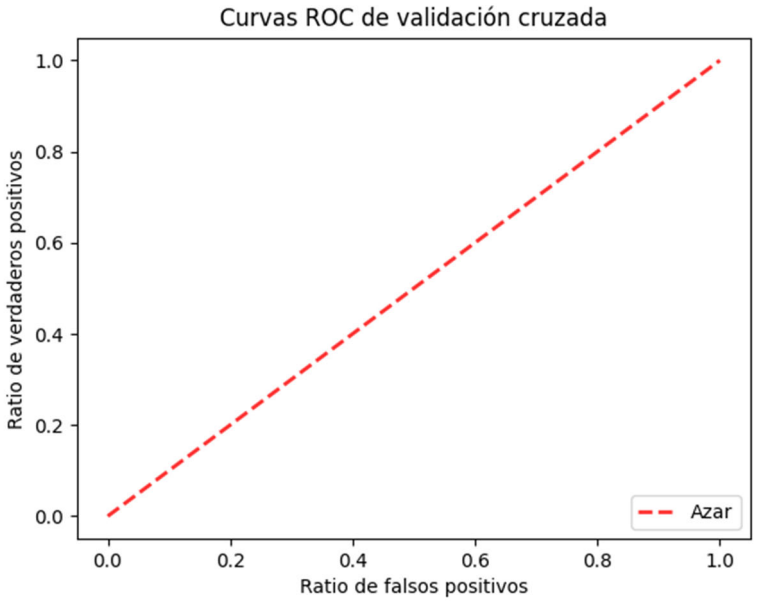


Figura 43. Espacio de curvas ROC

Tal y como se observa en la Figura 43, los ejes que componen el espacio donde se representan las curvas ROC están asociados a las dos ratios comentadas anteriormente, siendo el eje *x* el de la ratio de verdaderos positivos y el eje *y* el de la ratio de falsos positivos. En dicho espacio

la diagonal representa la clasificación aleatoria, es decir, la tasa de acierto en caso de utilizar el azar para clasificar un individuo, y dicha diagonal divide en dos el espacio de la curva de manera que en la parte superior estarán aquellos algoritmos que sean mejor que el azar y por debajo los que son peores. De esta manera se puede entender que el punto superior izquierdo del espacio correspondería a una clasificación perfecta, donde el 100% de los individuos se clasifican correctamente.

Un último aspecto a tener en cuenta en las curvas ROC es lo que se denomina espacio o área bajo la curva (AUC o *area under the curve*) que representa la probabilidad de que el algoritmo de clasificación asociado a dicha curva clasifique correctamente un individuo positivo cualquiera.

En la Figura 44 se pueden observar las curvas ROC asociadas a los tres algoritmos de árboles de decisión analizados y bajo el ejercicio de validación cruzada detallada en el punto anterior.

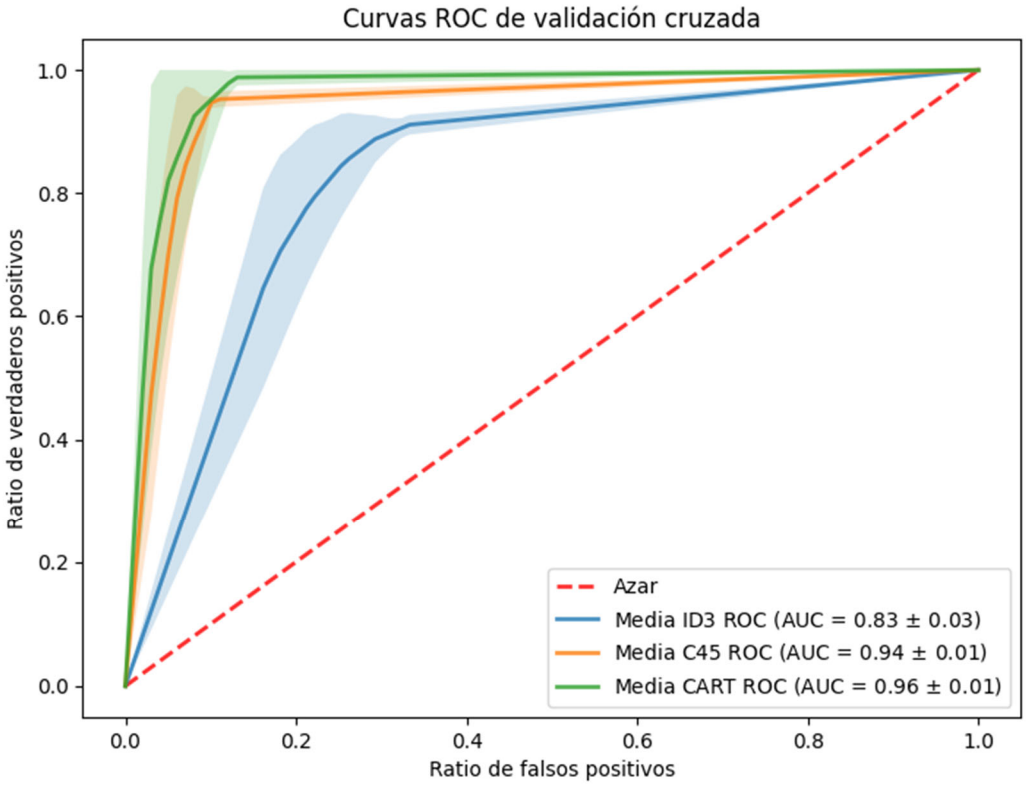


Figura 44. Curvas ROC asociadas a la validación cruzada de los algoritmos

El detalle de la información que se puede observar en la Figura 44 es la siguiente:

- El espacio o área sombreada alrededor de la curva representa la oscilación de resultados de ese algoritmo a lo largo de las diferentes iteraciones de la validación cruzada.
- Cada curva representa la media de las diferentes iteraciones de la validación cruzada.
- En la leyenda se indica el AUC de cada algoritmo.

Estas curvas ROC consolidan lo observado anteriormente de manera que CART es el algoritmo con mejor probabilidad de acierto elevándolo a un 96%, seguido de C4.5 que tiene una probabilidad del 94%. Lejos queda el algoritmo ID3 cuya ratio de falsos positivos es más elevada.

Un apunte interesante que se puede extraer de estas curvas es la sensibilidad del algoritmo CART al conjunto de entrenamiento ya que sus valores oscilan bastante en cada iteración de la validación cruzada, tal y como explica el área sombreada de verde, siendo bastante más estable el algoritmo C4.5 con una oscilación menor. Esto se puede traducir en que la efectividad del algoritmo CART es dependiente del conjunto de entrenamiento en mayor medida que lo que lo es el C4.5.

7.3.3. Conclusiones

Por todo lo anteriormente expuesto, se podrían resumir las siguientes conclusiones acerca de la aplicabilidad de los árboles de decisión al problema de identificación de prestaciones:

- Entendiendo que la identificación de prestaciones es una ayuda o herramienta para que los técnicos de EAPN Madrid puedan dar mejor ayuda a soporte a las personas que acuden a sus centros, se puede considerar entonces que una ratio del 100% de exactitud no es imprescindible porque el proceso no deja de ser una referencia a tomar en cuenta y que puede ser errónea.
- En ese caso, para proporcionar un nivel de confort suficiente, dado el comportamiento de los tres modelos estudiados y la complejidad que supone elaborar manualmente elementos para proporcionarlos al algoritmo, lo ideal sería apostar por una solución basada en CART y requerir un conjunto de datos de entrenamiento que rondara los 1000 elementos para asegurar porcentajes de éxito cercanos al 96%.
- Sin embargo, este modelo tiene una limitación muy importante ya que los árboles de decisión solo soportan que un elemento pertenezca únicamente a una clase y el

problema inicial de identificador de prestaciones es un problema que puede tener múltiples soluciones para un mismo individuo por lo que no sería aplicable a la totalidad del alcance, sino a la simplificación planteada en este capítulo en la que se limita a una las prestaciones a las que un determinado individuo puede acceder.

- Para poder aplicar árboles de decisión con aprendizaje automático a este problema, una solución futura podría ser la de generar tantos árboles de decisión como prestaciones haya, de manera que se use un mismo conjunto de entrenamiento para construir el árbol de cada prestación y donde la clase objetivo sea si satisface o no las condiciones de dicha prestación. De esta manera, para los datos de un determinado individuo, sería necesario evaluar cada árbol para obtener el conjunto final de prestaciones a las que podría tener acceso.

8. Conclusiones y trabajo futuro

8.1. Conclusiones

El objetivo principal que se perseguía con este trabajo era el de desarrollar una aplicación web que permitiera a cualquier persona consultar si cumple o no las condiciones necesarias para poder acceder a alguna de las diferentes prestaciones disponibles por las Administraciones del Estado y de la Comunidad de Madrid. Para llegar a ese objetivo, el usuario tan solo debería proporcionar información a través de un cuestionario dinámico que se iría adaptando en base a la información proporcionada previamente. El cumplimiento de este objetivo ha proporcionado una herramienta web sencilla, minimalista y muy orientada a los potenciales usuarios de la misma que son personal técnico de la red de asociaciones de EAPN de Madrid que no tienen conocimientos profundos ni de tecnología ni de aspectos legales, por lo que la aplicación desarrollada se puede considerar como solución de Legal Tech.

En el transcurso de este diseño y desarrollo igualmente se han satisfecho los objetivos específicos marcados al inicio, ya que, tras analizar y estudiar las diferentes técnicas y algoritmos de clasificación, se ha diseñado una solución en la que la lógica que discurre detrás de la herramienta aplica principios similares a los de los árboles de decisión y riqueza de información.

De igual forma se han aplicado criterios metodológicos al proceso de análisis, diseño, desarrollo y validación acordes a un estándar como el de Proceso de Desarrollo Unificado, elaborando y ejecutando las diferentes fases del ciclo de vida de un proyecto informático. Y todo apalancado en una elección justificada de tecnologías a aplicar, acordes a los requerimientos de la necesidad descrita, asegurando su flexibilidad y sencillez.

También se han estudiado e investigado alternativas basadas en árboles de decisión de aprendizaje automático, comparando los distintos métodos y analizando la aplicabilidad de estas técnicas al problema inicial planteado.

Por tanto, la conclusión final se puede resumir en una consecución de todos los objetivos definidos y marcados al principio proporcionando una solución de código libre que puede tener otras aplicaciones relativas a problemas de clasificación similares al planteado en este trabajo.

8.2. Trabajo futuro

Aunque la aplicación en su conjunto puede considerarse completa para la necesidad inicial descrita, se pueden considerar posibles extensiones o mejoras futuras tales como:

- Permitir agrupar las prestaciones por tipología para que se puede realizar una simulación por dicha tipología en lugar de hacerla sobre todas. Esto podría ser especialmente relevante cuando se amplíe el alcance de prestaciones a otros ámbitos que no sean los de pobreza o exclusión social como, por ejemplo, prestaciones laborales, de familia (maternidad/paternidad, familia numerosa, etc.), subvenciones para emprendedores, etc.
- Permitir un criterio de prelación o dependencia entre los diferentes atributos definidos para que la secuencia de obtención de información pueda tener una lógica más próxima al mapa mental del usuario (por ejemplo, solicitar antes si se está cursando estudios que cuál es el centro de estudios).
- Ampliar el ámbito de las prestaciones a todo tipo de ayuda de cualquier estamento, organismo y administración pública. Incluso podría ser un servicio que proporcionaran las propias administraciones para que fuera de dominio público, tal y como ocurre en Francia[39].
- Incorporar criterios de incompatibilidad en las prestaciones para que, en caso de ser beneficiario de una, se descarten todas las que son incompatibles con ella.
- En lo referente a la aplicabilidad de árboles de decisión y aprendizaje automático existen varias líneas de investigación futura: estudiar la incorporación de criterios de bosques de decisión o *random forest* que aumenten la efectividad de las predicciones y clasificaciones; profundizar en una solución que soporte problemas de multiclase, es decir, que un elemento pueda pertenecer a más de una clase o categoría; reaprendizaje en caso de cambio de alguna legislación o prestación sin necesidad de repetir todo el proceso de aprendizaje, etc.
- Proporcionar métricas de uso, de porcentaje de aciertos, de coberturas, etc. para poder medir la aplicabilidad de la herramienta y su utilidad efectiva.

[39] El Gobierno francés dispone de una web pública que permite simular todas las prestaciones disponibles en el estado: https://www.mesdroitssociaux.gouv.fr/accueil/

Referencias bibliográficas

Barrio Andrés, M. (2019). *Legal Tech. La transformación digital de la abogacía.* Wolters Kluver.

Berzal Galiano, F. (22 de 03 de 2021). *Data Mining.* Obtenido de http://elvex.ugr.es/idbis/dm/slides/3%20Classification.pdf

Breiman, L., Friedman, J., Olshen, R., & Stone, C. J. (1984). *Classification and Regression Trees.* Chapman and Hall.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies, 27*(4), 349-370.

Concejero Cerezo, P. (2004). *Comparación de modelos de curvas ROC para la evaluación de procedimientos estadísticos de predicción en investigación de mercados.*

EAPN Madrid. (2021). *EAPN Madrid*. Obtenido de http://www.eapnmadrid.org/

EAPN-ES. (22 de Marzo de 2021). *EAPN España*. Obtenido de https://www.eapn.es

European Commission. (22 de Marzo de 2021). *Employment, Social Affairs & Inclusion*. Obtenido de Employment, Social Affairs & Inclusion: https://ec.europa.eu/social/main.jsp?catId=818&langId=en

Ghimire, D. (5 de Mayo de 2020). Comparative study on Python web frameworks: Flask and Django. Metropolia University of Applied Sciences. Obtenido de https://www.theseus.fi/bitstream/handle/10024/339796/Ghimire_Devndra.pdf?sequence=2&isAllowed=y

Han, J., Pei, J., & Kamber, M. (2011). *Data Mining : Concepts and Techniques.* Elsevier Science & Technology.

Instituto Nacional de Estadística. (2020 de Julio de 21). *Instituto Nacional de Estadística | Encuesta de Condiciones de Vida 2019.* Obtenido de https://www.ine.es/prensa/ecv_2019.pdf

Jacobson, I., Booch, G., & Rumbaugh, J. (2000). *El proceso unificado de desarrollo software.* Addison Wesley.

Jadhav, S. D., & Channe, H. P. (2013). Comparative Study of K-NN, Naive Bayes and. *International Journal of Science and Research (IJSR)*, 1842-1845.

Kerbs, R. W. (2001). The Extraction of Classification Rules and Decision Trees from Independence Diagram. *The Extraction of Classification Rules and Decision Trees from Independence Diagrams. Doctoral dissertation.* Obtenido de NOVA SOUTHEASTERN UNIVERSITY - CCE THESES AND DISSERTATIONS: https://nsuworks.nova.edu/gscis_etd/630

Metsis, V., Androutsopoulos, I., & Paliouras, G. (Julio de 2006). Spam filtering with naive bayes-which naive bayes? *CEAS, 17*, 28-69.

Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill.

Patel, H. H., & Prajapati, P. (2018, October 31). Study and Analysis of Decision Tree Based Classification Algorithms. *International Journal of Computer Sciences and Engineering, 6*(10), 74-78. Retrieved from https://www.researchgate.net/profile/Purvi-Prajapati/publication/330138092_Study_and_Analysis_of_Decision_Tree_Based_Classification_Algorithms/links/5d2c4a91458515c11c3166b3/Study-and-Analysis-of-Decision-Tree-Based-Classification-Algorithms.pdf

Priyam, A., Abhijeet, Gupta, R., Rathee, A., & Saurabh, S. (1 de June de 2013). Comparative Analysis of Decision Tree Classification Algorithms. *International Journal of Current Engineering and Technology, 3*(2), 334-337.

Ross Quinlan, J. (Marzo de 1986). Induction of decision trees. *Machine Learning, 1*, 81-106.

Ross Quinlan, J. (1992). *C4.5: Programs for Machine Learning.* Morgan Kaufmann.

Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (s.f.). A Bayesian Approach to Filtering Junk E-Mail. *Stanford University*. Obtenido de http://robotics.stanford.edu/users/sahami/papers-dir/spam.pdf

Song, Y.-Y., & Lu, Y. (2015, February). Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry, 27*(2). Retrieved from https://doi.org/10.11919/j.issn.1002-0829.215044

Tan, P.-N., Steinbach, M., Karpatne, A., & Kumar, V. (2006). *Introduction to Data Mining (Second Edition).* Addison-Wesley.

Theodoridis, S. (2015). *Machine Learning : A Bayesian and Optimization Perspective.* Elsevier Science & Technology.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). *Data Mining : Practical Machine Learning Tools and Techniques.* Elsevier Science & Technology.

Anexo A. Encuestas realizadas

Para la recopilación de información por parte de usuarios que han probado la aplicación se ha llevado a cabo una encuesta a través de un formulario definido en *Google Forms* disponible en el siguiente enlace: https://forms.gle/M9u95vJR83U5TU2b9.

El formulario se divide en cuatro secciones:

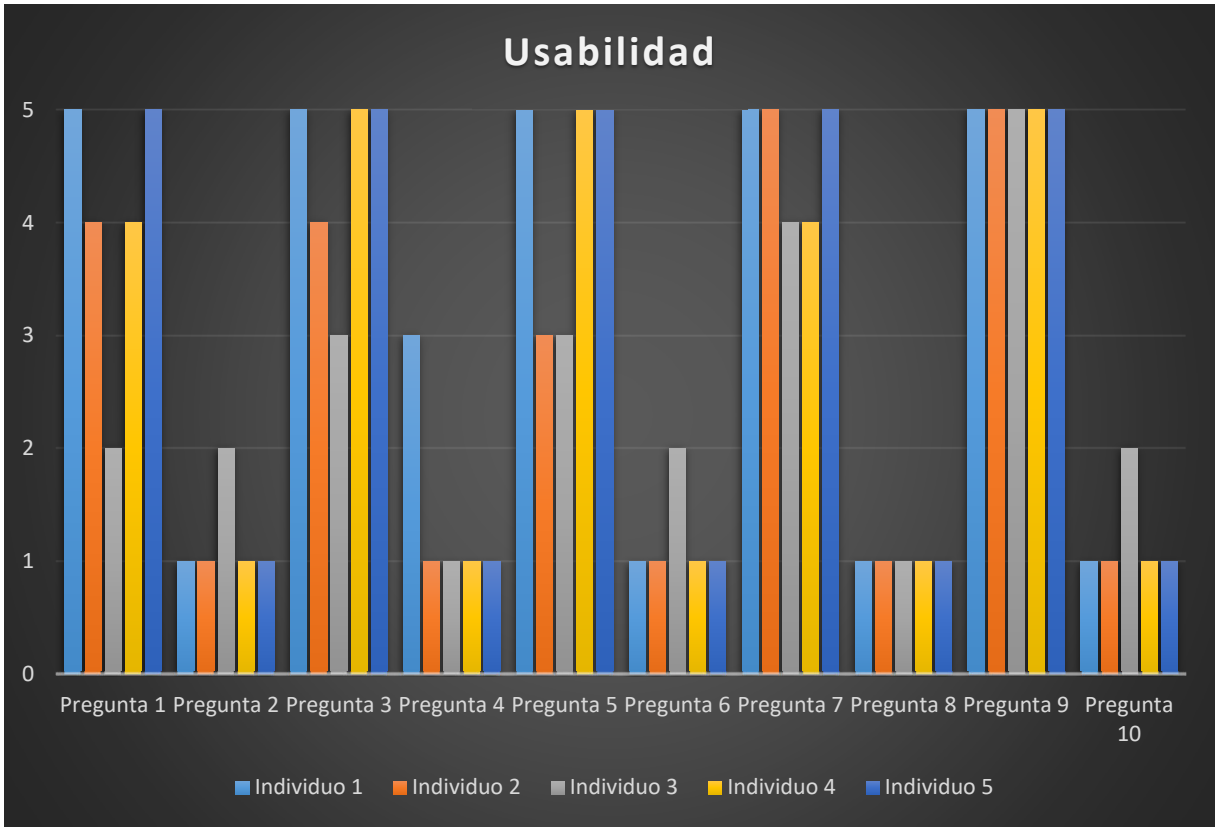
1. **Usabilidad**: utilizando el Sistema de Escalas de Usabilidad se realizan las siguientes 10 preguntas cuya valoración va desde el 1 (Totalmente en desacuerdo) al 5 (Totalmente de acuerdo):
    1. Creo que podría usar esta aplicación con frecuencia
    2. Encuentro la aplicación innecesariamente compleja
    3. Creo que la aplicación es fácil de usar
    4. Creo que necesitaría soporte de algún técnico para poder usar la aplicación
    5. Las diferentes funcionalidades de la aplicación están bien integradas
    6. Creo que hay demasiada incoherencia en la aplicación, es decir, el contenido mostrado no se corresponde con el contexto o con lo esperado.
    7. Creo que se aprende fácilmente a utilizar la aplicación
    8. La aplicación no es cómoda de usar
    9. Me he sentido seguro/a usando la aplicación
    10. He necesitado aprender muchas cosas antes de poder usar la aplicación

    Para cuantificar el resultado, este sistema define una metodología por la que a la respuesta de las preguntas impares se les resta 1 y las respuestas a las preguntas pares son restadas a 5. La suma total de todas las preguntas se multiplica por 2,5, obteniendo un valor que está entre 0 y 100[40].

    Si el valor está por debajo de 60 la usabilidad es pobre (calificación F); si está por debajo de 70 es una usabilidad aceptable (calificación D); si está por debajo de 80 es una buena usabilidad (calificación C); si está por debajo de 90 es una usabilidad excelente (calificación B); y superior a 90 es una usabilidad casi perfecta (calificación A).

[40] Se puede utilizar el siguiente simulador para comprender esta métrica: https://stuart-cunningham.github.io/sus/

En las encuestas realizadas a 5 personas, el resultado obtenido para cada pregunta es el que se observa en la siguiente figura:

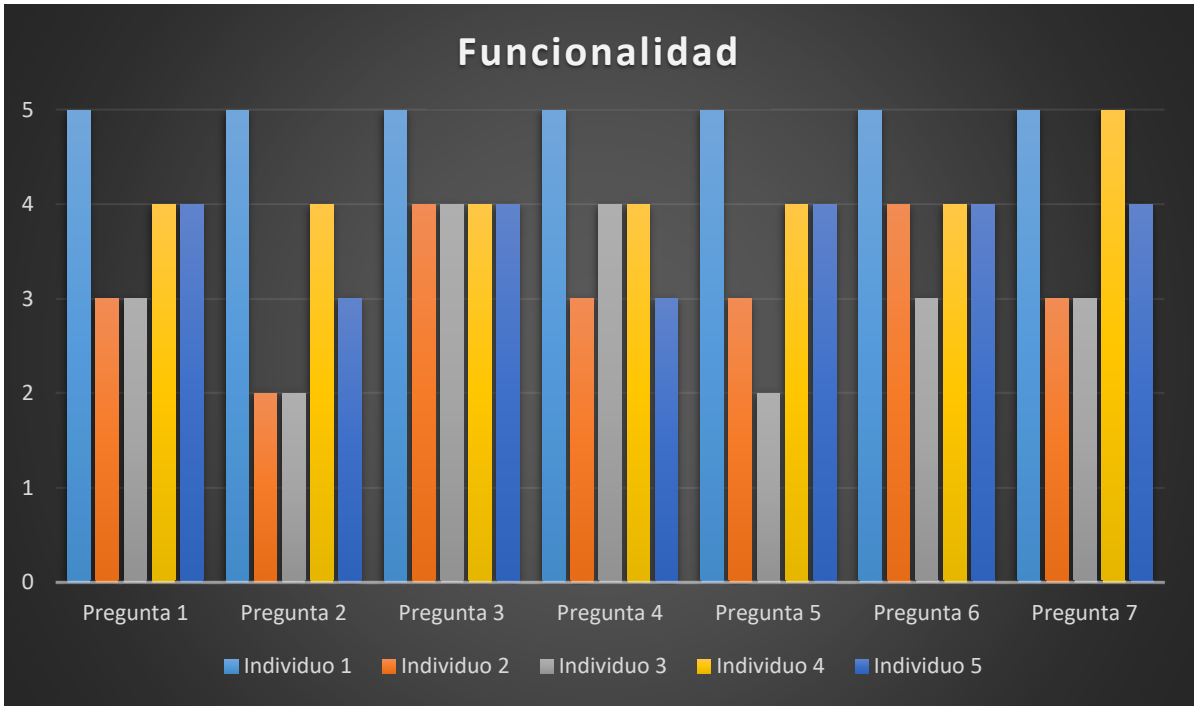


Calculando la métrica media final se obtiene un 90.5 que es una calificación de tipo A.

2. **Funcionalidad:** para medir la funcionalidad se definen 7 preguntas con opciones idénticas a las del apartado anterior, es decir, desde 1 para Totalmente en desacuerdo hasta 5 para Totalmente de acuerdo. Las preguntas son:
    1. La información que proporciona la aplicación es coherente y entendible, es decir, está relacionada con el contexto de uso y está bien descrita.
    2. Las opciones que proporciona en cada momento son comprensibles.
    3. El catálogo de prestaciones es útil y está bien estructurado.
    4. El proceso de captura de información (secuencia de preguntas) me parece correcto y coherente.
    5. Las preguntas están bien definidas y se sabe en cada momento lo que se está solicitando.
    6. Los resultados obtenidos coinciden con lo esperado.
    7. El diseño de la aplicación.

Aparte se establecen dos preguntas de texto libre para indicar lo que más gusta de la aplicación y lo que menos.

En la siguiente figura se observan los resultados para este apartado, destacando como puntos débiles la comprensibilidad de las preguntas y su definición.



Los aspectos más valorados son:

- Que es sencilla y fácil de usar. Explica con detalles breves y concisos toda la información necesaria.
- Su utilidad.
- Cubre muchas prestaciones.

Los menos valorados:

- Falta información en algunos conceptos
- Las prestaciones no tienen un orden lógico en el catálogo

3. **Uso potencial**: en este apartado se pretende analizar la utilidad que tiene la aplicación para los usuarios habituales y sus posibles usos futuros. Para ello se establecen las siguientes preguntas:
    1. ¿Crees que esta aplicación es útil en tu trabajo diario? (Sí / No / Tal vez)
    2. ¿Qué uso esperas hacer de esta aplicación? (Poco o ninguno / La usaré a menudo / La usaré a diario)

3. ¿Qué incorporarías en la aplicación para que te fuera más útil? (Texto libre)

Los resultados se pueden ver en las siguientes figuras:

Los puntos solicitados a incorporar en la aplicación son:

- Otras ayudas de otros ámbitos
- Información de cuantías

4. **Sugerencias:** el último apartado del formulario es para recoger posibles sugerencias relativas a la aplicación. Tan solo se han registrado dos, una relacionada con el cambio de la imagen principal de la aplicación y otra que solicita que el catálogo de prestaciones debería estar ordenado por tipología de las mismas.

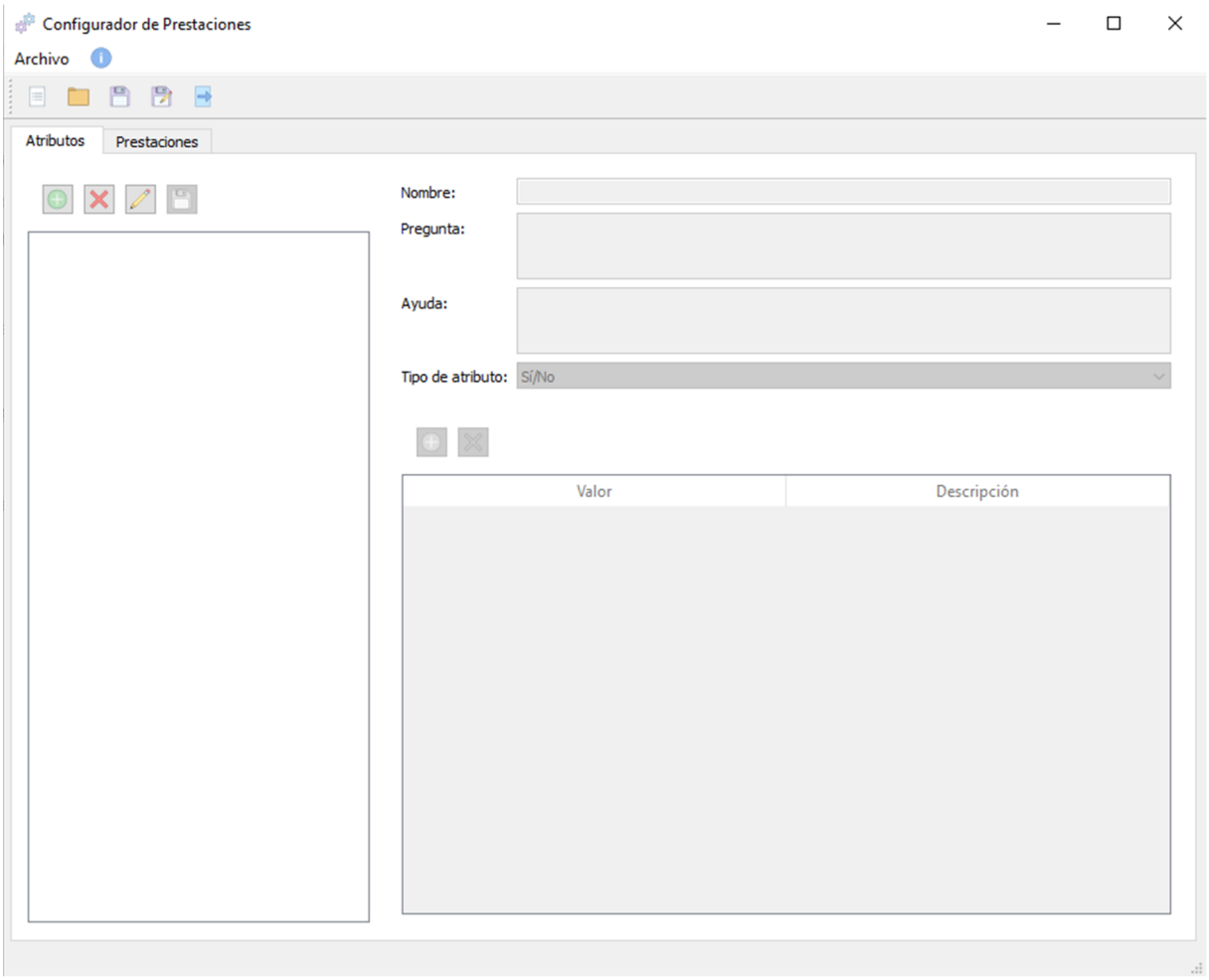
Anexo B. Manual de Usuario del Módulo de Configuración

Configurador de Prestaciones es una herramienta que permite la configuración de las prestaciones y los atributos que se usan en la aplicación web de Identificación de Prestaciones de manera que puede editar y crear el fichero de configuración que esa aplicación web utiliza para su funcionamiento.

En el presente manual se explicarán las diferentes opciones de uso y funcionalidades que ofrece.

1. INICIO DEL CONFIGURADOR

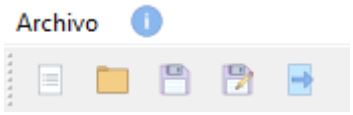
Para ejecutar *Configurador de Prestaciones* basta con lanzar el ejecutable **ConfiguradorPrestaciones.exe**, tras lo cual aparecerá la siguiente ventana:



2. DESCRIPCIÓN DE LA ZONA DE TRABAJO

En la ventana principal se encuentran los siguientes elementos:

- **Menú principal y barra de acciones**: tanto el menú *Archivo* como la barra de acciones tienen las mismas funciones, que son:

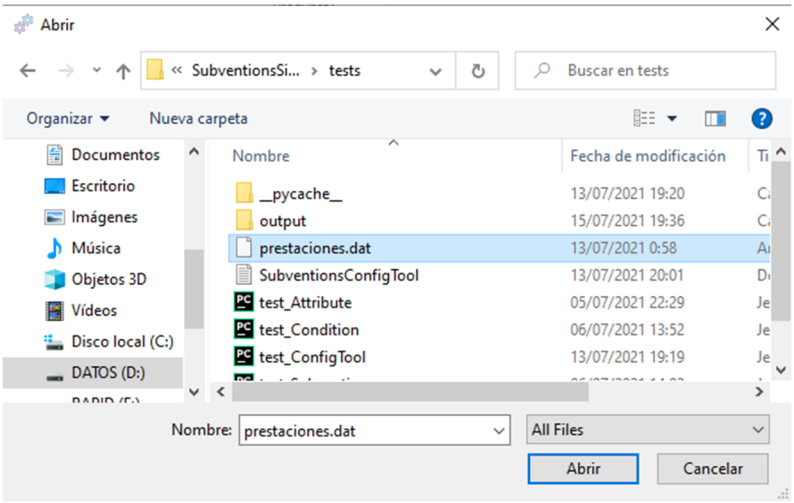


  - **Nuevo:** permite crear un nuevo fichero de configuración desde cero.
  - **Abrir:** permite abrir un fichero de configuración ya existente.
  - **Guardar:** guarda la configuración en el archivo actual o en uno nuevo.
  - **Guardar como:** guarda la configuración en un archivo nuevo.
  - **Salir:** cierra la aplicación.
- **Sección Atributos**: en esta sección se gestionan los atributos definidos en la configuración, permitiendo acciones tales como añadir uno nuevo, borrar uno existente, editar uno existente y guardar la modificación realizada sobre un atributo.
- **Sección Prestaciones**: en esta sección se gestionan las prestaciones definidas en la configuración, permitiendo acciones tales como añadir uno nuevo, borrar una existente, editar una existente y guardar la modificación realizada sobre una prestación. En esta sección se configuran las condiciones que una prestación debe satisfacer.

3. CARGA DE FICHERO EXISTENTE

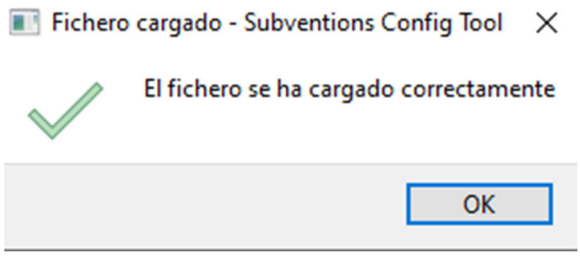
Para cargar un fichero de configuración ya existente los pasos son:

1. Pulsar el icono de abrir archivo  o elegir la opción **Abrir** del menú **Archivo**.
2. Aparecerá una ventana para seleccionar el fichero:

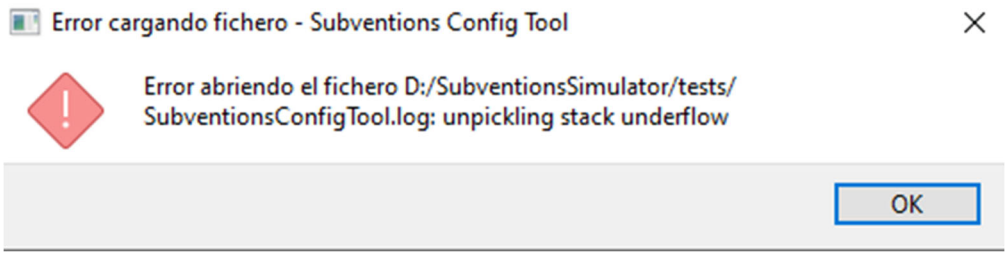


3. Tras elegir el fichero a abrir, pulsar en **Abrir**.

4. Si el fichero es correcto y se ha podido cargar aparecerá el siguiente mensaje:



5. En caso de error, aparecerá una ventana con la descripción del mismo:

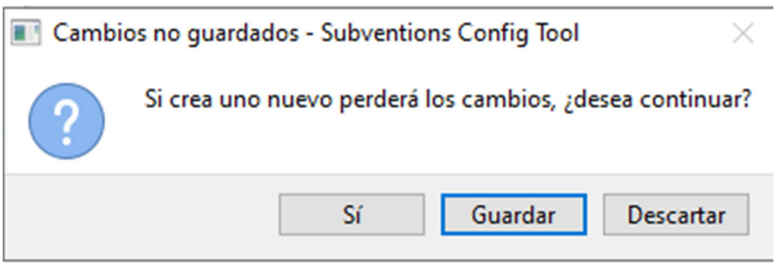


6. Tras la carga correcta, se verá la información disponible en las pestañas de atributos y prestaciones.

4. CREAR UN NUEVO FICHERO

Para crear un nuevo fichero de configuración:

1. Pulsar el icono de nuevo o elegir la opción **Nuevo** del menú **Archivo**.
2. Si hay datos cargados previamente con modificaciones pendientes de guardar, aparecerá una ventana pidiendo confirmación:



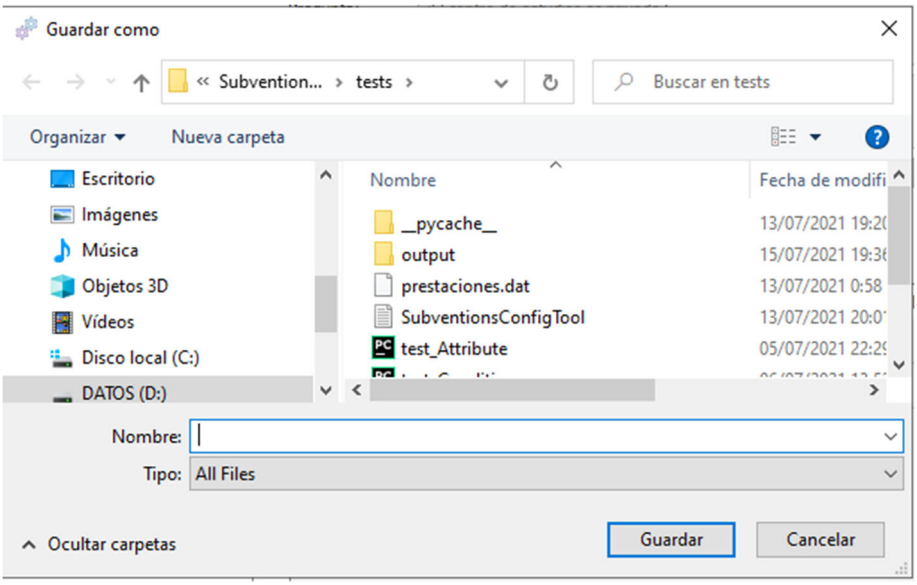
3. Finalmente se mostrarán vacías las pestañas de atributos y prestaciones.

5. GUARDAR UN FICHERO

Para guardar un nuevo fichero de configuración:

1. Pulsar el icono de guardar  , guardar como  , o elegir la opción **Guardar** o **Guardar como** del menú **Archivo**.
2. La opción de **Guardar** guarda los cambios en el mismo fichero cargado previamente o en un nuevo si no había ninguno cargado.

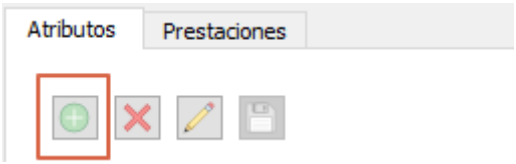
3. La opción de **Guardar como** guarda los cambios en un nuevo fichero, solicitando su nombre y ubicación:



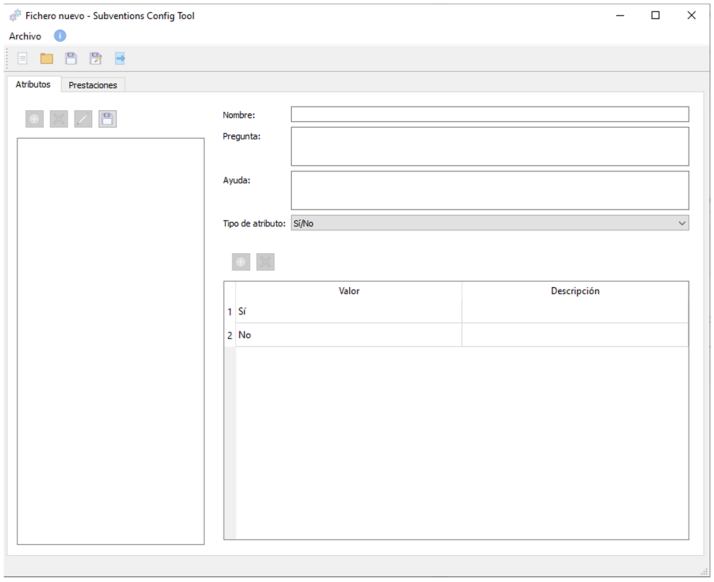
6. CREAR UN ATRIBUTO

Para crear un atributo nuevo el proceso es el siguiente:

1. Pulsar el botón de **Nuevo atributo**:



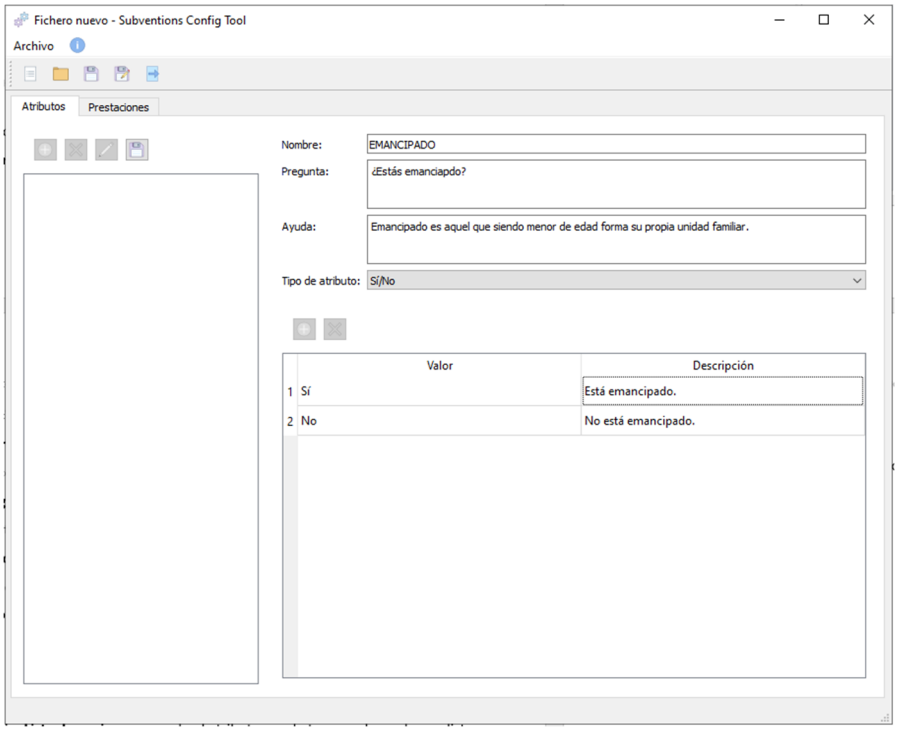
2. La sección de la derecha se habilitará para permitir introducir los datos:



3. La información a proporcionar es:

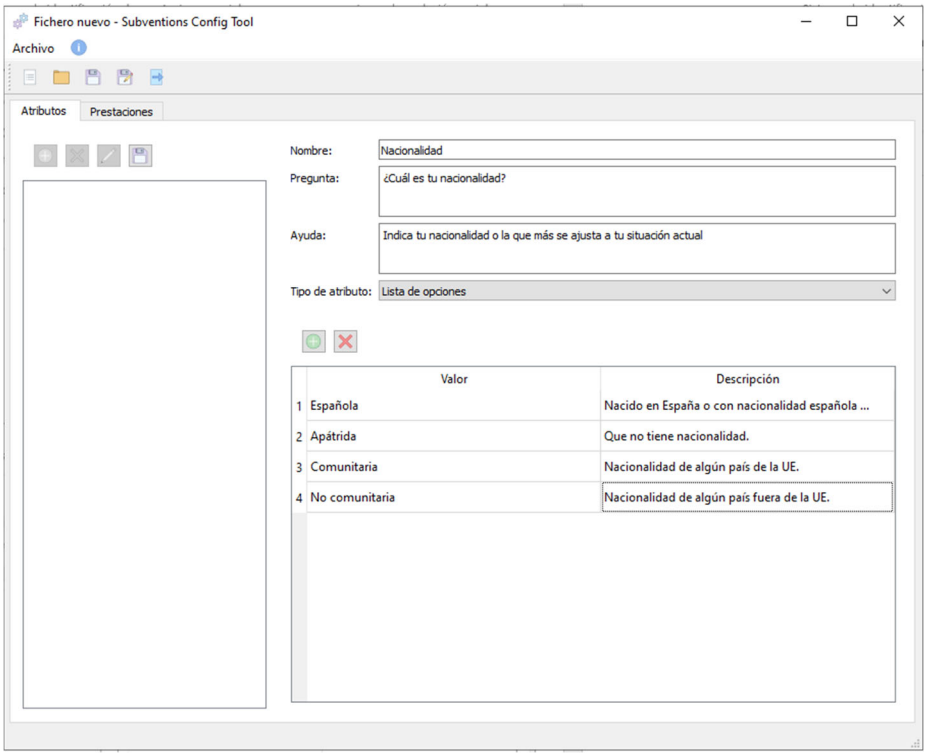
a. **Nombre**: es obligatorio y es el nombre del atributo por el que luego se crearán las condiciones en las prestaciones.
b. **Pregunta**: es obligatorio y es la pregunta asociada al atributo que será mostrada en la aplicación web cuando se solicite al usuario.
c. **Ayuda**: es opcional y proporciona un texto que aparecerá en la aplicación web como aclaratorio o contextualización del dato que se solicita.
d. **Tipo de atributo**: elegir entre las opciones:
    i. **Sí/No**: cuando el atributo puede tomar valores de Sí o No.
    ii. **Numérico:** cuando el atributo puede tomar valores numéricos (por ejemplo, la edad o los ingresos anuales).
    iii. **Lista de opciones**: cuando el atributo puede tomar valores de una lista de varias opciones.

4. Cuando el atributo es de tipo **Sí/No** se habilita la tabla inferior para permitir modificar el valor de la opción y su descripción:



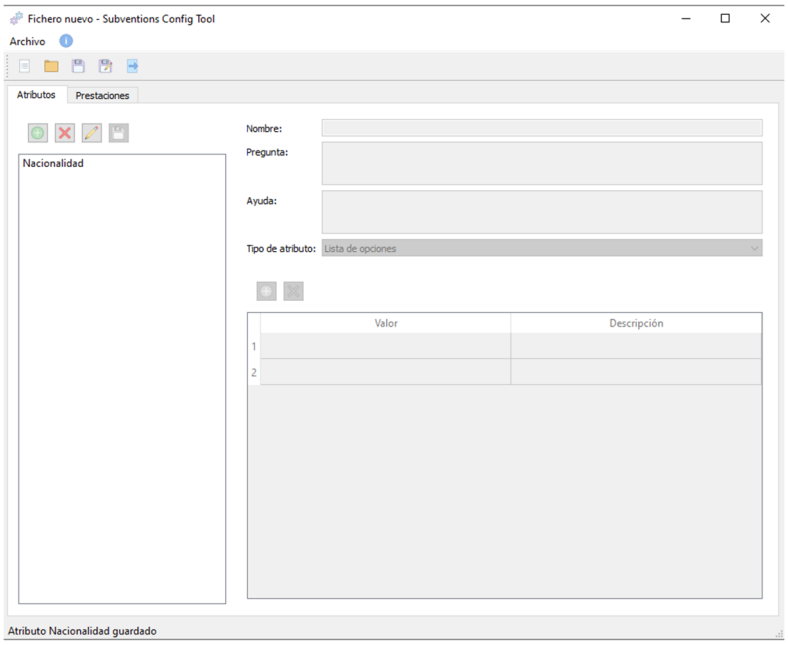
La descripción de cada valor posible del atributo es opcional y se utilizará en la aplicación web para mostrar un pequeño texto explicativo sobre cada opción que proporcione información al usuario de lo que significa esa opción.

5. Cuando el atributo es de tipo **Lista de Opciones**, además de habilitarse la tabla inferior, también se habilitan los botones de añadir nuevo valor  y eliminar un valor  , de manera que permitan añadir un nuevo registro en la tabla o eliminar uno previamente seleccionado, respectivamente.



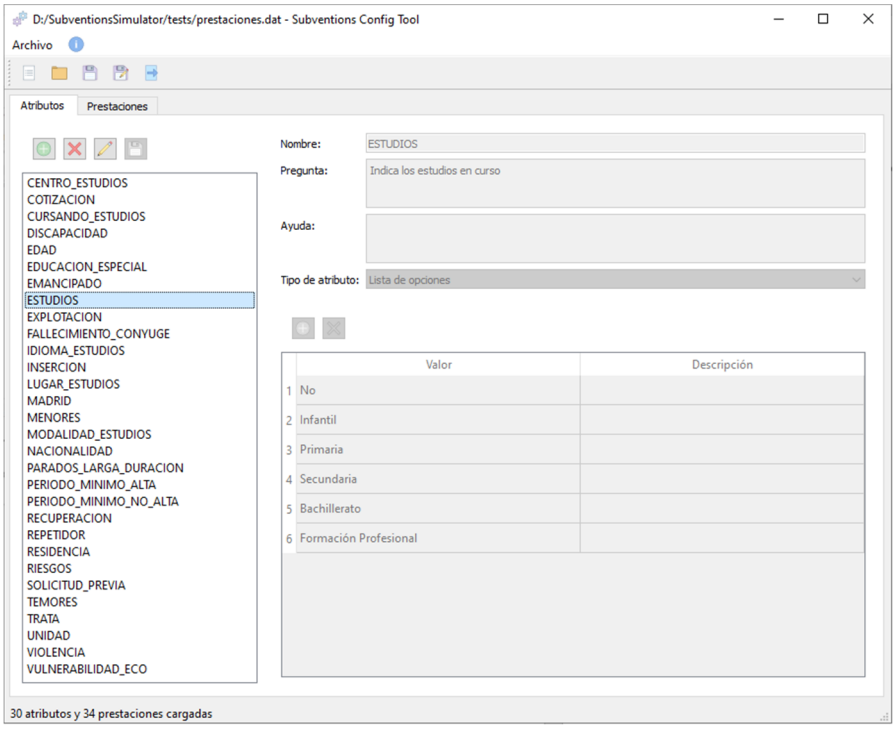
6. Una vez cumplimentada la información mínima necesaria, el atributo se puede guardar usando el botón de guardar:

7. Si el nombre del atributo ya existe, aparecerá un mensaje indicando esta situación por lo que será necesario elegir un nombre diferente.
8. Una vez guardada la configuración del atributo, este aparecerá en el panel de la izquierda:



7. VER EL DETALLE DE UN ATRIBUTO

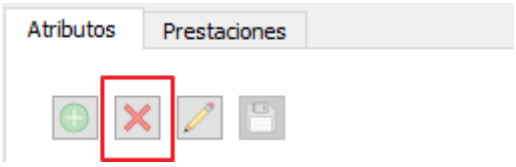
Para ver la información de un atributo solo basta con seleccionarlo en la lista de la izquierda y los datos asociados aparecerán en el panel de la derecha:



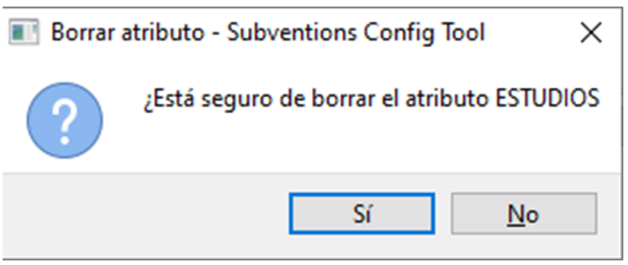
8. ELIMINAR UN ATRIBUTO

Para eliminar un atributo de la lista el proceso es el siguiente:

1. Seleccionar el atributo en la lista de atributos de la izquierda.
2. Pulsar el botón de eliminar atributo:



3. Aparecerá un cuadro de confirmación:



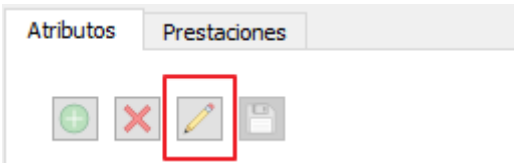
4. Confirmar el borrado pulsando **Sí.**

NOTA: cuando un atributo se elimina, se eliminan todas las condiciones asociadas a dicho atributo en todas las prestaciones configuradas.

9. EDITAR UN ATRIBUTO

Para editar un atributo de la lista:

1. Seleccionar el atributo en la lista de atributos de la izquierda.
2. Pulsar el botón de editar atributo:



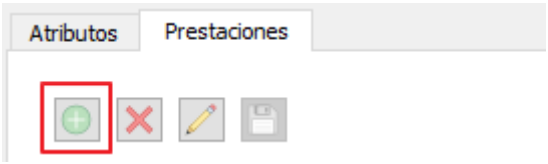
3. Se activan la sección derecha y se pueden modificar los datos del atributo.
4. Una vez finalizada la edición, se pulsa el botón de guardar:

NOTA: cuando un atributo cambia su nombre, se actualizan todas las condiciones asociadas a dicho atributo en todas las prestaciones configuradas.

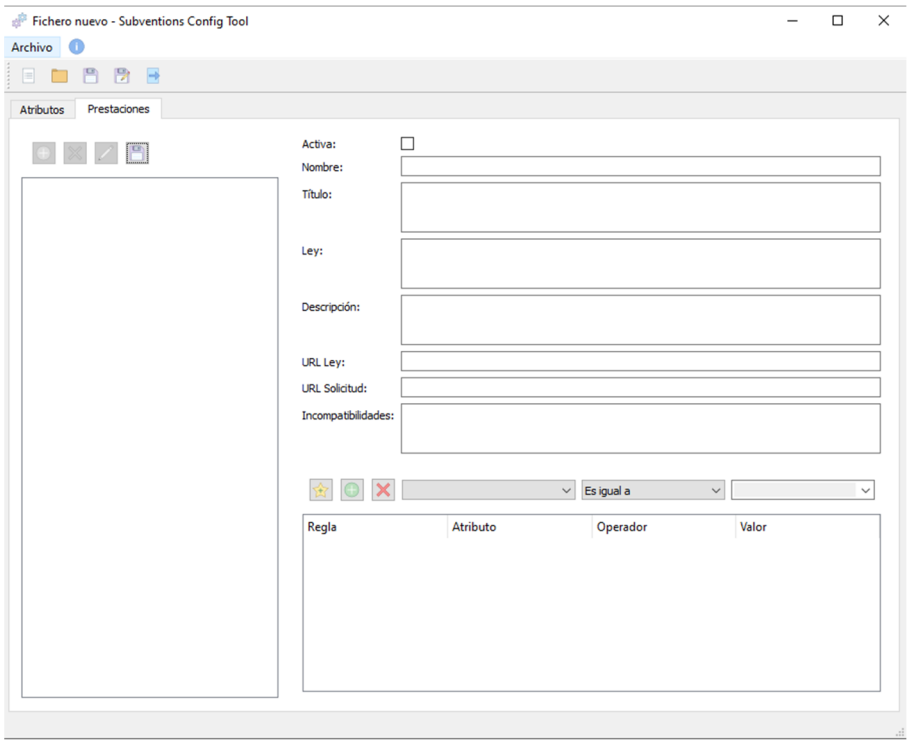
10. CREAR UNA PRESTACIÓN

Para crear una prestación nueva el proceso es el siguiente:

1. Pulsar el botón de **Nueva prestación** en la pestaña de Prestaciones:



2. La sección de la derecha se habilitará para permitir introducir los datos:



3. La información a proporcionar es:
    a. **Activa**: define si la prestación está activa o no, en función de este atributo se mostrará en la aplicación web o no.
    b. **Nombre**: es obligatorio y es el nombre corto de la prestación. Sirve para identificarla en la lista de la izquierda.
    c. **Título**: es obligatorio y es el título de la prestación. Se usará en la aplicación web como título en el catálogo.
    d. **Ley:** es opcional y contiene la información sobre la ley que describe esta prestación y sus condiciones. Si hay información en este campo se mostrará en la aplicación web.
    e. **Descripción:** es opcional y contiene una descripción algo más detallada del alcance de la prestación, a quien va dirigida, etc. Si hay información se mostrará en la aplicación web.

f. **URL Ley:** es opcional y es la dirección web donde está descrita la ley que define la prestación. Si contiene información se mostrará en la aplicación web un botón o enlace a esta dirección.

g. **URL Solicitud:** es opcional y es la dirección web donde se puede solicitar la prestación. Si contiene información se mostrará en la aplicación web un botón o enlace a esta dirección.

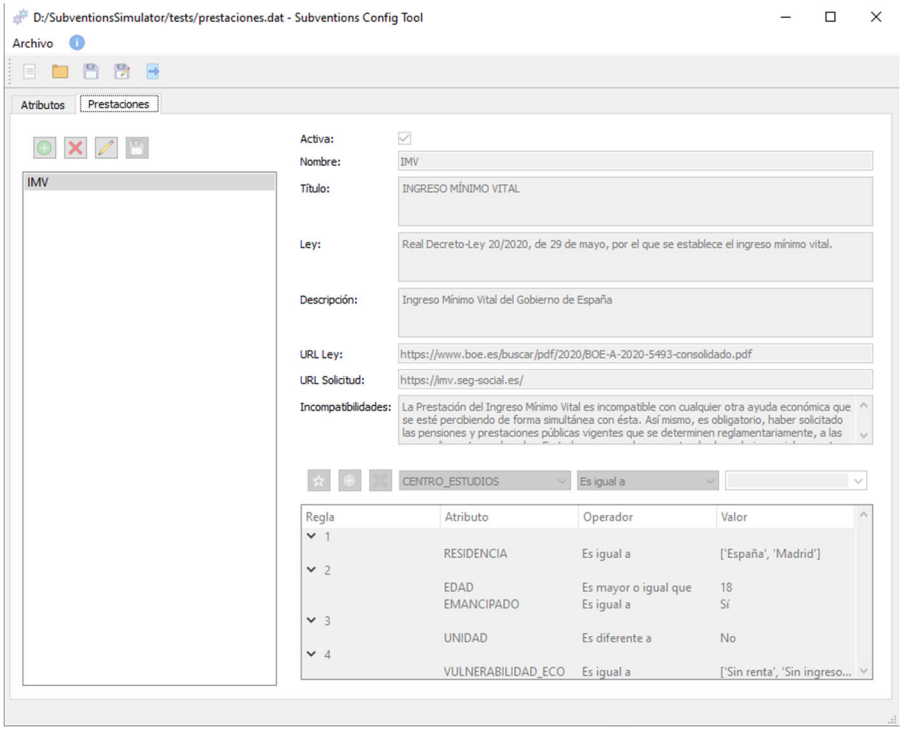
h. **Incompatibilidades:** es opcional y es un texto que explica si esta prestación tiene alguna incompatibilidad. En caso de contener información, en la página de resultados de la aplicación web se mostrará esta información.

i. **Reglas**: en esta subsección se definen las reglas y condiciones que deben cumplirse para poder acceder a esta prestación (ver punto 14 de este manual).

4. Una vez cumplimentada la información mínima necesaria, la prestación se puede guardar usando el botón de guardar:

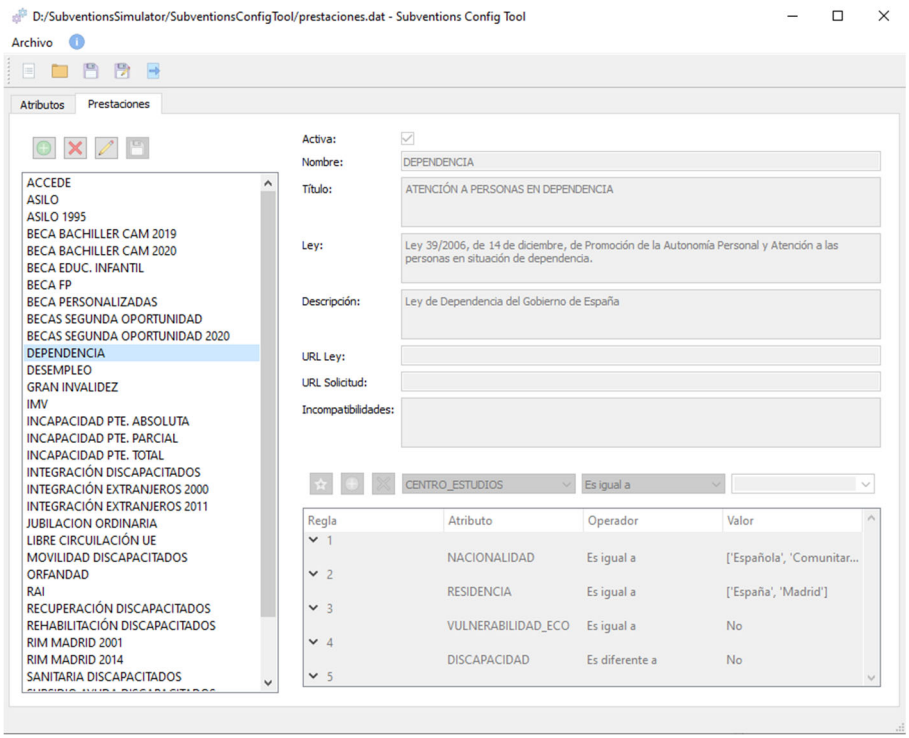
5. Si el nombre de la prestación ya existe, aparecerá un mensaje indicando esta situación por lo que será necesario elegir un nombre diferente.

6. Una vez guardada la configuración de la prestación, esta aparecerá en el panel de la izquierda:



11. VER EL DETALLE DE UNA PRESTACIÓN

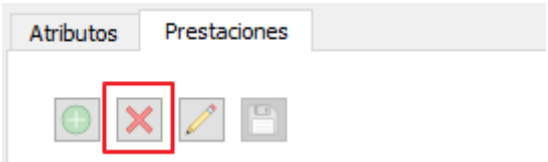
Para ver la información de una prestación solo basta con seleccionarla en la lista de la izquierda y los datos asociados aparecerán en el panel de la derecha:



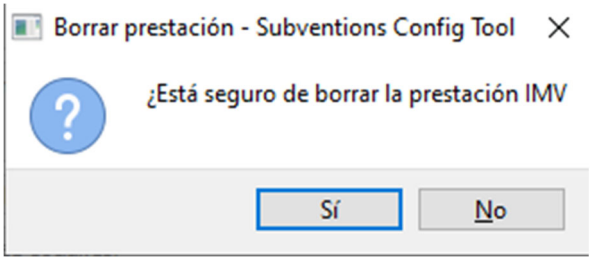
12. ELIMINAR UNA PRESTACIÓN

Para eliminar una prestación de la lista el proceso es el siguiente:

1. Seleccionar la prestación en la lista de prestaciones de la izquierda.
2. Pulsar el botón de eliminar prestación:



3. Aparecerá un cuadro de confirmación:

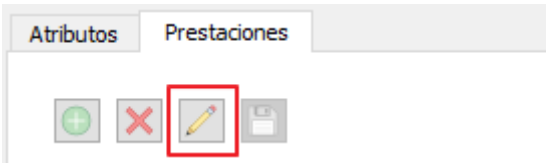


4. Confirmar el borrado pulsando **Sí.**

13. EDITAR UNA PRESTACIÓN

Para editar una prestación de la lista:

1. Seleccionar la prestación en la lista de prestaciones de la izquierda.
2. Pulsar el botón de editar prestación:



3. Se activan la sección derecha y se pueden modificar los datos de la prestación.
4. Una vez finalizada la edición, se pulsa el botón de guardar:

14. GESTIÓN DE REGLAS Y CONDICIONES

Las prestaciones están definidas en base a reglas y condiciones de atributos, de manera que si se satisfacen esas reglas y condiciones se puede tener acceso (teóricamente) a dicha prestación o no. Para ello es necesario definir lo que son estos conceptos:

- **Condición**: una condición es la combinación de un atributo con un operador y un valor de tal manera que se pueda evaluar para un valor dado si satisface o no esa combinación. Por ejemplo, una condición es:

  ```
  NACIONALIDAD Es igual a Española
  ```

  Donde NACIONALIDAD es el atributo, `Es igual a` es el operador y `Española` es el valor. Esta condición exige que el atributo NACIONALIDAD tenga un valor Española para que se cumpla, en caso contrario esta condición no se satisface. De esta manera, cuando se solicita el valor de NACIONALIDAD al usuario, el valor ofrecido se somete a esta condición para saber si la satisface o no.

- **Regla**: una regla es una combinación de condiciones de manera que es suficiente que se cumpla al menos una de las condiciones para que la regla sea válida. Por ejemplo, una regla podría ser:

  ```
  NACIONALIDAD Es igual a Española
  EDAD Es mayor o igual que 35
  ```

Esta regla será válida si el atributo NACIONALIDAD es Española o si la EDAD es mayor o igual que 35.

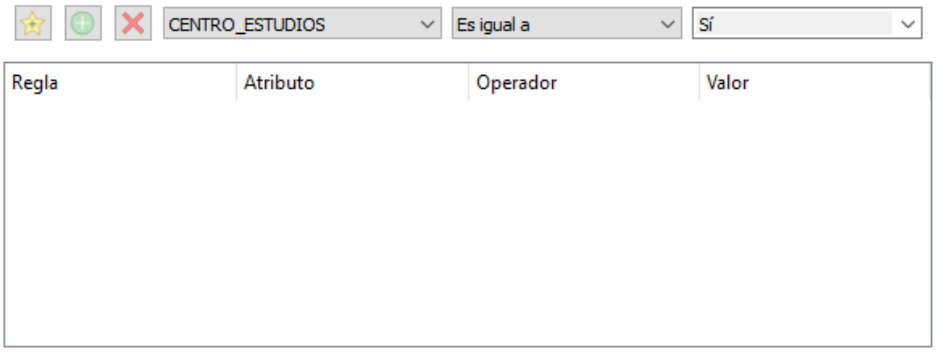
Una prestación se describe como una serie de reglas que **deben ser cumplidas todas** y cada una de ellas para que se tenga acceso a la misma.

Los operadores disponibles para definir condiciones son:

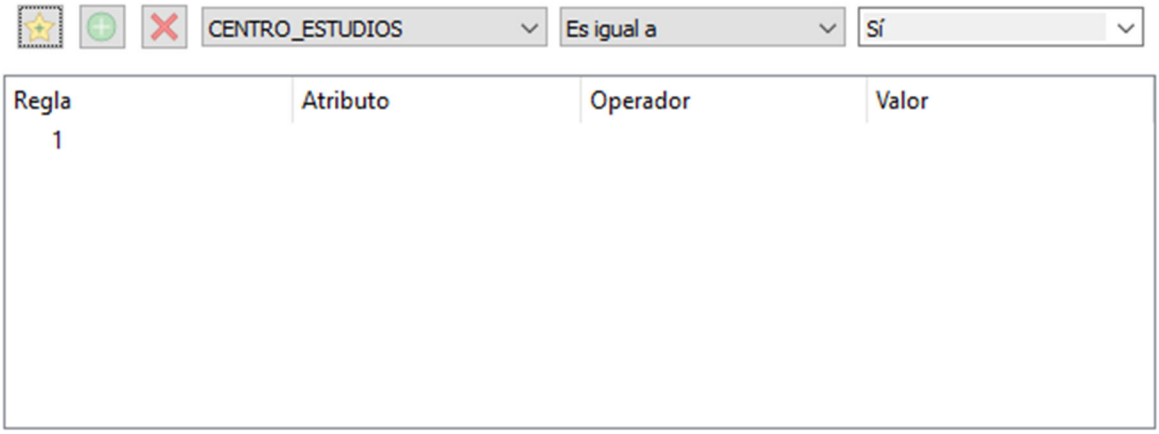
- **Es igual que**: sirve para exigir que el valor del atributo sea idéntico a uno determinado (tanto numérico como textual) o esté dentro de una lista de opciones posibles.
- **Es diferente que**: al contrario que el anterior, sirve para exigir que el valor del atributo no sea idéntico a uno determinado (tanto numérico como textual) o no esté dentro de una lista de opciones posibles.
- **Es menor que**: para valores numéricos sirve para exigir que el valor del atributo es estrictamente menor que uno determinado.
- **Es menor o igual que**: para valores numéricos sirve para exigir que el valor del atributo es menor o igual que uno determinado.
- **Es mayor que**: para valores numéricos sirve para exigir que el valor del atributo es estrictamente mayor que uno determinado.
- **Es mayor o igual que**: para valores numéricos sirve para exigir que el valor del atributo es mayor o igual que uno determinado.

14.1. Añadir reglas

Para añadir una regla a una prestación es necesario que esta prestación esté en modo edición o se esté creando una. En ese caso se habilitará el subpanel inferior derecho:



Para añadir una regla basta con pulsar el botón  y se creará una nueva línea en la tabla inferior numerándose de manera consecutiva.



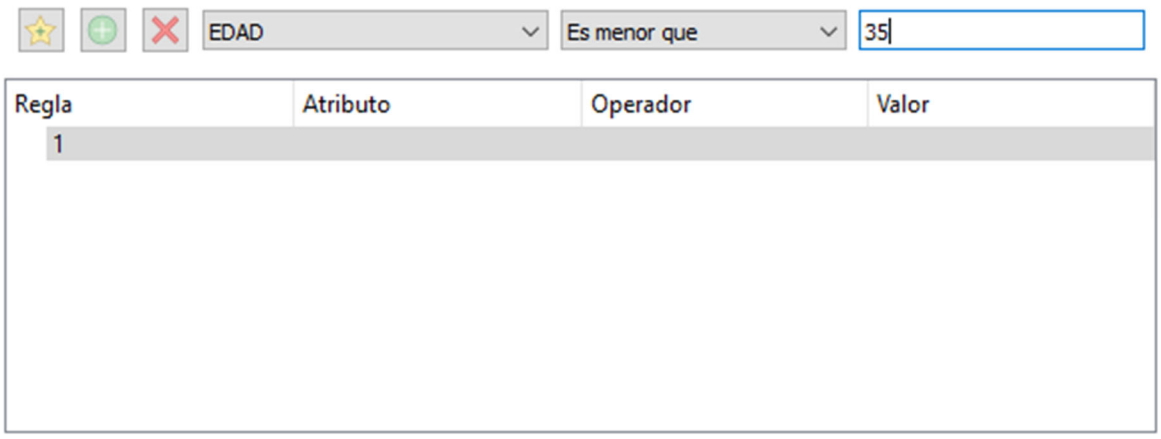
A partir de ese momento se pueden añadir condiciones a esa regla recién creada.

14.2. Añadir condiciones

Para añadir condiciones el procedimiento a seguir es el siguiente:

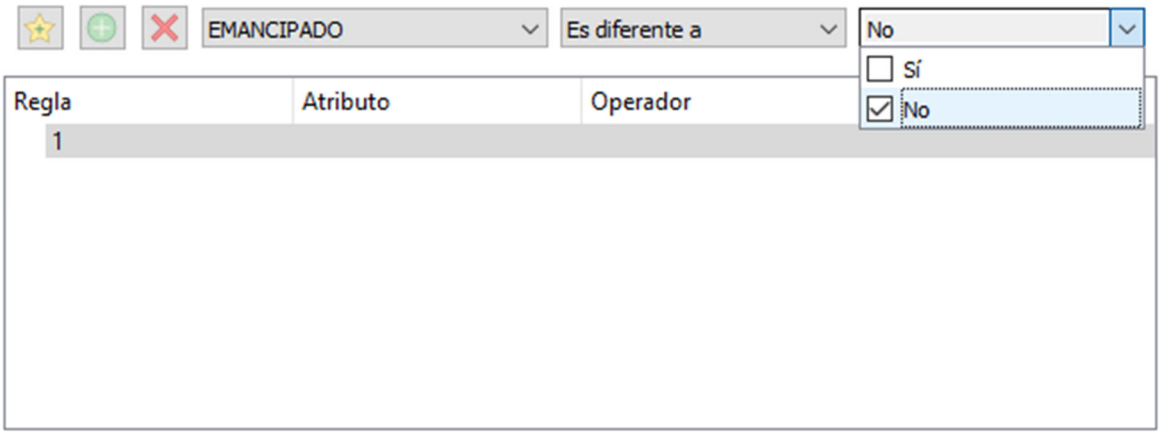
1. La prestación tiene que estar en modo edición o creación de una nueva.
2. Se debe seleccionar la regla sobre la que se quiere añadir una condición.
3. Se establecen los criterios de la condición usando los campos de Atributo, Operador y Valor. Al elegir el atributo la lista de valores se actualizará con los posibles valores que acepta o, en caso de ser de tipo numérico, aparecerá un campo de introducción de texto para indicar el valor numérico.

   Ejemplo de condición con atributo numérico:

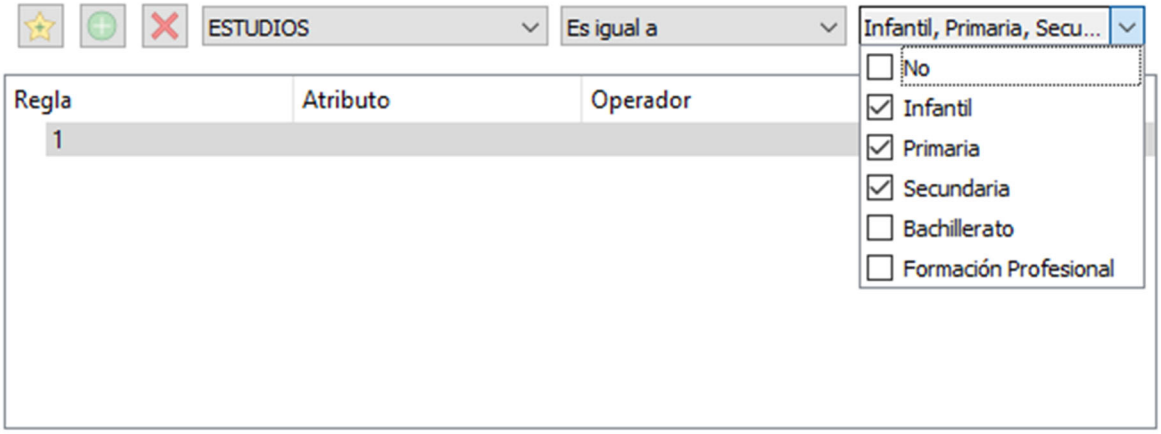


4. Del listado de valores es necesario marcar al menos un valor.

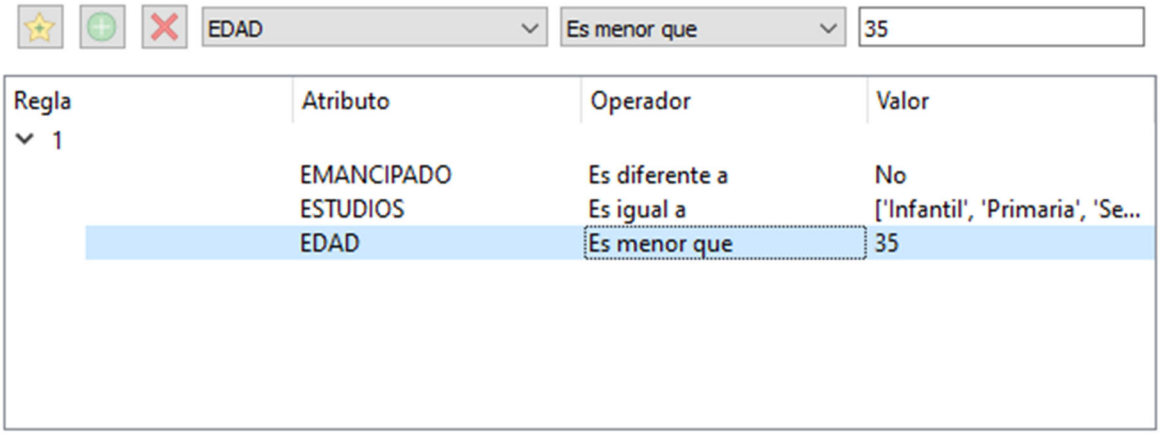
Ejemplo de condición con selección de un único valor:



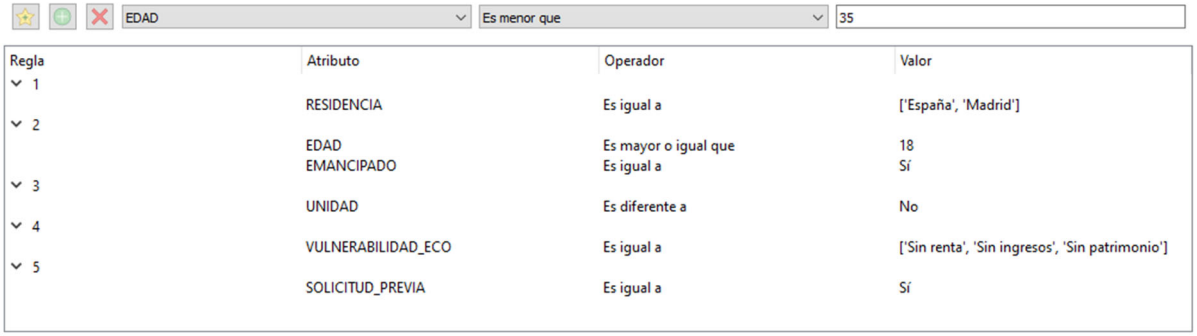
Ejemplo de condición con selección múltiple de valores:



5. Por último, se pulsa el botón de añadir condición  y la condición se añade a la tabla inferior.



Con este mecanismo se pueden definir todas las reglas que debe cumplir una prestación, como en el ejemplo siguiente, que está compuesto por 5 reglas que deben ser satisfechas todas donde una de ellas tiene dos alternativas para ser cumplida.



14.3. Eliminar reglas y condiciones

Para eliminar reglas o condiciones el procedimiento es:

1. La prestación tiene que estar en modo edición o creación de una nueva.
2. Se debe seleccionar la regla o la condición que se quiere eliminar.
3. Se pulsa el botón  y se eliminará la condición o la regla entera, según sea el caso.
4. Cuando una regla se queda sin condiciones es eliminada.

Índice de acrónimos

A

AROPE
: At-Risk-Of Poverty and Exclusion, i, ii, 1, 2, 38, 39

C

C4.5
: C4.5 Algorithm, 14, 15, 19, 31, 92, 96, 99, 100, 102, 103, 104, 107, ii

CART
: Classification and Regression Trees, 14, 15, 21, 31, 96, 100, 102, 103, 104, 107

CC
: Centímetro Cúbicos, 8, 9, 11

CSS
: Cascading Style Sheets, 76, 77, 78

CU
: Caso de Uso, 58, 59

E

EAPN
: European Anti Poverty Network, i, ii, iv, 2, 3, 4, 5, 33, 34, 35, 36, 37, 39, 44, 47, 50, 51, 57, 58, 59, 67, 73, 89, 91, 107, 109, i

H

HTML
: Hyper Text Markup Language, 76, 77, 82

I

ID3
: Iterative Dichotomiser 3, 14, 15, 18, 23, 31, 92, 96, 97, 98, 99, 100, 103, 104, 107

IDE
: Integrated Development Environment, 77

INE
: Instituto Nacional de Estadística, 1, 2, 38, 39

P

PUD
: Proceso Unificado de Desarrollo, 52

R

ROC
: Receiver Operating Characteristics, 105, 106, 107, i

T

TFG
: Trabajo Fin de Grado, 3, 4, 5, 35, 37, 46, 50, 91

TIOBE
: The Importance Of Being Earnest, 48, 49

U

UNIR
: Universidad Internacional de La Rioja, i, ii, iv, 2, 3, 4, 36, 37, 51, 73, 89, 91

URL
: Uniform Resource Locator, 69, 89, xvii