UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Aplicación de Machine Learning en productos software mediante un enfoque de Model-Driven Development

TRABAJO FIN DE MÁSTER

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

Autor: Víctor Alberto Iranzo Jiménez
Tutor: Patricio Orlando Letelier Torres

Curso 2020-2021

Agradecimientos

En primer lugar, quisiera agradecer a mi tutor, Patricio Letelier, por darme la idea del trabajo, por el impulso y la guía para llevarlo a cabo. Gracias por la confianza que depositas en mí y por la cercanía.

Gracias también a Paco, por permitirme hacer este trabajo dentro de la empresa. Quisiera agradecerte el compromiso y las facilidades que me has dado para hacer el proyecto, además del buen ánimo que siempre transmites.

Me gustaría dedicar este trabajo a mi madre y a mi hermana. Gracias por ayudarme con algunos detalles médicos que tiene la memoria, por la confianza y el apoyo. También va dedicado a mi abuelo, por ser el espejo en el que me quiero reflejar.

Por último, me quiero acordar de mis compañeros en el departamento de I+D. El teletrabajo nos ha separado físicamente, pero seguimos siendo una piña. Gracias por enseñarme y acompañarme todos estos años.

Resum

En l'actualitat, cada vegada és més rellevant i demandada la figura del científic de dades en el sector de les TIC. Amb la finalitat de satisfer aquesta demanda, han sorgit eines que permeten dotar d'intel·ligència a les aplicacions seguint estratègies de *low-code* o *no-code*. A grans trets, permeten el desenvolupament d'aplicacions que resolen problemes d'intel·ligència artificial (IA) sense necessitat de programació, esquivant la seua pronunciada corba d'aprenentatge. Aquestes estratègies de desenvolupament de programes són derivades del desenvolupament dirigit per models (MDD).

En aquest TFM es donarà un primer pas en el desenvolupament d'una solució MDD per a dotar a un producte *software* de funcionalitats cobertes mitjançant *machine learning*. En concret, s'abordaran escenaris d'anàlisis de sentiment i de predicció de successos. El projecte s'ha fet en col·laboració amb una PYME dedicada al desenvolupament d'un ERP per al sector sociosanitari. Aquesta empresa compta amb diferents eines de modelatge i generació de codi, amb les quals es realitzarà una integració.

Finalment, els escenaris s'usaran per a modelar dos casos d'ús dins del mateix ERP: l'anàlisi de valoracions mèdiques i la predicció de caigudes en residències.

Paraules clau: Desenvolupament guiat per models, Intel·ligència artificial, *Low-code*, Processament del llenguatge natural

Resumen

En la actualidad, cada vez es más relevante y demandada la figura del científico de datos en el sector de las TIC. Con el fin de satisfacer esta demanda, han surgido herramientas que permiten dotar de inteligencia a las aplicaciones siguiendo estrategias de *low-code* o *no-code*. A grandes rasgos, permiten el desarrollo de aplicaciones que resuelven problemas de inteligencia artificial (IA) sin necesidad de programación, esquivando su pronunciada curva de aprendizaje. Estas estrategias de desarrollo de *software* son derivadas del desarrollo dirigido por modelos (MDD).

En este TFM se dará un primer paso en el desarrollo de una solución MDD para dotar a un producto *software* de funcionalidades cubiertas mediante *machine learning*. En concreto, se abordarán escenarios de análisis de sentimiento y de predicción de sucesos. El proyecto ha sido realizado en colaboración con una PYME dedicada al desarrollo de un ERP para el sector socio-sanitario. Esta empresa cuenta con diferentes herramientas de modelado y generación de código, con las que se realizará una integración.

Finalmente, los escenarios se emplearán para modelar dos casos de uso dentro del propio ERP: el análisis de valoraciones médicas y la predicción de caídas en residencias.

Palabras clave: Desarrollo dirigido por modelos, Inteligencia artificial, *Low-code*, Procesamiento del lenguaje natural

Abstract

Nowadays, the figure of the data scientist in the ICT sector is increasingly relevant and in demand. In order to meet this demand, frameworks have emerged that allow cognifying applications following low-code or no-code strategies. Broadly speaking, they allow the development of applications that solve artificial intelligence (AI) problems without the need of programming, avoiding its difficult learning curve. These software development strategies are derived from Model Driven Development (MDD).

In this TFM, a first step is done in the development of an MDD solution to provide a software product with functionalities covered by machine learning. Specifically, sentiment analysis and event prediction scenarios will be addressed. The project has been carried out in collaboration with an enterprise dedicated to the development of an ERP for the socio-sanitary sector. This company has different modeling and code generation tools. An integration will be done with them.

Finally, the scenarios will be used to model two use cases within the ERP: the analysis of medical progress notes and the prediction of falls in nursing homes.

Key words: Model-driven development, Artificial intelligence, Low-code, Natural language processing

Índice general

Índice de figuras

Índice de tablas

Acrónimos empleados

- **ABVD**: Actividades Básicas de la Vida Diaria.
- **AIVD**: Actividades Instrumentales de la Vida Diaria.
- **ATL**: Atlas Transformation Language.
- **BD**: Base de datos.
- **CASE**: Computer Aided Software Engineering.
- **CIE**: Clasificación Internacional de Enfermedades.
- **CRUD**: Crear, leer, actualizar y eliminar.
- **CUDA**: Arquitectura unificada de dispositivos de cómputo.
- **DDD**: Desarrollo guiado por el dominio.
- **DL**: Deep Learning o aprendizaje profundo.
- **DSML**: Lenguaje de modelado específico de un Dominio.
- **DTO**: Objeto para la transferencia de datos.
- **EGL**: Epsilon Generation Language.
- **EMF**: Eclipse Modeling Framework.
- **ERP**: Sistema de planificación de recursos empresariales.
- **ETL**: Extraer, Transformar y Cargar.
- **EVL**: Epsilon Validation Language.
- **FPR**: False Positive Rate.
- **GPU**: Unidad de procesamiento gráfico.
- **IA**: Inteligencia artificial.
- **IDE**: Entorno de desarrollo integrado.
- **M2C**: Modelo a código.
- **M2M**: Modo a modelo.
- **MBE**: Ingeniería basada en modelos.
- **MDA**: Arquitectura dirigida por modelos.
- **MDD**: Desarrollo dirigido por modelos.
- **MDE**: Ingeniería dirigida por modelos.
- **ML**: Machine learning o aprendizaje automático.
- **MOF**: Meta-Object Facility.
- **NLP**: Procesamiento del lenguaje natural.

- **OCL**: Object Constraint Language.
- **OMG**: Object Management Group.
- **ONNX**: Open Neural Network Exchange.
- **OWL**: Web Ontology Language.
- **PIM**: Modelo independiente de la plataforma.
- **PMML**: Predictive Model Markup Language.
- **PR**: Precision-Recall.
- **PSM**: Modelo específico de la plataforma.
- **PYME**: Pequeña y mediana empresa.
- **QVT**: Query/View/Transformation.
- **ROC**: Receiver operating characteristic.
- **SDK**: Kit de desarrollo de *software*.
- **SER**: Speech Emotion Recognition.
- **STT**: Speech to Text.
- **TN**: True Negative.
- **TP**: True Positive.
- **TPR**: True Positive Rate.
- **TTS**: Text to Speech.
- **UI**: Interfaz de usuario.
- **UML**: Lenguaje unificado de modelado.

CAPÍTULO 1
Introducción

1.1 Motivación

Durante el año 2020, los puestos de trabajo en ciencia de datos e inteligencia artificial (IA) aumentaron en un 64 % [1]. Los conocimientos que estos perfiles requieren son muy variados y diferentes de los que poseen otros profesionales como son los ingenieros de *software* o los analistas de un dominio concreto. Mientras que en el contexto del ingeniero de *software* predominan los lenguajes de programación como Java o C#, el científico de datos se caracteriza por usar lenguajes como Python o R. Las librerías que se emplean, como Keras [1], TensorFlow [2] o PyTorch [3], suponen una curva de aprendizaje empinada incluso para programadores experimentados [2].

El desarrollo de modelos de inteligencia artificial se realiza muchas veces de manera iterativa en ciclos de prueba y error, donde se van ajustando los datos y parámetros para el entrenamiento [3]. Este proceso manual es ineficiente y poco sostenible. Por ejemplo, se estima que para el entrenamiento de grandes modelos de IA se emiten a la atmósfera el equivalente a las emisiones de dióxido de carbono hechas por 5 vehículos durante todo su período de vida [4]. Si tenemos en cuenta la escasez de profesionales, se hace evidente la necesidad de automatizar los procesos manuales de construcción de modelos de IA.

Por otra parte, el desarrollo de *software* ha estado ligado, desde sus inicios, con un esfuerzo continuo por agilizar y automatizar sus procesos [5]. Con este fin, se puede observar a otras industrias para ver cómo solventan sus problemas, sobre todo de escalabilidad. De manera análoga, se ha buscado siempre transitar de construcciones de *software* artesanales hacia desarrollos más industriales, basados en la reutilización para satisfacer la demanda [6]. Como veremos en la memoria, la propuesta del desarrollo dirigido por modelos (en inglés, *Model-Driven Development* o MDD) ha demostrado grandes beneficios en la industrialización elevando el nivel de abstracción [7]. ¿Por qué no aplicar esta estrategia para el desarrollo de inteligencia artificial?

El presente trabajo se ha desarrollado en colaboración con una PYME dedicada al desarrollo de *software*. Esta empresa comercializa un ERP para el sector socio-sanitario y apuesta por el uso de la generación automática de código basada en modelos para implementar la nueva versión de su producto. La motivación de este trabajo es crear las herramientas necesarias para modelar escenarios de IA, integrando los nuevos modelos con otros ya existentes. Así, cualquier usuario de estas herramientas podrá construir aplicaciones incorporando **valor añadido** sin necesidad de conocimientos en el campo de la inteligencia artificial.

[1] Página oficial de Keras: keras.io
[2] Página oficial de TensorFlow: tensorflow.org/?hl=es-419
[3] Página oficial de PyTorch: pytorch.org

Si bien ya existen varias herramientas para el desarrollo de inteligencia artificial orientadas a usuarios noveles o ciudadanos (*citizen developers*), los ámbitos de aplicación de cada una de ellas son muy concretos o están ligadas a una plataforma específica [8]. En este trabajo se revisarán algunas y se dará el primer paso para construir una solución usando una estrategia MDD más integrada dentro del desarrollo de un producto *software*. En concreto, se abordarán dos casos de uso: el primero, para el análisis de sentimiento en un texto; el segundo, para la predicción de sucesos a futuro.

1.2 Objetivos

El objetivo de este proyecto es desarrollar una herramienta de modelado que permita incorporar funcionalidades resueltas mediante *machine learning* (ML) en un producto *software*. En concreto, los objetivos específicos son:

- Permitir la representación de tareas de análisis de sentimiento en campos de texto usando la herramienta desarrollada.
- Soportar con dicha herramienta la construcción de modelos de ML para la predicción de sucesos a futuro.
- Integrar la herramienta con otras ya existentes, que emplean el modelado para representar otros elementos de una aplicación *software*.
- Validar los beneficios de usar una estrategia MDD para la incorporación de funcionalidades que utilizan ML.

1.3 Estructura de la memoria

A continuación, se presenta la estructura de la memoria:

El capítulo 1 introduce la memoria a través de la motivación del trabajo y los objetivos a cumplir.

Los capítulos 2 y 3 son una revisión del desarrollo dirigido por modelos y la inteligencia artificial, respectivamente. Se presentarán solo los términos que se usarán a lo largo del trabajo, ya que ambos campos de estudio son de extensión inmensurable. Un lector avanzado puede saltarse estas secciones.

El capítulo 4 revisa algunos artículos de la literatura en los que se desarrolla inteligencia artificial siguiendo una estrategia basada en modelos, además de herramientas para este propósito.

En el capítulo 5 se analizan los requisitos para la construcción de la nueva herramienta y se resume el contexto de desarrollo.

Los capítulos 6 y 7 resumen, respectivamente, el diseño y desarrollo de los escenarios planteados, que son los escenarios para el análisis de sentimiento y predicción de sucesos. También se valoran los resultados de aplicar los escenarios a 2 casos de uso concretos.

El capítulo 8 presenta las conclusiones del trabajo respecto a los objetivos planteados inicialmente. Finalmente, el último capítulo sintetiza el trabajo pendiente de los dos escenarios.

CAPÍTULO 2
Desarrollo dirigido por modelos

En este capítulo vamos a aportar algunas definiciones relacionadas con la ingeniería dirigida por modelos. También se presentarán algunos conceptos que se usarán a lo largo de la memoria, como son los términos de desarrollo *low-code* o las factorías de *software*.

2.1 Definiciones

El uso de modelos es una praxis extendida en muchos campos de estudio: en física podemos recordar el modelo atómico de Bohr al igual que en biología podemos recordar modelos del sistema respiratorio o digestivo. En todos estos casos se busca elevar el nivel de abstracción mediante la reducción de características, para reflejar en el modelo solo aquellas que son más relevantes, y el mapeo de características, que consiste en construir un prototipo generalizado de una población para describir una realidad [9].

Aplicado al contexto de la ingeniería de *software*, el objetivo esencial es derivar valor a partir de modelos para lidiar con la complejidad de los sistemas. Los modelos representan información sobre la que se puede hacer consultas, validaciones, simulaciones y transformaciones a otros formatos útiles, como código ejecutable [10].

Uno de los términos principales de la literatura es el de **desarrollo dirigido por modelos**, que representa el paradigma donde los modelos son considerados artefactos centrales en lugar de mera documentación durante el desarrollo [9]. Los modelos son usados a lo largo del ciclo de vida del *software*. Durante la toma de requisitos se pueden emplear diagramas de casos de uso para representar la funcionalidad de la que se dotará al usuario. Por otro lado, las actividades de pruebas e implementación pueden llevarse a cabo mediante la generación automática de código [11].

La **arquitectura orientada a modelos** (*Model-Driven Architecture* o MDA) es la visión particular que hace el *Object Management Group* (OMG) respecto al paradigma MDD. MDA es una forma específica de MDD que utiliza los estándares definidos por el OMG tales como MOF (*Meta Object Facility*), UML (*Unified Modeling Language*) o QVT (*Query/-View/Transformation*). Otras visiones son posibles, como la que veremos en la sección **2.3.1 Factorías de *software***.

Como superconjuntos de MDA y MDD se encuentran la **ingeniería dirigida por modelos** (*Model-Driven Engineering* o MDE), donde los modelos se emplean para representar procesos adicionales más allá de los ligados al desarrollo de *software*, y la **ingeniería basada en modelos** (*Model-Based Engineering* o MBE), en la que los modelos no tienen por qué ser el artefacto central del proceso de ingeniería. Estos términos se representan en la figura 2.1 [9].

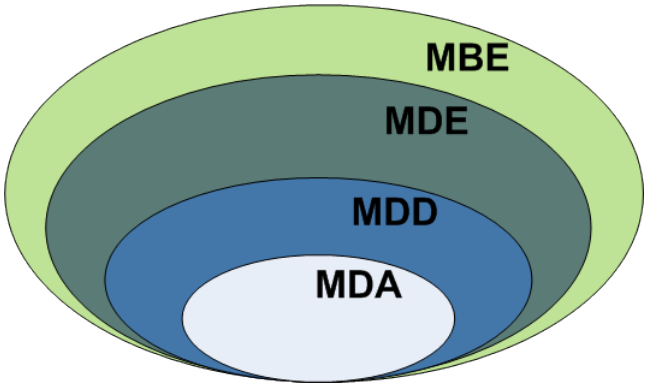

Figura 2.1: Conjuntos de la ingeniería de modelos [9].

Los modelos deben expresarse de tal forma que transmitan información que los *stakeholders* involucrados entiendan, que son tanto los miembros del equipo de desarrollo como los clientes. Por tanto, deben estar expresado en un lenguaje usando una notación, estructura, términos, sintaxis, semántica y reglas de integridad que sean consistentes. El conjunto de todos estos artefactos es llamado **lenguaje de modelado**. En los lenguajes de modelado formales, se puede determinar si un modelo es conforme a dicho lenguaje. Cuando el lenguaje de modelado se define para un dominio o contexto concreto, se le conoce también como **lenguaje de modelado específico de dominio** (*Domain Specific Modeling Language* o DSML) [9].

La mejor manera de expresar un lenguaje de modelado es un modelo, denominado metamodelo. Un **metamodelo** es un modelo que define un lenguaje de modelado y es expresado usando así mismo un lenguaje de modelado, el metametamodelo [10]. En MDA, los lenguajes se pueden definir formalmente a través de MOF o utilizando perfiles, estereotipos y restricciones de UML, que permiten extender el vocabulario de UML [12]. El propio UML está definido usando MOF.

En este trabajo vamos a seguir un enfoque MDD. Esto se debe a que no vamos a hacer uso de los estándares que propone OMG, por lo que no podemos catalogar nuestra propuesta como MDA. Tampoco tiene sentido etiquetar nuestra propuesta únicamente como MDE porque los modelos que construiremos tienen como fin único el desarrollo de *software*. Los modelos serán el eje vertebrador del producto *software*, que se deriva de ellos a partir de generación automática de código.

2.2 Beneficios de seguir una estrategia MDD

En la literatura, hemos observado consenso en cuanto a los beneficios de usar estrategias MDD, aunque estos son formulados por los autores de diversas formas. En esta memoria hemos querido resumir los principales beneficios en los siguientes 3 puntos [7, 9, 10]:

- **Estandarización**: los modelos emplean unos términos y unas notaciones bien definidas. Esto mejora la colaboración entre las personas y organizaciones. Son el medio de comunicación entre diferentes *stakeholders*.
- **Transformaciones, ahorro de costes y consistencia**: a partir de los modelos se pueden generar diferentes artefactos. La automatización de este proceso reduce el tiempo y coste del diseño, la implementación y el mantenimiento. Además, garantiza la consistencia entre todos los artefactos derivados. Los artefactos generados no tienen por qué ser necesariamente código. Pueden ser otros modelos con diferentes niveles de abstracción o parametrizados para una plataforma concreta.
- **Mejoras analíticas**: diferentes análisis pueden realizarse usando los modelos, como son validaciones o métricas. El análisis de los modelos es relevante para garantizar la calidad, tomar decisiones, evitar errores en la generación de artefactos, monitorizar, hacer análisis de impacto, etc. Si los modelos son ejecutables o pueden producir una simulación rápida del sistema, el cliente validará el sistema más rápido y agilizará el ciclo de desarrollo.

2.3 Contexto y evolución de MDD

En [13], Bucchiarone *et al.* revisan la evolución de MDD en los últimos años. Las primeras herramientas que emergieron para soportar la filosofía MDD fueron las herramientas CASE (*Computer Aided Software Engineering*) alrededor de los años 80. Estas surgieron por la necesidad de traer orden y abstraer grandes sistemas *software*.

Entre los años 90 hasta aproximadamente el 2007, la comunidad centró su esfuerzo en la estandarización de las técnicas existentes, dando lugar a estándares como MOF u OCL (*Object Constraint Language*). Entre los retos cubiertos del 2007 hasta hoy podemos mencionar la construcción de repositorios para modelos, metamodelos y transformaciones, como es ReMoDD [1]. También se han desarrollado entornos integrados (IDE) para el desarrollo de lenguajes, así como mejoras en el rendimiento de las transformaciones que permiten escalar MDD a grandes sistemas *software*.

Actualmente, está en boga la integración de MDD con otros campos de estudio como la inteligencia artificial. La IA es un dominio que requiere un alto número de conocimientos que muchos profesionales no poseen. MDD puede aportar grandes beneficios, abstrayendo aspectos tecnológicos de ML y fomentando la generación automática de código [13]. MDD también se beneficia del campo de la IA. La **síntesis de programas** es un área de investigación que se remonta hasta los años 60 y se define como el proceso de automatización para construir programas ejecutables a partir de especificaciones de alto nivel. Esta especificación, que puede ser entendida como un modelo, podría ser provista utilizando un lenguaje natural y procesada mediante técnicas de procesamiento de lenguaje natural (*Natural Language Processing* o NLP) [14]. De esta forma, un *citizen developer* es capaz de desarrollar *software*, abstraído de conocimientos de programación.

Por último, la expansión del COVID-19 ha manifestado la importancia de MDD. En Nueva York, por ejemplo, el portal para gestionar la emergencia sanitaria se desarrolló en apenas 3 días sin escribir una sola línea de código gracias a herramientas de este paradigma [14].

[1] Página del repositorio ReMoDD: cs.colostate.edu/remodd/v1/

2.3.1. Factorías de *software*

Las **factorías de *software*** son un enfoque para el desarrollo de *software* que integra varias técnicas de la ingeniería de *software* como son el desarrollo basado en componentes, el desarrollo dirigido por modelos y las familias de productos *software*. Su objetivo es industrializar el desarrollo de *software* para acelerar la producción de productos adaptados a las necesidades de cada cliente.

Las familias de productos *software*, que son un conjunto de diferentes productos con características comunes, se construyen mediante el modelado a través de un DSML y el ensamblaje de componentes, como son librerías comunes o *core*. Una factoría de software captura los requisitos que representan estas características comunes y construye herramientas y componentes que puedan ser empleados para desarrollar dicha familia de productos. De esta forma, se fomenta la abstracción y la reutilización ya que los desarrolladores solo deben encargarse de implementar la funcionalidad específica que no queda cubierta por la factoría [6].

Si se compara MDA con la propuesta de las factorías de *software*, MDA es más explícito en los **estándares** que pueden usarse para gestionar modelos: UML, QVT, MOF, etc. También incide claramente en hacer modelos abstraídos de plataforma, dando lugar a los modelos independientes de plataforma (PIM) y a los específicos de plataforma (PSM). Las factorías de *software* no proponen ninguna técnica o estándar en concreto, solo el uso de DSML concretos para el problema a resolver. En cuanto a las herramientas disponibles, MDA no es explícito en cuanto a herramientas o entornos de trabajo con los que llevar a cabo el desarrollo, pero cuenta con un número mayor de ellos: Eclipse EMF[2], Acceleo [3], etc [15].

2.3.2. Estrategias *low-code* y *no-code*

Una autocrítica común en la que coincide la comunidad sobre la falta de adopción de MDD es no haber sabido evangelizar sobre sus beneficios [13]. En los últimos años, han proliferado las denominadas herramientas *low-code* y *no-code*. Según Cabot [16], estos términos no están científicamente definidos y su definición varía según las herramientas que así se denominan.

Estas herramientas buscan acelerar el desarrollo y entrega de aplicaciones reduciendo el código a desarrollar a mano, incluso al mínimo en el caso de las catalogadas como *no-code*. Por este motivo, Cabot asegura que ambas estrategias pueden tratarse como un sinónimo o evolución de MDD. El término *low-code* transmite un mensaje más claro que MDD: es más cercano a la comunidad de desarrolladores y evita usar términos complejos, tales como 'transformaciones' o 'validaciones' de modelos.

Se estima que el 65 % de aplicaciones en 2024 usarán de alguna forma estrategias *low-code* [14, 17]. Como se muestra en [17], son muchos los dominios en los que han proliferado este tipo de aplicaciones, accesibles mayoritariamente desde la web. De la misma forma que MDD tiene un origen vinculado con las herramientas CASE, las aplicaciones *low-code* lo tienen con MDD. Su popularidad debe verse como una oportunidad para hacer crecer la comunidad [16].

[2]Página de Eclipse Modeling Framework: eclipse.org/modeling/emf/
[3]Página de Acceleo: eclipse.org/acceleo/

CAPÍTULO 3
Inteligencia artificial

Son muchos los términos que salen a colación cuando hablamos de inteligencia artificial. En este capítulo estableceremos la definición de algunos de ellos para facilitar la lectura del resto del documento.

3.1 Definiciones

La **inteligencia artificial** puede definirse como "el conjunto de ciencias, teorías y técnicas con el propósito de reproducir en una máquina las habilidades cognitivas de un ser humano" [18]. El término no es nuevo; ya en 1948 Alan Turing dió una de las definiciones más extendidas a través de la prueba de Turing, que informalmente se define como la capacidad de una máquina de hacerse pasar por un humano sin que una tercera persona sea capaz de discernir si está interactuando con una máquina u otro humano [19].

Dentro del concepto global de IA reside un campo más concreto, el **aprendizaje automático** o *machine learning*. Mientras que IA es la ciencia para que las máquinas imiten las habilidades humanas, ML se centra en como enseñar a la máquina a aprender y adaptarse mediante la experiencia. ML es la ciencia que permite a los ordenadores aprender de los datos, sin ser explícitamente programados. Formalmente, un programa se dice que aprende de una experiencia E con respecto a una tarea T y una medida de rendimiento P si su rendimiento medido con P para la tarea T mejora con la experiencia E [20].

Para problemas complejos, usar la programación tradicional se convierte en una larga lista de reglas. Por ejemplo, el problema del reconocimiento del habla (*speech recognition*) mediante programación tradicional resultaría difícil de escalar debido a los miles de palabras a procesar. En general, las soluciones de ML se adaptan mejor a entornos fluctuantes donde aparecen continuamente nuevos datos a procesar. El uso de *machine learning* también puede estar enfocado a encontrar patrones en los datos que aparentemente son difíciles de prever. Este tipo de actividad se conoce como **minería de datos** o *data mining*[20].

Diferentes clasificaciones se pueden hacer de las técnicas de ML según si se crean modelos predictivos, si los modelos aprenden incrementalmente o de acuerdo con el grado de supervisión humana durante el entrenamiento [20]. En esta memoria nos vamos a centrar en técnicas de ML para el **aprendizaje supervisado**, que es aquel en el que los datos del entrenamiento contienen la solución esperada. Ejemplos típicos de este tipo de aprendizaje son los problemas de clasificación o regresión. Los algoritmos que se emplean para estos problemas son k-Vecinos cercanos, regresiones lineares o logísticas, árboles de decisión, etc [21].

En el capítulo **4 MDD aplicado en el contexto de IA** haremos un inciso en las **redes neuronales**, ya que son una de las estructuras que más se ha querido vincular con MDD. Aunque sus orígenes se remontan a los años 50 como una forma de crear un modelo inspirado en la biología del cerebro, su auge ha sido en la última década gracias en parte a la proliferación de tarjetas gráficas (GPU) para las tareas de entrenamiento [21].

Las redes neuronales se componen de neuronas o nodos, que se organizan en diferentes capas interconectadas a través de arcos con un peso llamados sinapsis. La capa de entrada representa los parámetros de entrada del modelo, que se usarán para predecir un valor de salida. Los valores de entrada atraviesan la red neuronal siguiendo las sinapsis. En cada nodo se harán una serie de operaciones sencillas y si el resultado obtenido supera un valor umbral (función de activación), la neurona transmitirá el resultado a los nodos de la siguiente capa que tenga conectados. La función de activación puede representarse como el producto entre una serie de parámetros libres o pesos y los valores de entrada del nodo. Durante el entrenamiento de la red neuronal, los pesos son ajustados para minimizar de manera iterativa el error asociado a la predicción de la salida [22].

La aplicación de redes neuronales con un gran número de capas suele denominarse **aprendizaje profundo** o *deep learning* (DL). *Deep learning* es un subconjunto dentro de ML que se aplica para resolver problemas más complejos como el reconocimiento del lenguaje natural o el reconocimiento de imágenes, problemas que con otros algoritmos de ML no se pueden abordar. En la imagen 3.1 se observa la relación entre los términos IA, ML y DL.

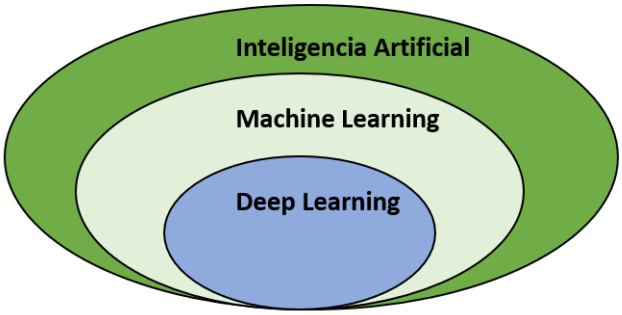

Figura 3.1: Conjuntos de la inteligencia artificial [1].

[1] An Introduction to Deep Learning algorithmia.com/blog/introduction-to-deep-learning

3.2 Ciclo de desarrollo de ML

En primer lugar, el proceso de creación de un modelo de ML comienza con la **recolección** de los datos para el entrenamiento. Los datos pueden provenir de diferentes fuentes que pueden ser integradas siguiendo procesos como ETL (Extraer, Transformar y Cargar)[23].

Una vez hecho esto, se prosigue con una primera **inspección o análisis** de los datos. Es interesante utilizar herramientas de visualización que nos permitan rápidamente detectar inconsistencias en los formatos, sobre todo si se han integrado datos de diferentes fuentes, y ver la distribución de los datos así como valores anómalos o *outliers*, que pueden ser más tarde eliminados.

Posteriormente, se sigue con el **preprocesado** y limpieza de los datos. Aquí los datos son normalizados, se reemplazan propiedades categóricas por una codificación como la *one-hot*, que representa estas propiedades como combinaciones de bits, se eliminan observaciones con valores faltantes (o se reemplazan estos por los valores medios) y se crean nuevas propiedades que aporten valor al modelo, entre otras transformaciones [21]. Esta etapa es la que más tiempo consume a lo largo del proceso, como se observa en la figura 3.2 [24].

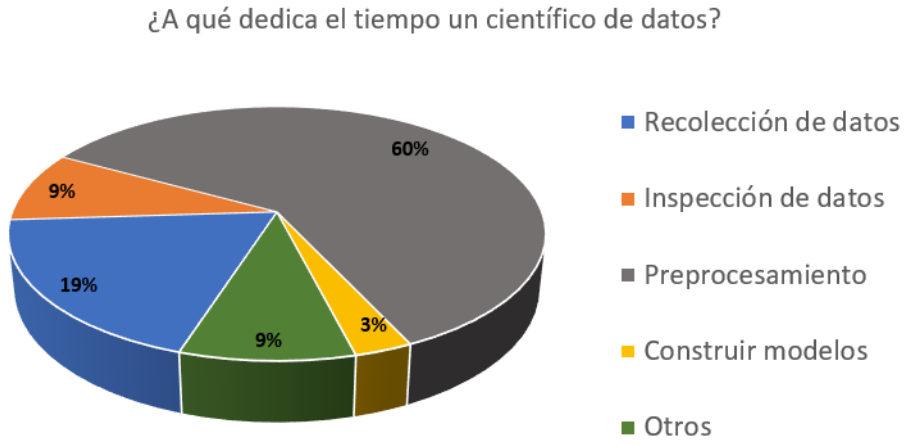

Figura 3.2: Etapas y dedicación de tiempo en el proceso de desarrollo de ML [24].

El algoritmo a usar para la creación del modelo dependerá de los datos y el problema a resolver. Existen algoritmos más rígidos como las regresiones lineares o las logísticas, que cuentan con un menor número de hiperparámetros. Los **hiperparámetros** son parámetros del algoritmo de aprendizaje. Estas regresiones ofrecen una representación sencilla, intuitiva y fácil de visualizar, al igual que los árboles de decisión. Otros algoritmos como las redes neuronales o los *random forests* [2], que es una composición de árboles de decisión, son menos explicativos y pueden verse como una caja negra, pero funcionan bien para la mayoría de las tareas [21].

La construcción del modelo debe ser evaluada mediante diferentes métricas para garantizar su rendimiento. Generalizar es un error típico humano, pero las máquinas también pueden caer en él. El **sobreentrenamiento** (o en inglés, *overfitting*) significa que el modelo se ejecuta correctamente con los datos de prueba, pero no generaliza bien. Esto ocurre cuando el modelo es muy complejo, hay pocos datos o tienen mucho ruido, es

[2]Los random forest son una representación bastante extendida gracias a que introduce aleatoriedad y *boosting*, donde un conjunto de clasificadores débiles (árboles de decisión) forman un clasificador fuerte cuando este sigue el voto de la mayoría.

decir, que no son significativos para el problema. Como solución, se pueden reducir los atributos, obtener más datos o corregir errores en los datos, como *outliers*. Este proceso se llama **regularización**. La regularización puede controlarse mediante los hiperparámetros. Por ejemplo, en un modelo linear, podemos definir menos grados de libertad, siendo los dos primeros la altura y la pendiente de una recta [20].

Una buena manera de probar cómo de bien generaliza nuestro modelo es dividiendo el total de los datos en 2 conjuntos: uno para el entrenamiento (80 %) y otro para las pruebas (20 %). Con estos *sets* se puede aproximar el error de generalización (también llamado *out-of-sample error*), que es el error al ejecutar el modelo con nuevos casos. Es decir, cómo se comporta el modelo con instancias que no conoce [20]. Un tercer conjunto de validación se puede crear para ayudar a seleccionar la arquitectura del modelo o los hiperparámetros [3].

Para evitar gastar muchos datos entre el entrenamiento y la validación se emplea la técnica de **validación cruzada** o *cross-validation*, donde el total de datos para el entrenamiento se divide en *subsets* y se entrena con una combinación de ellos y se valida con la parte restante. Una vez seleccionados los hiperparámetros, un modelo final es evaluado con los datos para pruebas.

Por último, una vez se realiza el despliegue del modelo para su uso, por ejemplo, a través de un servicio web, el rendimiento de este se puede monitorizar para detectar fluctuaciones en los datos, predicciones inesperadas o errores. La retroalimentación recibida puede producir que la arquitectura del modelo cambie o que deba reentrenarse con los nuevos datos obtenidos, convirtiendo esto en un proceso iterativo [23].

3.3 AutoML

El desarrollo de soluciones con inteligencia artificial sigue un proceso que requiere de conocimientos del ámbito de ML (transformaciones, algoritmos o métricas) y conocimientos del dominio del problema. Adicionalmente, requiere también de imaginación y un esfuerzo a base de prueba y error porque las relaciones entre los atributos no son siempre evidentes y los modelos no se comportan igual durante el desarrollo y su despliegue a producción [3].

AutoML engloba todas las técnicas para automatizar el proceso de desarrollo de un modelo de ML. Esta automatización se puede aplicar a fases concretas del ciclo de desarrollo. Por ejemplo, la simulación o síntesis de datos es muy importante en tareas que son difíciles de experimentar en el mundo real, como es el caso de la conducción autónoma porque es muy costoso realizar experimentos en el mundo físico de atropellos o accidentes. Para este propósito concreto existen herramientas como OpenAI Gym [3], que son capaces de simular un entorno lo más parecido al real [25].

También puede aplicarse la automatización al proceso en general, como hacen soluciones de grandes proveedores como Google Cloud AutoML [4] o los *frameworks* de Microsoft integrados en Visual Studio [5]. Por ejemplo, usando estas herramientas se puede seleccionar una tabla SQL y construir un modelo de predicción en pocos *clicks*. Solo es necesario indicar la columna a predecir y las columnas que serán la entrada del modelo[3].

La comunidad ha desarrollado una mayor automatización sobre todo en el campo de la optimización de hiperparámetros aplicado a algoritmos sencillos, como árboles de

[3] Página oficial de OpenAI Gym: gym.openai.com

[4] Página oficial de Google Cloud AutoML: cloud.google.com/automl

[5] Página oficial de ML.NET Model Builder: dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet/model-builder

decisión. Sin embargo, el aprendizaje profundo ha suscitado que el foco se mueva sobre todo a la búsqueda y configuración automática de arquitecturas de redes neuronales y su optimización [25].

El beneficio mayor de emplear *AutoML* es la rápida capacidad de desarrollar y hacer operativas soluciones de ML que funcionan, por ejemplo, desplegadas en la nube. Se debe hacer balance entre la precisión que puede ofrecer un modelo entrenado mediante *AutoML* y el que puede ofrecer un modelo entrenado siguiendo el proceso manual, especialmente cuando el equipo de desarrollo no cuenta con expertos en ML [3].

3.4 MLOps

El mantenimiento de aplicaciones es la actividad del proceso de desarrollo de *software* que más recursos consume, alrededor del 60 % o 70 % del coste original de un proyecto[26]. Como hemos visto en el apartado **3.2 Ciclo de desarrollo de ML**, los modelos de ML no son una excepción en cuanto a los artefactos a mantener. Cada versión nueva de estos artefactos debe desplegarse para hacer llegar los cambios al cliente y es crítico que se haga con celeridad. ¿Qué ocurre si se pudiera automatizar la actividad de despliegue? *MLOps* es el conjunto de prácticas que nace de la intersección entre *machine learning* y DevOps, que a su vez es el conjunto de prácticas para combinar los procesos de trabajo de los equipos de desarrollo y operaciones. La aplicación de *MLOps* y *DevOps* permite reducir los costes del mantenimiento y el despliegue, asegurando la entrega continua del producto [23].

Según la finalidad del modelo será necesario un tipo de despliegue u otro. Por ejemplo, un modelo para el reconocimiento facial que se ejecuta en un móvil debe adaptarse a las especificaciones y limitaciones del dispositivo, mientras que un modelo para la predicción climática debe estar adaptado para un flujo constante de datos. Una forma de clasificar una aplicación de ML es su capacidad de aprender incrementalmente. Según Geron [20]:

- Los sistemas *offline* son aquellos que una vez entrenados, se lanzan a producción y no se reentrenan nunca más, a menos que se haga esto específicamente (utilizando de nuevo todo el conjunto de datos actualizado con las nuevas instancias) y se despliegue una nueva versión. Este entrenamiento consume mucho tiempo y recursos, por lo que puede automatizarse para hacerlo semanalmente con todo el conjunto de datos (incluidos los nuevos de esa semana) y así poder adaptarse a nuevos cambios.
- En los sistemas *online*, el sistema se entrena incrementalmente, alimentándolo con las nuevas observaciones de manera individual o en pequeños bloques (*mini-batch*). Es un paradigma útil en sistemas que reciben flujos de datos continuos o que trabajan con enormes cantidades de datos. El **ratio de aprendizaje** define cómo de rápido debe adaptarse el sistema a cambios en los datos. Un ratio alto implica que el sistema se adaptará rápidamente a los nuevos datos, pero olvidará también rápidamente los viejos. Con un ratio bajo, el sistema aprenderá más despacio y será menos sensible al ruido o datos no representativos.

En ambos tipos de sistemas se puede aplicar *MLOps* con diferentes estadios de automatización. En la figura 3.3 se muestra un ejemplo de implementación.

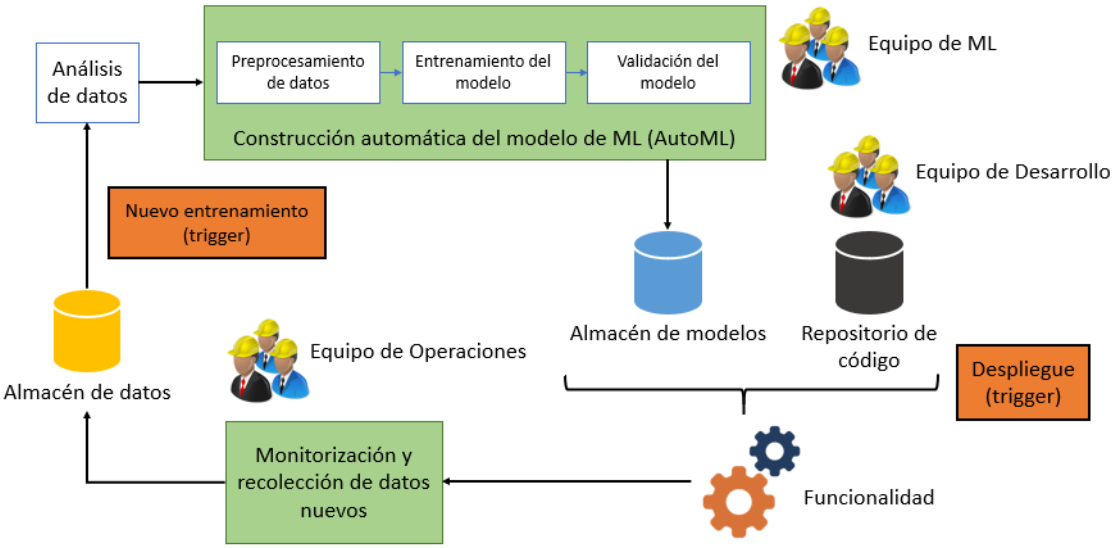

Figura 3.3: Despliegue siguiendo estrategia *MLOps* adaptado de [23].

En la iteración inicial, se construye un modelo utilizando como datos para el entrenamiento los persistidos en nuestro almacén. Usando *AutoML*, se puede automatizar el proceso de construcción del modelo, aunque debe hacerse todavía un análisis manual de los datos y hacer un esfuerzo por automatizar el preprocesamiento de los datos ya que algunas plataformas no cubren aspectos de la limpieza de datos. Una vez construido el modelo, este se persiste en un repositorio. El equipo de desarrollo será el encargado de construir una aplicación que recupere ese modelo y exponga algún tipo de servicio para ofrecer cierta funcionalidad al usuario.

Una vez desplegada, la aplicación capturará datos sobre el rendimiento del modelo en ejecución y observaciones que puedan emplearse más adelante para un reentrenamiento posterior. Estas observaciones se persistirán en el almacén de datos. La responsabilidad de estos dos pasos recae sobre el equipo de operaciones. Por último, periódicamente, cuando se detecte la degradación del modelo o de manera manual puede lanzarse un nuevo entrenamiento. Este nuevo entrenamiento puede tomar las nuevas observaciones para reentrenar el modelo existente o puede hacer un nuevo entrenamiento con todos los datos disponibles reconfigurando algunos hiperparámetros [23].

3.5 *Transfer Learning*

Las personas tenemos la capacidad de aplicar el conocimiento que tenemos de nuestra experiencia realizando una tarea para resolver fácilmente un problema nuevo con alguna similitud. Esta idea es la base del **aprendizaje por transferencia** (*transfer learning* en inglés o TL), aplicar el conocimiento aprendido de tareas previas a una tarea nueva. Por ejemplo, un modelo capaz de reconocer manzanas puede emplearse con pocos cambios para reconocer otras frutas, en lugar de crear un modelo nuevo desde el principio.

Su aplicación puede venir motivada por la falta de datos [6]. Si se cuenta con un modelo para realizar una tarea relacionada, se puede optar por una estrategia TL, independientemente de que la distribución de los datos de los dominios de las tareas sea diferente[27].

Las técnicas de TL son aplicadas por muchos proveedores de servicios, que cuentan con modelos privados entrenados para tareas genéricas que pueden ser reentrenados con los datos que aporta el cliente para una tarea más específica. También son habituales en tareas de **procesamiento de lenguaje natural**, donde encontramos repositorios [7] de modelos que representan un lenguaje (BERT [8], RoBERTa [9], T5 [10], etc.) y que pueden aprovecharse luego para tareas concretas como el análisis de sentimiento. Con modelos multilenguaje pueden incluso construirse modelos capaces de resolver una tarea en diferentes idiomas usando para el entrenamiento frases únicamente en inglés [28].

[6] Este problema también puede solucionarse mediante el **aprendizaje semisupervisado**, donde tenemos una pequeña cantidad de datos etiquetados con la salida esperada por el modelo y una mayor cantidad de datos sin etiquetar.

[7] Repositorio de modelos de HuggingFace: huggingface.co/models

[8] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding arxiv.org/abs/1810.04805

[9] RoBERTa: A Robustly Optimized BERT Pretraining Approach arxiv.org/abs/1907.11692

[10] Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer arxiv.org/abs/1910.10683

CAPÍTULO 4
MDD aplicado en el contexto de IA

Existen diferentes ámbitos de aplicación realizados hasta la fecha donde se integran estrategias basadas en modelos para el desarrollo de *software* que incluye funcionalidades con algún grado de IA. En este capítulo no se pretende realizar una revisión exhaustiva de todos ellos, simplemente se desea exponer la variedad de propuestas. En la literatura no se han encontrado revisiones sistemáticas sobre la combinación de estos dos ámbitos. Únicamente, en [29] se presenta una breve comparativa sobre diferentes herramientas MDD para los sistemas con conocimiento del contexto.

El capítulo se divide como sigue: una primera parte donde se presentan algunos artículos sobre MDD aplicado a ámbitos concretos de la IA como el procesamiento del lenguaje natural o las redes neuronales. Después, se presentan diferentes herramientas *low-code* y *no-code* para generar modelos de ML. Por último, se revisan las propuestas de los *big techs* como Amazon o Google para el desarrollo de IA.

4.1 MDD aplicado a redes neuronales y DL

4.1.1. DeepDSL para la optimización de redes de DL

En [30] y [31], Zhao y Huang presentan **DeepDSL** [1], un lenguaje específico de dominio desarrollado en el lenguaje de programación Scala para representar redes de *deep learning*. La representación de la red es luego transformada a código Java para las tareas de entrenamiento del modelo e inferencia. Así, se abstrae al usuario de plataformas y librerías como JCuda [2], que se emplean para optimizar la ejecución del programa usando tarjetas gráficas Nvidia CUDA (*Compute Unified Device Architecture*).

Las técnicas de *deep learning* requieren de grandes recursos de computación y es por ello por lo que muchas soluciones delegan en plataformas de computación paralela como servidores de GPU para obtener un mejor rendimiento. La optimización es crítica y fruto de ello nacen librerías como PyTorch, Caffe [3] o TensorFlow. La mayoría de ellas otorgan mucha flexibilidad a la hora de representar los tensores, que son las estructuras de datos de entrada de la red, pero ocultan los grafos que emplean para la red ya que no están diseñados para ser usados por el usuario, ofuscando su depuración. Además, algunas

[1] DeepDSL está disponible en el siguiente repositorio: github.com/deepdsl/deepdsl
[2] Página de JCuda: jcuda.org/
[3] Página oficial de Caffe: caffe.berkeleyvision.org/

tienen un gran número de **dependencias específicas de plataforma** [4]. Por ejemplo, al depender directa o indirectamente de librerías implementadas en C++ se necesita compilar en una plataforma específica.

DeepDSL supera estas limitaciones generando código Java, que puede ejecutarse en cualquier plataforma a través de la **Java Virtual Machine** (JVM), es fácil de entender y fácil de depurar. Asimismo, incorpora validaciones del modelo para detectar errores en el diseño de la red como dimensiones incompatibles entre tensores contiguos. Cuenta con 2 posibles modos de ejecución: el primero para optimizar la CPU y el segundo para optimizar la memoria. El segundo es el más interesante, ya que a través del análisis estático del modelo es capaz de predecir la memoria que consumirá la red en tiempo de ejecución y reorganizar la red para liberar recursos cuanto antes. Por último, el entrenamiento de la red de DL se puede parametrizar ajustando el ratio de aprendizaje, el número de iteraciones, el tamaño del lote, etc.

4.1.2. Otros metamodelos para representar redes neuronales

La aplicación de MDD en el contexto de ML puede conducir al desarrollo de nuevo estándares, menos ligados a herramientas y tecnologías específicas. En [11] Al-Azzoni presenta también un metamodelo para representar redes neuronales, donde se vislumbran las ventajas de una aproximación basada en modelos. Al igual que en DeepDSL [30], el trabajo desarrollado incluye validaciones de modelos y da un primer paso en la generación de código para entrenar y usar la red neuronal usando la herramienta Neuroph [5]. Neuroph es una herramienta para el desarrollo de redes neuronales en Java.

En la figura 4.1 se muestra el metamodelo desarrollado. Está inicialmente pensado para representar **perceptrones multicapa**, aunque se puede adaptar fácilmente para representar otro tipo de redes neuronales añadiendo invariantes. El metamodelo soporta tener una o más capas ocultas.

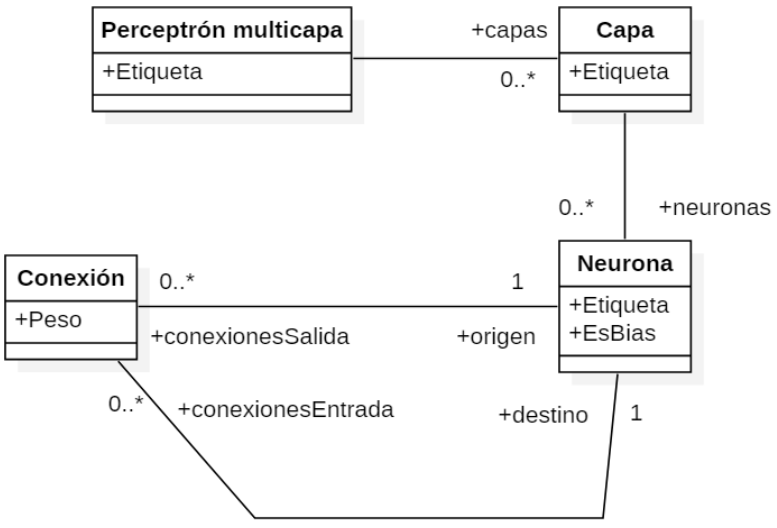

Figura 4.1: Metamodelo desarrollado por Al-Azzoni [11].

La herramienta ha sido implementada mediante el *framework* de modelado de Eclipse (EMF). Las validaciones del modelo y la generación de código también están implemen-

[4] Instalación y prerrequisitos de Caffe: caffe.berkeleyvision.org/installation

[5] Guía de inicio de Neuroph: http://neuroph.sourceforge.net/Getting%20Started%20with%20Neuroph%202.7.pdf

tadas dentro del entorno de Eclipse, a través del lenguaje de validación Epsilon [6] (EVL) y el generador basado en plantillas de Epsilon [7] (EGL) respectivamente.

En otro artículo [22], Lechevalier *et al.* proponen otro metamodelo para representar cualquier tipo de red neuronal. En la figura 4.2 se muestra el metamodelo. Ambos metamodelos vistos se centran en tener como metaentidades conceptos como las neuronas y sus conexiones y solo se distinguen en pequeños detalles como los conceptos de capas y nodos independientes o *bias*, que son nodos que representan un valor constante y permiten desplazar la función de activación a izquierda o derecha. El metamodelo en este caso ha sido desarrollado usando GME [8] (Generic Modeling Environment), un conjunto de herramientas para desarrollar entornos de desarrollo usando un DSML. Una vez realizado un modelo, la construcción y entrenamiento de la red derivada se hace usando Weka[9], una librería de ML para Java.

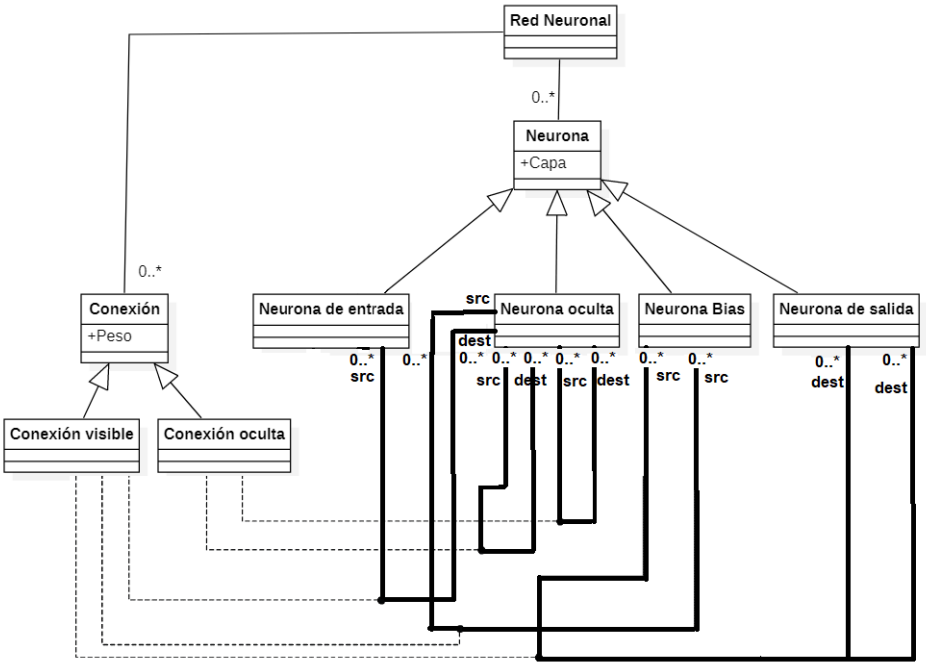

Figura 4.2: Metamodelo desarrollado por Lechevalier *et al.* [22].

En definitiva, los 3 trabajos presentan capacidades similares para representar redes neuronales y se diferencian principalmente en las herramientas sobre las que se apoyan luego para construir la red. Además, DeepDSL está más optimizado para la construcción de redes de DL de mayor dimensión.

[6] Página del Epsilon Validation Language: www.eclipse.org/epsilon/doc/evl/
[7] Página del Epsilon Generation Language: eclipse.org/epsilon/doc/egl/
[8] Página oficial de GME: isis.vanderbilt.edu/Projects/gme/
[9] Página oficial de Weka: cs.waikato.ac.nz/ml/weka/

4.2 MDD aplicado a Big Data

4.2.1. BiDaML

En [32] y [33] Khalajzadeh *et al.* presentan **BiDaML**, un conjunto de DSML para el análisis de soluciones de Big Data. La herramienta busca solucionar problemas de comunicación entre los diferentes *stakeholders* durante la toma de requisitos. Son muy diferentes los intereses que el cliente, un programador o un analista de datos pueden tener. También es diferente la manera de comunicar sus necesidades. Estas diferencias se hacen más presentes a lo largo del proceso de desarrollo conforme se pierde trazabilidad entre la toma de requisitos y las decisiones de diseño derivadas. Por tanto, es necesario capturar, mediante modelos, desde requisitos de alto nivel hasta detalles de tareas derivadas, con diferentes niveles de abstracción.

Mediante una aproximación basada en modelos se mejora el **diálogo** entre las diferentes partes, es más fácil la captura y refinamiento de requisitos y aumenta la velocidad de desarrollo. En BiDaML se presentan 5 tipos de modelos, que resumimos a grandes rasgos. El primero, se emplea para dar una visión general del proyecto y recoge parte de la lluvia de ideas inicial del proyecto. El segundo representa los diferentes procesos de curación y análisis de datos que se realizarán durante el proyecto, mientras que el tercero desgrana estos procesos en tareas más pequeñas. Los dos últimos modelos se centran en los artefactos generados. Uno de ellos se emplea para representar las fuentes de datos que se usarán y su estructura y el otro para representar los artefactos de salida, como informes, gráficos, etc.

En la figura 4.3 se muestra a la izquierda un ejemplo del primero tipo de modelos y a la derecha la notación empleada. En el modelo se separa claramente la responsabilidad de cada equipo y se vislumbran algunos detalles de implementación.

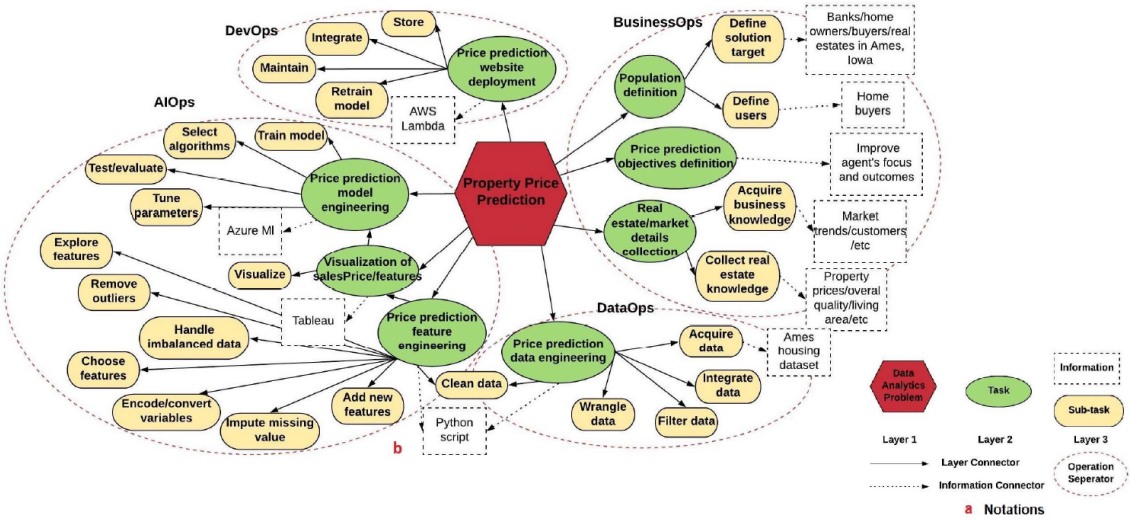

Figura 4.3: Modelo y notación para la lluvia de ideas de BiDaML [33].

4.2.2. Representación de modelos bayesianos

Breuker [34] propone una notación visual para representar modelos bayesianos. Los **modelos bayesianos** o modelos probabilísticos son grafos dirigidos donde se representa visualmente una distribución probabilística, donde los nodos son variables aleatorias y los arcos son las dependencias entre ellas. Tienen un ámbito de aplicación general en la ciencia de datos para la inferencia. La notación de la que se nutre para definir su DSML cuanta con algunas extensiones adicionales obtenidas de la literatura para añadir azúcar sintáctico. Esta notación se muestra en la figura 4.4. El azúcar sintáctico está presente, por ejemplo, en los platos, que se emplean para representar nodos y relaciones repetidas, o las puertas, para representar áreas del modelo que se incluyen solo bajo cierta condición.

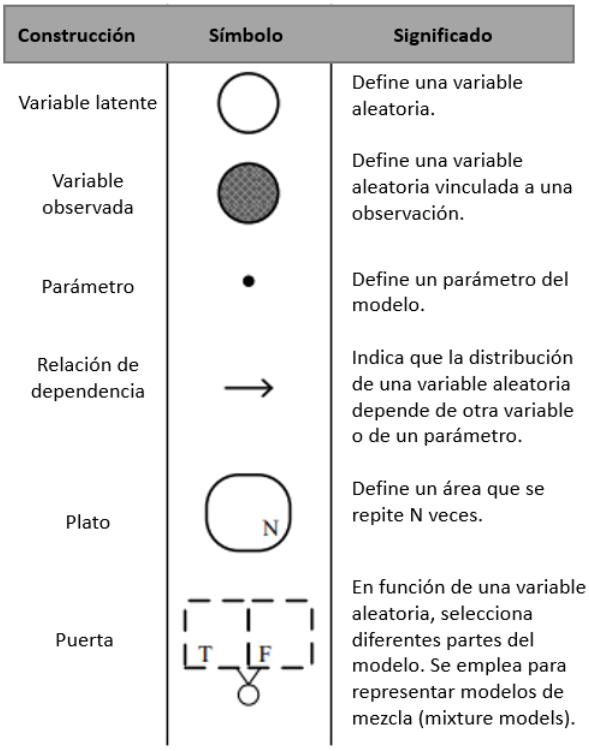

Figura 4.4: Notación para representar modelos de probabilidad [34].

Para construir su lenguaje, el autor se inspira en Infer.NET [10], una herramienta desarrollada en el entorno .NET para aplicar inferencia bayesiana en modelos en grafo. A pesar de que se define un DSML, el artículo posterga como trabajo futuro representar este lenguaje usando el SDK de Modelado de Visual Studio y la generación de código a partir de los modelos, usando la librería de Infer.NET. En este sentido, no se han encontrado trabajos relacionados que continúen su labor.

[10] Página oficial de Infer.NET: dotnet.github.io/infer

4.2.3. Big Data con MapReduce a través de modelos

MapReduce es un paradigma para el tratamiento de grandes volúmenes de datos siguiendo arquitecturas distribuidas. Se basa en dos funciones: la función *map()* procesa en paralelo las diferentes observaciones y hace un mapeo de cada entrada asignando una clave, mientras que la función *reduce()* toma después todos los valores con la misma clave y los procesa generando una salida. Rajbhoj *et al.* proponen una herramienta basada en MDD para reducir la complejidad de herramientas con este paradigma y fomentar la consistencia.

En su artículo, se presenta un metamodelo para representar este tipo de problemas y un motor para generar código para la herramienta Hadoop [11]. El metamodelo cuenta con metaentidades para representar los problemas de clasificación y *clustering*. La notación es únicamente textual y no cuenta con un diseñador para representar las tareas. Por último, concluye a través de un caso de uso que el código generado es similar al que se haría manualmente y la **velocidad de desarrollo se triplica** [35].

4.3 MDD aplicado a NLP

Hemos encontrado diferentes formas de combinar MDD y NLP. En primer lugar, en el artículo de Allala *et al.* [36] se presenta la combinación de ambos campos para la transformación de **requisitos a casos de prueba**. Los requisitos, aunque se pueden recoger en documentos estructurados siguiendo plantillas, se redactan en lenguaje natural, por lo que pueden ser a veces ambiguos, difíciles de interpretar o de mantener. En su trabajo, se presenta un metamodelo para representar tanto casos de uso como historias de usuario. A través de una transformación M2M utilizando ATL [12] (Atlas Transformation Language) se traducen los requisitos a un caso de prueba. En esta transformación, se extraen utilizando técnicas de NLP los sustantivos y verbos que aparecen, mapeando los primeros como actores en el *test* y los segundos como las acciones de la prueba. Este proceso se representa en la figura 4.5. Aunque resulta interesante, el prototipo convierte únicamente una especificación en otra, sin llegar a autogenerar estas acciones en código.

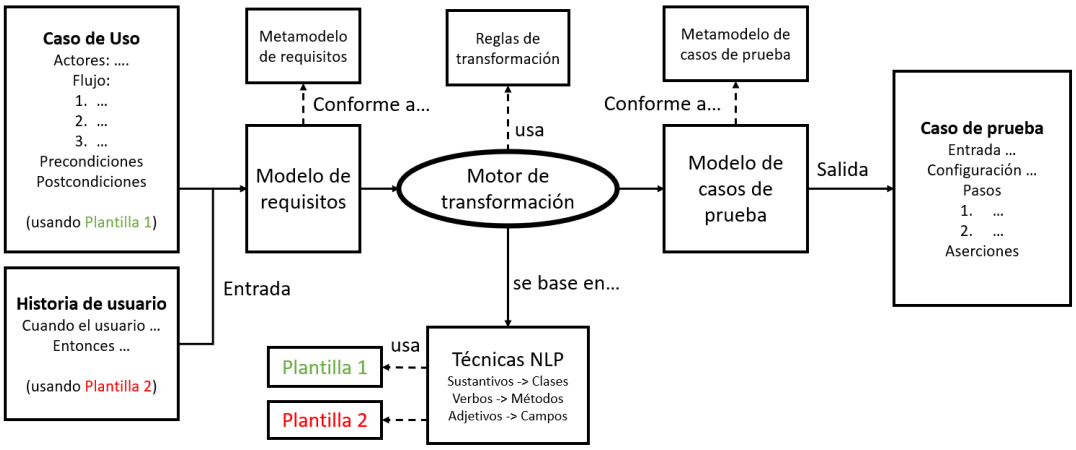

Figura 4.5: Proceso de transformación de requisitos a casos de prueba [36].

En segundo lugar, en [37] se plantea el problema de interoperabilidad entre los diferentes servicios *software* para NLP que forman un sistema. Una aplicación software real cuenta con un gran número de servicios de NLP, como las bases de conocimiento o los

[11] Página oficial de Hadoop: hadoop.apache.org/
[12] Página de ATL: eclipse.org/atl/

motores de reglas de negocio. Si el sistema sigue una arquitectura basada en microservicios, cada servicio está cargo de un equipo y posiblemente estará implementado con una tecnología diferente. El modelo de lenguaje en los que estos servicios se basan debe ser común, aunque se ejecuten en plataformas distintas. Como solución, se plantea el uso de modelos PIM para representar el dominio común e implementar diferentes transformaciones, usando el entorno de Eclipse, para generar los artefactos (un conjunto de modelos PSM) que cada servicio necesita. De esta forma, el modelo de lenguaje se representa formalmente, el nivel de abstracción aumenta y se garantiza la **consistencia** en todos los servicios.

En tercer lugar, encontramos interesante el trabajo de Burgueño *et al.* en [38], que incorpora técnicas de NLP en el proceso de desarrollo de modelos de dominio. Como saben la mayoría de los programadores, las técnicas de autocompletado están ampliamente integradas en los IDE, aumentando la velocidad de programación y previniendo errores. La herramienta desarrollada ofrece **sugerencias** durante el desarrollo de modelos de dominio. Estas sugerencias, presentadas al usuario en forma de clases, atributos y relaciones recomendadas, se obtienen a partir de dos fuentes de conocimiento. La primera fuente es general y común para todos los dominios, y la segunda es contextual al modelo, obtenida a partir de palabras relacionadas (*relatedness*) a las ya modeladas y provenientes de un corpus externo. Por ejemplo, si tuviéramos modeladas las entidades Aviador y Avión, la herramienta sugerirá una asociación entre ambas para representar que un aviador pilota un avión, al igual que sugerirá nuevas entidades como Pasajero.

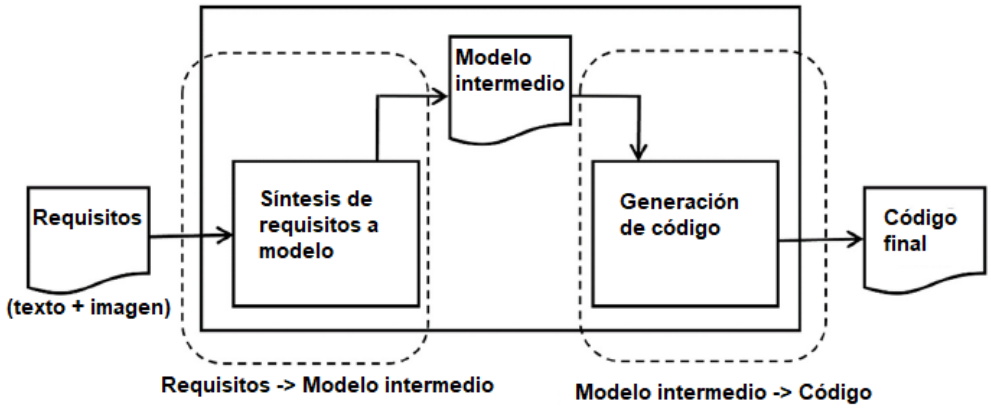

Figura 4.6: Proceso de síntesis de programas con capacidad cognitiva [39].

Por último, en [39] se realiza una revisión sistemática sobre el proceso de cognificación en la síntesis de programas. Dos aproximaciones se siguen mayormente: la aproximación **inductiva** trata de derivar el programa mediante la búsqueda de trazas o patrones, por lo que puede abordarse mediante técnicas de ML; la aproximación **deductiva** construye el programa formalizando la especificación de entrada y salida y resolviendo el problema, por ejemplo, a través de un DSML y transformaciones M2M. Subahi, el autor, propone combinar MDD, ML y NLP para desarrollar la siguiente generación de sintetizadores con capacidades cognitivas. Su propuesta se muestra en la figura 4.6. El modelo de entrada es una especificación de requisitos textual que puede incluir imágenes. En la fase de síntesis, a través de NLP se capturan las características especificadas como texto y se interpretan las imágenes mediante algoritmos de visión. La información recogida se representa en un modelo intermedio que es conforme a un DSML, que será la entrada del proceso de generación de código para autogenerar un programa, que es la salida del proceso.

4.4 MDD aplicado a sistemas *context-awareness*

La **computación pervasiva** es un paradigma con la finalidad de embeber la tecnología en el día a día de las personas de manera transparente. El sistema debe ser capaz de adaptarse a nuevas situaciones sin necesidad de la intervención del usuario. Las tareas que ejecuta las realiza de una manera no intrusiva. Los sistemas deben ser conscientes de su contexto (*context-awareness*), que engloba aspectos del usuario, el entorno, la temporalidad, etc.

Actualmente, el desarrollo de los sitemas pervasivos sigue mayormente una estrategia *ad-hoc*, forzando al desarrollador a trabajar en un nivel bajo de abstracción. Muchas veces se programa directamente sobre los dispositivos, hecho que resulta un problema para el mantenimiento ya que *hardware* y *software* evolucionan muy rápido.

Siguiendo una estrategia MDD, la productividad y calidad aumentan. A su vez, el mantenimiento es más sencillo y el sistema se adapta mejor a los cambios. **PervML** [13] es un lenguaje de modelado que permite representar sistemas conscientes del contexto con el suficiente nivel de abstracción. Los modelos de PervML son luego traducidos a código en Java OSGi [14] a través de transformaciones. También a partir de los modelos se genera una especificación OWL (*Web Ontology Language*) que permite adaptarse a los cambios del contexto. La especificación OWL es usada por algoritmos de *machine learning* para predecir las acciones que tienen que ejecutarse automáticamente, detectar patrones, etc.

PervML está desarrollado sobre EMF y GMF para ser usado dentro del entorno de Eclipse. El *framework* obliga a especificar diferentes tipos de modelos a diferentes niveles de abstracción: modelos de clases para representar los servicios, modelos de interacción para reflejar la comunicación que se da cuando se contacta con un servicio, modelos de casos de uso, de transición entre estados, etc. Esto hace que personas con diferentes roles definan modelos diferentes para la implementación de un sistema. En la figura 4.7 se observa uno de estos modelos, el modelo de localización, implementado como una extensión del modelo UML de paquetes y que representa el espacio del mundo real en el que dispositivos y usuarios se pueden encontrar [40].

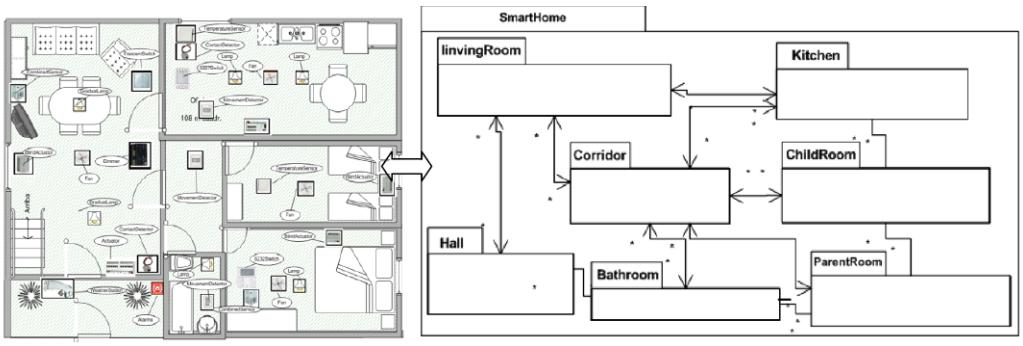

Figura 4.7: Modelo de localización de PervML [40].

En [29] se presenta un DSML llamado **AocML** para aplicaciones *context-awareness* orientadas a actividades. Un ejemplo de aplicación de este tipo es un hogar inteligente: el sistema debe ser capaz de inferir las actividades que están desarrollando los usuarios y adaptar el entorno, como controlar la temperatura o ajustar la luz cuando se está viendo una película o leyendo. El artículo contiene una comparación con otras herramientas MDD para implementar sistemas *context-awareness*, entre los que se encuentra PervML. Ambos están orientados a generar código Java y una ontología OWL, con la diferencia

[13] Página de descarga de PervML: pros2.webs.upv.es/es/pervml/modeling/pervml-language
[14] Página de OSGi: osgi.org/

de que PervML define diferentes modelos gráficos que extienden UML y AocML solo especifica un DSML textual.

4.5 Herramientas *low-code* y *no-code* para incorporar IA en aplicaciones

En esta sección vamos a revisar algunas de las herramientas comerciales que existen para incorporar inteligencia artificial siguiendo estrategias *low-code* o *no-code*. Al término de la sección haremos una comparación de todas ellas. A la mayoría de las librerías se ha llegado a partir de los artículos de Gavrilova [41], Patodi [8] y Chugh [42]. Como dice Cabot, estas herramientas no aparecen muchas veces en la literatura debido a que el término *low-code* no es un término científico en sí [16].

4.5.1. Teachable Machine

Teachable Machine [15] es una aplicación web muy sencilla destinada a resolver, por ahora, tres problemas habituales de clasificación: la detección de objetos (y personas), el reconocimiento de personas por voz y el reconocimiento de posturas corporales. Su desarrollo está apoyado por Google y hace uso de TensorFlow.js [16], la versión en JavaScript de TensorFlow. Está orientada a cualquier usuario ya que su funcionalidad para entrenar un modelo es completamente *no-code*. Después, el modelo entrenado puede exportarse usando diferentes formatos de TensorFlow, Keras o TensorFlow Lite [17], que se emplea para móviles e IoT.

Las imágenes y audios que se emplean pueden subirse directamente desde la *webcam* o usando el sistema de ficheros. La validación del modelo una vez entrenado también se puede hacer usando estas entradas. Una vez subidas, solo es necesario indicar a qué clase pertenecen para hacer la clasificación. En la figura 4.8 se muestra la interfaz de usuario de la aplicación, donde se está entrenando un modelo con 2 clases. Algunos parámetros del entrenamiento como las épocas (en inglés, *epochs*), el tamaño del lote o la tasa de aprendizaje pueden ajustarse.

En definitiva, Teachable Machine es una herramienta muy interesante para resolver los problemas que soporta por gente con pocos conocimientos de *machine learning* o para razonar sobre proyectos durante su fase de análisis. Futuros escenarios serán cubiertos conforme avance su desarrollo.

[15] Página oficial de Teachable Machine: teachablemachine.withgoogle.com/
[16] Página de TensorFlow.js: tensorflow.org/js?hl=es-419
[17] Página de TesorFlow Lite: tensorflow.org/lite?hl=es-419

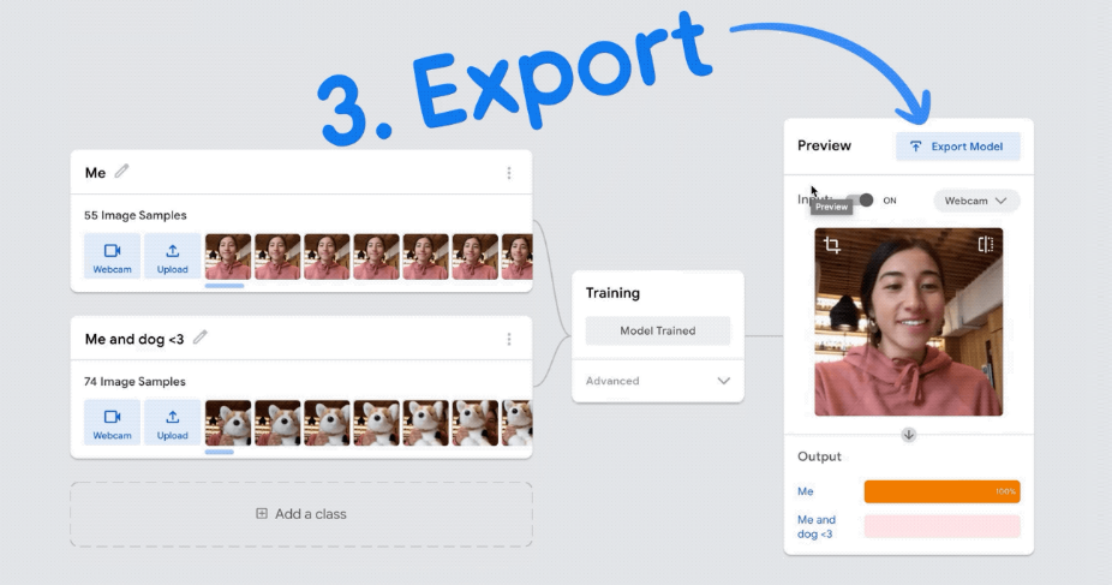

Figura 4.8: Entrenamiento de un modelo con Teachable Machine.

4.5.2. BigML

BigML [18] es una herramienta mayormente web para la construcción de modelos de ML. Las tareas que ofrece son clasificadas según si emplean un aprendizaje supervisado o no. Dentro del aprendizaje supervisado resuelve problemas de clasificación, regresión y predicciones de sucesos temporales, usando algoritmos como árboles de decisión o redes neuronales. En cuanto al aprendizaje no supervisado, resuelve problemas de *clustering*, detección de anomalías o descubrimiento de asociaciones.

La herramienta ofrece una serie de *datasets* para familiarizarse con la plataforma y permite subir fuentes de datos propias a partir de bases de datos como MySQL [19], PostgreSQL [20] o SQL Server [21]. Sin embargo, la cuenta gratuita no permite trabajar con *datasets* propios de más de 16 MB. Las cuentas gratuitas también tienen limitado a 2 el número de entrenamientos de modelos que pueden realizarse simultáneamente.

La interfaz de usuario que ofrece es muy intuitiva. Los *datasets* pueden ser explorados fácilmente y los modelos construidos se representan de manera interactiva. En la figura 4.9 se muestra un modelo de ML construido para predecir la gravedad de las heridas en un accidente. El nodo seleccionado en color verde representa una hoja del árbol donde se clasifica como de 'Gravedad Fatal' cuando la persona involucrada es un peatón con una edad entre 35 y 64 años.

Los datos pueden ser transformados fácilmente gracias a la extensibilidad de la aplicación, que ofrece su propia línea de comandos para crear scripts, BigMLer [22]. Además, los modelos pueden consumirse a través de una API REST que tiene clientes para la mayoría de lenguajes de programación: Python, Java, C#, etc.

[18] Página oficial de BigML: bigml.com
[19] Página de MySQL: mysql.com/
[20] Página de PostgreSQL: postgresql.org/
[21] Documentación de SQL Server: docs.microsoft.com/es-es/sql/sql-server/?view=sql-server-ver15
[22] Página de BigMLer: bigml.com/tools/bigmler

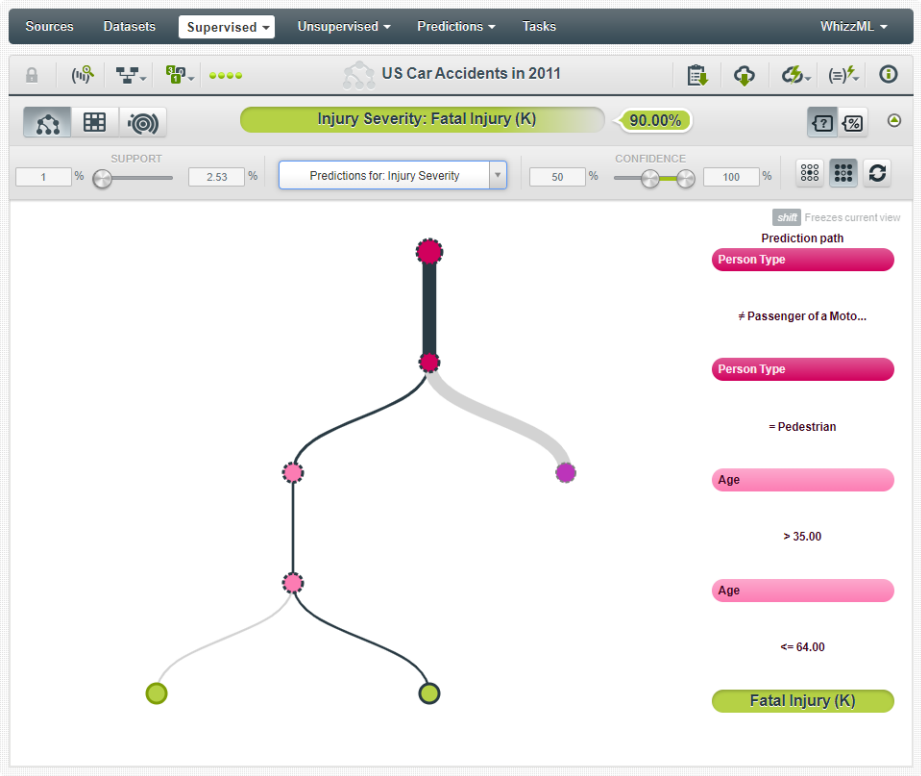

Figura 4.9: Predicción de accidentes usando BigML.

4.5.3. Runway

Runway [23] es una plataforma para la aplicación de IA orientada principalmente a diseñadores, editores y creadores de contenido en el dominio de edición de imágenes. Por ejemplo, ofrece una serie de acciones rápidas como añadir máscaras a objetos en movimiento en un vídeo o extraer a las personas de una imagen. En la figura 4.10 se muestran estas acciones rápidas en ejecución. A la izquierda, una funcionalidad para extraer personas de una imagen. A la derecha, la aplicación de una máscara basada en puntos para extraer objetos en movimiento de un vídeo.

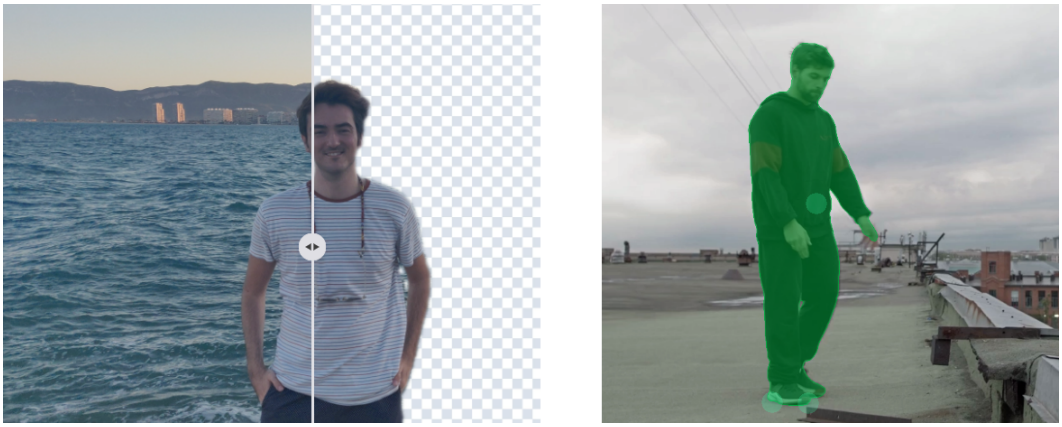

Figura 4.10: Acciones rápidas disponibles en Runway.

Runway permite crear modelos para la detección de objetos y para la generación automática de imágenes y texto. La plataforma también cuenta con la infraestructura para

[23] Página oficial de Runway: runwayml.com

desplegar en la nube un modelo e invocarlo a través de una API. Además, los modelos creados se pueden compartir y hacer accesibles a la comunidad.

Otros modelos más complejos no pueden ser customizados, pero sí pueden ejecutarse. Por ejemplo, encontramos un modelo entrenado usando guiones de películas que convierte cualquier texto que le introduzcamos en el guion de una película. También hay modelos para aplicar estilos a imágenes, que transforman la imagen que le introduzcamos a estilos de pintores como Picasso o Van Gogh. Por último, hay implementaciones para la detección de objetos de diferentes categorías, para la segmentación de imágenes o para detectar posturas humanas.

4.5.4. Obviously AI

Al igual que las herramientas presentadas anteriormente, Obviously AI [24] es una aplicación web, pero para las tareas de regresión y análisis de datos. El flujo de trabajo de la aplicación es el siguiente. Primero, se debe subir el conjunto de datos a analizar a la aplicación, ya sea a través de un fichero CSV, bases de datos u otros formatos soportados. Una vez hecho esto, podemos indicar una columna sobre la que hacer predicciones. La interfaz de usuario en la que se desarrollan estos pasos se muestra en la figura 4.11. La aplicación automáticamente hará la limpieza de los datos introducidos y encontrará el mejor modelo de predicción ejecutando diferentes algoritmos. Finalmente, podremos visualizar diferentes informes y gráficos para el análisis de datos y se podrá ejecutar el modelo de predicción introduciendo en un formulario las entradas de una nueva instancia [25].

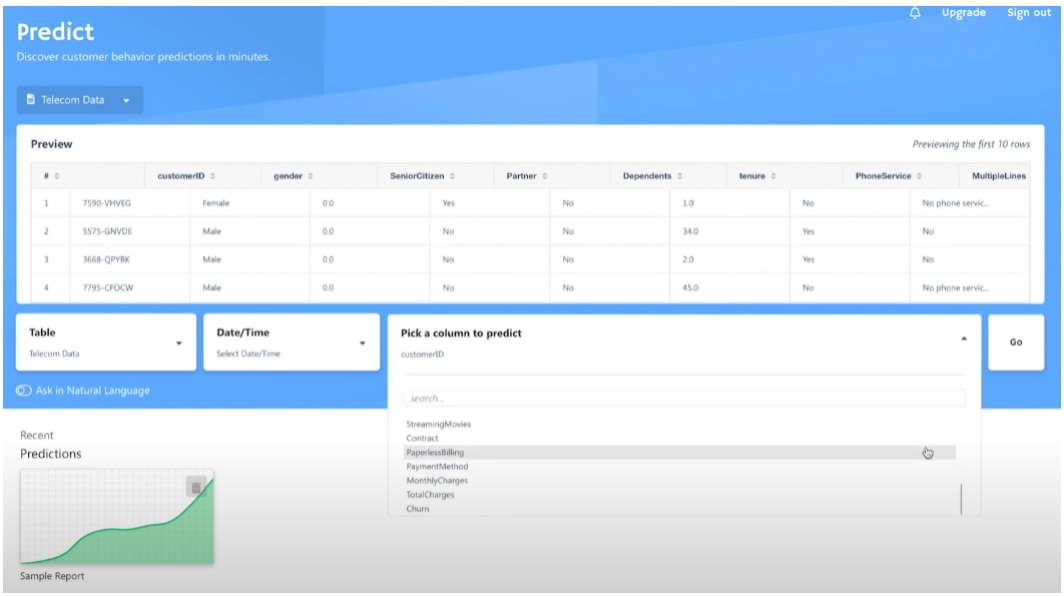

Figura 4.11: Predicción de ventas usando Obviously AI.

Una de las características más interesantes que ofrece es el procesamiento de lenguaje natural. La aplicación es capaz de responder a consultas formuladas en lenguaje natural sobre los datos. Además, el modelo entrenado puede desplegarse y consumirse a través de una API securizada.

Observamos que la aplicación está centrada en un escenario muy particular. Su intención no es resolver el mayor número de problemas que aborda la IA, solo se centra en uno

[24] Página oficial de la Obviously AI: obviously.ai
[25] Vídeo de demostración: youtube.com/watch?v=d_1pzpzEP6I

y ofrece una interfaz amigable para resolverlo y facilitar el análisis de datos a pequeños y medianos negocios.

4.5.5. Fritz AI

En las redes sociales están en auge los **filtros de realidad aumentada**, animaciones que se mezclan con la imagen capturada por la cámara del móvil. Para su desarrollo, surge Fritz AI [26], una aplicación orientada al desarrollo de modelos para dispositivos Android y iOS. Permite crear modelos para la segmentación de imágenes, la detección de objetos o la transferencia de estilos. Las librerías que utiliza por debajo están diseñadas para poder ser ejecutadas en estos dispositivos, como TensorFlow Lite.

Actualmente, Fritz AI cuenta con una serie de características en fase beta y de acceso gratuito que permiten su integración con el entorno de desarrollo de Lens Studio. Lens Studio [27] es un *software* de la compañía Snapchat, una de las pioneras en los filtros de realidad aumentada, que permite la creación y publicación de estos filtros. Utilizando Fritz AI, podemos generar un modelo y referenciarlo luego dentro de Lens Studio. En la figura 4.12 podemos ver el entrenamiento de un modelo para la transferencia de estilo. El entrenamiento ha tenido una duración de 2 horas. A la izquierda, hemos seleccionado una imagen a partir de la cual se generará el filtro, el cuadro 'Las mujeres de Argel' de Picasso. Para el entrenamiento se pueden ajustar diferentes parámetros como su duración o diferentes pesos para ajustar cómo de relevante son sus formas y colores. Una vez entrenado, a la derecha, podemos ver una visualización del modelo aplicado a diferentes imágenes.

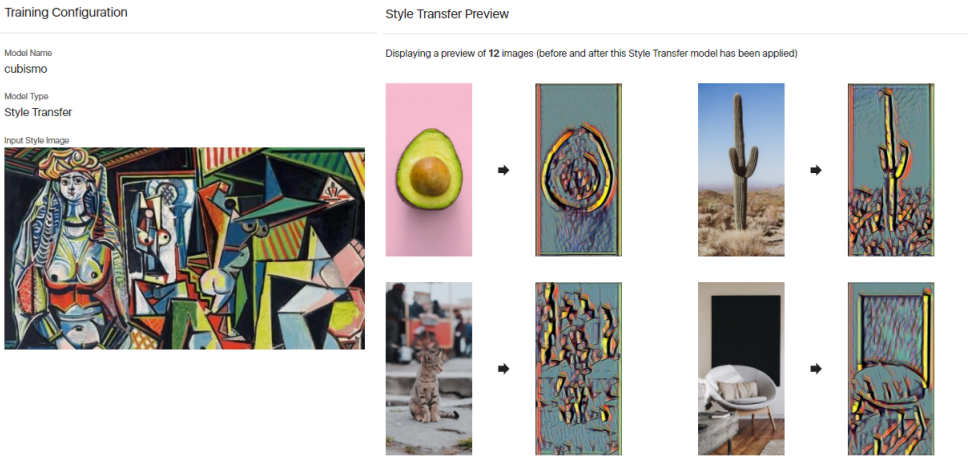

Figura 4.12: Transferencia de estilos con Fritz AI.

A pesar de que el ámbito de esta aplicación es reducido y derivado principalmente del ocio, gracias a seguir una estrategia *no-code* puede ser empleada por cualquier persona sin necesidad de conocimientos especiales. El despliegue del modelo en las redes sociales también es muy sencillo y es solo necesario descargar el modelo a través del explorador de filtros integrado en la *store* de aplicaciones como Instagram.

[26] Página oficial de Fritz AI: fritz.ai/
[27] Página de Lens Studio: lensstudio.snapchat.com/

4.5.6. Comparación de herramientas

En [43] se comparan 8 plataformas para el desarrollo *low-code*. El propósito de estas plataformas es incluir al ciudadano de a pie en el desarrollo de sistemas *software*. Sin embargo, existen tantas plataformas para este propósito general que es difícil decidir cual emplear. Sahay *et al.* describen las propiedades que todas estas plataformas tienen y hacen una comparación sistemática, revisando algunas de sus características como la interoperabilidad, reusabilidad, escalabilidad, etc.

En esta sección vamos a hacer una comparación similar pero aplicada a las plataformas que hemos descrito para desarrollar IA. La comparación la haremos de acuerdo con estas características:

- **Características generales**: evaluaremos si son *low-code* o *no-code* y el coste de emplearlas.
- **Escenarios cubiertos**: se resumen los diferentes escenarios que cubren: regresión de valores, clasificación, detección de objetos, etc.
- **Plataformas y lenguajes**: se describe si los modelos que generan son multiplataforma o están ligados a una plataforma en particular y si el entorno de desarrollo es una plataforma web o se realiza el entrenamiento en local.
- **Funcionalidades**: se resumen algunas facilidades como si ofrece la posibilidad de desplegar el modelo entrenado usando algún tipo de servicio, si cuenta con modelos preentrenados como ejemplos o punto de inicio y si ofrece herramientas para visualizar los datos del entrenamiento.

El resultado de la comparación se resume en la tabla 4.1. La mayoría de herramientas *low-code* se caracterizan por ofrecer una estrategia *no-code* donde se pueden insertar pequeñas porciones de código para personalizar la solución generada. Tanto Runway como Obviously AI ofrecen un tipo de despliegue similar: el proveedor del servicio *hostea* nuestro modelo y nuestra aplicación puede ejecutar ese modelo a través de una API HTTP. Teachable Machine permite exportar el modelo entrenado en diferentes formatos de TensorFlow y proporciona código de ejemplo, en JavaScript y Java, que podemos añadir en nuestra aplicación para usar el modelo. Por último, Fritz AI facilita el despliegue a través del entorno de desarrollo de Lens Studio.

Característica	TeachableMachine	BigML	Runway	Obviously AI	Fritz AI
Generales					
Tipo	No Code	Low Code	No Code	No Code	No Code
Coste	Gratuito	Mayormente, gratuito	Mayormente, de pago	De pago	Mayormente, gratuito
Suscripciones		Personal: desde 30 €/mes Organización: dese 55 €/mes	Personal: desde 35 €/mes	Personal: desde 65 €/mes	Pago por uso
Escenarios cubiertos					
Regresión		X		X	
Clasificación	X	X			
Clustering		X		X	
NLP			X		
Detección de objetos	X	X	X		X
Reconocimiento de voz	X	X			
Realidad aumentada			X		X
Plataformas y lenguajes					
Modelos entrenados específicos de plataforma					Android/iOS
Modelos entrenados específicos de librería	TensorFlow, Keras, TensorFlow Lite, TensorFlow.js				PyTorch, TensorFow, SnapML
Uso de formatos o lenguajes estándares		Sí, PMML y JSON PML			Sí, ONNX[28]
Entorno de desarrollo	Web	Web	Web	Web	Web
Funcionalidades					
Visualización de datos		X		X	
Ofrece modelos preentenados		X		X	X
Consumo del modelo como API		X	X	X	
Facilita el despliegue		X	X	X	X

Tabla 4.1: Comparación de herramientas *low-code* y *no-code* para el desarrollo de ML.

En las herramientas vistas no vemos una estrategia clara basada en modelos. El término *low-code* es muy genérico. Es cierto que permiten llevar a cabo tareas interesantes con poco o nada de código. Muchas veces, el código necesario es solo el que hace falta para integrar el modelo generado en nuestra aplicación. Sin embargo, es evidente que detrás de las herramientas no existe un metamodelo y no se está modelando en el dominio del problema (en este caso, IA) de manera gráfica o textual. En definitiva, no se construye un artefacto o modelo en un formato exportable, como es XML. Más bien, se sigue algún tipo de *wizard* que nos asiste en la resolución del problema o bien tenemos un entorno sencillo donde podemos cargar datos y construir luego diferentes modelos de ML a partir de ellos.

4.6 La propuesta de grandes proveedores de servicios

El término **Machine Learning as a Service** (MLaaS) se aplica para definir a las plataformas en la nube que ofrecen capacidades de preprocesamiento de datos, entrenamiento de modelos, evaluación de modelos y ejecución de estos. Las aplicaciones pueden gestionar la infraestructura relacionada y ofrecer una API REST para el consumo del modelo entrenado [44].

Estas soluciones, asociadas con grandes proveedores, no requieren un gran conocimiento de lenguajes de programación y permiten al usuario, a través de simples interacciones como el *drag-and-drop* de entidades y tareas, construir y desplegar modelos de ML. [33] Las capacidades que ofrecen son muy similares, aunque algunas tienen un menor nivel de madurez. Normalmente, estos proveedores cuentan con una plataforma para MLaaS que ofrecen modelos entrenados que pueden reentrenarse mediante TL. También tienen capacidad de hacer modelos siguiendo estrategias de AutoML y ofrecen una API con servicios concretos como el procesamiento de vídeos, imágenes o texto [44]. Por último, es común contar con algunas características MLOps para automatizar tareas de infraestructura. Vamos a repasar en concreto las propuestas de Microsoft, Google, Amazon e IBM. Si se desea consultar un artículo con más profundidad, recomendamos el siguiente[44].

4.6.1. Azure AI, Azure Cognitive Services y ML.NET

Azure AI [29] es la plataforma que ofrece Microsoft. Está compuesta por diferentes componentes, entre los que destaca Azure ML. Azure ML [30] está orientada tanto a usuarios noveles como a científicos de datos. Cuenta con hasta 100 algoritmos diferentes para realizar tareas de clasificación, detección de anomalías o *clustering*. Ofrece también un diseñador gráfico, Azure ML Designer [31], que a través de interacciones *drag-and-drop* permite visualizar y transformar los datos, construir un modelo y desplegarlo siguiendo una estrategia *low-code* (figura 4.13). Una característica diferenciadora del resto es Azure AI Gallery [32], una plataforma donde la comunidad puede compartir soluciones de ML.

En cuanto a las capacidades de su API, Microsoft es de las que más funcionalidades ofrece a través de Azure Cognitive Services [33]. Entre las capacidades compartidas con sus competidores encontramos: traducción de textos, *text-to-speech* (TTS), *speech-to-text* (STT), reconocimiento de entidades en vídeo, moderación de contenido, etc. Queremos destacar la API de LUIS [34] (Language Understanding Intelligent Service), que se emplea para analizar la intención del usuario al hablar, que es transcrita a comandos para implementar luego un *chatbot*.

Por último, ML.NET [35] es una librería multiplataforma para la aplicación de ML por parte de los desarrolladores .NET. La librería puede emplearse en la nube a través de los servicios de Azure o dentro de Visual Studio, donde destaca la extensión de ML.NET

[29] Página de Azure AI: azure.microsoft.com/es-es/overview/ai-platform/
[30] Página de Azure ML: azure.microsoft.com/es-es/services/machine-learning/
[31] Página de Azure ML Designer: azure.microsoft.com/es-es/services/machine-learning/designer/
[32] Página de Azure AI Gallery: gallery.azure.ai/
[33] Página de Azure Cognitive Services: azure.microsoft.com/es-es/services/cognitive-services/
[34] Página oficial de LUIS: azure.microsoft.com/es-es/services/cognitive-services/language-understanding-intelligent-service/
[35] Repositorio de código de ML.NET: github.com/dotnet/machinelearning

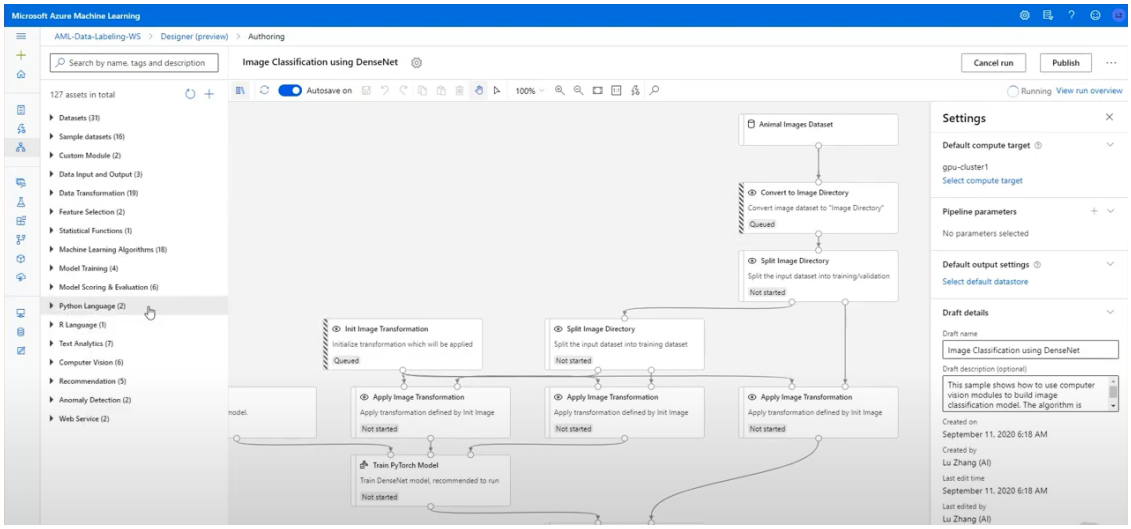

Figura 4.13: Uso de Azure ML Designer para clasificación de imágenes.

Model Builder [36] para realizar una aproximación AutoML. Los escenarios por ahora cubiertos por ML.NET Model Builder son: clasificación de textos, predicción de valores, clasificación de imágenes, recomendación y detección de objetos.

4.6.2. Google AI y Google Cloud ML

En la mayoría de las plataformas se observa un esfuerzo por unificar su oferta de MLaaS en una única herramienta web que permita la combinación de sus capacidades. Este esfuerzo no se observa tanto en Google ya que una librería como TensorFlow, desarrollada por el propio Google, no se integra en la misma plataforma igual de bien que otras capacidades como Google Cloud AutoML.

Vertex AI [37] es la plataforma anunciada en Mayo de 2021 para solucionar este problema. Vertex AI cuenta con funcionalidades de AutoML, permite crear *pipelines* de ML y permite emplear Vizier [38], un optimizador de hiperparámetros en redes de DL.

Aunque hay algunas tareas que su versión de AutoML no soportan, como la detección de anomolías, se posiciona por delante de sus competidores en tareas de NLP. Por ejemplo, su API es capaz de reconocer más de 120 idiomas y de traducir a más de 100. Para tareas como STT, permite al usuario dar un contexto de la conversación para ofrecer un mejor servicio. Es decir, si se indica un contexto de un bar, entenderá la palabra copa como una bebida y no como el conjunto de ramas de un árbol. En cuanto al procesamiento de vídeo, su API soporta el seguimiento de objetos o detectar contenido explícito, pero otras como detectar rostros o reconocer actividades y poses corporales están todavía en fase beta.

[36] Documentación de Model Builder: docs.microsoft.com/es-es/dotnet/machine-learning/automate-training-with-model-builder

[37] Página de Vertex AI: cloud.google.com/vertex-ai

[38] Documentación de Vizier: cloud.google.com/ai-platform/optimizer/docs/overview

4.6.3. ML en AWS y Amazon SageMaker

Amazon cuenta con 2 plataformas para ML: Amazon ML [39] y SageMaker [40], la más nueva y superior. Hacia la segunda plataforma se desea encaminar a los nuevos usuarios de estos servicios.

Amazon ML ofrece un abanico similar a Microsoft en cuanto a tareas soportadas y AutoML. SageMaker está más orientado a un usuario experimentado, ofreciendo un IDE para el desarrollo de algoritmos (SageMaker Studio [41]), que permite depurar los artefactos construidos o explorar datos a través de notebooks como Jupyter [42]. En la figura 4.14 se observa la complejidad de la interfaz de usuario de SageMaker Studio.

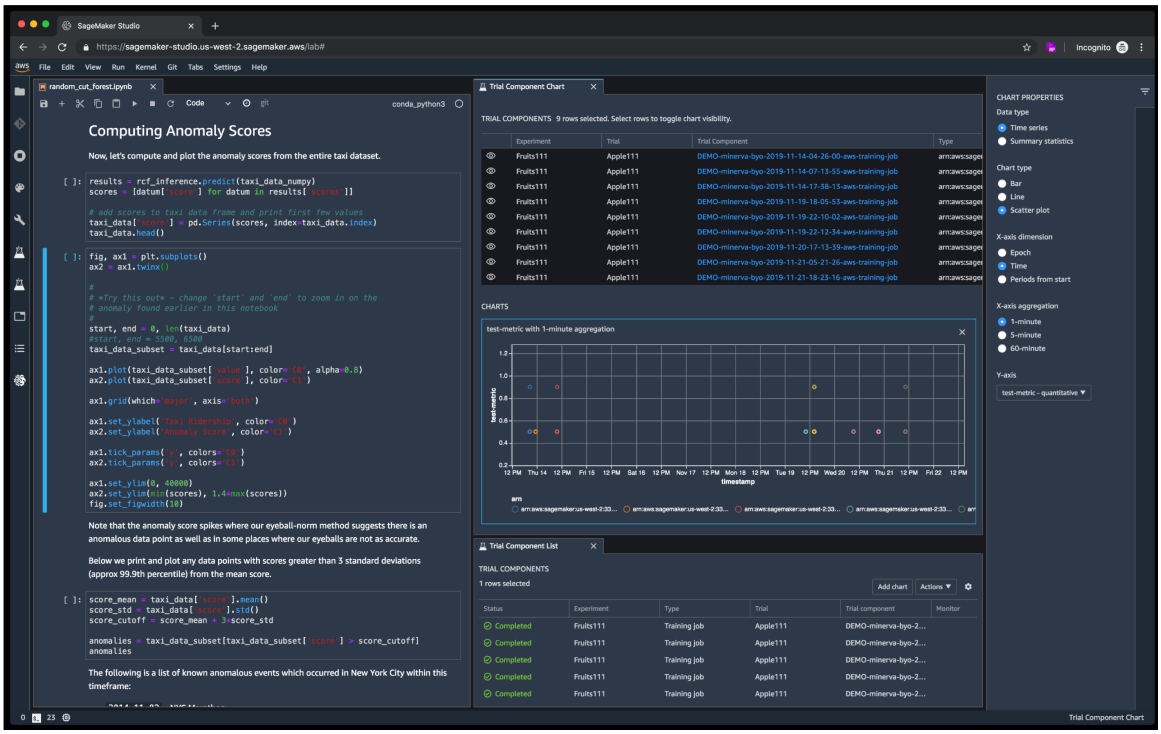

Figura 4.14: Interfaz de usuario de SageMaker Studio.

En cuanto a las APIs ofrecidas, puede resultar confuso porque algunas superponen sus capacidades. Por ejemplo, Amazon Lex [43] es la herramienta diseñada para construir *bots* textuales y por voz. Soporta el reconocimiento del habla al igual que Amazon Transcribe [44], otra herramienta pero orientada solo al STT y con mayor potencia para aislar al usuario de ruidos del contexto u otros interlocutores. Las capacidades de reconocimiento de vídeo e imágenes se realiza a través de Amazon Rekognition [45]. Como curiosidad, permite reconocer sentimientos en las expresiones faciales y se ha adaptado para reconocer personas que llevan equipos de protección personales (EPI) como mascarillas o cubiertas para la cabeza.

[39] Servicios de ML en AWS: aws.amazon.com/es/machine-learning/
[40] Página de Amazon SageMaker: aws.amazon.com/es/sagemaker/
[41] Página de SageMaker Studio: aws.amazon.com/es/sagemaker/studio/
[42] Página del proyecto Jupyter: jupyter.org/
[43] Página de Amazon Lex: aws.amazon.com/es/lex/
[44] Página de Amazon Transcribe: aws.amazon.com/es/transcribe/
[45] Página de Amazon Recognition: aws.amazon.com/es/rekognition/

4.6.4. Watson Studio y Watson Visual Recognition

La plataforma que ofrece IBM se llama Watson Studio [46] y es, sin duda, la menos desarrollada de todas las vistas. Por ejemplo, para las tareas de STT y TTS solo están soportados 9 idiomas. Otras tareas corrientes como el *clustering* o la detección de anomalías no están soportadas en su plataforma y siguiendo una estrategia AutoML solo se soportan problemas de clasificación y regresión, empleando para ello un bajo número de algoritmos.

La API para el tratamiento de imágenes, Watson Visual Recognition [47], expone funcionalidades básicas como reconocimiento facial o de objetos, pero únicamente en imágenes y no en vídeos. Esto es debido a que está a la espera de unificarse con su servicio para el tratamiento de redes neuronales. Quizás su herramienta más divulgada por su potencia sea Watson Assistant [48], para la construcción de *bots* en *chats*.

4.7 Crítica al estado del arte

Cada dominio tiene sus peculiaridades, lo que conduce a la comunidad a generar librerías con algoritmos de ML lo más configurables posible para que los usuarios finales puedan enfocarse en tener un corpus de datos correcto y amplio [34, 20]. Sin embargo, sigue siendo necesario dar un paso de la especificación imperativa típica de un problema de ML **hacia una especificación más declarativa**, como hemos visto en la mayoría de las propuestas.

En **4.1.1 DeepDSL para la optimización de redes de DL** se presenta DeepDSL. Esta herramienta, a pesar de resolver un gran número de limitaciones del contexto actual de DL, creemos que no tiene todavía la **suficiente madurez**. Primero, porque no ofrece una notación visual, solo aporta un nuevo lenguaje textual que nos abstrae de las librerías que utilizará el código generado, pero sigue siendo necesario un importante conocimiento del dominio de ML. Segundo, la extensibilidad de la herramienta se realiza modificando el código autogenerado, algo que algunos autores consideraran una mala praxis. Tercero y último, la herramienta no ha sido modificada desde 2018 y no mantener un *software* en un entorno tan cambiante como el de la IA hace sospechar que no se está trabajando en ella [30, 31].

En cuanto al trabajo realizado por Al-Azzoni en [11] para representar redes neuronales, este es muy completo e interesante. Sin embargo, creemos que le falta una notación visual para representar los modelos, ya que el XML que lo representa es poco eficiente como mecanismo de comunicación entre los interesados de un proyecto. Como referencia puede tomar el editor gráfico que proporciona Neuroph y se muestra en la figura 4.15. El editor de Neuroph no es agnóstico de plataforma mientras que la herramienta propuesta por Al-Azzoni sí lo es. Además, se plantea como líneas de trabajo futuro la generación de código para Python y R, lo que demuestra que el metamodelo no está ligado a una herramienta particular como es Neuroph.

[46] Página de Watson Studio: ibm.com/es-es/cloud/watson-studio
[47] Página de Watson Visual Recognition: ibm.com/es-es/cloud/watson-visual-recognition
[48] Página de Watson Assistant: ibm.com/es-es/cloud/watson-assistant

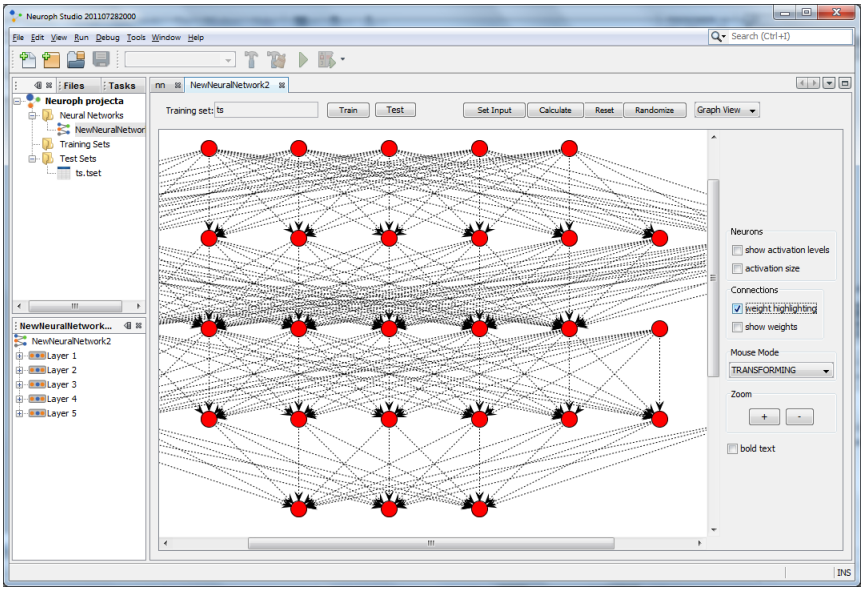

Figura 4.15: UI de Neuroph Studio.

Sobre el desarrollo de la herramienta BiDaML, a pesar de ser muy interesante el problema que abordan, los autores no sugieren ninguna forma de realizar transformaciones de modelos, por lo que los modelos realizados se emplearán como mera documentación. Como opinión personal, una herramienta así tiene poco recorrido en muchas empresas donde los equipos dedicados a la explotación de datos no requieren un nivel de organización tan elaborado al estar formados por un pequeño número de personas.

Respecto a las propuestas de grandes proveedores, ofrecen una gran variedad de servicios, tantos que puede resultar confuso cuál se debe usar para resolver cierto problema. Eso, sumado a interfaces de usuario complejas o el esfuerzo por soportar su extensión con cuantos más *frameworks* de ML mejor (por ejemplo, Keras, TensorFlow, PyTorch, etc.) obliga a pensar que estas herramientas están pensadas para ser usadas por personas con el suficiente conocimiento IA. Caso aparte son las API de servicios: ofrecen una gran variedad de parámetros y la complejidad se mueve a las aplicaciones que las consumen para procesar la salida que devuelven. Estas API resultan adecuadas para alguien con conocimientos de programación y no de IA, pero no lo recomendaríamos para personas sin experiencia con la informática.

Tareas como el preprocesamiento de datos tampoco están del todo cubiertas. La entrada a muchas de las herramientas es un fichero CSV o una tabla SQL y la construcción de este *dataset* la debe realizar el usuario, que posiblemente deba construirlo consultando diferentes tablas o bases de datos. Con Azure ML Designer se puede diseñar la *pipeline* que realiza esta transformación, pero vemos el proceso algo tedioso para un usuario con pocos conocimientos de ML [49]. Por norma general, tareas de preprocesamiento sencillas como transformaciones de valores categóricos, normalizaciones o sustitución de valores faltantes sí pueden realizarse en herramientas como Obviosly AI [50] o BigML [51].

[49] Transformación de datos en el diseñador de Azure Machine Learning: docs.microsoft.com/es-es/azure/machine-learning/how-to-designer-transform-data

[50] 4 Easy Ways to Improve Your Machine Learning Model: obviously.ai/post/4-easy-ways-to-improve-your-machine-learning-model

[51] Galería de scripts para la limpieza de datos en BigML: bigml.com/gallery/scripts/data_transformation

En definitiva, los DSML propuestos en la literatura son muy interesantes, pero o bien están muy ligados a una estructura como las redes neuronales y hacen una representación muy similar a ellas o bien se quedan como una representación textual. Por otro lado, tanto en las herramientas de *low-code* como en las propuestas de los grandes proveedores se observa una propuesta alejada del MDD tradicional. En todas las herramientas se evidencia que estas no se incluyen dentro de una plataforma para el desarrollo de un producto *software* completo. En otras palabras, las herramientas construyen un modelo de ML que luego un desarrollador debe incorporar a la aplicación. Por tanto, podemos concluir que es necesario ofrecer una propuesta más cercana a MDD, con una notación visual y fácil de entender e integrada dentro de una plataforma para la creación de aplicaciones *software*.

CAPÍTULO 5
Análisis del caso práctico

A continuación, se presenta el entorno en el que se desarrollará el nuevo DSML de IA. Los requisitos que este lenguaje tendrá también serán presentados. Por último, se plantearán dos casos prácticos para validar los escenarios que se pueden modelar.

5.1 Contexto de desarrollo

Como hemos mencionado en la **1.1 Motivación**, el presente trabajo se realiza en colaboración con una PYME que comercializa un ERP dirigido al sector socio-sanitario. La herramienta que vamos a desarrollar estará integrada con algunas existentes que vamos a describir a continuación. El *stack* tecnológico que usaremos también vendrá determinado por el código existente, para mantener la consistencia del sistema y facilitar su interoperabilidad.

En la empresa se apuesta por el uso de estrategias MDD para el desarrollo de su nuevo producto, que seguirá una arquitectura basada en microservicios. En [45] se da una definición de este tipo de arquitecturas y se expone una arquitectura interna del microservicio similar a la usada por la empresa. En la figura 5.1 se muestran algunos de ellos para la generación de informes, aplicar seguridad o recolectar datos de telemetría. Se observa la consistencia entre ellos gracias a que son autogenerados usando la misma herramienta y por tanto tienen la misma **arquitectura interna**, formada por las capas de contratos, aplicación y persistencia, entre otras. Las flechas representan la interacción que entre ellos existe.

Estos microservicios, separados siguiendo los principios del desarrollo guiado por el dominio (en inglés, *Domain-Driven Development* o DDD), son autogenerados a partir de diferentes modelos que representan entidades, acciones, pruebas y formularios. No todo el código del microservicio es autogenerado. Por ejemplo, para las acciones modeladas se genera toda la infraestructura necesaria para ser invocadas, pero la lógica de la acción debe implementarla un programador.

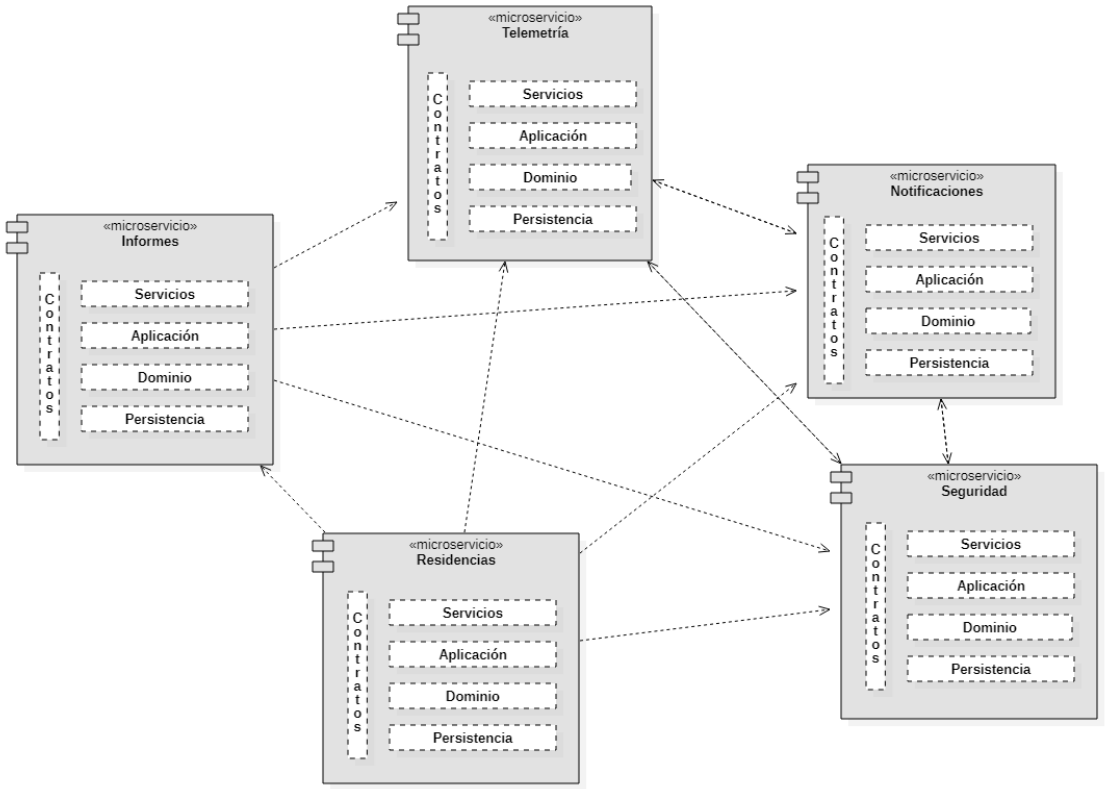

Figura 5.1: Diagrama de componentes con algunos microservicios.

En cuanto a la tecnología empleada, los microservicios son generados en C# y utilizan .NET Core [1], mientras que los formularios se generan en Xamarin [2], un *framework* multiplataforma que permite reutilizar las vistas entre diferentes plataformas móviles y *desktop*. Para el modelado se emplean las DSL Tools de Visual Studio (VS).

En las siguientes secciones presentaremos las capacidades que ofrecen las DSL Tools y las herramientas existentes brevemente.

5.1.1. DSL Tools

Las DSL Tools son una parte del SDK (kit de desarrollo de *software*) que ofrece Visual Studio desde su versión 2005 para el desarrollo de DSML. Son una pieza fundamental en la apuesta de Microsoft por las factorías de *software*, que integra el uso de patrones y librerías comunes con lenguajes de dominio para generar componentes concretos.

La creación de una DSL Tool se realiza a través de un *wizard* integrado en VS. Se pueden emplear diferentes plantillas para crear directamente un DSML que represente clases o *workflows*. Como resultado de este *wizard*, se creará una solución con 2 proyectos: el proyecto Dsl, donde se ubica el metamodelo del lenguaje, y el proyecto DslPackage, que se encarga de empaquetar el anterior para su despliegue como una extensión en Visual Studio a través de un fichero .vsix [46].

El metamodelo se edita a través de un editor gráfico, esquematizado en la figura 5.2. La parte central del editor está divida en dos por una línea vertical. En la parte izquierda se muestran las metaentidades y las relaciones entre ellas. En la parte derecha se define como estas se van a renderizar gráficamente en los modelos como formas y conectores.

[1] Documentación de ASP.NET: docs.microsoft.com/es-es/aspnet/core/?view=aspnetcore-5.0
[2] ¿Qué es Xamarin? docs.microsoft.com/es-es/xamarin/get-started/what-is-xamarin

Las metaentidades de la izquierda y los elementos gráficos de la derecha se relacionan a través de *mappings* [3]. Para la representación gráfica están disponibles un conjunto de formas como son formas con compartimentos (útiles para representar diagramas de clases), formas geométricas (por ejemplo, para representar formularios de UI) o piscinas (si se desea hacer un lenguaje para representar flujos de trabajo) [4].

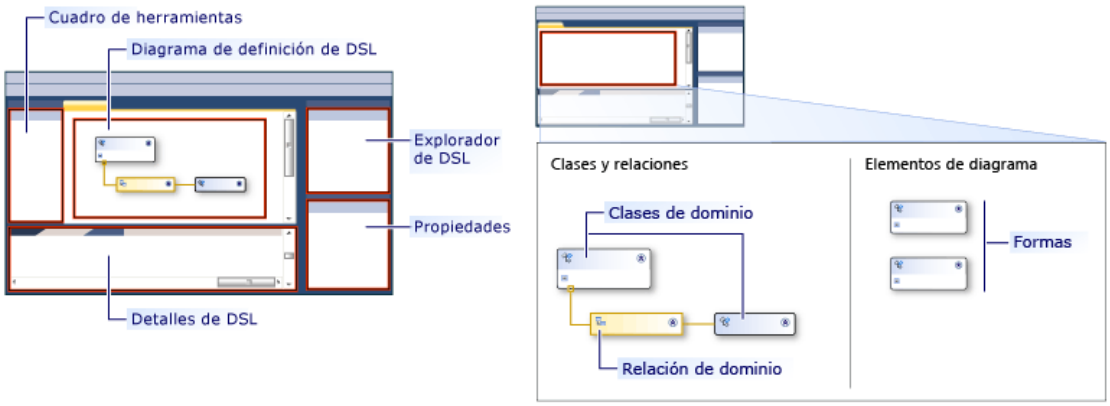

Figura 5.2: Esquema del editor de metamodelos de las DSL Tools.

A través del explorador del DSL se pueden editar comportamientos más avanzados. Por ejemplo, se pueden editar aspectos relacionados con la serialización del modelo en XML, los elementos que aparecerán en la *toolbox* de modelado para hacer *drag-and-drop* o las directivas que marcan cómo se combina un elemento cuando se deja caer sobre otro.

Como es lógico, las DSL Tools tienen integrado un motor de validaciones que permite comprobar, a través de código, que la definición de un modelo es correcta. También cuenta con un motor de plantillas, que permite a través de plantillas T4 [5] explotar un modelo para generar código a partir de él.

[3]Información general sobre la UI de las DSL Tools docs.microsoft.com/en-us/visualstudio/modeling/overview-of-the-domain-specific-language-tools-user-interface

[4]Definición de formas y conectores: docs.microsoft.com/en-us/visualstudio/modeling/defining-shapes-and-connectors

[5]Generación de código y plantillas de texto T4: docs.microsoft.com/es-es/visualstudio/modeling/code-generation-and-t4-text-templates?view=vs-2019

5.1.2. Herramientas MDD en uso

Actualmente, en nuestra PYME contamos con los siguientes DSML creados utilizando las DSL Tools:

DSML de Dominio

Permite modelar las **entidades** de un microservicio, que serán aquellas que se persisten en base de datos. Las entidades están relacionadas entre ellas a través de **asociaciones**, con su respectiva cardinalidad, y cadenas de **herencia**. Como se muestra en la figura 5.3, las entidades están formadas por **campos**, que tienen asociado un tipo (*string*, *DateTime*, *int*, etc.) y una serie de propiedades para indicar si estos son requeridos, si son únicos, sus valores máximos, etcétera. Las entidades pueden definir validaciones y campos calculados. Las cajas amarillas representan **enumerados**, campos de la entidad con valores prefijados. En el ejemplo, se utiliza un enumerado para representar el género del usuario, que solo acepta los valores hombre o mujer.

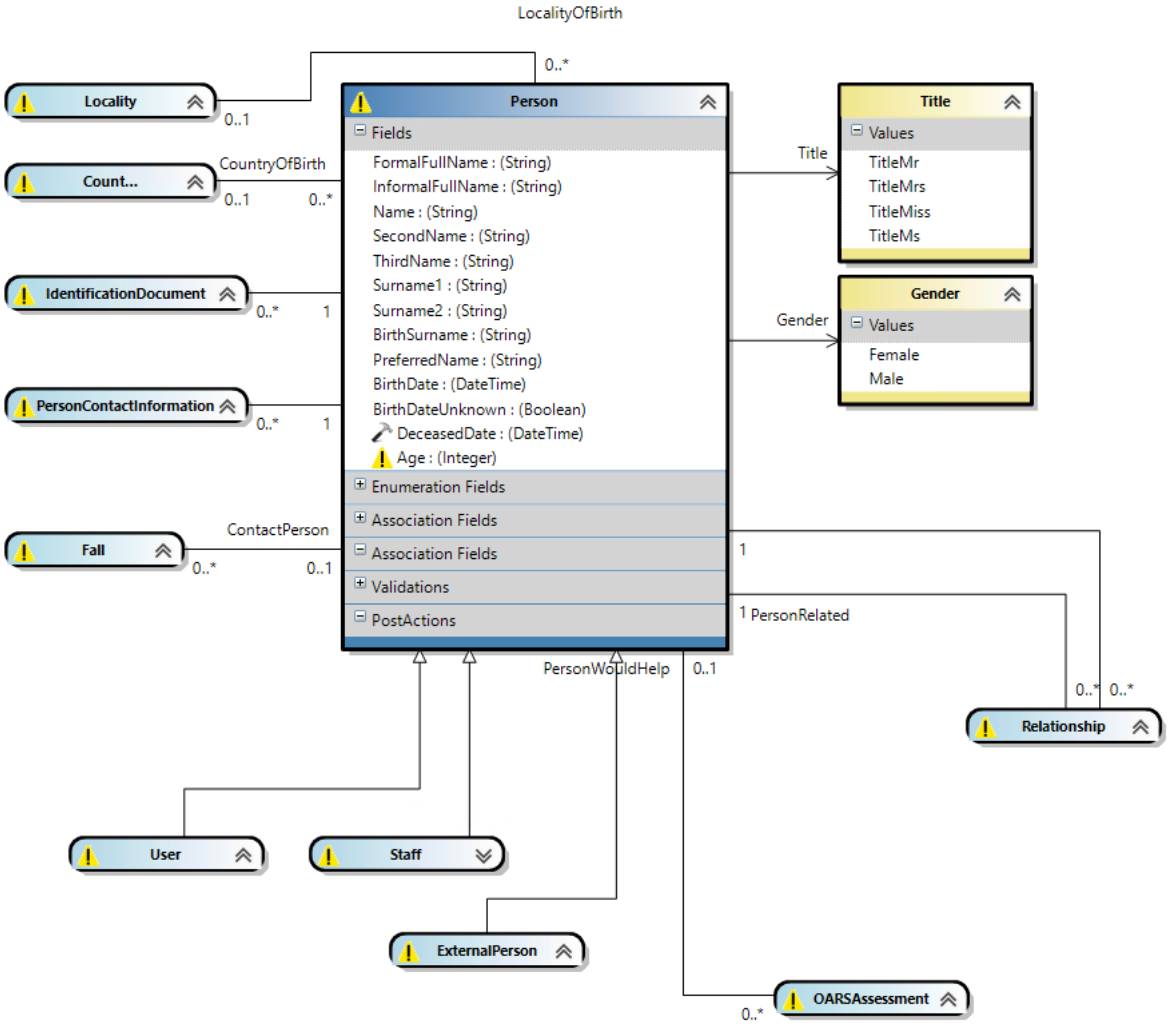

Figura 5.3: Modelo de dominio de la entidad Persona.

DSML de Aplicación

Esta herramienta permite modelar acciones y DTO. Los **objetos para la transferencia de datos** (DTO) se utilizan para desacoplar los servicios expuestos de la representación interna que da el sistema a sus entidades, ocultando aquellas propiedades que no necesitan ser transferidas. También pueden cambiar el formato de algunas propiedades para que sea más cómodo su procesamiento por parte de clientes. Los DTO pueden estar asociados a una entidad o no, en cuyo caso son llamados DTO de parámetros.

Las **acciones** reciben DTO como elementos de entrada y salida. La lógica de la acción debe ser implementada por un programador. La acción se puede marcar como pública o privada, según si son expuestas como puntos de entrada por el servicio en ejecución o se emplean solo internamente para modularizar la lógica. La figura 5.4 muestra un ejemplo donde las figuras moradas representan una acción y el resto son DTO, donde se muestran los campos que lo componen. Las flechas representan si el DTO es la entrada (la dirección es del DTO a la acción) o la salida de la acción (la flecha va de la acción al DTO).

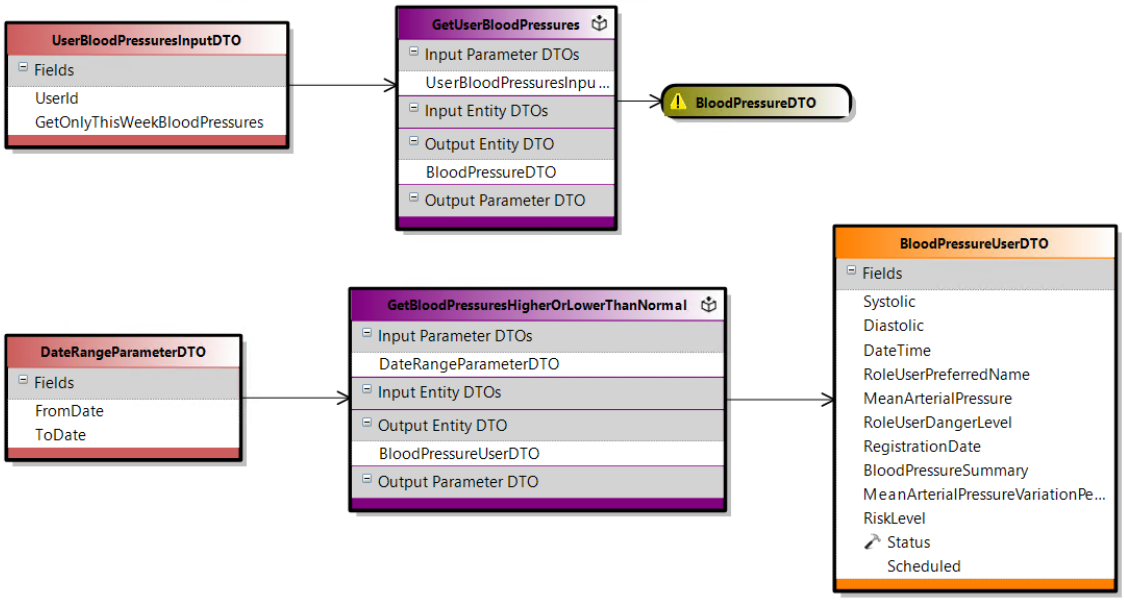

Figura 5.4: Modelo de aplicación para obtener los controles de tensión de un usuario.

DSML de Pruebas

Para las acciones, validaciones y campos calculados modelados se pueden modelar **casos de prueba**. Estos casos de prueba se traducen en pruebas unitarias generadas automáticamente, por lo que la definición de pruebas se abstrae del *framework* de *testeo* y cualquier usuario puede definir casos sin conocimientos de programación.

En la figura 5.5 se muestra para una acción (contenedor morado) un conjunto de casos de prueba (contenedores azules). El rectángulo amarillo representa la instanciación del parámetro de entrada de la acción con unos valores concretos (que se introducen a través de un editor que no se muestra). El rectángulo rojo representa la salida esperada de la acción, mientras que las cajas verdes representan instancias de entidades que existen durante la ejecución de la prueba.

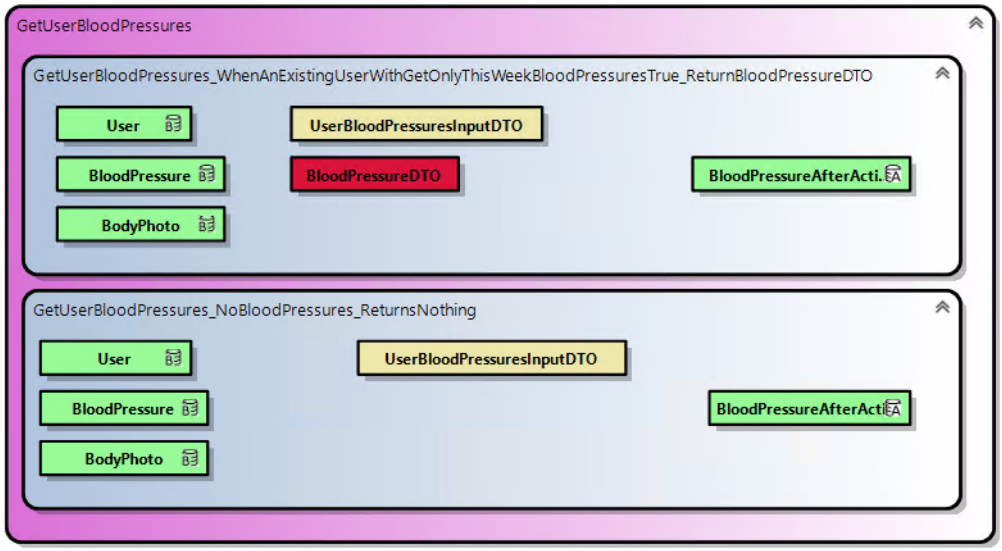

Figura 5.5: Modelo de pruebas de una de las acciones anteriores.

DSML de Formularios

Este DSML permite modelar los **formularios** que se autogenerarán para la aplicación de escritorio y móvil. En ella, un formulario se vincula a un DTO, del que se nutre para renderizar la información. Los DTO se emplean para nutrir listas, tablas y controles como **selectores** (o *combobox*). Con esta herramienta también se define la navegación entre los formularios.

En el ejemplo de la figura 5.6 se muestra un patrón típico, el maestro-detalle. Se describe que al formulario de la izquierda se accede a través de una navegación del formulario principal (caja azul en la esquina inferior izquierda). Los rectángulos verdes definen el diseño de los elementos que contiene, de tal forma que puede ordenar estos por columnas o por filas. El formulario de la izquierda contiene una **tabla** (*Grid*) donde se renderizan 2 campos. Cuando un elemento de la tabla se selecciona, el espacio de su derecha representado por el rectángulo rojo (***Form Container***) cargará el formulario de la derecha, que contiene el detalle de dicho elemento.

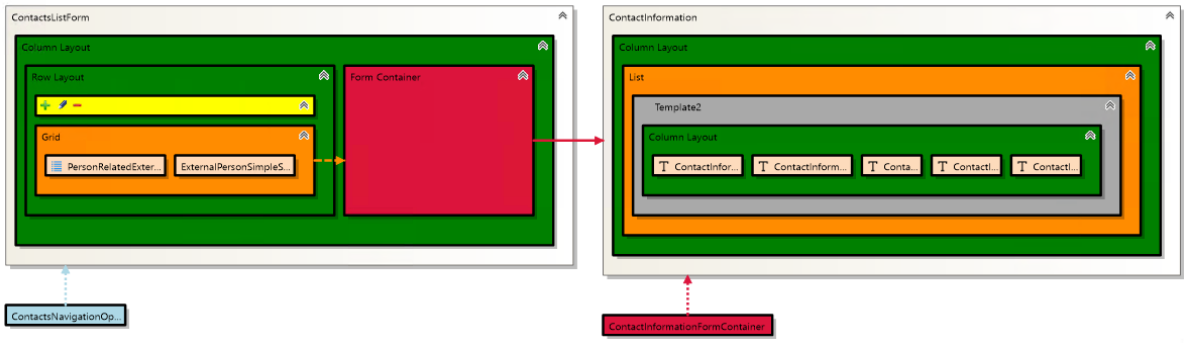

Figura 5.6: Modelo de formularios de los contactos de un usuario.

5.2 Presentación del caso práctico

En la PYME, se quiere implementar un nuevo DSML para agilizar la incorporación de inteligencia artificial en la aplicación. Algunos escenarios ya han sido cubiertos de manera manual en el pasado, como el reconocimiento de entidades en texto, el reconocimiento facial o la ejecución de comandos por voz.

Sin embargo, se ha observado que son muchos los sitios de la aplicación donde se quieren añadir funcionalidades basadas en inteligencia artificial. Aprovechando la experiencia del equipo en el campo de MDD, se plantea hacer lo mismo para diferentes escenarios de IA. En concreto, se plantea soportar los siguientes escenarios: reconocimiento de entidades en texto (esta vez, mediante la generación automática de código), detección de anomalías, detección de emociones, búsqueda de respuestas (*question answering*), predicción de sucesos a futuro y sistemas recomendadores.

La solución propuesta en todos los escenarios debe ser genérica. Por ejemplo, el escenario de análisis de sentimiento se plantea usar en diferentes textos como valoraciones médicas u observaciones. La predicción de sucesos debe soportar en un futuro predecir los días que un usuario permanecerá hospitalizado o predecir la demanda de un fármaco para su provisión.

En este trabajo, nos centraremos únicamente en dos: el escenario de análisis de sentimiento, que aplicaremos a las valoraciones médicas, y el de predicción de sucesos, aplicado a caídas. Hemos seleccionado el primer escenario porque se puede emplear para su implementación modelos públicos ya entrenados, librerías de código abierto o servicios de una API como los expuestos de proveedores de servicios, como Amazon y Microsoft. Lo importante en este caso será hacer un diseño modular para poder reemplazar una implementación por otra conforme evolucione el estado del arte. El escenario de predicción ha sido escogido porque implica entrenar un modelo propio. Tendremos que definir cómo llevar a cabo todo el ciclo de ML. Además, resulta interesante el ámbito de aplicación ya que no es un caso tan habitual, como son otros problemas de regresiones o clasificaciones, y los proveedores de servicios no le dan un soporte concreto y fácil de usar.

5.2.1. Especificación de requisitos

A continuación, se describen los casos de uso, que se definen siguiendo una plantilla en modo tabla. Para cada caso de uso se va a proveer un identificador, un título y una descripción breve. El caso de uso para el escenario de análisis de sentimiento es:

CU001	Aplicar análisis de sentimiento a un campo en una entidad
En el DSML de Dominio aparecen modeladas entidades con campos de tipo textual. Se debe permitir aplicar el análisis a cualquiera de estos campos sin importar la entidad o el servicio al que pertenezca. Aplicar el análisis a un campo significa que, cuando se realice una operación de crear o actualizar una instancia de la entidad y se cambie el valor del campo, el texto provisto se mandará a analizar. Cuando el analizador detecta, con el suficiente grado de certeza, sentimientos negativos en un texto, se debe notificar a un usuario de la aplicación sobre este hecho. Los clientes de la empresa son de diferentes regiones de España donde el español comparte ser lengua oficial con otras como el catalán, el vasco o el gallego. También se cuenta con clientes anglosajones. La primera versión del producto debe soportar como mínimo la lengua inglesa, pero debe seguir un diseño modular para incorporar el análisis de textos en otras lenguas, como las descritas arriba.	

En cuanto al escenario de predicción de sucesos, los casos de uso son los siguientes:

CU002	Aplicar predicción con la ocurrencia de una entidad
Se debe permitir seleccionar una entidad para evaluar cuando se creará una instancia de esta. Por ejemplo, si seleccionamos la entidad Usuario, estaremos prediciendo cuando un usuario nuevo será registrado en el sistema. En este proceso será necesario identificar las entradas (*features*) que se creen influyen en el evento a predecir. Siguiendo el ejemplo, un experto del Dominio podría determinar que la fecha influye, ya que en el mes de Septiembre suele haber un aumento en los usuarios solicitando información. En el sistema, los datos que influyen en el evento a predecir provienen de entidades a las que se puede navegar desde la seleccionada, y más en concreto de sus **campos, enumerados y asociaciones**. Por ejemplo, imaginemos un dominio en el que aparecen las entidades Usuario, Patología y Hospitalización. Un usuario sufre una serie de patologías y tiene vinculadas sus hospitalizaciones anteriores. Para tratar de deducir los días que un usuario permanecerá ingresado en una nueva hospitalización podemos consultar sus patologías (a través de la asociación entre ambas entidades) y ver el tiempo que han permanecido otros usuarios con las mismas patologías, así como calcular la media de días que ese usuario está hospitalizado.	

CU003	Experimentos para probar diferentes configuraciones
Los expertos de dominio no pueden determinar muchas veces las relaciones entre atributos, aunque se apoyen en herramientas de visualización para consultar los datos. Se debe soportar el concepto de **experimento**, donde para una misma tarea (por ejemplo, predecir una caída) se realizan diferentes configuraciones con diferentes entradas para el modelo e hiperparámetros. Estos experimentos pueden desplegarse a producción para su evaluación (estado de **evaluación**), de tal forma que son transparentes al usuario de la aplicación mientras recolectan datos y son ejecutados con el objetivo de ver la precisión en un escenario real. Una vez elegido el modelo con el que se desea llevar a cabo la tarea, se cambiará el estado del experimento seleccionado a **producción** y el resto de los experimentos se marcarán con el estado **descartado**, proveyendo una breve justificación de estas decisiones.	

CU004	Operaciones para crear atributos nuevos
En la predicción, se seleccionan los campos y asociaciones de entidades relacionados que influyen en el evento a predecir. Sin embargo, la entrada del modelo no es siempre el valor de estos campos tal y como aparecen en las instancias de la entidad. Por ejemplo, supongamos que queremos predecir la hospitalización de un usuario. En nuestro ERP se recolecta información de los usuarios como sus pesos (que van acompañados de la fecha en la que se realizan) o sus controles de tensión. El sistema debe ofrecer la posibilidad de marcar como entrada del modelo de predicción el **último** peso o la **media** y el **máximo** de los controles de tensión que se le han hecho, porque un experto de dominio determina que pueden influir. Aplicado a un campo, las operaciones que interesan soportar son: máximo, mínimo, último y primero. El concepto de operación también aplica a las asociaciones. Es necesario una operación 'está relacionado con' para determinar, por ejemplo, si un usuario sufre una patología. Lo mismo aplica para los enumerados. Por ejemplo, el género de un usuario se suele representar como un enumerado y es una entrada muy interesante para los modelos de predicción.	

CU005	Visualización de las predicciones
En la mayoría de los casos de uso planteados, el usuario es quien juega un papel principal. La API del servicio debería ofrecer un método para poder hacer una consulta únicamente proveyendo al usuario y que se obtuvieran automáticamente todos los datos necesarios que son la entrada del modelo. Esta acción será utilizada para crear un formulario donde se muestren todos los usuarios acompañados de la probabilidad de que les ocurra el evento a predecir, por ejemplo, una caída.	

Como requisito no funcional, se espera que las implementaciones de ambos escenarios puedan usarse por gente sin conocimientos de programación. Los usuarios que se espera tener a corto plazo son los usuarios de los DSML presentados, que no son programadores como tal.

5.3 Escenario de análisis de sentimiento en valoraciones

El sector de las residencias en España ha sido uno de los más faltos de transparencia y control hasta el inicio de la pandemia, cuando se puso el foco en ellos por su criticidad. Un 21 % de las residencias han sido sancionadas desde 2014. De estas sanciones, un 25 % son por falta de personal, uno de los principales problemas de las residencias, un 19 % por una asistencia inadecuada, un 10 % por falta de higiene, un 8 % por trato degradante o maltrato y un 4 % por obstruir la labor inspectora. Hasta hace bien poco, se podían ver periódicamente vídeos con cámara oculta reflejando estas situaciones [47].

Como es lógico, la excepción no confirma la regla, pero nuestro papel ofreciendo un *software* de gestión para estas instituciones debe ser firme. Un pequeño paso que se desea dar en la futura aplicación es el análisis automático de textos para la detección de situaciones anómalas. En concreto, vamos a reproducir un caso de uso concreto aplicado a las valoraciones médicas usando la implementación basada en modelos. Las valoraciones son las anotaciones que realiza un profesional recogiendo los detalles de una consulta con un paciente. Se espera, al tener un fin únicamente profesional, un uso correcto y formal del lenguaje en el texto libre que se introduce.

En el siguiente capítulo discutiremos el diseño y la implementación. Una evaluación satisfactoria se traducirá en emplear el análisis de sentimiento en más partes de la aplicación como mensajería, informes u observaciones en general.

5.4 Escenario de predicción de caídas

Según la Organización Mundial de la Salud (OMS), las **caídas** son sucesos involuntarios que hacen perder el equilibrio y dar con el cuerpo en el suelo u otra superficie que lo detenga. Las lesiones causadas por las caídas pueden resultar mortales, aunque la mayoría no lo son, sobre todo si tenemos en cuenta las que sufre la población joven. Sin embargo, es la segunda causa mundial de muerte por lesiones accidentales o involuntarias, solo por detrás de las colisiones de tránsito [48].

Las consecuencias obvias de una caída son a nivel físico: contusiones, heridas, fracturas faciales o coágulos. Además, también producen efectos psicológicos como el miedo a volver a caer, que se traduce en una dependencia del mayor si no puede o no quiere seguir viviendo solo y requiere de cuidados [48, 49, 50].

Existen factores que predisponen una caída. La mayoría de los autores los clasifican en **extrínsecos**, que son aquellos ajenos a la persona como obstáculos en el camino, falta de iluminación o suelos resbaladizos, e **intrínsecos**, de tipo fisiológico y propios del envejecimiento, que afectan al equilibrio y a la marcha [50].

Debido a la naturaleza del ERP del que vamos a obtener los datos, nos centraremos principalmente en factores intrínsecos, ya que los controles, escalas y patologías del residente son los datos que mayormente se registran. Los factores que vamos a tomar de la literatura para la predicción de caídas son:

- **Edad**: aproximadamente el 30 % de las personas mayores con más de 65 años sufren una caída al año, número que aumenta cuando los ancianos viven en instituciones[49]. Este porcentaje aumenta al 50 % en personas con más de 80 años [50].
- **Sexo**: la mayoría de estudios coinciden en que las mujeres son las que padecen un mayor riesgo de caída [48, 50]. Sin embargo, el estudio de Silva-Fohn *et al.* [51] refleja un predominio de caídas en hombres que todavía no se han jubilado, sin problemas cognitivos, pero con síntomas depresivos.
- **Problemas de movilidad**: artritis, artrosis, atrofia muscular [52], pérdida reciente de masa muscular o sobrepeso [49] son algunas de las patologías que pueden influir.
- **Trastornos neurológicos**: como son sufrir la enfermedad de Parkinson, haber superado un accidente cerebrovascular (ictus), sufrir demencia, depresión, tener episodios de desmayos o pérdidas del conocimiento [52, 50].
- **Problemas cardiovasculares**: mientras que en [50] y [52] se incide en que la hipotensión arterial y las arritmias son un factor de riesgo, en [49] es la hipertensión determinante, donde el 53 % de las personas que se caían la padecían.
- **Problemas visuales y auditivos**: todos los autores coinciden en que el déficit visual aumenta la probabilidad de sufrir una caída, como son cataratas o glaucoma. Suárez *et al.* recogen también la relación con problemas auditivos, tales como presbiacusia, otitis u otosclerosis, aunque no llegan a ser significativos [49].
- **Escalas**: en [51] se menciona que el resultado de algunas escalas como las de Barthel, Lawton y Brody o la escala de Depresión Geriátrica de Yesavage pueden emplearse para la predicción de caídas. La escala de Barthel [6] valora la capacidad de la persona para realizar las Actividades Básicas de la Vida Diaria (ABVD). La escala

[6]Cuestionario de la escala Barthel: hipocampo.org/Barthel.asp

de Lawton y Brody [7] valoran la autonomía física y realización de las Actividades Instrumentales de la Vida Diaria (AIVD). Por su parte, la escala de Yesavage [8] mide el grado de depresión del paciente. Estas escalas están disponibles en nuestro ERP, pero será importante considerar valores medios para sustituir valores faltantes, es decir, cuando al usuario no se le ha realizado la escala bajo análisis.

Otros factores que menciona la literatura no se van a incluir por ahora como entrada del modelo de IA porque no están disponibles en el ERP para ser explotados fácilmente. Por ejemplo, la polifarmacia, es decir, el uso simultáneo de varios fármacos o con una dosis excesiva, puede influenciar en el residente [51]. Algunos autores como Blake [50] especifican fármacos concretos como los antidepresivos, tranquilizantes o inductores de sueño. Machado *et al.* relacionan el consumo de café también [53]. Por otro lado, [49] incide más en la movilidad como factor de riesgo: si el residente sale de la institución, hace ejercicio regularmente o usa soportes de apoyo para caminar.

Con todo esto, en el capítulo **7 Diseño y desarrollo del escenario de predicción de sucesos** utilizaremos este conocimiento recabado para construir nuestro modelo de predicciones. Evaluaremos también su precisión y cómo de bien se adapta el DSML para este propósito.

[7] Cuestionario de la escala de Lawton y Brody: hipocampo.org/lawton-brody.asp
[8] Cuestionario de la escala Yesavage: hipocampo.org/yesavage.asp

CAPÍTULO 6
Diseño y desarrollo del escenario de análisis de sentimiento

En este capítulo veremos el desarrollo del escenario de análisis de sentimiento. Las actividades que se explican son las de diseño, implementación y pruebas. Previo a todo, se presentan unas consideraciones iniciales para tener en cuenta sobre los microservicios y librerías existentes.

6.1 Diseño de la solución

El diseño del escenario fue estudiado en diferentes reuniones a las que asistieron el jefe del Departamento de Desarrollo y la tupla formada por el programador que implementaría los cambios y la persona encargada de revisar dicho código. Es necesario mencionar algunos aspectos antes de proseguir:

- **Existencia de un microservicio para IA**: previo a todo el trabajo, existe ya en el sistema un microservicio de inteligencia artificial que expone algunas funcionalidades para el reconocimiento de imágenes o el reconocimiento por voz. Este microservicio está construido usando los DSML presentados anteriormente, aunque parte de su lógica se ha implementado manualmente. Es en este servicio donde se quiere incorporar todo aquel código relacionado con el procesamiento del lenguaje natural, para tener las funciones de IA centralizadas y gobernadas por un equipo especializado.
- **Microservicio para notificaciones**: en el sistema se ha planeado la creación de un microservicio para gestionar las notificaciones que se envían a un usuario. El microservicio también gestionará la mensajería interna entre usuarios. Cuando se cree, existirá un método HTTP disponible que podemos invocar desde otros servicios para crear notificaciones, que por ahora se simulará a través de una interfaz.
- **Existencia de una librería *core***: las factorías de *software* se apoyan en los DSML para construir sistemas de un dominio concreto, pero el código generado para estos sistemas suele apoyarse en un conjunto de librerías *core* compartidas entre todos ellos. Este principio aplica también a nuestro sistema, donde la librería *core* gestiona aspectos transversales como la gestión de datos, permisos o el tratamiento de errores. Las operaciones CRUD siempre pasan a través de esta librería, por lo que se ha consensuado que puede modificarse según sea necesario.
- **Acuerdo en crear un DSML para IA**: debido a todos los casos que se desean cubrir en un futuro relacionados con la IA, se ha acordado que se implemente un nuevo

DSML para esta finalidad, en lugar de incorporar los conceptos necesarios dentro de los DSML existentes como el de Aplicación o Dominio. Esto mantiene la cohesión y separa los intereses de manera clara entre las diferentes herramientas. Como desventaja, supone un mayor esfuerzo ya que la creación del nuevo DSML sigue un proceso largo. Por ejemplo, se debe garantizar la entrega continua de la nueva herramienta, por lo que a nivel de infraestructura es necesario la creación de *builds* en el repositorio de código, *build policies* para garantizar que no se integre un error que haga que el código de la herramienta no compile o no ejecute satisfactoriamente sus *tests*, etc. Estas tareas se han realizado, pero no se documentan con más detalle en el trabajo porque queda fuera de su alcance.

De acuerdo con los requisitos, que también se revisaron en las reuniones, la propuesta de diseño para coordinar a todos los componentes implicados es la siguiente, que se muestra en la figura 6.1:

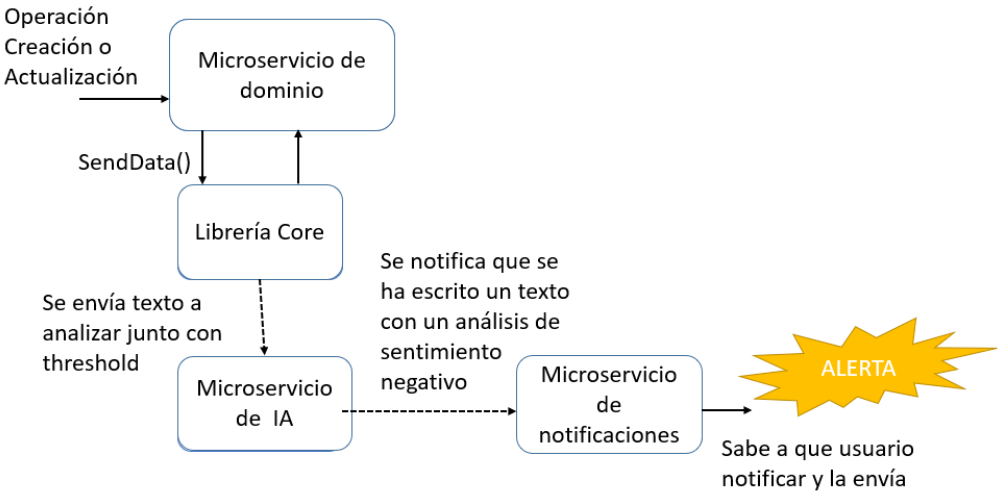

Figura 6.1: Diseño del escenario de análisis de sentimiento.

1. Cuando se realiza una operación de actualización o creación de una instancia de una entidad de un microservicio (operación *SendData()*), esta es procesada por nuestra librería *core*.

2. La librería *core* es capaz, delegando en una interfaz recibida por inyección de dependencias y que se implementará en el servicio, de saber si alguno de los campos textuales de esa instancia debe enviarse al microservicio de IA para su análisis.

3. En caso afirmativo, la librería *core* invoca una acción expuesta por el microservicio de IA para enviar el texto a analizar. El microservicio de IA procesa el texto y obtiene como resultado el análisis de sentimiento del texto.

4. Si el resultado del análisis es negativo con cierto grado de confianza, el microservicio de IA contacta con el de notificaciones a través de una acción, comunicando el evento que ha ocurrido.

5. El microservicio de notificaciones almacena, como una configuración dinámica, a quién se debe notificar cuando un evento ocurre. Se puede notificar a un usuario particular o a un grupo de usuarios. En este caso, el microservicio creará la notificación correspondiente, finalizando así el caso de uso.

Para llevar a cabo el caso de uso CU001, es necesario añadir en nuestro DSML de IA las metaentidades para relacionar un campo de texto, que es un concepto del DSML de Dominio, con el escenario de análisis, que será en si la metaentidad nueva. En la figura 6.2 se muestra el metamodelo. Un modelo de nuestro DSML contiene **escenarios**, una metaentidad base de la que heredarán los escenarios que vamos a crear y los futuros para clasificación, sistemas recomendadores, etc. Un escenario referencia siempre a una entidad y a un campo, que en este escenario será siempre un campo textual. Por último, se ha añadido un parámetro adicional para representar un valor umbral, que se empleará para determinar el grado de seguridad que se debe superar para enviar una notificación. Es decir, un valor umbral del 80 % representa que, si el analizador concluye que el texto tiene una connotación negativa con una seguridad del 60 %, una notificación no será enviada porque no se supera el valor umbral.

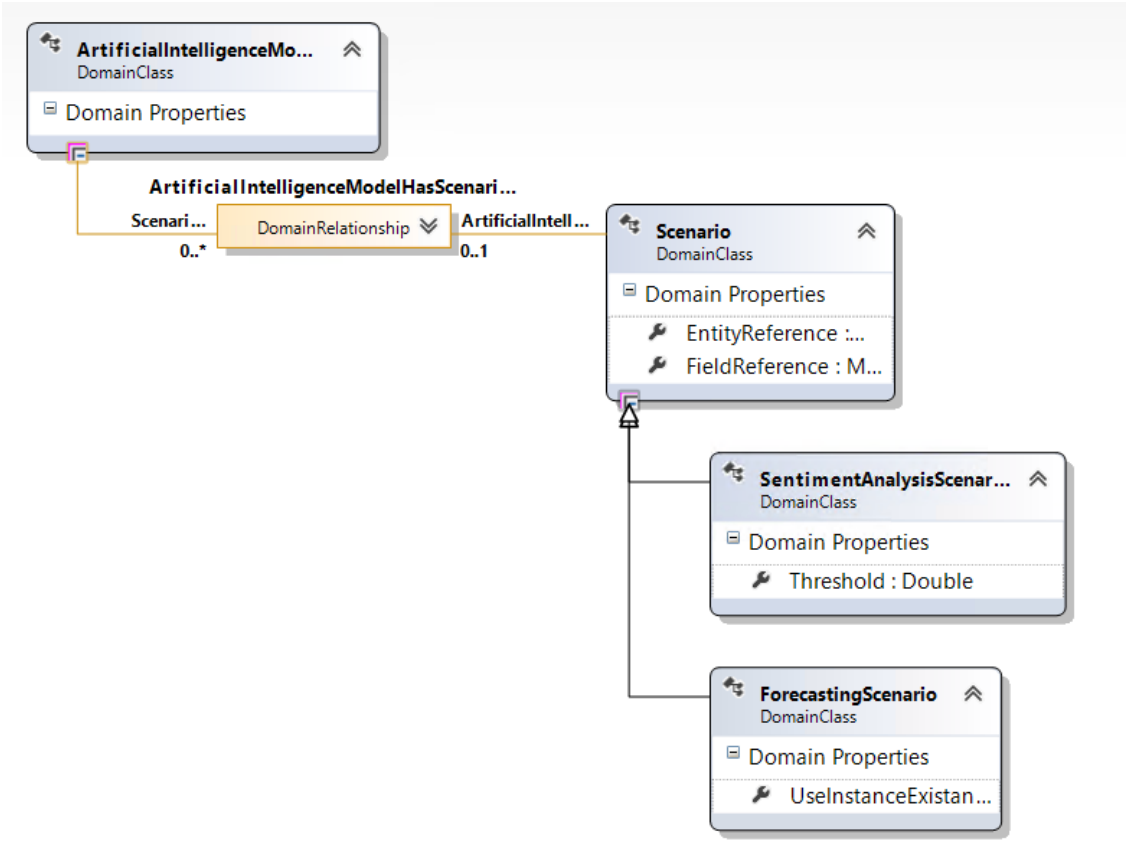

Figura 6.2: Metaentidades para el escenario de análisis de sentimiento.

6.2 Detalles de la implementación

Para cubrir la funcionalidad necesaria, hemos comenzado modelando 2 nuevas acciones (figura 6.3). La primera de ellas, *AnalyzeTextSentimentAndNotifyWhenThresholdExceed*, será la acción invocada desde la librería *core* para llevar a cabo el análisis de un texto. Esta acción recibe una colección de DTO con diferentes textos a analizar y sus respectivos valores umbrales, con el objetivo de realizar una única llamada con todos los textos a analizar de una vez. La acción es asíncrona, por lo que no devolverá nada y su invocación se hará a través de un bus de mensajería en lugar de a través de un método HTTP. La segunda acción, *DetectLanguages*, es una acción privada. Esto se representa por el contenedor que la envuelve, que representa la clase donde se implementará. En este caso, para un texto la acción devuelve un conjunto de DTO con los diferentes lenguajes que ha detectado en el texto y el nivel de confianza con el que determina la presencia de cada uno.

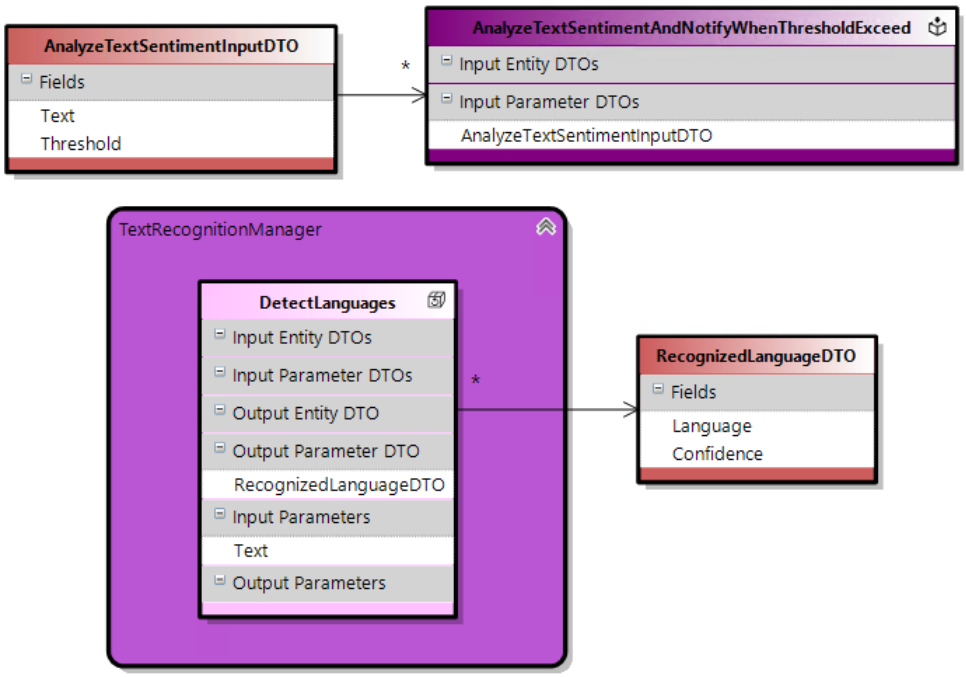

Figura 6.3: Acciones para el análisis de sentimiento.

La implementación de las acciones es manual. En el *snippet* de código de la figura 6.4 se muestra la implementación de la primera acción. En ella se itera sobre los textos a analizar y se comprueban los idiomas en el texto invocando a la segunda acción representada. Posteriormente, si se detecta un idioma con el suficiente nivel de confianza, se prosigue con el análisis. Por ahora, el análisis solo está soportado en inglés (en el código se aprecia un *TODO*) y si este análisis sobrepasa el valor umbral, una notificación es enviada invocando al microservicio de notificaciones. El código está diseñado para soportar diferentes idiomas haciendo uso de diferentes modelos de lenguaje.

En una primera versión, quisimos hacer uso de una estrategia como la presentada en[28], donde un modelo multilenguaje se entrena para la tarea de análisis de sentimiento. Sin embargo, nuestra falta de experiencia en Python hizo que no obtuviéramos un rendimiento aceptable del modelo. Tras esto, se decidió emplear librerías existentes en C# para el análisis de sentimiento. De todas las librerías evaluadas, solo algunas que es-

taban en inglés funcionaban como deseábamos, por lo que se decidió postergar el soporte multilenguaje como trabajo futuro.

```
internal async Task AnalyzeTextSentimentAndNotifyWhenThresholdExceedAsync(
    IEnumerable<AnalyzeTextSentimentInputDTO> textsToAnalyze)
{
    foreach (AnalyzeTextSentimentInputDTO textToAnalyze in textsToAnalyze)
    {
        decimal sentimentNegativeScore;
        IEnumerable<RecognizedLanguageDTO> recognizedLanguages = this.textRecognitionManager.DetectLanguages(textToAnalyze.Text);
        RecognizedLanguageDTO dominantLanguage = recognizedLanguages
            .Aggregate((firstItem, secondItem) => firstItem.Confidence > secondItem.Confidence ? firstItem : secondItem);

        if (dominantLanguage.Confidence.Value < 80) {
            throw new FunctionalValidationException();
        }

        switch (dominantLanguage.Language) {
            case Language.English:
                sentimentNegativeScore = this.englishSentimentAnalyzer.GetSentimentNegativeScore(textToAnalyze.Text);

                break;
            case Language.Spanish:
                // TODO: Support spanish language.
                throw new FunctionalValidationException();
        }

        if (sentimentNegativeScore > textToAnalyze.Threshold) {
            await this.cocktailMessagingManager.SendNotificationAsync(Event.NegativeSentimentExceeded, textToAnalyze);
        }
    }
}
```

Figura 6.4: Porción de código para el análisis de sentimiento.

Las librerías que hemos empleado para implementar la acción son las siguientes:

- **BrokenEgg.LanguageDetection**[1]: una librería para la detección de idiomas en un texto en C#, traducida desde una librería original en Java. La librería cuenta con soporte para detectar 53 idiomas, representados como diferentes ficheros JSON. El entrenamiento de la librería ha sido realizado empleando artículos de Wikipedia mediante un clasificador bayesiano.
- **VaderSharp**[2]: es también una librería traducida a C# desde Python. Se emplea para el análisis de sentimiento en textos, únicamente en inglés. Está basada en el uso de lexicones y reglas.

Tal como se menciona en el paso 5 del proceso descrito en el apartado **6.1 Diseño de la solución**, cuando en un texto se encuentran sentimientos negativos se debe crear una notificación empleando el microservicio de notificaciones. No obstante, este microservicio no está desarrollado todavía, por lo que en una primera versión que nos permita evaluar nuestro caso de uso almacenaremos las sentencias que deberían ser enviadas como notificaciones. Con este fin, se ha creado la entidad de la figura 6.5, que tiene como campos la fecha en la que se realiza el análisis, el texto analizado y la puntuación obtenida.

[1]Repositorio de LanguageDetection: github.com/kaiidams/LanguageDetection
[2]Repositorio de VaderSharp: github.com/codingupastorm/vadersharp

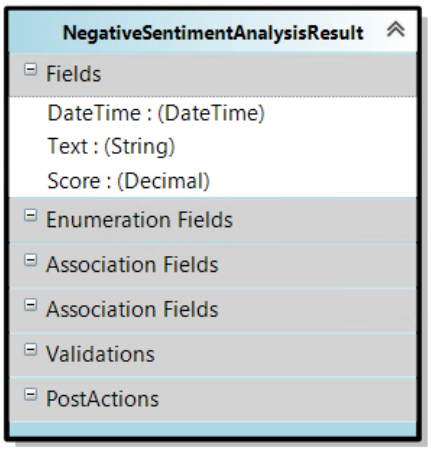

Figura 6.5: Entidad para almacenar sentencias con análisis negativo.

Esta entidad puede emplearse luego para construir un formulario accesible por los usuarios de la aplicación. El formulario se muestra en la figura 6.6, donde se indica que los 3 campos de la entidad se renderizarán como columnas dentro de una tabla. En esta tabla solo estará soportada la operación de lectura, de tal forma que desde la UI no se podrán crear nuevas instancias.

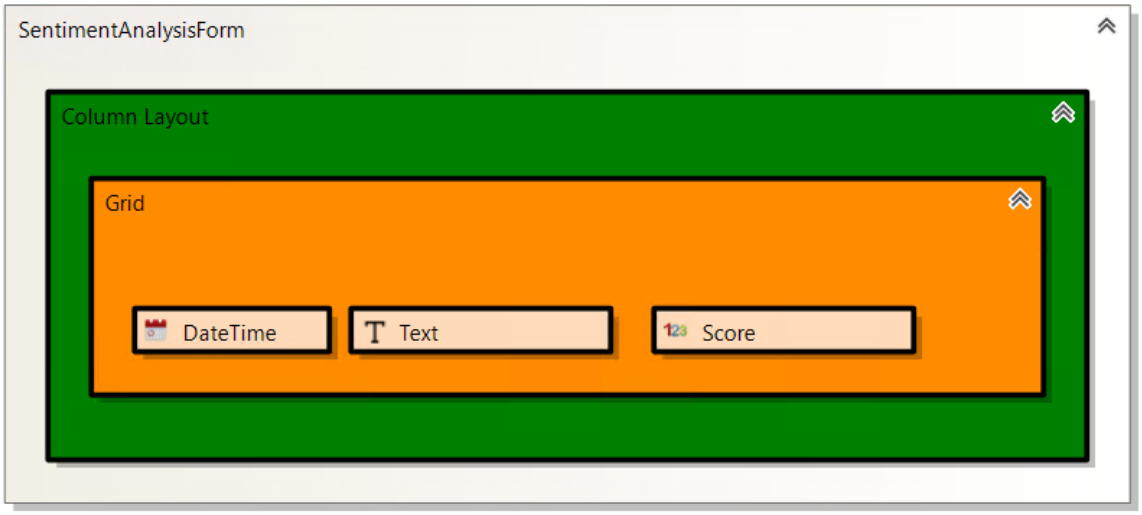

Figura 6.6: Modelo del formulario con los análisis negativos.

Para determinar si un texto debe ser analizado se debe autogenerar una clase en el microservicio del dominio, es decir, el microservicio en cuyo dominio está el campo sobre el que se realiza el análisis. La generación de código se realiza mediante plantillas T4, donde se instancian los elementos del modelo y se puede mezclar el código de control (por ejemplo, iterar sobre los escenarios de análisis de sentimiento) con el código de la clase que se autogenerará. En la figura 6.7 se incluye una porción de la clase a generar. El código entre los signos '<#' y '#>' representa el código de la transformación. En el microservicio se itera sobre los escenarios de análisis que tiene modelados y para cada uno se crea un *case* del *switch* para representar que las instancias de la entidad que tienen asociadas deben analizarse. El símbolo '<#=' toma el valor de la propiedad que contiene y lo imprime en la clase generada.

```
[System.CodeDom.Compiler.GeneratedCode("<#= CodeGeneratorName #>", "1.0.0.0")]
internal sealed class <#= artificialIntelligenceModel.FullCompositeApplicationName #>ArtificialIntelligenceFilter : ArtificialIntelligenc
{
    /// <inheritdoc/>
    protected override AnalyzeTextSentimentInputDTO GetTextToAnalyze(EntityBase entityInstance)
    {
        switch (entityInstance)
        {
<#
foreach (SentimentAnalysisScenario sentimentAnalysisScenario in sentimentAnalysisScenarios)
{
#>
            case <#= sentimentAnalysisScenario.Entity.Name #> <#= sentimentAnalysisScenario.Entity.Name.ToLowerCamelCasing() #>:
                if (string.IsNullOrEmpty(<#= sentimentAnalysisScenario.Entity.Name.ToLowerCamelCasing() #>.<#= sentimentAnalysisScenario.
                {
                    return null;
                }

                return new AnalyzeTextSentimentInputDTO()
                {
                    Text = <#= sentimentAnalysisScenario.Entity.Name.ToLowerCamelCasing() #>.<#= sentimentAnalysisScenario.Field.Name #>,
                    Threshold = <#= sentimentAnalysisScenario.Threshold #>,
                };
<#
}
#>
            default:
                return null;
        }
    }
}
```

Figura 6.7: Fragmento de una plantilla.

El resto de los detalles, relacionados sobre todo con la librería *core*, no se consideran de relevancia para mencionarse en la memoria.

6.3 Pruebas

Para las pruebas de la librería Vadersharp se ha construido una batería utilizando un subconjunto de sentencias de un *dataset* de Kaggle [3] de críticas de películas y otro de *tweets*. La librería ha sido entrenada utilizando únicamente datos de las redes sociales, por lo que introducir parte del *dataset* de críticas de películas nos puede dar una aproximación de cómo de bien se adapta a otros dominios.

Se ha obtenido un 95.9 % de resultados bien clasificados. La realización de esta prueba nos ha permitido perfeccionar la fórmula para clasificar un texto como negativo. La librería VaderSharp nos devuelve 4 valores del análisis: 3 ratios con la proporción de palabras consideras como positivas, negativas y neutras respectivamente, además de un cuarto valor que representa la puntuación compuesta. En un principio, habíamos considerado como negativo los textos cuya puntuación negativa fuera mayor de 0.8, obteniendo así un porcentaje de sentencias bien clasificadas de 92.2 %. La puntuación compuesta hace un balance dando más peso a algunas palabras dentro del conjunto léxico y se normaliza luego para que su valor oscile entre -1 y 1. Según los autores, es mejor emplear esta puntuación, de tal forma que se puede considerar un texto como negativo si tiene una puntuación compuesta inferior a -0.05. Al emplear esta fórmula se observa un aumento de la precisión hasta el 95.9 % descrito arriba.

En la figura 6.8 se muestra la prueba realizada. Las sentencias se encuentran en un fichero CSV. Cada sentencia está etiquetada con el resultado esperado de la clasificación binaria, positivo o negativo. Se observa en el código el acceso a la puntuación compuesta o *compound*.

Además, utilizando nuestro DSML para pruebas se han modelado los siguientes casos de prueba para las acciones mostradas en **6.2 Detalles de la implementación**. Los casos

[3] Página de Kaggle: kaggle.com/

```
using (StreamReader reader = new StreamReader(
    Path.Combine(Directory.GetCurrentDirectory(), "..\\..\\..\\test.csv")))
{
    while (!reader.EndOfStream)
    {
        string line = reader.ReadLine();
        string[] split = line.Split(',');

        if (split.Length > 3 || split[0] == "id")
        {
            continue;
        }

        bool isNegativeExpected = split[1] == "1";
        string sentence = split[2];

        SentimentAnalysisResults results = analyzer.PolarityScores(sentence);

        if (isNegativeExpected)
        {
            if (results.Compound < -0.05) { countWellClassified++; } else { countBadClassified++; }
        }
        else
        {
            if (results.Negative < -0.05) { countBadClassified++; } else { countWellClassified++; }
        }

        count++;
    }

    double percentageWellClassified = (double)countWellClassified / (double)count;

    // 95.9 % of well classified.
    Assert.IsTrue(percentageWellClassified > 0.9);
}
```

Figura 6.8: Prueba para evaluar la librería VaderSharp.

de prueba se muestran en la figura 6.9. A simple vista no son muy inteligibles, ya que los datos de las instancias que se crean durante el *test* no se representan visualmente y solo se ven a través del explorador de propiedades de las cajas verdes, rojas y amarillas. El objetivo de estos *tests* es probar el código garantizando la cobertura de sentencias, es decir, que todas las instrucciones de los métodos se ejecuten al menos una vez. De esta forma, quedan cubiertos y probados casos especiales como los que ocurren cuando no se detecta un idioma con el suficiente nivel de confianza (ver figura 6.4) o cuando se introduce un texto en un lenguaje no soportado.

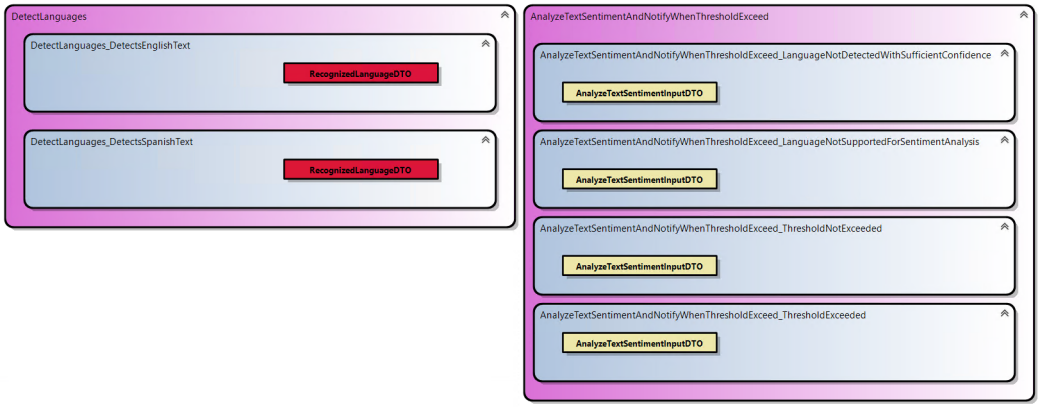

Figura 6.9: Casos de prueba modelados para las acciones de análisis de sentimiento.

Estos modelos se traducen en *pruebas* unitarias que se ejecutan automáticamente cuando se introduce un cambio en el repositorio de código que pueda impactar en su resultado.

6.4 Caso práctico: análisis de sentimiento en valoraciones

6.4.1. Modelado del problema

En este apartado vamos a repasar los pasos necesarios para añadir análisis de sentimiento a un campo en una entidad de dominio. En el ejemplo, vamos a emplear la entidad Valoración médica:

1. En primer lugar, se debe crear un diagrama de IA. Esto es tan sencillo como crear un fichero nuevo y solo es necesario tener instalada en Visual Studio la extensión del DSML de Inteligencia Artificial.

2. En la *toolbox* de Visual Studio (figura 6.10) nos aparecerá un elemento que representa el escenario de análisis de sentimiento. Lo único que se debe hacer con él es *drag-and-drop*.

Figura 6.10: *Toolbox* de Visual Studio para el análisis de sentimiento.

3. El escenario se representa dentro del diagrama como un contenedor azul, que se muestra a la izquierda en la figura 6.11. En la ventana de propiedades de Visual Studio debemos dar valor a algunas propiedades, como se muestra en la figura. Debemos prestar especial atención a las propiedades en la categoría *References*, donde a través de un editor seleccionamos que la entidad sobre la que se realiza el análisis es *DoctorProgressNote* y el campo en concreto es *Diagnosis*. El resto de las propiedades marcadas con un asterisco son requeridas y en caso de no proveerlas fallarán las validaciones del modelo. Por último, es necesario indicar el *Threshold*, un valor entre 0 y 100 que se indica para configurar el grado de confidencia para clasificar un texto como negativo.

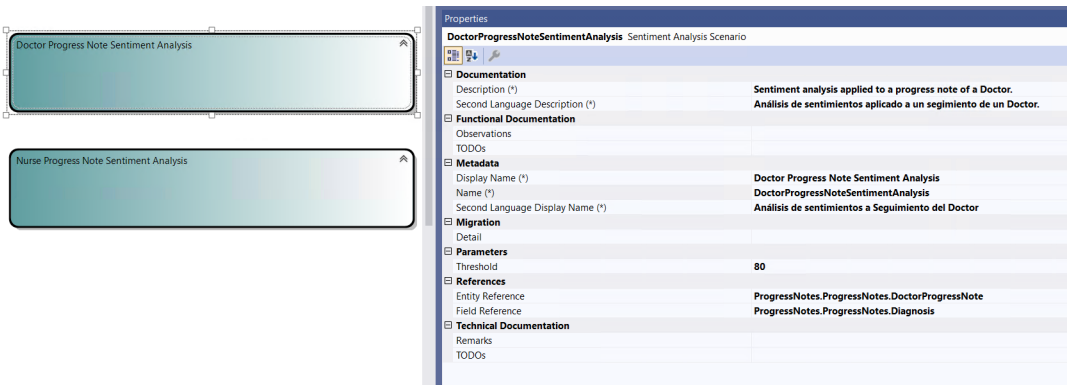

Figura 6.11: Modelo para el análisis de sentimiento en valoraciones.

4. Una vez modelado el escenario, se debe generar código a partir de los modelos. En nuestro caso, este proceso lo tenemos automatizado a través de diferentes *scripts* que no vamos a detallar. El código generado y el modelo deben subirse al repositorio de código.

5. Cuando se despliegue la aplicación, el análisis de sentimiento sobre una valoración médica y todo lo descrito en el caso de uso se realizará automáticamente y de manera transparente.

6.4.2. Resultados obtenidos

Tras los cambios realizados en los modelos de nuestra aplicación, podemos consultar a través del formulario de la figura 6.12 las sentencias que han sido clasificadas como negativas. A través de los permisos de la aplicación, se podrá configurar quién puede acceder a este formulario. El formulario ha sido autogenerado a partir del modelo de la figura 6.6.

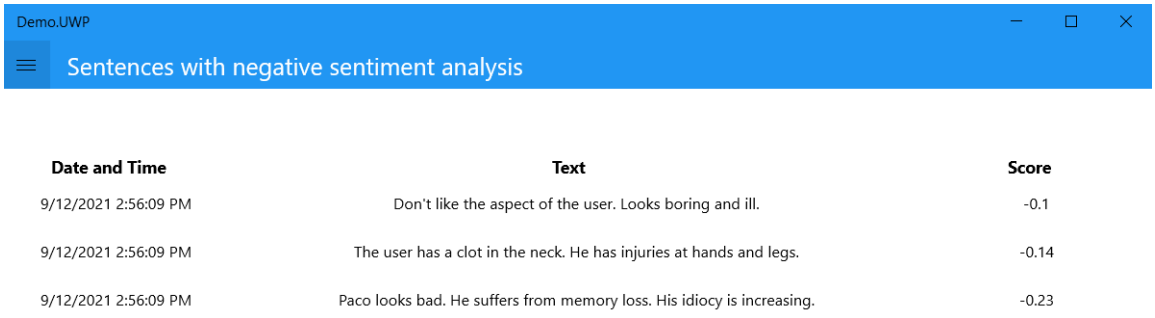

Figura 6.12: Formulario de sentencias con un análisis negativo.

Por último, queremos mencionar que se han observado, haciendo pruebas manuales, algunos textos que han sido clasificados de manera errónea. Para mejorar estos resultados, en futuras iteraciones se reemplazará la librería *Vadersharp* que se emplea por un modelo entrando utilizando textos de nuestro dominio. También puede aumentarse el valor umbral para minimizar el error de clasificación.

CAPÍTULO 7
Diseño y desarrollo del escenario de predicción de sucesos

En este capítulo se presenta el proceso de construcción de los escenarios de predicción. En primer lugar, se introduce la evolución de la notación del DSML. A continuación, se resumen los cambios realizados y por último se realiza un caso práctico para la predicción de caídas.

7.1 Diseño de la solución

De nuevo, la tarea de diseñar la solución corrió a cargo del equipo de IA a través de diferentes reuniones. En este escenario, el modelado a través de las DSL Tools cobra mucho mayor peso ya que es necesario indicar las *features* que se emplearán en la construcción del modelo de ML. Esto no pasaba con el escenario de análisis de sentimiento, donde solo era necesario especificar la entidad y campo al que realizar el estudio y la mayoría del código implementado es genérico, relacionado sobre todo con la infraestructura.

La notación visual cobra mucho peso. ¿Cómo se representa una tarea de clasificación o regresión en las herramientas vistas? Algunas representaciones están muy ligadas al algoritmo que se usará. Esto se aprecia sobre todo en herramientas como Neuroph donde directamente se dibuja la red neuronal. Obviamente, la arquitectura de una red neuronal no puede ser diseñada por un usuario sin conocimientos en la materia.

La propuesta de los grandes proveedores tampoco nos termina de convencer. Se centran en soportar el mayor número posibles de fuentes como origen de datos (CSV, Redis[1], Cosmos DB [2], etc.). Sin embargo, a la hora de la verdad, todos los datos que se desean usar se deben ofrecer a la aplicación en un formato tabular. Por lo general, aunque se pueda integrar con muchas bases de datos, luego solo se puede usar una tabla o fichero. No obstante, en las herramientas más potentes se puede especificar la transformación para combinar las diferentes entradas como si de una *pipeline* ETL se tratara. El modelo relacional es muy rico y poder navegar hacia otras tablas a través de instrucciones JOIN en SQL o hacer filtros lo vemos esencial. Al final, la herramienta construida debe ser cercana a la representación entidad-relación de nuestro DSML de Dominio porque va a explotar esos datos y es la que conocen nuestros usuarios.

[1]Página de Redis: redis.io/
[2]Página de Azure Cosmos DB: azure.microsoft.com/es-es/services/cosmos-db/

7.1.1. Evolución de la notación gráfica de modelado

La representación que se busca debe asemejarse a representaciones ya existentes como la del DSML de Dominio. En esta dirección fueron las primeras propuestas, que definimos a través de formas de PowerPoint. En una primera iteración, se propuso la notación visual de la figura 7.1. Se han representado dos escenarios de predicción diferentes: uno para predicción de caídas y otro para predecir los días que un usuario permanece en un hospital cuando sufre un accidente. Esto se hace con el objetivo de abstraer mejor la representación de un problema concreto e introducir variabilidad.

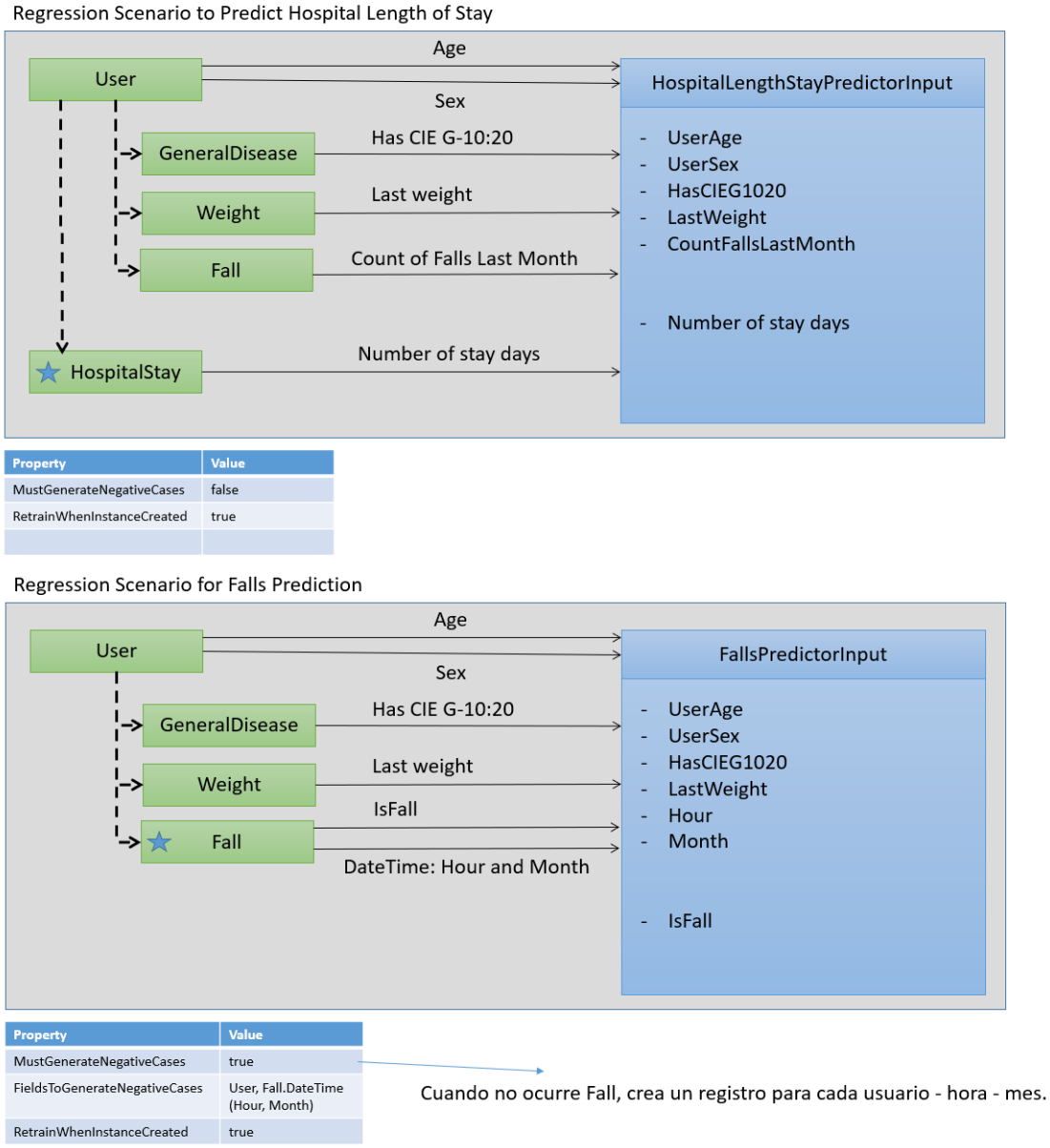

Figura 7.1: Primer boceto del escenario de predicción.

Las cajas verdes son referencias a entidades del dominio. La entidad marcada con una estrella representa la entidad sobre la que se realiza la predicción, en estos casos la caída o la estancia en el hospital. Las líneas discontinuas representan las asociaciones que existen en el dominio entre las entidades. Las líneas continuas representan el mapeo entre las entidades y la entidad aplanada que se usará para el entrenamiento, que se representa

mediante una caja azul. La entidad aplanada es necesaria para la librería ML.NET, que se usará para el entrenamiento y que requiere una entrada tabular. Las líneas continuas pueden ser proyecciones o transformaciones. Cuando se proyecta un valor, estamos copiando este tal como es. Por ejemplo, en el *dataset* tendremos una fila para cada caída y tendremos una columna *UserAge* que toma como valor la edad del usuario que sufre la caída. Las transformaciones son un conjunto de operaciones que proveemos con nuestro DSML para enriquecer el modelo. Por ejemplo, se van a ofrecer operaciones como 'el último', 'el primero' o 'el máximo'. Esto se usa en la figura para obtener el último peso del usuario que sufre la caída.

Esta representación no es la definitiva, ya que encontramos algunos inconvenientes en ella. En primer lugar, las líneas discontinuas no aportan valor al modelo. La asociación entre las entidades queda representada en el modelo de dominio y añadirlo aquí es redundante. En segundo lugar, observamos mucha información repetida: en la caja azul tenemos representado lo mismo que en las flechas continuas. Además, nuestra experiencia con el resto de DSML nos ha conducido a pensar que los conectores no son la mejor representación ya que generan mucho ruido. Por ejemplo, si hay muchas líneas, estas terminan cruzándose sin entenderse realmente. Por este motivo, se decide cambiar esta notación por una con compartimentos, donde se combina la entidad y las transformaciones de las *features*. La evolución del prototipo se muestra en la figura 7.2.

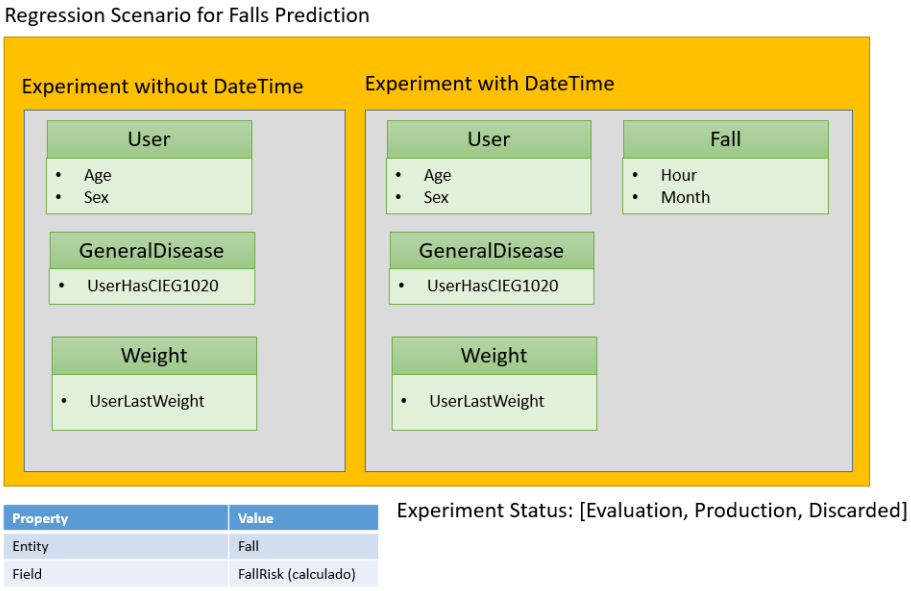

Figura 7.2: Segundo boceto del escenario de predicción.

En esta segunda notación se ha introducido el concepto de **Experimento**. Los expertos del dominio a veces no saben con exactitud qué campos pueden influir en el suceso. El proceso de encontrar relaciones entre características a primera vista no aparentes puede englobarse en el proceso de *data mining*. Aunque las tareas de visualización y análisis de los datos hemos visto que es fundamental, nuestra herramienta no va a dar soporte a este tipo de tareas, que están cubiertas ya por muchas herramientas del mercado como las provistas por R, la librería Pandas [3] en Python, Rapid Miner [4], Weka o simplemente Excel.

Como se expone en el CU003, el concepto de experimento queremos que vaya un poco más allá: son diferentes configuraciones para el escenario de predicción donde po-

[3]Página de Pandas: pandas.pydata.org/
[4]Página de Rapid Miner: rapidminer.com/

demos usar distintas *features* para construir el modelo. Luego, todos los experimentos construidos podremos desplegarlos de manera transparente al usuario para evaluar su precisión con datos reales. El experimento tiene asociado un estado, que para este propósito sería Evaluación. Con los resultados fruto de monitorizar los experimentos bajo evaluación, podremos decidir qué modelo da mejores resultados, cambiando sus estados a Producción o Descarte respectivamente.

7.1.2. Metamodelo

Una vez está clara la representación, se identifican ya mejor las metaentidades que necesitamos. En la figura 7.3 se muestra el metamodelo hecho con las DSL Tools. Identificamos las siguientes metaentidades:

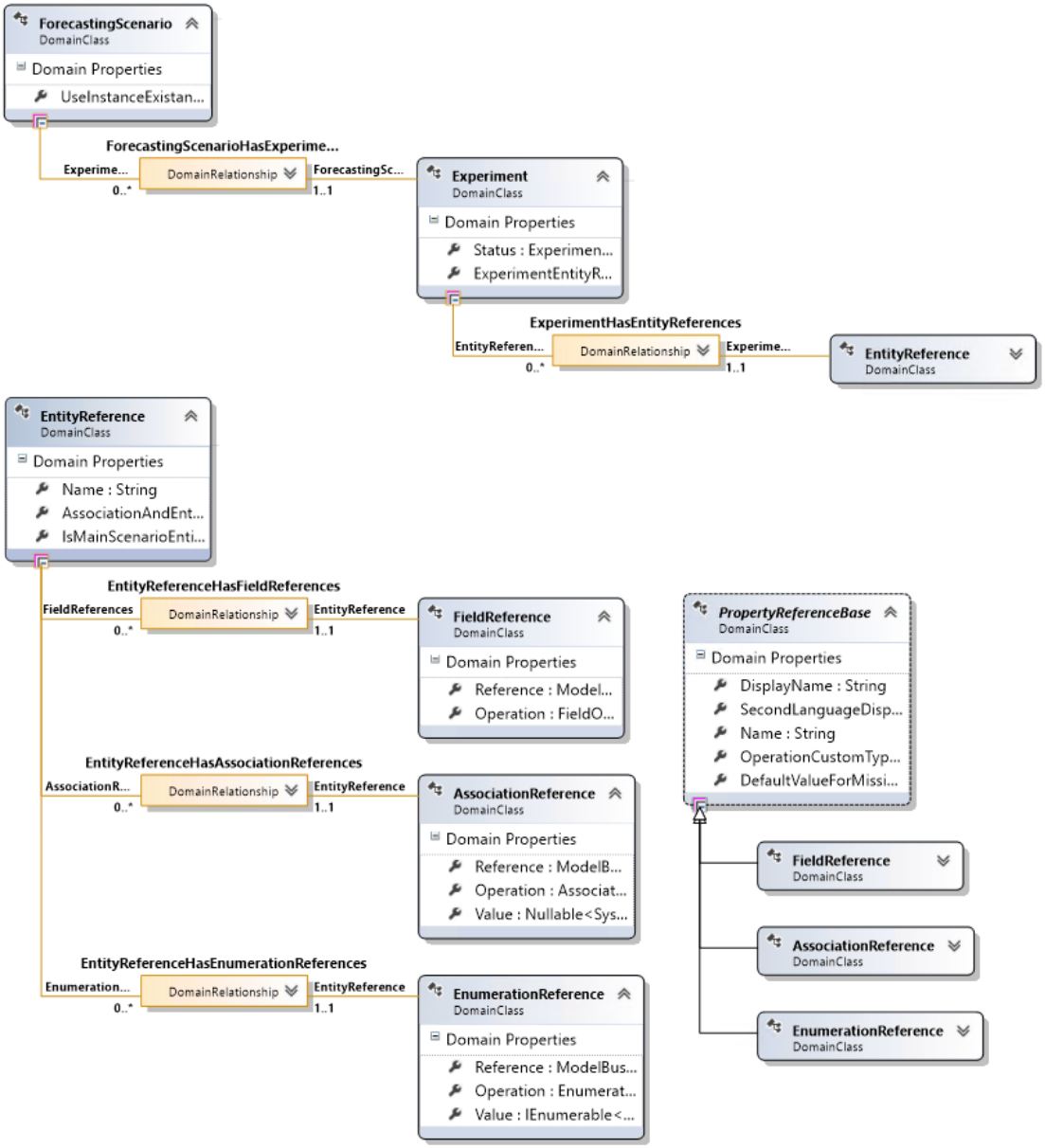

Figura 7.3: Metamodelo del escenario de predicción de sucesos.

- **Escenario de Predicción**: representa el contenedor del resto de conceptos. Está compuesto por al menos un experimento. Esto no se expresa así en el metamodelo, pero se asegura a través de una validación. Hereda de la base creada para todos los escenarios, por lo que contiene una propiedad para relacionar la entidad sobre la que se realiza la predicción y otra para indicar el campo sobre el que se realiza la predicción, si lo que se desea predecir es el valor de un campo. En nuestro caso de predicción de caídas, queremos predecir la futura existencia de una instancia, por lo que marcaremos a *true* la propiedad *UseInstanceExistanceForPrediction*, que representa esto mismo.

- **Experimento**: tiene una enumeración para representar los estados de Evaluación, Producción y Descarte que hemos explicado en el análisis. Además, tiene una propiedad extra, *ExperimentEntityReference*. Esta propiedad no se muestra al usuario del DSML y es calculada. Se emplea para apuntar a la entidad que se autogenera para representar el *dataset* del entrenamiento, como veremos en el apartado de implementación.

- **Referencias a propiedades**: en el DSML de Dominio gobiernan 5 conceptos principalmente: la entidad, la asociación, la generalización, los campos y los enumerados. Salvo la generalización, que se puede obviar ya que una entidad hereda todos los campos, asociaciones y enumerados de su base, todos los demás han sido representado en este DSML para mantener la consistencia. Cada tipo de propiedad posee unas operaciones diferentes.

  Las operaciones permitidas sobre los campos las hemos descritos antes: *Min*, *Max*, *First* y *Last*. Si no se indica ninguna operación, se realizará una proyección del valor, siempre que las validaciones lo permitan. En cuanto a los enumerados, nos interesa solo la operación *Equal* para saber si la propiedad toma un valor concreto. Por otro lado, para las asociaciones nos interesa una operación del tipo *IsRelatedTo*. Esto dota de expresividad sobre todo a las asociaciones con **entidades maestras** [5]. De esta forma, podremos añadir *features* como si un usuario sufre cierta patología, que se registra como una instancia maestra.

[5]Las entidades maestras son entidades convencionales que se proveen al cliente con una serie de instancias por defecto. Por ejemplo, para la entidad País, en el modelo de dominio se proveen por defecto instancias de países como 'España' o 'Francia'. Esto se hace así para facilitar la configuración de los clientes y mejorar la integración de datos, ya que dicha instancia en diferentes clientes tendrá el mismo identificador y será más fácil su explotación. Un cliente puede añadir instancias maestras nuevas si las necesita.

7.1.3. Ciclo de modelado

Vamos a analizar ahora el proceso de desarrollo que se deberá seguir para desarrollar un modelo de predicción usando nuestro DSML de IA. El proceso se detalla en la figura 7.4:

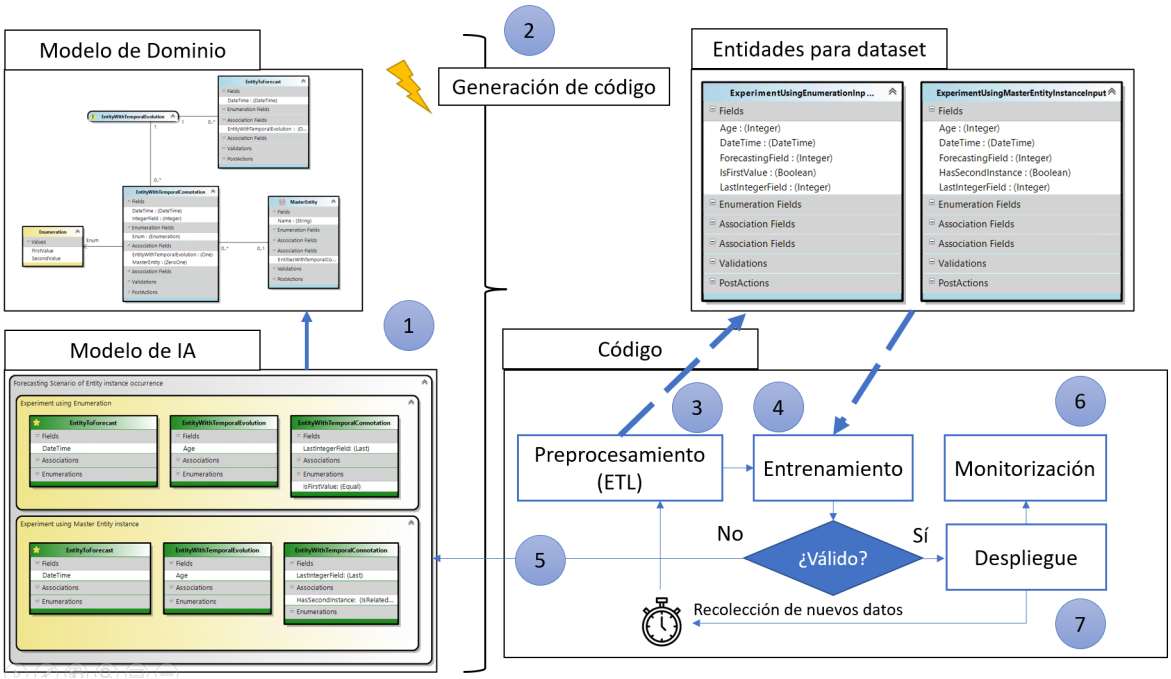

Figura 7.4: Ciclo de desarrollo de escenarios de predicción.

1. El primer paso consiste en modelar el escenario de predicción y los diferentes experimentos. El diseñador, gracias a diversas validaciones que se han implementado, garantizará que el modelo que se construya pueda transformarse en código.

2. Tras esto, a través de un motor de plantillas se realizarán 2 transformaciones. La primera es una transformación M2M, donde a partir del modelo de IA se generarán automáticamente dos entidades de Dominio para cada experimento: una representando la entrada para el entrenamiento y otra para almacenar su salida, es decir, las predicciones hechas usando el modelo. En la tabla SQL de la entidad de entrada se almacenará el resultado del preprocesamiento. La segunda transformación es de modelo a código (M2C) y se encargará de generar todo el código relacionado con el preprocesamiento, entrenamiento y despliegue del modelo.

3. En tercer lugar, pasaremos a ejecutar el código autogenerado. La primera fase de preprocesamiento se encargará de transformar el modelo relacional en el que nos vienen los datos en instancias de la entidad que hemos descrito antes. Aquí se aplicarán las transformaciones que hemos definido para campos, asociaciones y enumerados y se tratarán valores faltantes. Todos estos pasos los hemos englobado en una *pipeline* ETL.

   La adquisición de datos no la contemplamos en la herramienta. Se espera que la entrada del proceso sea una BD siguiendo el esquema de nuestro ERP. Esta BD puede ser de un cliente o el resultado de combinar diferentes BD de clientes.

4. Una vez realizado el preprocesamiento, se podrá lanzar el entrenamiento. La implementación de este se hará con ML.NET, en concreto su versión AutoML, para

abstraerse de los algoritmos disponibles. Parte de los datos disponibles se destinarán al entrenamiento y parte a la validación del modelo resultante.

5. En caso de que el modelo de ML no valide, el modelo hecho usando el DSML debe ir refinándose hasta obtener un resultado válido. La salida de la fase de validación debe ser intuitiva para que sea fácil de entender por qué el modelo no se comporta de acuerdo con lo esperado.

6. En caso de que el modelo de ML tenga el rendimiento esperado, este se podrá desplegar. En cualquier caso, será necesaria su monitorización para reaccionar a errores de predicción, que nos conducirán de nuevo a refinar nuestro modelo en el DSML.

7. Por último, conforme se usa el ERP se irán añadiendo nuevas instancias que se pueden usar para reentrenar el modelo. Periódicamente, se puede programar para que se construya un nuevo modelo. Situaciones más avanzadas donde se requiere que el modelo se reentrene continuamente debido al flujo continuo de datos no se contemplan todavía.

7.2 Detalles de la implementación

7.2.1. Cambios en DSML de IA

Más allá de metamodelar los conceptos de Escenario, Experimento y Referencias, a nivel de DSML se ha hecho un mayor esfuerzo para obtener el comportamiento deseado. Queremos hacer especial mención a los siguientes detalles:

- **Formas y combinación de formas**: las metaentidades descritas en el diseño deben representarse conforme a la notación propuesta. Esta será finalmente como muestra la figura 7.5. El escenario se representa mediante una forma gris y cada experimento mediante una forma amarilla. Las referencias a entidades (*EntityReference*) se representan mediante una forma verde con 3 compartimentos, que representan los campos, enumerados y asociaciones seleccionados. Si sobre ellos se aplica una operación, el nombre de la propiedad irá seguido por el de la operación entre paréntesis. Si la *EntityReference* tiene relacionada la entidad vinculada al escenario, esta aparece con un decorador con una estrella amarilla. Es necesario también indicar qué forma es el padre de qué otra, de tal forma que el escenario sea la única que pueda dejarse caer sobre un diagrama vacío, el experimento solo pueda quedar encima de un escenario y las referencias a entidades queden por encima de todas las anteriores.

- **Editores**: usando las DSL Tools, el editor de una propiedad puede abrir un formulario personalizado. Diferentes formularios han sido necesarios, entre los que destacamos el necesario para seleccionar la entidad asociada a una *EntityReference*. Este formulario parte de la entidad asociada con el escenario de predicción y construye un árbol donde los nodos son entidades relacionadas y la relación padre-hijo representa la asociación entre una entidad y otra. La herencia entre las diferentes entidades también es considerada.

  En la figura 7.6 se muestra este editor para el problema de predicción de caídas. La entidad Caída (*Fall*) es la raíz del árbol. Esta entidad está relacionada con otras como el lugar donde ocurre la caída (*Location*) o el usuario que la sufre. Si seguimos expandiendo el árbol, vemos las entidades relacionadas con *User*. Por ejemplo, el usuario duerme en una cama (*Bed*) que se ubica en una habitación (*Room*).

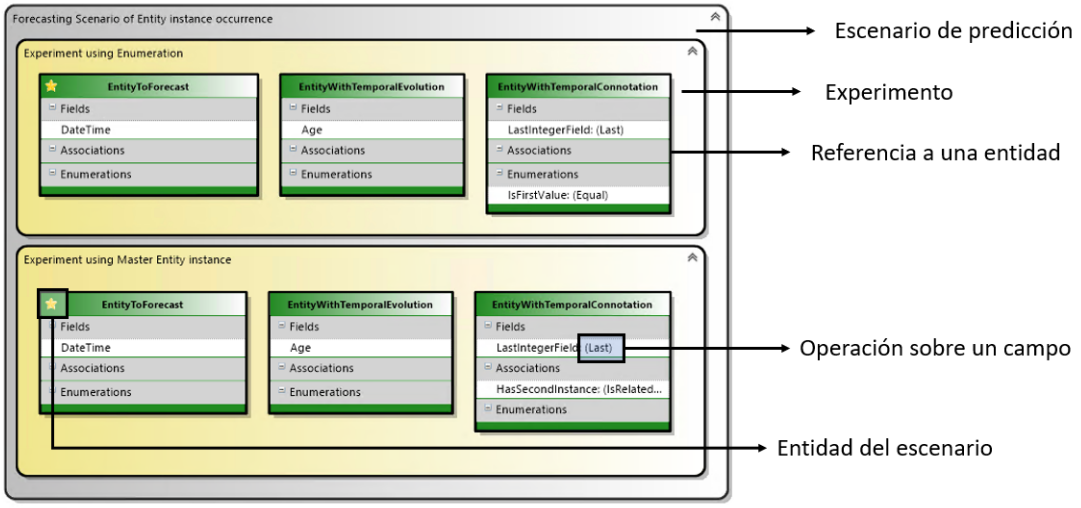

Figura 7.5: Componentes del modelo de predicción.

Cualquiera de estas entidades puede ser seleccionada para emplearse como entrada del modelo de ML. Un detalle importante es almacenar en el modelo del DSML el camino que se ha seguido para llegar desde la entidad a predecir hasta la seleccionada. Esto se debe a que múltiples caminos son posibles para llegar desde una entidad a otra.

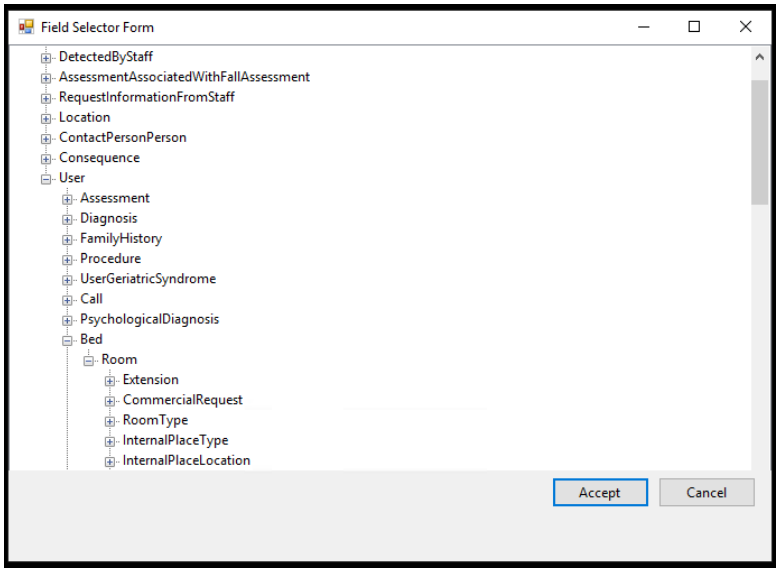

Figura 7.6: Editor para seleccionar entidad.

- **Transformaciones de modelo a modelo**: las DSL Tools de Visual Studio cuentan con un buen soporte para realizar transformaciones de modelo a texto a través de plantillas T4, pero no para realizar transformaciones de modelo a modelo. En nuestro lenguaje, necesitamos transformar los experimentos representados a entidades del DSML de Dominio para representar la entrada y salida del entrenamiento. Este proceso se quiere hacer de manera transparente para el usuario de la herramienta.

  Es cierto que un modelo no deja de ser un fichero XML que se puede editar a través del diseñador del DSML, pero representar la transformación de M2M como una transformación M2T es difícil de mantener. En su lugar, la transformación puede

hacerse de manera programática. Para ello, es necesario modificar el código de serialización de los modelos del nuevo DSML para que cada vez que se serialice un experimento se creen sus entidades relacionadas. La serialización ocurre cuando el modelo representado como una jerarquía de objetos es transformado para persistirse como un fichero en XML. Hacer esto penaliza el rendimiento del diseñador, pero la experiencia de trabajo sigue siendo buena y la transformación es más fácil de evolucionar.

- **Validaciones**: en el apartado **2.2 Beneficios de seguir una estrategia MDD** hemos mencionado que uno de los beneficios de usar estrategias MDD es la capacidad de incluir validaciones sobre los modelos. En nuestro DSML también se han añadido validaciones, que se hacen mediante programación. En la figura 7.7 se muestran dos validaciones relacionadas con las operaciones sobre campos. Las operaciones Primero y Último no pueden llevarse a cabo sobre entidades si estas no tienen connotación temporal porque es necesario ordenar por un campo de tipo fecha para tomar el primer o el último valor. Lo mismo ocurre para las operaciones Máximo y Mínimo, que solo pueden aplicarse a campos de tipo numérico.

  Mediante validaciones se controlan también los valores faltantes. Por ejemplo, si se selecciona el resultado de una escala, se obliga a que se introduzca un valor para asignarlo en caso de que al usuario no se le haya realizado nunca esa escala. El usuario del DSML puede introducir cualquier valor, aunque es habitual emplear valores medios. Los valores introducidos también se validan para asegurar la consistencia ya que esa propiedad acepta texto libre, de tal forma que no se permite poner como 'Valor para valores faltantes' un *true* si el campo es numérico.

```
/// <summary> Validates that the Last/First operations are only used in a Entity ...
[ValidationMethod(ValidationCategories.Menu | ValidationCategories.Save)]
0 references | Víctor Alberto Iranzo Jiménez, 87 days ago | 1 author, 1 change
internal void ValidateLastFirstOperation(ValidationContext context)
{
    if (this.Operation != FieldOperation.Last || this.Operation != FieldOperation.First)
    {
        return;
    }

    if (this.EntityReference.Entity.EntityType != EntityType.EntityWithTemporalConnotation)
    {
        context.LogError(
            string.Format(CultureInfo.CurrentCulture, ValidationResources.FirstAndLastOperationsCanBeUsedOnlyAtEntityWithTemporalConnotation),
            nameof(ValidationResources.FirstAndLastOperationsCanBeUsedOnlyAtEntityWithTemporalConnotation),
            this);
    }
}

/// <summary> Validates that the Min/Max operations are only used with numeric f ...
[ValidationMethod(ValidationCategories.Menu | ValidationCategories.Save)]
0 references | Víctor Alberto Iranzo Jiménez, 59 days ago | 1 author, 1 change
internal void ValidateMinMaxOperation(ValidationContext context)
{
    if (this.Operation != FieldOperation.Min || this.Operation != FieldOperation.Max)
    {
        return;
    }

    if (!this.Field.Type.IsNumericType())
    {
        context.LogError(
            string.Format(CultureInfo.CurrentCulture, ValidationResources.MinAndMaxOperationsMustBeUsedOnNumericFields),
            nameof(ValidationResources.MinAndMaxOperationsMustBeUsedOnNumericFields),
            this);
    }
}
```

Figura 7.7: Validaciones sobre operaciones.

7.2.2. Código autogenerado

A diferencia del escenario anterior, donde la mayoría de los cambios se han realizado a nivel de la librería *core*, en este escenario el peso de la implementación recae sobre todo en el código autogenerado.

Por cada experimento en estado de evaluación o producción se generará un proyecto .NET Core. El proyecto tendrá la siguiente estructura, que se muestra en la figura 7.8:

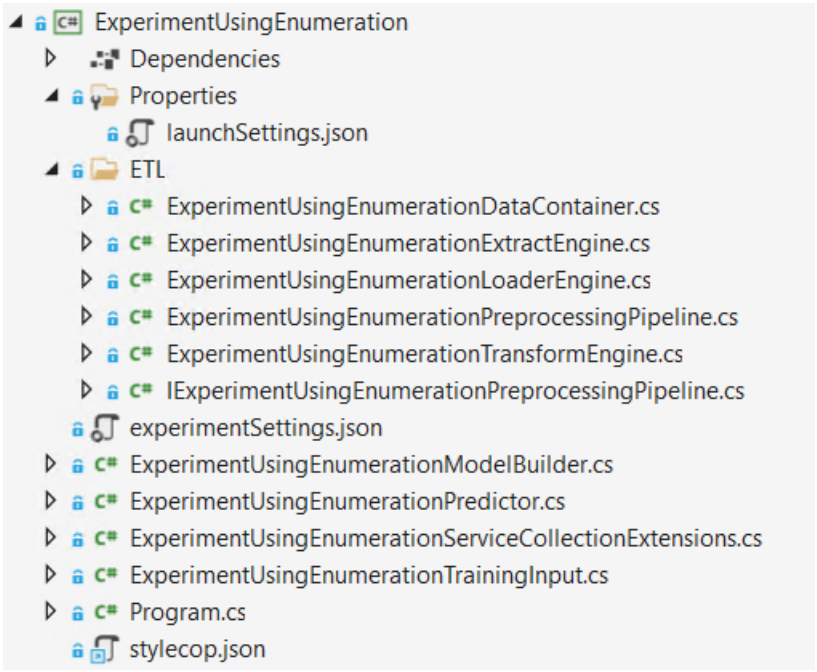

Figura 7.8: Estructura del código generado para un experimento.

- En la carpeta ETL se encuentran todas las clases para el preprocesamiento de datos. Las clases encargadas de la extracción de datos (*ExtractEngine.cs*) y de la carga de datos (*LoaderEngine.cs*) son triviales y rondan cada una las 100 líneas de código. Durante la extracción, se leen todas las instancias de entidades involucradas y se almacenan para facilitar su consumo en una estructura de datos que autogeneramos, el *DataContainer.cs*.

  Autogenerar la clase de transformación *TransformEngine.cs* ha requerido un mayor esfuerzo ya que es necesario prestar atención a las asociaciones entre las entidades para acceder a ellas de manera ordenada y atendiendo a su cardinalidad. Se hace un uso intensivo del lenguaje LINQ (lenguaje integrado de consultas), que varía según la operación seleccionada. Las plantillas encargadas de transformar el modelo en esta clase se organizan según el tipo de propiedad (enumerados, campos y asociaciones) para facilitar el mantenimiento. Para hacer una referencia a su complejidad, el código de esta clase ronda las 500 líneas.

  Por último, en esta carpeta encontramos una interfaz *IPreprocessingPipeline.cs* y una clase *PreprocessingPipeline.cs*. La clase implementa la interfaz y se encarga de orquestar la invocación de los componentes que forman el proceso ETL. Además, nos permite abstraer la interfaz de la implementación y usar inyección de dependencias.

- La clase *ModelBuilder.cs* es la encargada de construir un modelo de ML a partir de los datos preprocesados. Como hemos mencionado, empleamos ML.NET para el

entrenamiento. En la figura 7.9 se muestra el método *Train()* que expone. Primero, los datos preprocesados se cargan en memoria usando la clase *LoaderEngine.cs*. El conjunto de datos se divide en un conjunto para la validación y otro para en entrenamiento. Lo que se desea predecir lo interpretamos como un problema de clasificación binaria sobre un campo *booleano* llamado *ForecastingField*. El experimento lo configuramos para que haga un entrenamiento de un máximo de 20 minutos. Una vez terminado ese tiempo, se imprimirán las métricas que ML.NET devuelve y se guardará el modelo como un fichero ZIP.

```
public void Train()
{
    IEnumerable<ExperimentUsingEnumerationTrainingInput> data = this.loaderEngine.LoadData();
    IDataView dataView = mlContext.Data.LoadFromEnumerable(data);

    DataOperationsCatalog.TrainTestData dataSplit = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2);
    IDataView trainData = dataSplit.TrainSet;
    IDataView testData = dataSplit.TestSet;

    BinaryExperimentSettings experimentSettings = new BinaryExperimentSettings()
    {
        MaxExperimentTimeInSeconds = 1200,
    };

    BinaryClassificationExperiment experiment = mlContext.Auto().CreateBinaryClassificationExperiment(experimentSettings);

    ExperimentResult<BinaryClassificationMetrics> experimentResult = experiment.Execute(
        trainData: trainData,
        validationData: testData,
        labelColumnName: nameof(ExperimentUsingEnumerationInput.ForecastingField));

    this.logger.LogInformation($"Best algorithm found: {experimentResult.BestRun.TrainerName}");
    this.logger.LogInformation($"Accuracy: {experimentResult.BestRun.ValidationMetrics.Accuracy:0.##}");
    this.logger.LogInformation($"Area under Precision Recall Curve: {experimentResult.BestRun.ValidationMetrics.AreaUnderPrecisionRecallCurve:0.##}");
    this.logger.LogInformation($"Area under ROC Curve: {experimentResult.BestRun.ValidationMetrics.AreaUnderRocCurve:0.##}");
    this.logger.LogInformation($"F1 score: {experimentResult.BestRun.ValidationMetrics.F1Score:0.##}");
    this.logger.LogInformation($"Negative precision: {experimentResult.BestRun.ValidationMetrics.NegativePrecision:0.##}");
    this.logger.LogInformation($"Negative recall: {experimentResult.BestRun.ValidationMetrics.NegativeRecall:0.##}");
    this.logger.LogInformation($"Positive precision: {experimentResult.BestRun.ValidationMetrics.PositivePrecision:0.##}");
    this.logger.LogInformation($"Positive recall: {experimentResult.BestRun.ValidationMetrics.PositiveRecall:0.##}");

    mlContext.Model.Save(experimentResult.BestRun.Model, dataView.Schema, "..\\..\\ExperimentUsingEnumerationModel.zip");
}
```

Figura 7.9: Entrenamiento del modelo usando ML.NET.

- La clase *Predictor.cs* permite hacer predicciones usando el modelo entrenado. Las predicciones hechas se almacenarán automáticamente como instancias de la entidad de salida del experimento.
- La clase *ServiceCollectionExtensions.cs* contiene métodos auxiliares para el registro de dependencias necesario para hacer inyección de dependencias.
- La clase *TrainingInput.cs* es una clase auxiliar necesaria para usar ML.NET. Este es bastante restrictivo en cuanto al tipo de las *features* para hacer un entrenamiento. Por ejemplo, usando nuestro DSML de Dominio, representamos las propiedades numéricas con el tipo de C# *decimal*. ML.NET no acepta este tipo, por lo que es necesario hacer una conversión a *float*. Esta clase es la responsable de representar este tipo de conversiones necesarias.
- Por último, la clase Program representa el punto de entrada del proyecto al tener el método *Main()*. Esta clase invoca a las fases de preprocesamiento, entrenamiento o predicción según los *flags* que se pasen como argumento, para facilitar la explotación del experimento.

7.2.3. Generación de casos negativos

Hemos mencionado que la complejidad de la clase de transformación procede de la variedad de operación soportadas. Sin embargo, su complejidad también radica en la generación de **casos negativos**. En los casos por ahora soportados, los eventos que deseamos predecir están asociados a la creación de una instancia de una entidad. Sin embargo, nuestro *dataset* no puede constituirse únicamente por los datos relacionados a estas instancias, que representan solo los **casos positivos**. Por ejemplo, si queremos predecir la hospitalización de los usuarios de acuerdo con sus patologías, tomaremos las hospitalizaciones existentes y veremos las patologías que el usuario tiene asociadas para crear un caso positivo. De los usuarios que no han sido hospitalizados, tomaremos sus patologías y generaremos un caso negativo. En nuestro *dataset*, los datos positivos tendrán a *true* la *label* a predecir, llamada *ForecastingField*, mientras que los casos negativos lo tendrán a *false* porque representan situaciones donde no ocurre el evento de interés.

La estrategia empleada para la generación casos negativos es muy importante ya que puede dar lugar a un *dataset* no equilibrado (*imbalanced*). Esto significa que hay una diferencia considerable en el número de instancias que pertenecen a una clase y el número de las que pertenecen a otra. Algunos problemas son así por su propia naturaleza, como es el caso de la detección de fraudes ya que son eventos que ocurren muy ocasionalmente. Una de las técnicas más habituales para abordar este problema son las técnicas de **remuestreo** (*resampling*), que puede basarse en eliminar instancias de la clase más representada (*undersampling*) o en crear nuevas instancias de la clase menos representada (*oversampling*). Otras opciones son tratar de recolectar más datos, hacer un entrenamiento con sensibilidad a los errores de clasificación o ensamblar diferentes clasificadores débiles[54].

En nuestro caso, vamos a seguir una estrategia de *undersampling*. Siguiendo con el ejemplo de hospitalizaciones, podríamos generar un caso negativo para un usuario por cada día que el usuario no está hospitalizado, pero esto nos conduciría a un número desorbitado de casos negativos. ¿Por qué no generar el caso negativo por cada mes o año? En nuestro DSML, se puede configurar esto de tal forma que se generen casos negativos por cada día, mes o año que no ocurra el evento. El experto de dominio deberá considerar cómo de frecuente es el evento a predecir para optar por una estrategia u otra.

También es posible seguir una estrategia que denominamos **sin connotación temporal**. Usando esta estrategia, se divide el conjunto de usuarios según si han sido hospitalizados alguna vez o no. Con cada uno de los usuarios que nunca han sido hospitalizados se crea una instancia en el *dataset* como un caso negativo. La generación de casos positivos es la habitual: se toman las hospitalizaciones y se obtienen las patologías que el usuario tenía en dicho momento. La desventaja de usar esta estrategia es que el tamaño del *dataset* se reduce, lo que puede dar problemas de sobreentrenamiento al no ser capaz de generalizar bien.

Todos estos detalles forman parte de la configuración del experimento, que se realiza en el modelado a través de la ventana de propiedades de Visual Studio. Esta ventana se muestra en la figura 7.10.

En la imagen se observa que el experimento ha sido marcado para tener connotación temporal y los casos negativos se generarán mensualmente. También se indica que se invocará código *custom* para la generación de casos negativos. Este *flag* se traduce en un método *booleano* que deberá implementar un programador en una clase parcial [6]. En

[6]Las clases parciales son una característica de C# que permite dividir el contenido de una clase en varios ficheros. Esto es útil en escenarios de generación de código donde parte de la clase es autogenerada y otra parte mínima debe implementarse manualmente. Para más información, se puede consultar docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/partial-classes-and-methods.

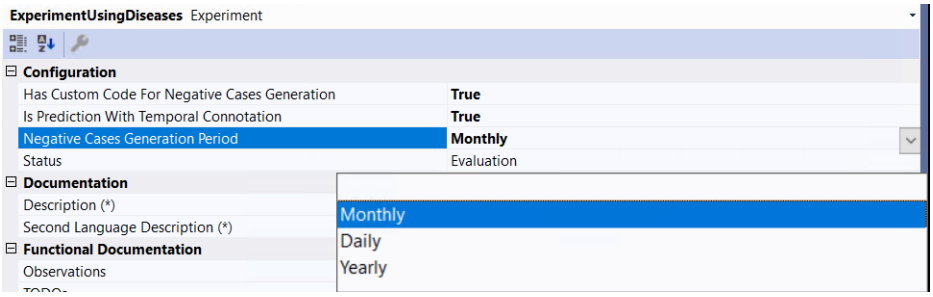

Figura 7.10: Configuración del experimento.

nuestro caso, necesitamos usar este *flag* para evitar generar casos de prueba cuando el usuario no está en la residencia o ha fallecido, que debe consultarse a través de una lógica especial y difícil de generalizar al DSML.

7.3 Pruebas

Los modelos hechos usando el DSML de IA se basan en modelos construidos con el DSML de Dominio. Para la realización de pruebas vamos a crear un dominio inventado para representar diferentes situaciones. Nos interesa probar especialmente la generación de código hecha a partir de los modelos, las operaciones disponibles y las diferentes configuraciones que un experimento puede tomar. El dominio que nos hemos inventado se muestra en la figura 7.11.

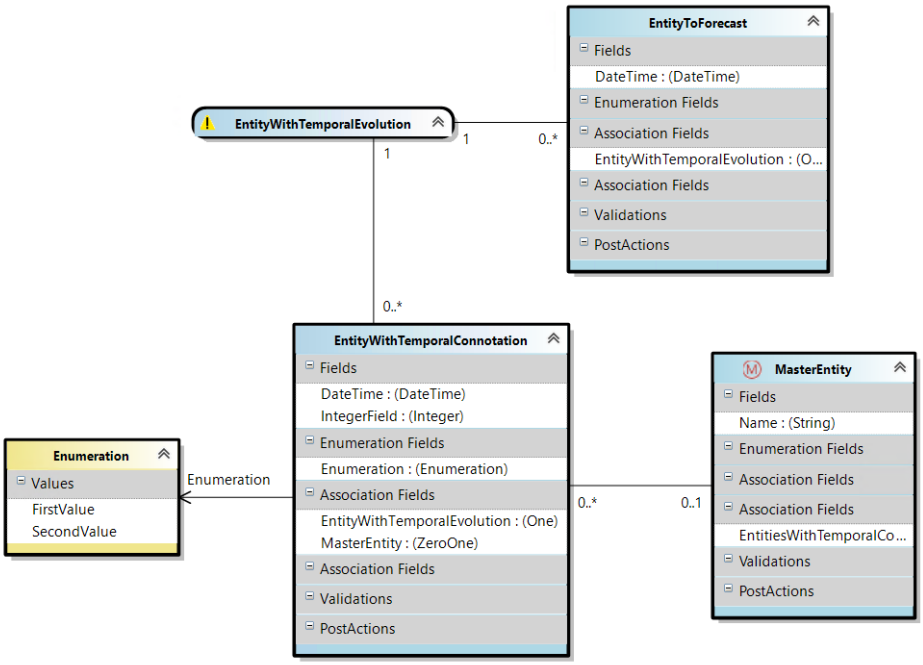

Figura 7.11: Dominio para la realización de pruebas.

Dentro de este dominio, el nombre de las entidades representa el propósito que se dará luego a ellas usando el DSML de IA. De esta forma, encontramos en el modelo la entidad sobre la que se hace la predicción (*EntityToForecast*), la entidad con evolución temporal (*EntityWithTemporalEvolution*), que es un símil de la entidad Usuario, una enti-

dad con connotación temporal (*EntityWithTemporalConnotation*), que es una entidad con un campo de tipo fecha, un enumerado y una entidad maestra asociada.

Sobre este modelo de dominio se ha construido un escenario de predicción con dos experimentos, que se muestran en la figura 7.5. Estos modelos se han usado para depurar la generación de código asociada al DSML, que se ha ido perfeccionando progresivamente hasta hacer que el código resultante compile y funcione.

Por último, un total de 10 pruebas unitarias se han creado probando las clases autogeneradas de estos experimentos. Se ha hecho inciso sobre todo en probar la *pipeline* para el preprocesamiento de datos.

7.4 Caso práctico: predicción de caídas

En el apartado **5.4 Escenario de predicción de caídas** hemos hecho un análisis de la literatura sobre los factores que pueden predisponer la caída de un residente. Ahora, vamos a ver paso a paso cómo podemos construir y explotar un modelo de ML usando nuestro DSML. Diferentes experimentos serán construidos para comparar luego sus resultados.

7.4.1. Modelado del problema

El proceso de modelado de escenarios de predicción se resume en los siguientes pasos:

1. En primer lugar, debemos crear un diagrama de IA usando la extensión de Visual Studio asociada al DSML de Inteligencia Artificial.

2. Utilizando la *toolbox* (figura 7.12) crearemos un escenario de predicción. En el escenario seleccionaremos la entidad de dominio sobre la que se realiza la predicción, en este caso la entidad Caída. Será necesario rellenar algunos metadatos como la descripción del escenario o su *DisplayName*.

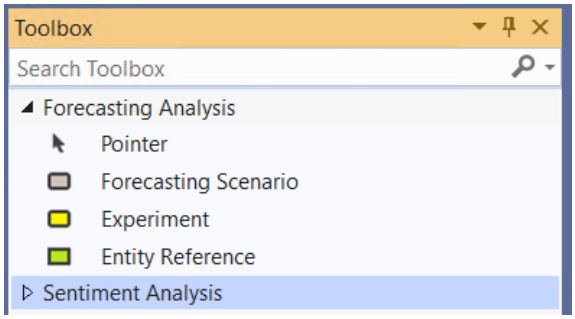

Figura 7.12: *Toolbox* de Visual Studio para la predicción de sucesos.

3. Haciendo también *drag-and-drop*, crearemos un experimento sobre el escenario. El estado del experimento lo dejaremos en 'Evaluación' y configuraremos el experimento según si queremos que tenga connotación temporal y con una estrategia para la generación de casos negativos. Para más detalle, se puede consultar la sección **7.2.3 Generación de casos negativos**.

4. Una vez hecho esto, podemos añadir las referencias a entidades y propiedades que convenga para construir nuestro modelo de ML. La entidad sobre la que se realiza la predicción, que es la entidad Caída, debe ser incluida o fallará una validación del modelo. Si se configura el experimento para que tenga connotación temporal, también se debe incluir el campo fecha y hora de la caída o en caso contrario fallará una validación del modelo.

   Siguiendo el análisis, las patologías del usuario son los factores que más predisponen a su caída. En nuestra aplicación, la entidad Caída está relacionada con un usuario. El usuario a su vez está relacionado con la entidad Patología con una cardinalidad 0..N para representar las enfermedades que sufre. La entidad Patología (*GeneralDisease*) está relacionada con otra entidad maestra denominada 'Enfermedad CIE' (en inglés, *ICDDiagnosis*). CIE-10 [7] es una clasificación internacional de enfermedades que otorga una categoría y un código único a cada afección. Como es una entidad maestra, encontraremos en ella los datos por defecto que necesitamos como instancias de la entidad, tales como 'G30 Alzheimer' o 'H40 Glaucoma'. Estas entidades están representadas en el modelo de dominio de la figura 7.13.

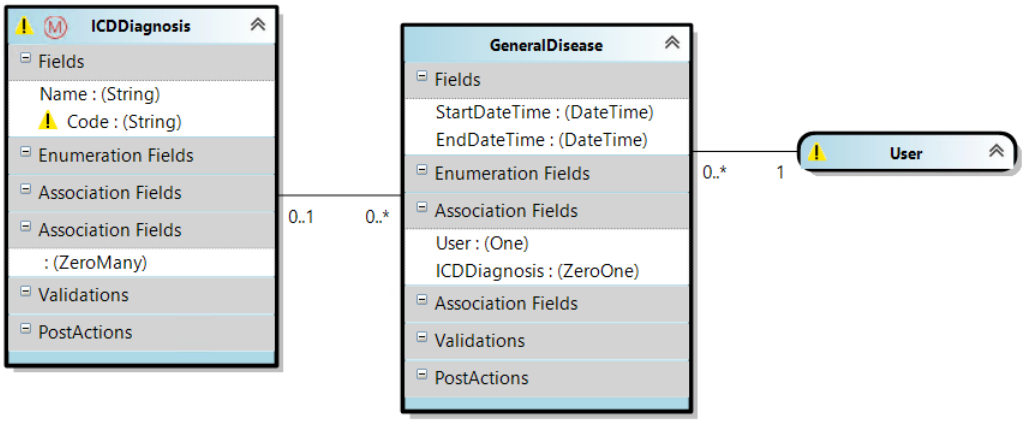

Figura 7.13: Modelo de dominio para representar las enfermedades de un usuario.

Por tanto, para tener en nuestro modelo de ML una *feature* como 'HasAlzheimer' debemos añadir una referencia a la entidad *GeneralDisease* y una referencia a la asociación entre *GeneralDisease* e *ICDDiagnosis*. A esta asociación le aplicaremos la operación 'está relacionado con' y seleccionaremos la instancia maestra 'G30 Alzheimer' a través del editor disponible.

Otra *feature* que nos resultará interesante es el resultado de una escala. En los modelos de nuestra aplicación, una escala como la de Barthel están representadas por una entidad (*BarthelAssessment*), que está relacionada con la entidad *User*. Una escala en nuestra aplicación puede interpretarse como un conjunto de preguntas, donde cada pregunta tiene un conjunto de respuestas posibles con un valor ponderado. El resultado de la escala es normalmente un campo numérico que se calcula sumando los valores asociados a las respuestas dadas. Así, en nuestro DSML tendremos que añadir la referencia a la entidad *BarthelAssessment*. En ella, añadiremos después la referencia a su campo *Result* y aplicaremos la operación 'Último' para que nos devuelva el resultado de la última escala Barthel hecha antes del momento de la caída.

[7] Buscador de CIE-10: https://eciemaps.mscbs.gob.es/ecieMaps/browser/index_10_mc.html

En la figura 7.14 se muestra el primer experimento hecho, donde se han seleccionado un buen número de enfermedades, siguiendo el análisis. Se observan algunas que indican si el usuario sufre de artrosis, si el usuario ha sufrido un infarto, si tiene epilepsia, etc. También se ha añadido el campo edad del usuario y su género, que viene representado como un enumerado.

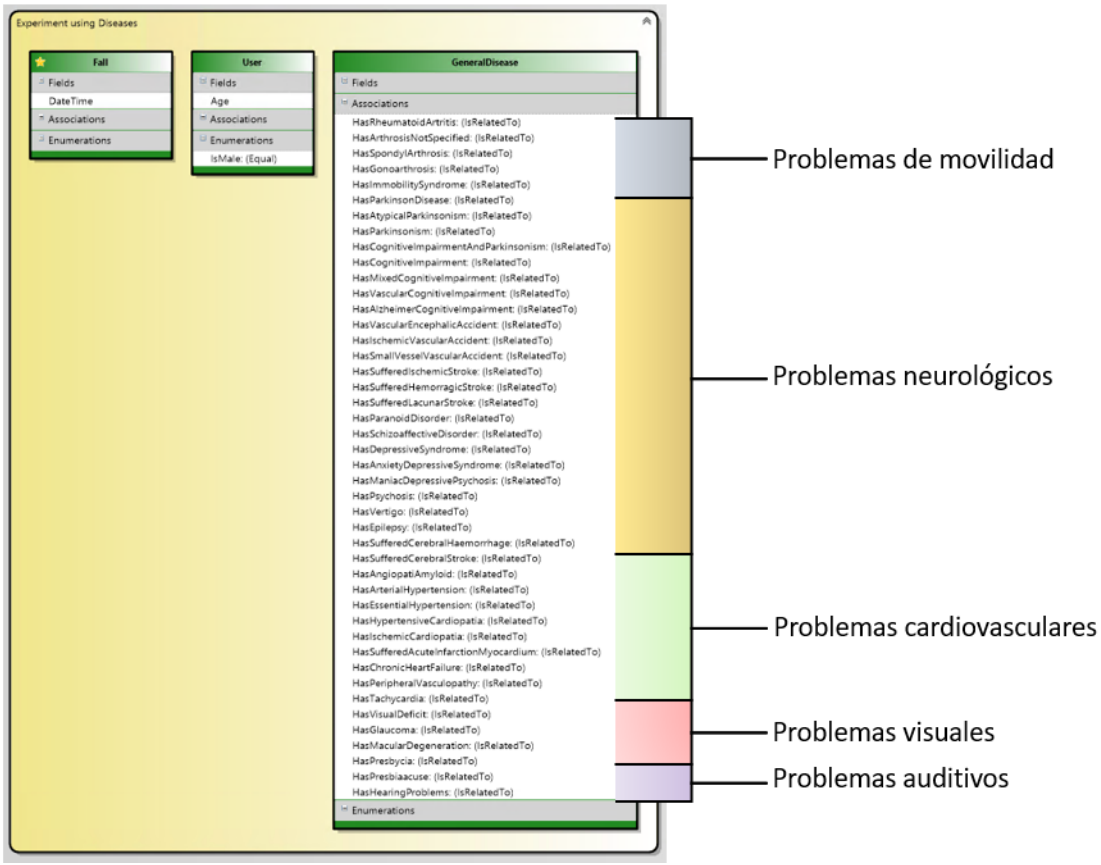

Figura 7.14: Experimento usando enfermedades del Usuario.

5. Conforme se modela usando el DSML, de manera transparente se realiza la transformación M2M para generar las entidades de entrada y salida del modelo de ML. En la figura 7.15 se muestran las entidades que están asociadas al experimento anterior.

6. Una vez modelados los experimentos, generaremos código del modelo del DSML. Este paso lo tenemos cubierto empleando diferentes *scripts* que no vamos a describir.

7. El código autogenerado lo emplearemos para realizar el preprocesamiento de datos, invocar el entrenamiento y realizar predicciones, tal y como se ha explicado en **7.1.3 Ciclo de modelado**. Estas acciones se llevan a cabo a través de Visual Studio, invocando al método *Main()* de la clase *Program* de cada experimento. La base de datos de la que se obtienen los datos para el entrenamiento, que es una BD siguiendo el esquema relacional del ERP, se indica como una cadena de conexión en un fichero llamado *experimentSettings.json*. Las predicciones hechas se almacenarán en la tabla de la entidad de salida del experimento.

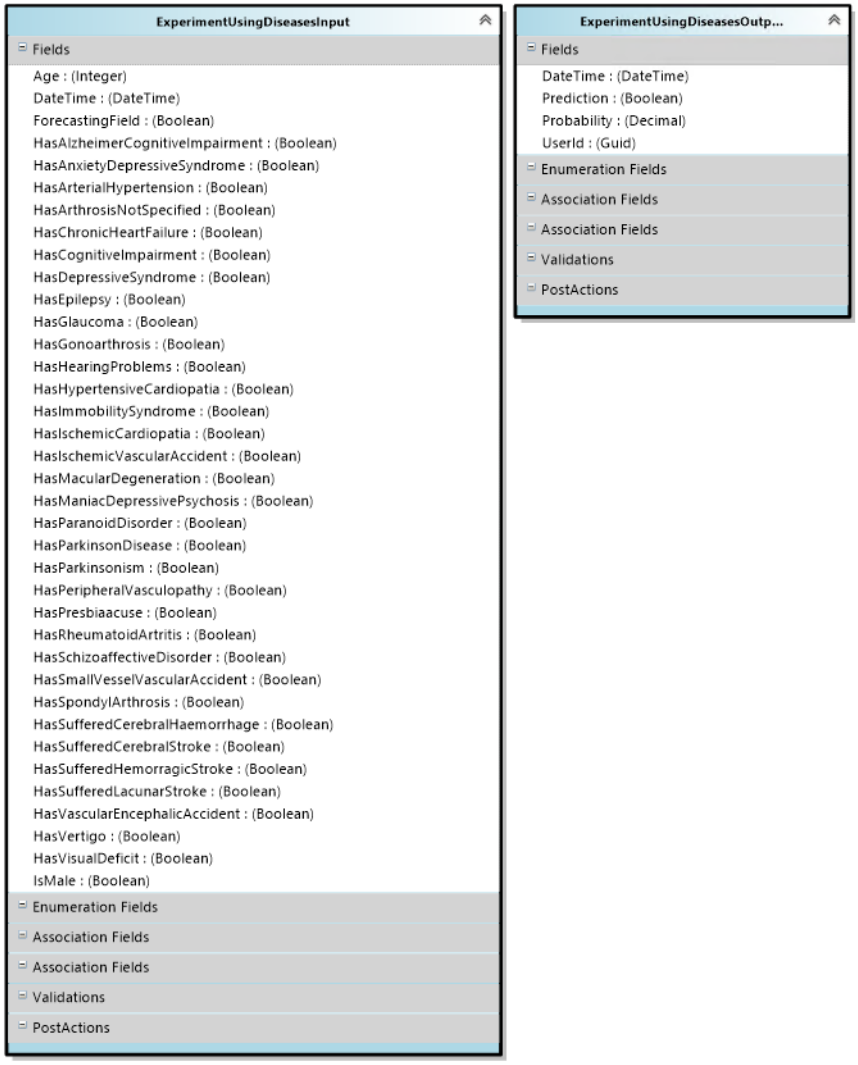

Figura 7.15: Entidades de entrada y salida del modelo de ML.

Siguiendo este proceso, se han creado para el escenario de predicción de caídas 4 experimentos diferentes. Se ha hecho esto para explotar la vertiente experimental que nos ofrece el DSML y probar diferentes configuraciones y entradas. En la próxima sección discutiremos los resultados que obtenemos con cada uno de ellos. Los experimentos hechos son los siguientes:

- **Experimento usando enfermedades** (figura 7.14): este experimento construye el modelo de ML solo con las enfermedades que se describen en **5.4 Escenario de predicción de caídas**: enfermedades cardiovasculares, neurológicas, de movilidad, visuales y auditivas. También se incluye el sexo y edad del usuario, que son factores que se incluirán en todos los experimentos. El experimento ha sido configurado para tener connotación temporal y generar casos negativos mensualmente.
- **Experimento usando enfermedades y escalas** (figura 7.16): al experimento anterior se le han añadido las escalas que describe la literatura como influyentes: la escala de Barthel, la escala de Lawton y Brody y la escala de Yesavage. Este experimento ha sido configurado para no tener connotación temporal.

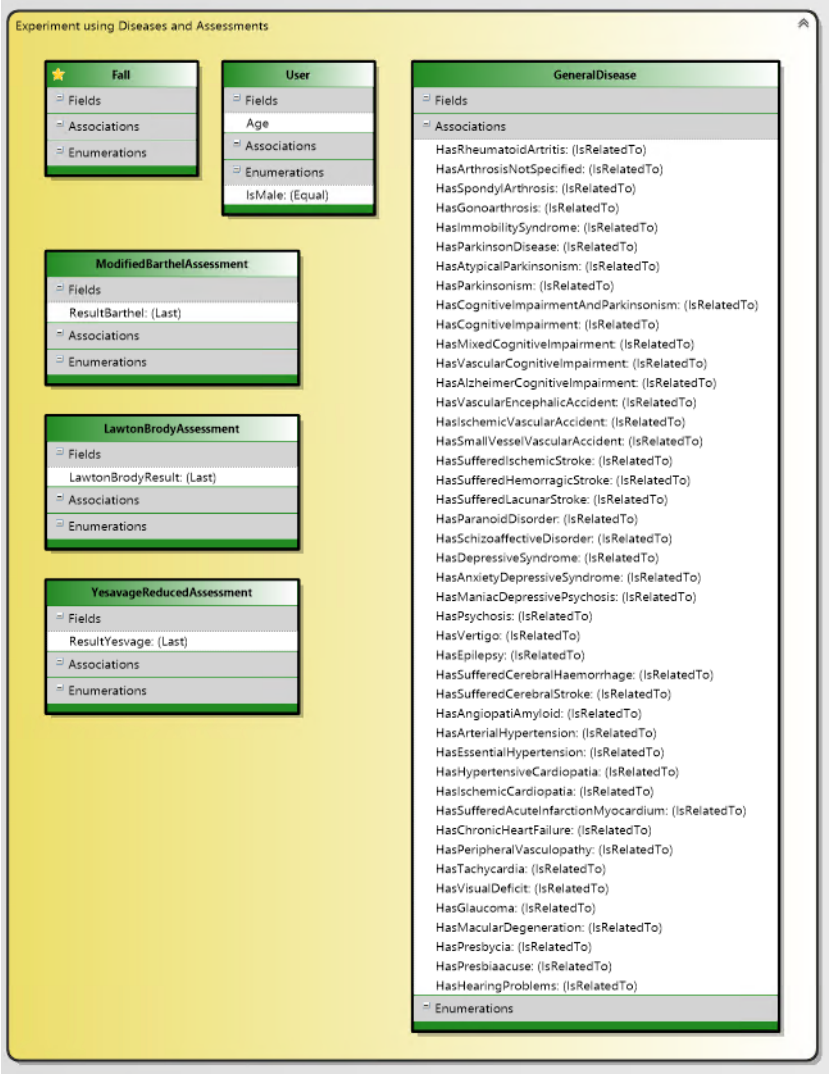

Figura 7.16: Experimento usando enfermedades y escalas del Usuario.

- **Experimento usando enfermedades y escalas adicionales** (figura 7.16): al experimento anterior se le han añadido las escalas de Downton, Tinneti, *Get Up & Go*, Lobo y Pfeiffer. La escala de Downton [8] valora el riesgo de caída de un usuario. Las escalas de Tinetti [9] y *Get Up & Go* [10] valoran la movilidad del paciente, mientras que las escalas de Lobo [11] y de Pfeiffer [12] miden su estado cognitivo. También se han añadido algunas patologías: si el usuario tiene asma, si tiende al aislamiento, si tiene incontinencia urinaria o si padece de insomnio. Aunque no hemos encontrado referencias a estos factores en la literatura consultada, diferentes personas de la empresa especializadas en este dominio nos han transmitido que pueden influir. El experimento está configurado para tener connotación temporal y generar casos negativos anualmente.

[8] Cuestionario de la escala Downton: campusvirtual.farmacoterapia-sanidadmadrid.org/CURSOS/logic/Consejeria_sanidad/osteoporosis/educacion_salud/pdf/EscaladeRiesgodecaidas.pdf

[9] Cuestionario de la escala Tinetti: csantantoni.com/wp-content/uploads/2015/11/Escala-Tinetti.-Equilibrio-y-Marcha.pdf

[10] Cuestionario de la escala *Get Up & Go*: inger.gob.mx/pluginfile.php/1690/mod_resource/content/4/Archivos/Instrumentos/22_Get_Up_And_Go.pdf

[11] Cuestionario de la escala Lobo: infogerontologia.com/documents/vgi/escalas/mini_mental.pdf

[12] Cuestionario de la escala Pfeiffer: hipocampo.org/pfeiffer.asp

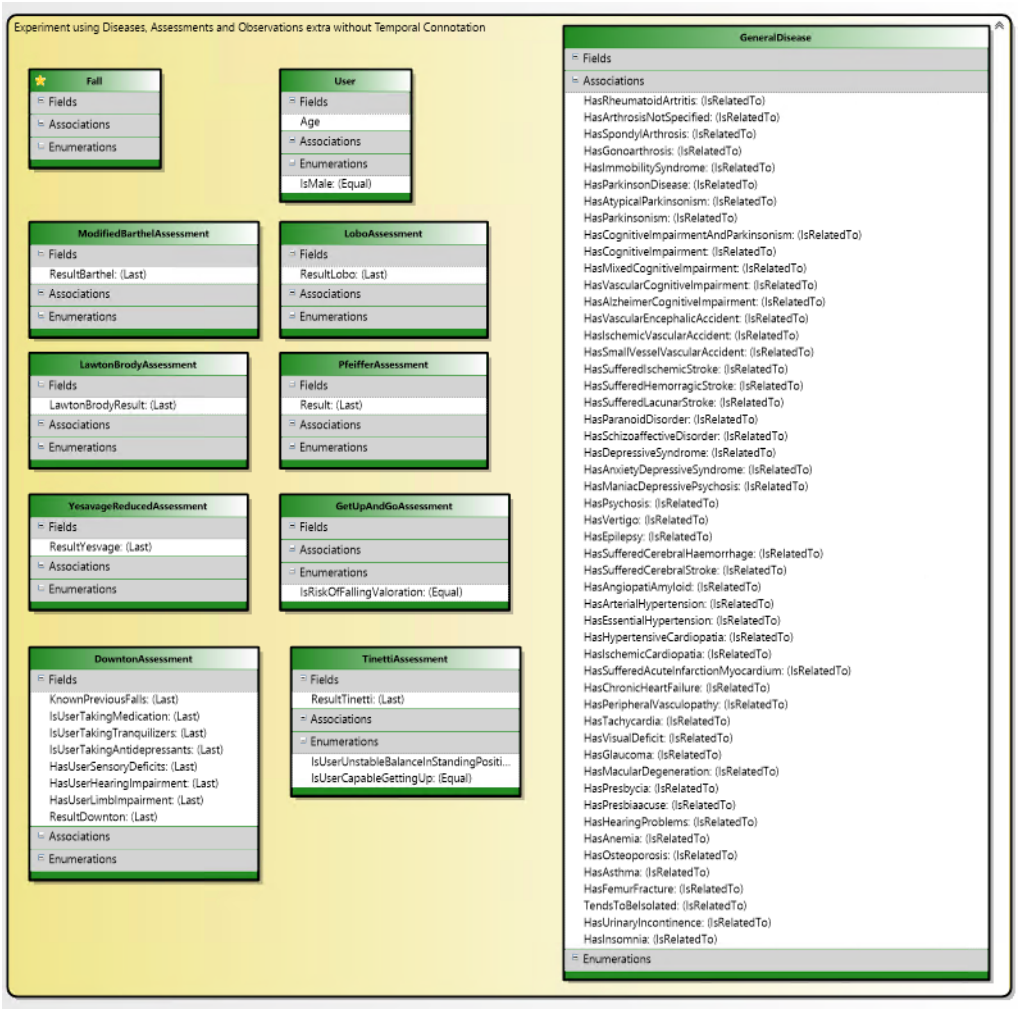

Figura 7.17: Experimento usando enfermedades y escalas adicionales.

- **Experimento usando enfermedades y escalas adicionales sin connotación temporal** (figura 7.16): este experimento es idéntico al anterior y solo se diferencia en que no tiene connotación temporal.

7.4.2. Resultados obtenidos

Para ejecutar los experimentos hechos se va a utilizar una BD de un cliente real del ERP. Los datos de esta residencia han sido anonimizados, borrando datos personales como nombres, apellidos o DNI que puedan identificar a una persona. En esta BD hay un total de 478 usuarios registrados, que han sido atendidos en la residencia en algún momento. Las caídas son eventos poco frecuentes y solo se tiene registro de 54 de ellas, teniendo la primera fecha de Mayo de 2015 y la última fecha de Abril de 2021.

A partir de estos datos y atendiendo a la configuración de cada experimento, se generará un conjunto de datos para el entrenamiento mayor o menor, tanto en *features* como en instancias. Como hemos mencionado en la sección anterior, las fases de preprocesamiento de datos y entrenamiento se ejecutan de manera independiente ya que los datos preprocesados son almacenados en una tabla SQL que luego es leída para realizar el entrenamiento. Por tanto, una vez hecho el preprocesamiento podemos inspeccionar esta tabla que conforma el *dataset* para el entrenamiento. El DSML en sí no soporta la inspección del *dataset*, pero usando scripts SQL podemos obtener información de gran valor, como la distribución de las *features*. En la tabla 7.1 se muestra la distribución para el experimento usando enfermedades y escalas adicionales sin connotación temporal.

Atendiendo a esta tabla, podemos eliminar directamente aquellas características que consideremos no representativas, para facilitar la construcción del modelo de ML. Por ejemplo, las características con algún valor 0 o cercano a 0, como si el usuario tiende al aislamiento o si el usuario tiene asma, podrían ser eliminadas. Este proceso de refinamiento de los datos se puede hacer de manera gradual hasta dejar solo las *features* más representativas. También se puede tratar de recolectar más datos. Por ejemplo, solo hay un registro de un usuario que tiene asma. Si este usuario ha sufrido una caída, podríamos caer en un error de sobreentrenamiento si el modelo asume que los usuarios con asma tienen mayor probabilidad de caerse cuando no tenemos suficientes datos para afirmar esto.

# Features	# Positivos	# Negativos
Campo a predecir	54	447
Usuario es hombre	168	333
Usuario tiene una valoración de Riesgo de Caída	29	472
Usuario está inestable en bipedestación	29	472
Usuario tiene caídas previas	141	360
Usuario tiene deterioro cognitivo por Alzheimer	45	456
Usuario tiene Anemia	1	500
Usuario sufre una Angiopatía Amiloide	1	500
Usuario sufre de Hipertensión Arterial	12	489
Usuario sufre Artrosis sin especificar donde	12	489
Usuario tiene Asma	1	500
Usuario tiene Parkinsonismo atípico	8	493
Usuario tiene insuficiencia cardiaca crónica	1	500
Usuario tiene deterioro cognitivo	56	445
Usuario sufre un síndrome depresivo	5	496
Usuario sufre de Epilepsia	2	499
Usuario sufre de Hipertensión Esencial	84	417
Usuario tiene el fémur fracturado	4	497
Usuario tiene Glaucoma	2	499
Usuario sufre de Gonartrosis	1	500
Usuario tiene Hipoacusia, sin más especificación	5	496
Usuario sufre una Cardiopatía Hipertensiva	3	498
Usuario padece del síndrome de Inmovilidad	1	500
Usuario tiene Insomnio	3	498
Usuario sufre Cadiopatía Isquémica	13	488
Usuario ha sufrido un accidente Vascular Isquémico	3	498
Usuario sufre Psicosis Maniaco-Depresiva	1	500
Usuario sufre Deterioro Cognitivo mixto	16	485
Usuario tiene Osteoporosis	5	496
Usuario sufre un trastorno Paranoide	1	500
Usuario tiene Parkinson	20	481
Usuario sufre de Parkinsonismo	7	494
Usuario sufre una Vasculopatía Periférica	2	499
Usuario tiene Presbiacusia	5	496
Usuario tiene Presbicia	4	497
Usuario sufre Psicosis, sin especificar su tipo	1	500
Usuario tiene Artritis Reumatoide	3	498
Usuario tiene una enfermedad cerebrovascular de pequeño vaso	1	500
Usuario sufre de Espondiloartrosis	2	499
Usuario ha sufrido un Infarto Agudo de Miocardio	0	501
Usuario ha sufrido una Hemorragia Cerebral	2	499
Usuario ha sufrido un Infarto Cerebral	3	498
Usuario ha sufrido un Ictus Hemorrágico	1	500
Usuario ha sufrido un Ictus Isquémico	5	496
Usuario tiene incontinencia urinaria	5	496
Usuario tiene alteraciones auditivas	31	470
Usuario problemas en extremidades	9	492
Usuario tiene alteraciones visuales	37	464
Usuario tiene deterioro cognitivo probablemente vascular	5	496
Usuario ha sufrido un accidente vascular encefálico	7	494
Usuario sufre de vértigo	1	500
Usuario sufre Déficit Visual	7	494
Usuario es Capaz de Levantarse	94	407
Usuario toma antidepresivos	63	438
Usuario toma tranquilizantes o sedantes	77	424
Usuario tiende al aislamiento	0	501

Tabla 7.1: Distribución de características *booleanas* en predicción de caídas.

El lenguaje SQL nos da mucha expresividad para explotar los datos preprocesados y generar vistas que aporten valor. Otras vistas, centradas únicamente en los casos positivos también serían útiles, así como la construcción de gráficos. Para este propósito se pueden utilizar otros lenguajes como R o se pueden migrar los datos a una hoja de Excel.

A continuación, vamos a dar de manera tabular las métricas que cada entrenamiento ha tenido con un tiempo máximo de 20 minutos. El entrenamiento, como hemos explicado, se hace usando AutoML, que nos devolverá también el algoritmo con el que mejores métricas se han obtenido:

Experimento usando enfermedades			
# Features	12	**# Instancias**	18732
Algoritmo	FastTreeBinary		
Precisión	1	**Valor F1**	0.75
aucROC	0.87	**aucPR**	0.62
Precisión positiva	1	**Exhaustividad positiva**	0.6
Precisión negativa	1	**Exhaustividad negativa**	1

Experimento usando enfermedades y escalas			
# Features	49	**# Instancias**	501
Algoritmo	LightGbmBinary		
Precisión	0.97	**Valor F1**	0.82
aucROC	0.86	**aucPR**	0.75
Precisión positiva	1	**Exhaustividad positiva**	0.7
Precisión negativa	0.97	**Exhaustividad negativa**	1

Experimento usando enfermedades y escalas adicionales			
# Features	71	**# Instancias**	1695
Algoritmo	LightGbmBinary		
Precisión	0.96	**Valor F1**	0.88
aucROC	0.94	**aucPR**	0.91
Precisión positiva	1	**Exhaustividad positiva**	0.79
Precisión negativa	0.95	**Exhaustividad negativa**	1

Experimento usando enfermedades y escalas extra sin connotación temporal			
# Features	70	**# Instancias**	501
Algoritmo	LightGbmBinary		
Precisión	0.96	**Valor F1**	0.83
aucROC	0.86	**aucPR**	0.79
Precisión positiva	1	**Exhaustividad positiva**	0.71
Precisión negativa	0.96	**Exhaustividad negativa**	1

Vamos a hacer un pequeño inciso sobre las métricas. La **precisión** es el ratio entre el número de predicciones correctas y el número de predicciones hechas. Cuanto más cercano de 1 sea, mejor. Sin embargo, un valor exacto o muy cercano a 1 es sinónimo de problemas como datos desequilibrados. En la sección **7.2.3 Generación de casos negativos** hemos hecho una reflexión sobre esto, discutiendo sobre la importancia de la estrategia para generar casos negativos. El primer experimento, que utilizaba una estrategia de generación de casos negativos mensual, debe descartarse o modificarse atendiendo a

la precisión que hemos obtenido. En el resto de los experimentos, el desequilibrio ha sido menor. Cuando la precisión es cercana a 1, se deben comprobar métricas adicionales, como es el valor F1.

Los valores F son una medida ponderada entre la precisión y la exhaustividad y es útil cuando se desea encontrar un equilibrio entre ambas. En concreto, la **puntuación F1** se calcula como sigue, donde los valores más cercanos a 1 representan un clasificador más preciso:

$$F_1 = 2 \cdot \frac{\textit{Precisión} \cdot \text{Exhaustividad positiva}}{\textit{Precisión} + \text{Exhaustividad positiva}}$$

La **exhaustividad** (*recall*) es también una medida muy importante en la predicción de sucesos. Intuitivamente, ser capaz de determinar que una persona no se va a caer (caso negativo) no es de tanta ayuda como lo es alertar de que una persona va a caerse (caso positivo). De manera informal, la exhaustividad es la habilidad de un modelo para determinar los casos relevantes. Se calcula usando las fórmulas siguientes:

$$\text{Exhaustividad positiva (TPR)} = \frac{\text{\#Verdaderos positivos (TP)}}{\text{\#Instancias positivas reales}}$$

$$\text{Exhaustividad negativa (FPR)} = \frac{\text{\#Verdaderos negativos (TN)}}{\text{\#Instancias negativas reales}}$$

En nuestro caso, no nos vamos a fijar en las métricas de precisión negativa y exhaustividad negativa porque lo que nos interesa es ser capaces de predecir caídas, que se encuadra como la clase positiva. Por otro lado, las métricas **aucROC** (área bajo la curva ROC) y **aucPR** (área bajo la curva de precisión-exhaustividad o PR) pueden emplearse como medida cuando el conjunto de datos está sesgado o poco equilibrado. En [55] se menciona que cuando esto ocurre de manera pronunciada, aucPR ofrece una mejor visión sobre el rendimiento del algoritmo. La curva ROC representa el ratio entre positivos reales (*True Positive Rate*, TPR) y el ratio de falsos positivos (*False Positive Rate*, FPR). TPR es sinónimo de exhaustividad positiva o sensibilidad y FPR lo es de exhaustividad negativa o especificidad.

Atendiendo a todo esto, el experimento que deberíamos desplegar a producción es el que emplea enfermedades y escalas adicionales ya que presenta unos valores superiores del valor F1, la exhaustividad positiva y las áreas bajo la curva ROC y PR. Sin embargo, un nuevo experimento debería ser creado y entrenado utilizando solo aquellas características representativas de acuerdo con la tabla 7.1 con el fin de tener mejores resultados en la exhaustividad positiva.

# *Features*	**Importancia medida con valor F1**
Usuario es hombre	-0,47812
Resultado de la escala Lawton y Brody	-0,41503
Resultado de la escala Barthel	-0,16357
Usuario tiene Parkinson	-0,08527
Usuario sufre Deterioro Cognitivo mixto	-0,0024
Edad	0,06405
Usuario sufre de Hipertensión Esencial	0,1366

Tabla 7.2: Importancia de características en predicción de caídas.

Adicionalmente, se puede obtener la puntuación con la importancia de las *features* en el modelo entrenado a través del algoritmo PFI [13] (*Permutation Feature Importance*). En la tabla 7.2 se presentan las *features* que más influyen dentro del experimento que ha obtenido mejores resultados. La importancia de las características se mide con una de las métricas que ofrece ML.NET, donde hemos seleccionado la puntuación F1. A mayor valor, mayor influencia en la predicción. Se debe interpretar de la siguiente forma: un valor negativo de la importancia indica que si la característica es verdadera (por ejemplo, el usuario es hombre), menor es la probabilidad de caída. Esto concuerda con la literatura consultada. Lo mismo ocurre con los valores numéricos: cuanto mayor es el resultado de la escala Barthel (que indica una mayor independencia) menor es la probabilidad de caída.

Los resultados de características como si el usuario sufre Parkinson o deterioro cognitivo no parecen coincidir con lo que indica el sentido común ya que uno esperaría que sufrir estas enfermedades se traduce en una mayor probabilidad de caída cuando se ha obtenido justo lo contrario. Por tanto, el modelo debería volverse a entrenar utilizando una cantidad mayor de datos.

[13]Interpretación de las predicciones del modelo mediante la importancia de características de permutación: docs.microsoft.com/es-es/dotnet/machine-learning/how-to-guides/explain-machine-learning-model-permutation-feature-importance-ml-net

Por último, como hemos dicho las predicciones hechas se almacenan como instancias de una entidad. Utilizando el DSML de formularios es posible generar un formulario de cualquier entidad, como hemos hecho en este caso (figura 7.18), construyendo un *grid* donde mostramos la fecha de la predicción, el usuario sobre el que se realiza la predicción, el resultado de la predicción y la probabilidad de esta.

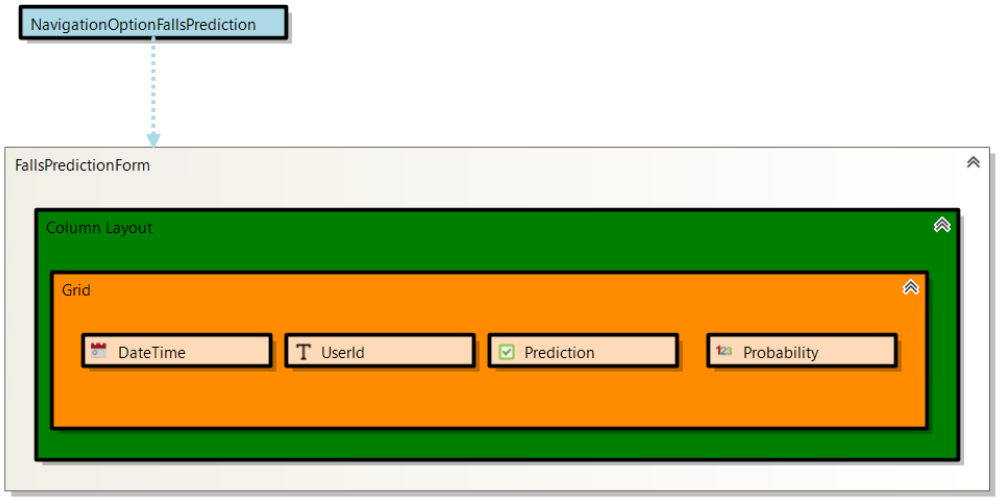

Figura 7.18: Modelo del formulario de predicciones.

Este modelo, a través de la generación de código, es convertido en un formulario de Xamarin accesible desde la aplicación y se renderiza como muestra la figura 7.19.

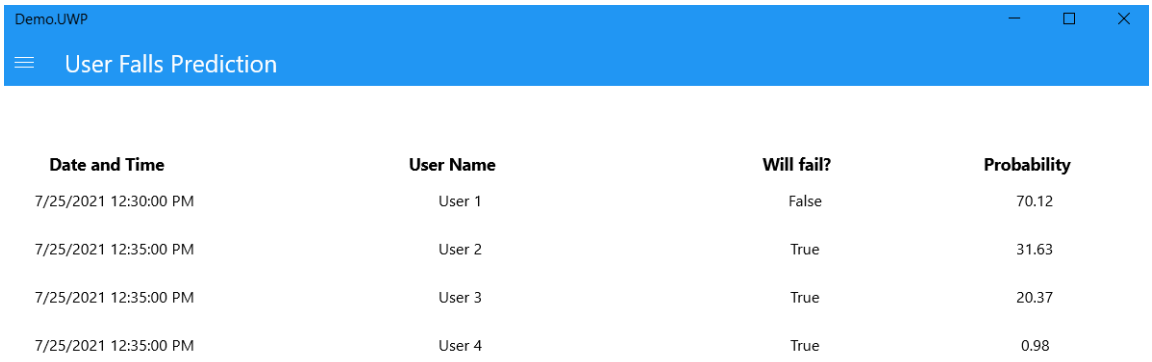

Figura 7.19: Formulario con las predicciones.

CAPÍTULO 8
Conclusiones

Respecto al objetivo principal de este trabajo, se ha desarrollado un DSML que nos permite añadir funcionalidades resueltas mediante *machine learning*. Este DSML está integrado con otros, que se emplean para el modelado de entidades o formularios, dotando al usuario de un conjunto de herramientas para el desarrollo íntegro de un producto *software* que incorpore IA. En cuanto a los objetivos de esta memoria, podemos hacer las siguientes reflexiones:

- En cuanto al **escenario de análisis de sentimiento**, se ha desarrollado una primera versión con los requisitos mínimos. El análisis de sentimiento se realiza únicamente sobre textos escritos en inglés. La funcionalidad desarrollada se ha empleado para analizar las valoraciones médicas dentro del ERP.

- Se ha desarrollado satisfactoriamente el soporte para la **predicción de sucesos a futuro** dentro del DSML de IA. Utilizando este tipo de escenarios se han modelado diferentes experimentos para la predicción de caídas de usuarios en una residencia. Sin embargo, con respecto al análisis inicial, ha quedado pendiente de implementar las funcionalidades de despliegue y monitorización de modelos de ML en un entorno real. Estas funcionalidades se engloban como trabajo futuro debido a que el ERP de la PYME en la que se ha realizado el proyecto está todavía en desarrollo.

- El DSML desarrollado se ha **integrado correctamente** con el resto de las herramientas de la empresa. Está construido sobre la base de un DSML de Dominio que representa modelos entidad-relación. Utilizando transformaciones M2M, los experimentos desarrollados en el escenario de predicción se representan como entidades, a partir de las cuales se pueden hacer formularios usando un tercer DSML. También se han modelado acciones y pruebas a través de los DSML de Aplicación y Pruebas, que se emplean para modelar el comportamiento del *software*. Así, se ofrece una solución integral para crear aplicaciones *software*. Apoyarnos en el DSML Dominio para construir el DSML de IA resta importancia a tareas de ML como la limpieza de datos porque los campos están tipificados y cuentan con validaciones como longitudes máximas, valores mínimos y máximos, etc. Sin embargo, la solución propuesta solo tiene sentido en el entorno de desarrollo, es decir, el DSML de IA no puede emplearse de manera aislada y solo genera código para una plataforma.

- Respecto a los **beneficios de seguir una estrategia MDD**, al inicio de la memoria se han resumido en 3 los beneficios. El primero de ellos, la estandarización, ha quedado acreditado. Términos como el de 'valor umbral' en el escenario de análisis de sentimiento u otros como son los de 'experimento' o 'caso negativo' en el escenario de predicción han quedado bien formalizados dentro del DSML y han mejorado la

comunicación entre los miembros del equipo. Se han realizado transformaciones de modelo a código que aceleran el desarrollo de funcionalidades de ML, aunque no se ha medido de manera rigurosa cuál es el ahorro en coste o tiempo. Por último, el uso de modelos ha permitido añadir diferentes validaciones para garantizar la corrección de los modelos hechos.

El trabajo realizado es solo un primer paso dentro de un proyecto que apuesta por el desarrollo de *software* a través de estrategias MDD. Dotar de inteligencia a una aplicación a través de modelos y generación automática de código acelera el desarrollo, nos abstrae de detalles de implementación y nos facilita el futuro mantenimiento.

Es cierto que solo se han cubierto dos escenarios muy concretos. No obstante, es muy **fácil de extrapolar** los conceptos de experimento, referencias a elementos del dominio u operaciones a otros escenarios como la detección de anomalías, la regresión de un valor o una clasificación multiclase. Por ejemplo, una anomalía puede interpretarse como un suceso muy poco frecuente, por lo que los cambios que deberán aplicarse al DSML serán mínimos, en su mayoría relacionados con el tratamiento de datos no equilibrados.

Existe un **nicho comercial** en la intersección entre los campos de MDD y la IA. En la situación excepcional en la que nos encontramos fruto de la pandemia se ha acelerado el proceso de transformación digital. La figura del *citizen developer* cobra fuerza y hace indispensable la existencia de las plataformas *low-code* y *no-code*. Ninguna de ellas ofrece un desarrollo integral de un producto *software* con capacidades cognitivas.

Personalmente, este trabajo refleja el conocimiento adquirido en la Universidad y en mi experiencia laboral. Por una parte, de la Universidad he aplicado mis estudios sobre ciencia de datos e ingeniería basada en modelos. Las asignaturas del máster que he cursado y más he podido aplicar en este trabajo son: Data Science (DAS) e Ingeniería de Sistemas de Información (ISI). DAS sintetiza muy bien la literatura que he consultado relacionada con IA y ML, mientras que ISI me ha ofrecido una visión muy práctica del desarrollo basado en modelos y sus beneficios. Por otra parte, he aplicado los conocimientos que tenía de usar tecnología de Microsoft fruto de mi experiencia laboral.

CAPÍTULO 9
Trabajo futuro

En este capítulo se resume el trabajo futuro, que se ha dividido en dos secciones siguiendo los escenarios presentados.

9.1 Escenario de análisis de sentimiento

Los ERP deben evolucionar hacia interacciones con el usuario ubicuas para facilitar la realización de sus tareas. En este sentido, nuestra aplicación permite la introducción por voz de campos textuales utilizando STT. Ahora mismo, solo permitimos el análisis de sentimiento en texto. El **reconocimiento de emociones en el habla** (*speech emotion recognition* o SER) tiene un amplio espectro de aplicación para recoger *feedback* en centros de atención telefónica y en productos como videojuegos o aplicaciones móviles [56]. En nuestra aplicación, podríamos aplicar un análisis SER siempre que se realiza STT, incorporando esta funcionalidad como un escenario nuevo del DSML de IA.

El trabajo realizado, que consideramos una primera versión de lo que será la herramienta final, solo soporta el análisis de sentimiento en textos de lengua inglesa. Hemos optado por seguir esta estrategia ya que en este escenario queríamos centrarnos primero en el diseño, a la espera de ver cómo evoluciona el estado del arte. Como trabajo futuro, se espera reemplazar las librerías usadas para el análisis por modelos entrenados en la empresa para lenguas específicas como el catalán o el vasco. Se espera colaborar con otras instituciones que nos puedan proporcionar un corpus para realizar este entrenamiento. Un entrenamiento semisupervisado podría llevarse a cabo, empleando datos de nuestros clientes para tal entrenamiento, siempre con su consentimiento.

Además, deberían desarrollarse **diferentes modelos** según el contexto al que luego se aplique el análisis. Por ejemplo, en nuestro caso práctico hemos aplicado el análisis sobre las valoraciones médicas. Se asume que aquí se usará un lenguaje más formal que el que se emplea en la mensajería interna, donde se asume un lenguaje más coloquial. Quizás esto pueda ajustarse únicamente a través del valor umbral que permite especificar el análisis, pero serán necesarias pruebas para determinarlo.

Por último, el análisis realizado se engloba en un problema de clasificación binaria con dos clases, positivo y negativo. Este análisis puede resultar bastante pobre y contraproducente. Por ejemplo, ¿tiene sentido notificar a un usuario Administrador cuando se introduce un texto donde el análisis da como resultado un sentimiento negativo debido a que el texto trata sobre el deceso de una persona? Obviamente, no. La herramienta desarrollada debería fluctuar desde el análisis de sentimiento actual a la **detección de emociones** tales como ira, tristeza o felicidad. De esta forma, solo se notificaría cuando se

detectara ira en los textos. De nuevo, encontrar datos para este objetivo será fundamental, que se espera poder abordar mediante *transfer learning*.

9.2 Escenario de predicción de sucesos

La información es clave en los escenarios de predicción. La integración con dispositivos que recolectan datos periódicamente puede ser muy útil, tanto para agilizar la entrada de datos en el ERP como para nuestro escenario de predicción. En [57] se hizo un estudio para la predicción de caídas empleando los datos de un acelerómetro para medir la actividad diaria. En el apartado **4.4 MDD aplicado a sistemas *context-awareness*** hemos visto como el ámbito de IoT puede beneficiarse de las estrategias MDD. Nuestro DSML de IA también debería ser capaz de explotar la información que cada vez más dispositivos dan.

Por otro lado, nos gustaría validar con diferentes profesionales del sector cómo de expresivo es el DSML construido. ¿Es fácil de usar para una persona con pocos conocimientos técnicos? ¿Qué otros modelos de predicción se pueden construir? ¿Son necesarias nuevas operaciones?

Algunos casos de uso del análisis no han sido cubiertos completamente todavía y otros nuevos requisitos han surgido durante la implementación del lenguaje. El siguiente listado recoge los cambios que se desean introducir en el DSML:

- **Visualización de datos**: la parte analítica del proceso propuesto se realiza por ahora a través de scripts SQL. Esto no es fácil de usar para usuarios no técnicos y el DSML debería dar facilidades para encontrar asociaciones entre las *features* y construir gráficos a partir de ellas. En este sentido, se está trabajando también en un nuevo DSML de inteligencia de negocio (*business intelligence*, BI) que permita definir hechos y dimensiones. Este DSML de BI explota, igual que el de IA, los modelos de dominio. La integración de ambos DSML facilitará la explotación y limpieza de datos en nuestro ciclo de ML.

- **Monitorización**: en el análisis del escenario se menciona la capacidad de desplegar 2 experimentos y comparar su rendimiento usando un flujo real de datos. También se deben controlar los modelos de ML en producción para reaccionar ante situaciones incorrectas como un gran número de predicciones incorrectas o un descenso del rendimiento. Ninguno de estos requisitos ha sido implementado todavía.

- **Datos anómalos**: el DSML tiene soporte para el tratamiento de datos anómalos, pero por ahora no se realiza ningún proceso automático para enmendar *outliers* o valores en formatos incorrectos. El DSML de IA está construido sobre el de Dominio. En este último encontramos propiedades para determinar si un valor es válido o no (valores máximos y mínimos de campos, longitudes máximas, restricciones de tipos, etc.), por lo que puede considerarse una tarea sencilla.

- **Sistema proactivo**: como se ha presentado en el caso de predicción de caídas, con los modelos de ML por ahora no podemos hacer mucho más allá de construir formularios donde mostrar las predicciones hechas. Se debe transitar a una funcionalidad más proactiva: si se predice que un usuario puede sufrir próximamente una caída, se deberá alertar al usuario del ERP para que tome las medidas que convenga. Para que esto no se convierta en una funcionalidad intrusiva, se debe ir refinando primero los modelos de predicción de forma iterativa.

- **Reentrenamiento**: actualmente, si tenemos un modelo de ML entrenado y quisiéramos reentrenarlo añadiendo los datos de una nueva residencia, no podríamos hacerlo de manera eficiente. Algunos algoritmos de ML.NET soportan el reentrenamiento, que consiste en tomar un modelo de ML entrenado y ajustarlo utilizando nuevos datos. Esto es crítico si el flujo de nuevos datos del problema es constante y es necesario reaccionar rápidamente a ellos. Será necesario cubrir esta funcionalidad para evitar realizar un nuevo entrenamiento desde cero.
- **Mejoras en la interacción con el DSML**: hemos observado que la construcción de modelos reales, como el de predicción de caídas, puede ser costosa debido al gran número de *features* que tiene. Por tanto, será interesante revisar la experiencia del usuario a la hora de usar el DSML para agilizar la construcción de los modelos. También será necesario añadir validaciones nuevas para guiar a usuarios noveles. Por ejemplo, ahora se podría añadir como *feature* el nombre o el DNI del usuario, cuando no tiene mucho sentido construir un modelo de ML que reciba como entrada esto.

Por último, se desea usar el modelo de predicción de caídas dentro del ERP. Serán necesarias futuras iteraciones con nuevos datos que permitan ir refinando y mejorando el modelo construido.

Referencias

[1] "Los 15 empleos más demandados en 2021, según LinkedIn". La Vanguardia, 2021. Consultado el día 06/07/2021, URL: https://www.lavanguardia.com/vida/formacion/20210413/6647317/15-empleos-mas-demandados-espana-2021-linkedin.html.

[2] Andrew Zola. *"Making the Transition from Software Engineer to Artificial Intelligence Engineer"*. DZone, 2020. Consultado el día 29/07/2021, URL: https://dzone.com/articles/making-the-transition-from-software-engineer-to-ar

[3] D. Esposito y F. Esposito. *"Introducing Machine Learning"*. Microsoft Press Store, 2020. ISBN: 9780135588338.

[4] Karen Hao. *"Training a single AI model can emit as much carbon as five cars in their lifetimes"*. *MIT Technology Review*, 2020. Consultado el día 16/08/2021, URL: https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/

[5] Frederick P. Brooks. *"No Silver Bullet-Essence and Accident in Software Engineering"*. *IEEE Computer*, 1986. DOI: https://doi.org/10.1109/MC.1987.1663532

[6] Jack Greenfield y Keith Short. *"Software factories: Assembling Applications with Patterns, Frameworks, Models and Tools"*. en *OOPSLA 2003 - 18th annual ACM Object-oriented programming, systems, languages, and applications*, 2003. DOI: https://doi.org/10.1145/949344.949348

[7] Bran Selic. *"The pragmatics of model-driven development"*. *IEEE Software*, 2003. DOI: https://doi.org/10.1109/MS.2003.1231146

[8] Prachi Patodi. *"Top 15 No-Code AI and ML Platforms for 2021"*. Data Flair, 2020. Consultado el día 18/04/2021, URL: https://data-flair.training/news/no-code-ai-ml-platforms/

[9] Marco Brambilla, Jordi Cabot, y Manuel Wimmer. *"Model-Driven Software Engineering in Practice"*. Morgan & Claypool, 2017. ISBN: 1681732335

[10] Jon M. Siegel. *"Model Driven Architecture (MDA) - MDA Guide rev. 2.0"*, OMG Group, 2014. URL: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01

[11] Issam Al-Azzoni. *"Model Driven Approach for Neural Networks"*. en *IDSTA 2020 - International Conference on Intelligent Data Science Technologies and Applications*, 2020. DOI: https://doi.org/10.1109/IDSTA50958.2020.9264067

[12] Stephen J Miller, Kendall Scott, Axel Uhl, y Dirk Weise. *"MDA Distilled"*. *Addison-Wesley Professional*, 2004. ISBN: 0201788918

[13] Antonio Bucchiarone, Jordi Cabot, Richard F. Paige, y Alfonso Pierantonio. *"Grand challenges in model-driven engineering: an analysis of the state of the research"*. *Software and Systems Modeling 2020*, 2020. DOI: https://doi.org/10.1007/s10270-019-00773-6

[14] Marcus Woo. *"The Rise of No/Low Code Software Development—No Experience Needed?"* *Engineering*, 2020. DOI: https://doi.org/10.1016/j.eng.2020.07.007

[15] Javier Muñoz y Vicente Pelechano. "MDA vs Factorías de Software". en *CEUR Workshop Proceedings*, 2005. URL: http://ceur-ws.org/Vol-157/paper01.pdf

[16] Jordi Cabot. *"Low-code vs model-driven: are they the same?"*. Modeling Languages, 2020. Consultado el día 29/06/2021, URL: https://modeling-languages.com/low-code-vs-model-driven/

[17] Setrag Khoshafian. *"No-code/low-code: Why you should be paying attention"*. VentureBeat, 2021. Consultado el día 29/06/2021, URL: https://venturebeat.com/2021/02/14/no-code-low-code-why-you-should-be-paying-attention/

[18] Glosario de conceptos de Inteligencia Artificial. Consejo de Europa. Consultado el día 30/06/2021, URL: https://www.coe.int/en/web/artificial-intelligence/glossary

[19] Alan Turing. *"Computing machinery and intelligence"*. *Mind*, 1950. DOI: https://doi.org/10.1007/s11245-013-9182-y

[20] Aurelien Geron. "Hands–On Machine Learning with Scikit–Learn and TensorFlow". O'REILLY, 2019. ISBN: 1491962291

[21] Daniel Hain y Roman Jurowetzki. *"Introduction to Rare-Event Predictive Modeling for Inferential Statisticians – A Hands-On Application in the Prediction of Breakthrough Patents"*, 2020. URL: https://arxiv.org/ftp/arxiv/papers/2003/2003.13441.pdf

[22] David Lechevalier, Steven Hudak, A. K. Ronay, Y. Tina Lee, y Sebti Foufou. *"A neural network meta-model and its application for manufacturing"*. en *2015 IEEE International Conference on Big Data*, 2015. DOI: https://doi.org/10.1109/BigData.2015.7363903

[23] Sridhar Alla y Suman Kalyan Adari. "Beginning MLOps with MLFlow". Apress, 2021. ISBN: 1484265483

[24] Oscar García-Olalla. "La importancia del preprocesamiento de datos en Inteligencia Artificial: Limpieza de datos.". Xeridida, 2020. Consultado el día 20/08/2021. URL: https://www.xeridia.com/blog/la-importancia-del-preprocesamiento-de-datos-en-inteligencia-artificial-limpieza-de-datos

[25] Xin He, Kaiyong Zhao, y Xiaowen Chu. *"AutoML: A Survey of the State-of-the-Art"*. *2019 Knowledge-Based Systems*, 2019. DOI: https://doi.org/10.1016/j.knosys.2020.106622

[26] Roger S. Pressman y Bruce R. Maxim. "Software engineering : a practitioner's approach". McGraw-Hill Education, 2010. ISBN: 9780078022128

[27] Sinno Jialin Pan y Qiang Yang. *"A survey on transfer learning"*. *IEEE Transactions on Knowledge and Data Engineering 2010*, 2010. DOI: https://doi.org/10.1109/TKDE.2009.191

[28] Sayak Misra. *"Guessing Sentiment in 100 Languages..."*, Medium 2020. Consultado el día 03/07/2021. URL: https://medium.com/analytics-vidhya/guessing-sentiment-in-100-languages-4574ceed3b67

[29] Xuan Song Li, Xian Ping Tao, Wei Song, y Kai Dong. *"AocML: A Domain-Specific Language for Model-Driven Development of Activity-Oriented Context-Aware Applications"*. *2018 Journal of Computer Science and Technology*, 2018. DOI: https://doi.org/10.1007/s11390-018-1865-9

[30] Tian Zhao y Xiaobing Huang. *"Design and implementation of DeepDSL: A DSL for deep learning"*. *2018 Computer Languages, Systems and Structures*, 2018. DOI: https://doi.org/10.1016/j.cl.2018.04.004

[31] Tian Zhao, Xiaobing Huang, y Yu Cao. *"DeepDSL: A Compilation-based Domain-Specific Language for Deep Learning"*. 2017. URL: https://arxiv.org/abs/1701.02284

[32] Hourieh Khalajzadeh, Andrew J. Simmons, Mohamed Abdelrazek, John Grundy, John Hosking, y Qiang He. *"End-User-Oriented Tool Support for Modeling Data Analytics Requirements"*. en *VL/HCC 2020 - Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing*, 2020. DOI: https://doi.org/10.1109/VL/HCC50065.2020.9127196

[33] Hourieh Khalajzadeh, Mohamed Abdelrazek, John Grundy, John Hosking, y Qiang He. *"BiDaML: A Suite of Visual Languages for Supporting End-User Data Analytics"*. en *2019 IEEE International Congress on Big Data*, 2019. DOI: https://doi.org/10.1109/BigDataCongress.2019.00025

[34] Dominic Breuker. *"Towards model-driven engineering for big data analytics - An exploratory analysis of domain-specific languages for machine learning"*. en *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2014. DOI: https://doi.org/10.1109/HICSS.2014.101

[35] Asha Rajbhoj, Vinay Kulkarni, y Nikhil Bellarykar. *"Early experience with model-driven development of mapreduce based big data application"*. en *APSEC - Proceedings Asia-Pacific Software Engineering Conference*, 2014. DOI: https://doi.org/10.1109/APSEC.2014.23

[36] Sai Chaithra Allala, Juan P. Sotomayor, Dionny Santiago, Tariq M. King, y Peter J. Clarke. *"Towards transforming user requirements to test cases using MDE and NLP"*. en *COMPSAC Proceedings - International Computer Software and Applications Conference*, 2019. DOI: https://doi.org/10.1109/COMPSAC.2019.10231

[37] Alessandro Di Bari, Alessandro Faraotti, Carmela Gambardella, y Guido Vetere. *"A model-driven approach to NLP programming with UIMA"*. en *CEUR Workshop Proceedings*, 2013. URL: http://ceur-ws.org/Vol-1038/paper_9.pdf

[38] Loli Burgueño, Robert Clarisó, Sébastien Gérard, Shuai Li, y Jordi Cabot. *"An NLP-Based Architecture for the Autocompletion of Partial Domain Models"*. en *CAiSE 2021 - International Conference on Advanced Information Systems Engineering*, 2021. URL: https://hal.archives-ouvertes.fr/hal-03010872

[39] Ahmad F. Subahi. *"Cognification of program synthesis—a systematic feature-oriented analysis and future direction"*. *Computers*, 2020. DOI: https://doi.org/10.3390/computers9020027

[40] Estefanía Serral, Pedro Valderas, y Vicente Pelechano. *"Towards the Model Driven Development of context-aware pervasive systems"*. *Pervasive and Mobile Computing*, 2010. DOI: https://doi.org/10.1016/j.pmcj.2009.07.006

[41] Yulia Gavrilova. *"Top 18 Low-Code and No-Code ML Platforms"*. Serokell, 2021. Consultado el día 18/04/2021, URL: https://serokell.io/blog/top-no-code-platforms

[42] Anupam Chugh. *"Top 8 "No-Code" Machine Learning Platforms You Should Use en 2020"*. Towards Data Science, 2020. Consultado el día 18/04/2021, URL: https://towardsdatascience.com/top-8-no-code-machine-learning-platforms-you-should-use-in-2020-1d1801300dd0

[43] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, y Alfonso Pierantonio. *"Supporting the understanding and comparison of low-code development platforms"*. en *SEAA 2020 Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications*, 2020. DOI: https://doi.org/10.1109/SEAA51224.2020.00036

[44] *Comparing ML as a Service (MLaaS): Amazon AWS, IBM Watson, MS Azure* AltexSoft, 2021. Consultado el día 08/07/2021. URL: https://www.altexsoft.com/blog/datascience/comparing-machine-learning-as-a-service-amazon-microsoft-azure-google-cloud-ai-ibm-watson/

[45] Víctor Iranzo. "Desarrollo de software basado en microservicios: un caso de estudio para evaluar sus ventajas e inconvenientes", 2018. URL: https://riunet.upv.es/handle/10251/111173

[46] Steve Cook, Gareth Jones, Stuart Kent, y Alan Cameron Wills. *"Domain-Specific Development with Visual Studio DSL Tools"*. Addison Wesley, 2007. ISBN: 0321398203.

[47] Iñigo Domínguez, María Sosa, y Daniele Grasso. "Las residencias en España: descontrol en un sistema opaco, con multas bajas y contra el que sirve de poco quejarse". El País, 2021. Consultado el día 04/07/2021. URL: https://elpais.com/sociedad/2021-07-04/las-residencias-en-espana-descontrol-en-un-sistema-opaco-con-multas-bajas-y-contra-el-que-sirve-de-poco-quejarse.html

[48] Caídas, Organización Mundial de la Salud, 2021. Consultado el día 04/07/2021. URL: https://www.who.int/es/news-room/fact-sheets/detail/falls

[49] Gabriel Suárez, Victor Velasco, Maria Limones, Hugo Reyes, y Blanca Zacarías. "Factores asociados con caídas en el adulto mayor". *Paraninfo Digital*, 2018. URL: http://www.index-f.com/para/n28/pdf/e025.pdf

[50] Eric Manuel Blake Pávez. "Caídas en adultos mayores: principales causas y cómo prevenir". Clínica Alemana, 2018. Consultado el día 04/07/2021. URL: https://www.clinicaalemana.cl/articulos/detalle/2018/caidas-en-adultos-mayores-principales-causas-y-como-prevenir

[51] J.R. Silva-Fhon, R. Partezani-Rodrigues, K. Miyamura, y W. Fuentes-Neira. "Causas y factores asociados a las caídas del adulto mayor". *Enfermería Universitaria*, 2019. DOI: https://doi.org/10.22201/eneo.23958421e.2019.1.576

[52] Sanitas. "Factores que predisponen a las caídas en personas mayores". Sanitas, 2020. Consultado el día 04/07/2021. URL: https://www.sanitas.es/sanitas/seguros/es/particulares/biblioteca-de-salud/tercera-edad/habitos-vida-saludable/factores-caidas-mayores.html

[53] Lidia Machado, Miriela Machado, y Marioneya Izaguirre. *"Principales factores de riesgo asociados a las caídas en ancianos del área de salud Guanabo"*. MEDISAN, 2014. URL: https://www.medigraphic.com/cgi-bin/new/resumen.cgi?IDARTICULO=47837

[54] Victoria López, Alberto Fernández, Salvador García, Vasile Palade, y Francisco Herrera. *"An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics"*, 2013. DOI: https://doi.org/10.1016/j.ins.2013.07.007

[55] Jesse Davis y Mark Goadrich. *"The Relationship Between Precision-Recall and ROC Curves"*. en *Proceedings of the 23rd international conference on Machine learning*, 2006. DOI: https://doi.org/10.1145/1143844.1143874

[56] Eva Lieskovská, Maroš Jakubec, Roman Jarina, y Michal Chmulík. *"A Review on Speech Emotion Recognition Using Deep Learning and Attention Mechanism"*. *Electronics 2021*, 2021. DOI: https://doi.org/10.3390/electronics10101163

[57] Ahmed Nait Aicha, Gwenn Englebienne, Kimberley S. van Schooten, Mirjam Pijnappels, y Ben Kröse. *"Deep learning to predict falls in older adults based on daily-life trunk accelerometry"*. *Sensors (Switzerland)*, 2018. DOI: https://doi.org/10.3390/s18051654