UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño y desarrollo de un framework para la clasificación de variaciones genómicas basado en modelos conceptuales

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Benavent Ribelles, Laia

Tutor/a: Costa Sánchez, Mireia

Cotutor/a: Pastor López, Oscar

Director/a Experimental: García Simón, Alberto

CURSO ACADÉMICO: 2023/2024

Resumen

El estudio de nuestro ADN permite prevenir, diagnosticar y tratar un gran abanico de enfermedades, contribuyendo a mejorar la calidad de vida de la sociedad. En este contexto, una de las áreas que más atención ha recibido es el estudio de cómo las variaciones genéticas se relacionan con distintas enfermedades, proceso conocido como clasificación de variaciones. Históricamente, la clasificación de variaciones ha sido un proceso complejo, subjetivo y ha requerido de una gran labor manual. Para intentar facilitar la labor a los expertos del dominio, se crearon un conjunto de guías que dieran soporte al proceso de clasificación. Aunque estas guías han sido una gran mejora, todavía presentan algunas limitaciones. Las principales son la falta de reproducibilidad y de automatización. El objetivo de este Trabajo de Fin de Grado es diseñar y desarrollar un framework para automatizar la clasificación de variantes e incrementar su reproducibilidad. Todo el framework se ha desarrollado siguiendo modelos conceptuales para asegurar la consistencia y la precisión del proceso. El framework está dividido en dos partes. Primero, un sistema de evaluación mediante una lógica basada en reglas predefinidas donde, a partir de los criterios de una guía de clasificación, estos se descomponen en métricas atómicas que pueden ser automatizadas. Segundo, una interfaz gráfica de usuario con la que el usuario puede introducir los datos asociados a una guía clínica de una manera mucho más fácil y usable. Ambos componentes han sido desarrollados en Python. Este framework ha sido evaluado mediante un caso de uso, en el que se instanció una de las guías clínicas más importantes (ACMG/AMP) y en el que se analizaron los perfiles genéticos de pacientes reales.

Palabras clave: Variantes Genéticas; Clasificación; Modelos Conceptuales.

Resum

L'estudi del nostre ADN permet prevenir, diagnosticar i tractar un gran ventall de malalties, contribuint a millorar la qualitat de vida de la societat. En aquest context, una de les àrees que més atenció ha rebut és l'estudi de com les variacions genètiques es relacionen amb distintes malalties, procés conegut com a classificació de variacions. Històricament, la classificació de variacions ha sigut un procés complex, subjectiu i ha requerit un gran treball manual. Per a intentar facilitar el treball dels experts del domini, es va crear un conjunt de guies que donaren suport al procés de classificació. Encara que aquestes guies han sigut una gran millora, s'hi poden trobar algunes limitacions. Les principals són la falta de reproductibilitat i d'automatització. L'objectiu d'aquest Treball de Fi de Grau és dissenyar i desenvolupar un framework per automatitzar la classificació de variants i incrementar la seua reproductibilitat. Tot el framework s'ha desenvolupat seguint models conceptuals per assegurar la consistència i la precisió del procés. El framework està dividit en dues parts. Primer, un sistema d'avaluació mitjançant una lògica basada en regles predefinides on, a partir dels criteris d'una guia de classificació, aquests es descomponen en mètriques atòmiques que poden ser automatitzades. Segon, una interfície gràfica d'usuari en la que l'usuari pot introduir les dades associades a una guia clínica d'una forma molt més fàcil i usable. Ambdós components han sigut desenvolupats en Python. Aquest framework ha sigut avaluat mitjançant un cas d'ús, en el que s'instancià una de les guies clíniques més importants (ACMG/AMP) i en el que s'analitzaren els perfils genètics de pacients reals.

Paraules clau: Variants Genètiques; Classificació; Models Conceptuals.

Abstract

The study of our DNA allows us to prevent, diagnose and treat a wide range of diseases, contributing to improving the quality of life of society. In this context, one of the areas that has received the most attention is the study of how genetic variations are related to different diseases, a process known as variation classification. Historically, variation classification has been a complex, subjective, and labour-intensive process. To try to facilitate the work of domain experts, a set of guides were created to support the classification process. Although these guides have been a great improvement, they still have some limitations. The main ones are the lack of reproducibility and automation. The objective of this Final Degree Project is to design and develop a framework to automate the classification of variants and increase its reproducibility. The entire framework has been developed following conceptual models to ensure the consistency and precision of the process. The framework is divided into two parts. First, an evaluation system using a logic based on predefined rules where, based on the criteria of a classification guide, these are decomposed into atomic metrics that can be automated. Second, a graphical user interface with which the user can enter the data associated with a clinical guide in a much easier and more usable way. Both components have been developed in Python. This framework has been evaluated through a use case, in which one of the most important clinical guidelines (ACMG/AMP) was installed and in which the genetic profiles of real patients were analyzed.

Keywords: Genetic Variations; Classification; Conceptual Models.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

El estudio del ADN ha revolucionado la medicina moderna, brindando una comprensión más profunda de la base genética de numerosas enfermedades. Esta información genética ha permitido el desarrollo de la medicina de precisión, un enfoque innovador que busca personalizar el tratamiento y la prevención de enfermedades basándose en las características genéticas individuales de cada persona [1]. La medicina de precisión promete tratamientos más efectivos y con menos efectos secundarios, ya que se adapta a las necesidades específicas de cada persona.

Una de las áreas más importantes de la medicina de precisión es la clasificación de variantes genéticas. Las variantes genéticas son alteraciones permanentes en la secuencia de ADN que forman un gen [2]. Con esta técnica se busca clasificar las variantes según la probabilidad de causar enfermedades. Típicamente, las variantes se clasifican como benignas (no causan enfermedades), probablemente benignas (existe al menos un 90% de probabilidad de no causar enfermedades), de significado incierto (se desconoce el efecto en el organismo), probablemente patogénicas (existe al menos un 90% de probabilidad de causar enfermedades) y patogénicas (causa confirmada de enfermedad) [3].

La clasificación de variantes se realiza a través de guías de interpretación. Las guías de interpretación son un conjunto de instrucciones que orientan el proceso de clasificación de una variante, evaluando si su información contextual cumple o no criterios específicos [4]. Entre todas las existentes destaca la guía ACMG/AMP (American College of Medical Genetics and Genomics / Association for Molecular Pathology) [2], la cual proporciona criterios y métricas para evaluar la significancia clínica de las variantes genéticas en enfermedades mendelianas o hereditarias.

En este Trabajo de Fin de Grado se busca cubrir algunos de los desafíos principales de la clasificación de variantes, tal y cómo se detalla en la siguiente sección.

1.1. Motivación

El proceso de clasificación de variantes genéticas es actualmente un procedimiento manual en el que los expertos deben revisar y evaluar la relevancia de cada variante de manera individual. Este método es costoso en términos de tiempo y recursos humanos necesarios.

Una de las principales dificultades en este proceso radica en la variabilidad de las interpretaciones de las guías utilizadas para clasificar las variantes. Aunque se utilicen los mismos criterios, el resultado puede variar de un experto a otro [5]. Esto se debe a que las definiciones en las guías a menudo son imprecisas, lo que deja espacio para interpretaciones subjetivas por parte de los expertos en el dominio.

Un ejemplo de esta variabilidad se puede observar en la interpretación de la frecuencia de una variante en la población. Mientras que una guía podría considerar una frecuencia de 0.1% como indicativa de una variante probablemente benigna, otro experto podría interpretar la misma frecuencia como insuficiente para descartar las posibilidades de patogenicidad, lo que lleva a clasificaciones inconsistentes.

En este contexto, los expertos clínicos sostienen que se requieren definiciones más claras para estandarizar la interpretación de variantes y minimizar las inconsistencias [4]. Estas inconsistencias son contraproducentes para los objetivos de la medicina de precisión, que busca ofrecer tratamientos personalizados basados en análisis genéticos exactos y fiables.

Como se puede comprobar mediante estudios como el de Wahbeh et al. [6], los enfoques automatizados mediante frameworks de clasificación ofrecen una solución prometedora a estos problemas al agilizar el proceso y mejorar la precisión. Utilizar un framework automatizado para la clasificación de variantes reduciría significativamente la carga de trabajo manual, aceleraría la toma de decisiones clínicas y disminuiría la subjetividad en el proceso, representando un avance considerable hacia la mejora de la medicina de precisión.

Al facilitar un análisis más rápido, preciso y reproducible de las variantes genéticas, se contribuiría a la implementación más efectiva de tratamientos personalizados, beneficiando en última instancia a los pacientes.

Por lo tanto, el framework para la automatización de la clasificación de variantes genéticas surge como una solución integral a estas limitaciones,

optimizando tanto el tiempo como los recursos y mejorando la consistencia y precisión en los resultados. Diseñar y desarrollar este framework es el objetivo principal de este Trabajo de Fin de Grado, como se expone en la siguiente sección.

1.2. Objetivos

El objetivo principal de este proyecto es diseñar y desarrollar un framework para automatizar la clasificación de variantes.

Además del objetivo principal, a lo largo del trabajo se pretenden alcanzar los siguientes objetivos específicos:

1) Investigar cómo implementar el framework realizando una revisión exhaustiva de la literatura existente.
2) Diseñar y desarrollar el framework de clasificación de variantes basado en un modelo conceptual con Python y Jupyter Notebook.
3) Instanciar el framework mediante un caso de prueba basado en las guías ACMG/AMP y pacientes de cáncer hereditario.

1.3. Metodología

Para la realización de este trabajo, se ha seguido la metodología Design Science –ciencia del diseño– propuesta por Roel J. Wieringa [7], con la que se pretende diseñar e investigar cierto artefacto en un contexto.

Wieringa utiliza el concepto de artefacto para referirse a cualquier objeto creado con un propósito específico para resolver un problema práctico [7]. Este artefacto se debe aplicar y evaluar en un entorno en concreto, el cual se conoce como el contexto.

Esta metodología se considera un proceso que itera entre actividades de diseño e investigación [7]. Este proceso está formado por cinco fases que componen el ciclo de ingeniería (engineering cycle). Las tres primeras fases de este comprenden el ciclo de diseño (design cycle) y las dos últimas están destinadas a la transferencia industrial.

En la Figura 1 se puede apreciar la división del ciclo de ingeniería que realiza Wieringa, destacando en colores las tres fases del ciclo de diseño seguidas en este trabajo y dejando en gris las dos fases de la transferencia industrial.

Figura 1 - Fases del ciclo de ingeniería

1) Investigación del problema: ¿Qué fenómenos se deben mejorar y por qué?
2) Diseño de la solución: Diseñar uno o más artefactos que podrían tratar el problema.
3) Validación de la solución: ¿Estos diseños tratarían el problema?
4) Implementación de la solución: Tratar el problema con uno de los artefactos diseñados.
5) Evaluación de la implementación: ¿Qué tan exitosa ha sido la solución? Esto podría ser el inicio de una nueva iteración a través del ciclo de ingeniería.

1.3.1. Investigación del problema

La investigación del problema es la primera fase del ciclo de diseño de la metodología Design Science. El objetivo de esta fase según Wieringa [7] es identificar, describir, explicar y evaluar el problema que se pretende mejorar.

Para llevar a cabo esta investigación, se debe revisar la literatura existente relacionada con el tema para comprender el estado del arte y las soluciones previas, usando herramientas como la investigación bibliográfica o las entrevistas a expertos. Si estas técnicas no proporcionan suficiente conocimiento, existen otras técnicas de investigación científica como las encuestas, los estudios de caso observacional, los

experimentos de mecanismo de caso único y los experimentos estadísticos de diferenciación.

En este trabajo, se ha abordado la investigación del problema con un enfoque integral, comenzando con una explicación detallada de la clasificación de variantes genéticas y los métodos empleados en este proceso. Posteriormente, se han analizado las dificultades inherentes a la clasificación de variantes y se han examinado las metodologías y algoritmos actuales, identificando sus limitaciones. Además, se introducen los modelos conceptuales y metamodelos, seleccionando un metamodelo específico como base para el futuro diseño y desarrollo del framework de clasificación.

1.3.2. Diseño de la solución

Una vez identificado y comprendido el problema, se procede a la fase de diseño de la solución. En esta fase se crean uno o más artefactos, utilizando las herramientas y técnicas apropiadas, que tienen como objetivo tratar el problema identificado.

El diseño debe considerar cómo interactuará el artefacto con su contexto, lo cual implica definir claramente los requisitos del artefacto, proponer varias soluciones potenciales, seleccionar el diseño más prometedor basado en criterios predefinidos y asegurarse de que el diseño sea visible y adecuado para el contexto en el que se implementará.

Según Wieringa [7], un requisito es una característica que una parte interesada desea que tenga la solución y para la cual ha asignado recursos con el fin de hacerla realidad. Dicho de otro modo, se trata de un objetivo que la solución a diseñar debe cumplir.

Dado que la metodología Design Science sigue un proceso iterativo, si se encuentra algún error significativo durante la fase del diseño, se debe volver a la fase de investigación del problema para reformularlo adecuadamente y luego proceder con el nuevo diseño.

En este proyecto, el diseño de la solución se ha abordado siguiendo un enfoque estructurado basado en una arquitectura de tres capas, que ha servido como guía para el desarrollo del framework. En lo que respecta a la capa de datos, se ha diseñado una base de datos robusta. Para la capa de aplicación, se ha implementado un algoritmo de evaluación que realiza el proceso de clasificar variantes genéticas.

Finalmente, en la capa de presentación, se ha utilizado Jupyter Notebook como una aproximación inicial a una futura interfaz gráfica.

1.3.3. Validación de la solución

La validación de la solución es una fase crucial que asegura que los diseños propuestos cumplirán con sus objetivos cuando se implementen en el contexto del problema. Para Wieringa [7], validar la solución implica demostrar que esta contribuirá a alcanzar los objetivos de las partes interesadas una vez sea implementada en el contexto del problema.

En este trabajo, la validación de la solución se ha llevado a cabo mediante un caso de prueba con datos reales de cinco pacientes de cáncer hereditario, y utilizando las guías ACMG/AMP para evaluar la efectividad del framework diseñado y desarrollado.

2. Investigación del problema

Este apartado explica la primera fase del ciclo de diseño, la investigación del problema. En esta fase, se profundizará sobre la clasificación de variantes genéticas, una disciplina esencial en la medicina de precisión. Se abordarán varios aspectos críticos para proporcionar un contexto amplio y detallado, necesario para el desarrollo del framework.

Primero, se describirá el proceso de clasificación de variantes genéticas, explicando su importancia en la medicina personalizada, detallando cómo se lleva a cabo. Es fundamental destacar que la clasificación de variantes genéticas depende no solo de los datos genéticos en sí, sino también de la información contextual que rodea a cada variante. Esta información contextual se refiere a los datos clínicos experimentales y bioinformáticos que acompañan a las variantes y que son cruciales para su correcta interpretación.

Seguidamente, se realizará una revisión bibliográfica exhaustiva sobre la problemática en la clasificación de variantes y sobre las metodologías y algoritmos actuales para la clasificación de variantes, incluyendo enfoques basados en aprendizaje automático y guías como las ACMG/AMP. También se evaluarán las herramientas bioinformáticas existentes.

Finalmente, se presentará un metamodelo que incluye los componentes y relaciones necesarios para la clasificación de variantes. Este metamodelo detallará los datos de entrada, criterios de clasificación y procesos de evaluación. Se explicará cómo este metamodelo va a guiar la implementación técnica del framework objetivo, asegurando un sistema técnicamente sólido y accesible.

En resumen, esta fase de investigación establecerá los cimientos para el diseño y desarrollo del framework objetivo, proporcionando una comprensión profunda del problema y del enfoque adoptado.

2.1. Clasificación de variantes genéticas

La clasificación de variantes genéticas es fundamental en genética clínica y medicina de precisión, ya que influye directamente en el diagnóstico, tratamiento y predicción del riesgo de enfermedades. Al identificar la relevancia clínica de variantes específicas, esta clasificación facilita la precisión de tratamientos y la adopción de estrategias de salud más efectivas, mejorando los resultados terapéuticos y reduciendo los efectos adversos [10, 11].

Este proceso no se basa únicamente en la secuencia genética de una variante, sino también en la información contextual que la acompaña, como datos clínicos, antecedentes familiares, estudios funcionales y otras evidencias científicas.

Para realizar una clasificación precisa, se emplean guías y criterios estandarizados que evalúan diversas características de las variantes. Estos criterios permiten determinar la significancia de la variación genética en función de la evidencia científica disponible, datos de laboratorio y estudios de asociación genética, lo que asegura una evaluación coherente y precisa [12].

2.1.1. Guías ACMG/AMP

La clasificación de variantes se realiza usando guías de interpretación, como se ha comentado en la introducción. Aunque existen otras guías, la publicada en 2015 por la ACMG/AMP es una de las más ampliamente utilizadas y conocidas para facilitar la interpretación de las variantes genéticas y la toma de decisiones clínicas informadas.

En las guías ACMG/AMP, la clasificación de variantes se determina evaluando el cumplimiento de criterios específicos, cada uno de los cuales se basa en un tipo de evidencia concreta. Por ejemplo, si una variante está presente en un gen que se sabe que está relacionado con una enfermedad específica y se encuentra en varias personas afectadas, este construiría un criterio fuerte de evidencia patogénica. La aplicación de estos criterios permite clasificar las variantes de manera estructurada y replicable, mejorando la consistencia y precisión de las interpretaciones genéticas.

Richards et al. [2] dividen los criterios para clasificar las variantes patogénicas y las variantes benignas en diferentes categorías según la fuerza de la evidencia, que se clasifica en muy fuerte, fuerte, moderada y débil. Cuanto mayor sea la fuerza de un criterio, más robusta será la evidencia que apoya la patogenicidad o benignidad

de la variante. En total, hay 28 criterios diferentes que permiten una evaluación detallada y precisa de cada variante, los cuales se organizan en estas categorías de fuerza. Esta división se puede apreciar en las tablas del Anexo 2.

2.2. Problemática

Como se comenta en la motivación, el proceso de clasificación de variantes es un método intensivo en tiempo y recursos humanos, lo que, en su estado actual, lo hace insostenible a largo plazo.

La reproducibilidad en la clasificación de variantes es un factor esencial para asegurar que los resultados sean fiables y consistentes, independientemente de quién realice el análisis. Las guías de clasificación de variantes, aunque estandarizadas, son cualitativas en muchos aspectos y no proporcionan la especificidad necesaria para una aplicación uniforme [13]. Esto introduce una considerable subjetividad en su implementación práctica, lo que puede resultar en interpretaciones divergentes entre diferentes expertos clínicos, como muestran en su estudio Rehm et al. [14].

Para abordar estas limitaciones, se han desarrollado diversas herramientas automatizadas que buscan mejorar la eficiencia y reproducibilidad del proceso de clasificación de variantes. La guía ACMG/AMP 2015 ha sido fundamental para el diseño de estos sistemas y herramientas, las cuales integran datos de múltiples fuentes y aplican algoritmos avanzados para proporcionar una interpretación más consistente de las variantes genéticas, como el caso de Tavtigian et al. [15], que aplica la guía siguiendo un marco bayesiano.

La automatización mediante frameworks de clasificación no solo reduce la carga de trabajo manual, sino que también mejora la precisión al emplear algoritmos de aprendizaje automático. Estos frameworks tienen beneficios significativos como la reutilización de código y la portabilidad, lo que disminuye el esfuerzo y los costos asociados al mantenimiento [16]. Además, permiten una evaluación uniforme y objetiva de los criterios establecidos por guías como la ACMG/AMP, disminuyendo la variabilidad introducida por la interpretación humana. Esto se puede apreciar en artículos como el de Harrison et al. [17].

Tras realizar una investigación bibliográfica, se han encontrado herramientas bioinformáticas que utilizan para la guía ACMG/AMP y algoritmos de aprendizaje

automático para facilitar la clasificación de variantes. Ya se ha comentado que la guía ACMG/AMP establece criterios estandarizados para la clasificación de variantes genéticas, fundamental para el diseño de sistemas y herramientas para la clasificación automatizada.

Los algoritmos de aprendizaje automático han emergido como una herramienta poderosa en la clasificación de variantes genéticas. El aprendizaje automático puede descubrir patrones complejos en los datos difíciles de detectar con métodos tradicionales [18]. Algunos enfoques incluyen redes neuronales [19], máquinas de soporte vectorial (SVM) [20] y modelos de bosque aleatorio [21].

Algunas de las herramientas más importantes encontradas en la investigación bibliográfica son las siguientes:

- VarSome [22], InterVar [23], CardioVAI [24] y CardioClassifier [25]: Plataformas que recopilan y proporcionan información sobre variantes genéticas y su relación con fenotipos humanos.
- SIFT [26], PolyPhen [27] y CADD [28] Herramientas especializadas en la anotación y clasificación de variantes genéticas relacionadas con enfermedades cardiovasculares.

A pesar de los avances significativos en la automatización de la clasificación de variantes genéticas, aún existen limitaciones que justifican la necesidad de un nuevo enfoque. Las herramientas actuales, aunque efectivas, suelen ofrecer soluciones cerradas y no personalizables por los usuarios. Esto significa que los expertos no pueden aplicar sus propios criterios de clasificación, lo que a menudo los obliga a rehacer las clasificaciones generadas por las herramientas automáticas. Como resultado, estas herramientas, aunque útiles, pueden llevar a interpretaciones subjetivas y no uniformes.

La propuesta de este Trabajo de Fin de Grado se centra en el diseño y desarrollo de un framework flexible que no solo automatice la clasificación de variantes genéticas, sino que también permita a los expertos definir sus propios workflows y personalizar sus análisis. Este framework proporcionará una plataforma unificada que integrará múltiples algoritmos y bases de datos, mejorando así la capacidad de análisis y la reproducibilidad de los resultados. Además, permitirá mejorar la accesibilidad y facilidad de uso para los profesionales de la salud, integrando una futura interfaz intuitiva.

Utilizando como base el metamodelo que se explicará a continuación, el framework abordará las limitaciones de las herramientas actuales, ofreciendo definiciones precisas y concretas en lugar de interpretaciones subjetivas.

2.3. Metamodelo

En el ámbito de la genómica, el modelado conceptual se ha convertido en una herramienta fundamental para gestionar y analizar la creciente cantidad de datos biológicos impulsados por proyectos genómicos y nuevas técnicas [29]. Estas técnicas permiten una representación estructurada y precisa de los datos y sus relaciones, facilitando la comprensión y la comunicación entre investigadores y profesionales del área.

En este contexto, Costa et al. propusieron el metamodelo VarClaMM, diseñado con el propósito de representar de manera estructurada todos los elementos que intervienen en la clasificación, y representado mediante el lenguaje UML (Unified Modeling Language). Una primera versión de este metamodelo está disponible en [4]. La versión que se va a describir aquí está bajo proceso de revisión en la revista *Data & Knowledge Engineering* bajo el título *"VarClaMM: A Reference Meta-Model to Understand DNA Variant Classification".*

Un metamodelo es un modelo que define los conceptos de un lenguaje, las relaciones que los vinculan y las reglas estructurales que limitan qué elementos pueden formar parte de los modelos válidos, así como las combinaciones de elementos que reflejan las reglas semánticas del dominio [30].

Este metamodelo, tal como se muestra en la Figura 2, tiene tres secciones principales. En la sección representada en rosa se detallan los tres componentes principales para la clasificación de variantes. La sección naranja está dedicada a los resultados de la evaluación de estos elementos. Por último, la sección azul proporciona la información contextual que se utiliza para la clasificación. Además, en color blanco, se representan tres tipos de datos personalizados diseñados para definir atributos únicos.

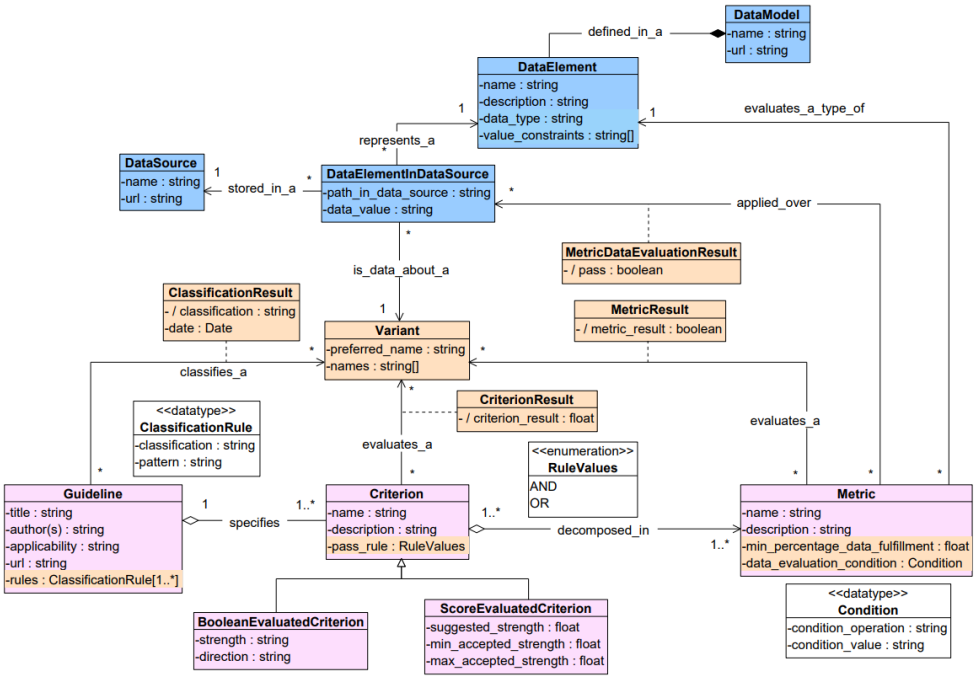

Figura 2 - Metamodelo VarClaMM

Para clasificar una variante, las guías determinan si esta se ajusta a un conjunto de especificaciones predefinidas denominadas criterios. Estos criterios pueden ser de dos tipos: *BooleanCriterion*, que se evalúa como verdadero o falso, y *ScoreCriterion*, que devuelve un valor numérico que representa el grado de cumplimiento del criterio. Ambos tipos de criterio definen métricas, que son condiciones específicas cuyo cumplimiento determina si se cumple el criterio.

Por ejemplo, un *BooleanCriterion* podría evaluar si una variante tiene una frecuencia alélica superior al 5% en bases de datos como el Exome Sequencing Project, 1000 Genomes o ExAC (criterio *BA1*), lo cual se evaluaría como verdadero o falso. Por otro lado, un *ScoreCriterion* podría calcular el porcentaje de co-segregación de una variante con la enfermedad en múltiples miembros afectados de una familia (criterio *PP1*), resultando en un valor numérico que refleje el grado de cumplimiento.

Los criterios y las métricas se aplican a una variante particular para obtener su clasificación. Esta clasificación se refleja en el atributo “classification” de la clase *ClassificationResult*, y se calcula aplicando un conjunto de reglas, definidas en el atributo “rules” de la clase *Guideline*, a los resultados de la evaluación de cada criterio o guía sobre la variante. Los resultados de estas evaluaciones se almacenan en el atributo “criterion_result” de la clase *CriterionResult*, que puede tener un valor de ‘0’ o ‘1’ para *BooleanCriterion*, o un valor numérico para *ScoreCriterion*. Por ejemplo,

para el criterio *BS2*, si una variante es observada en adultos sanos para trastorno dominante con penetrancia completa esperada en edades tempranas, el *CriterionResult* sería ‘1’ (verdadero).

Las reglas de clasificación se componen de una clasificación y un patrón representados en el tipo de datos personalizado *ClassificationRule*. Cada *CriterionResult* se determina aplicando la “pass_rule” del criterio a los resultados de las métricas definidas por el criterio, que se representan en el atributo “pass” de la clase *MetricResult*. Esta regla, según indica el tipo de datos personalizado *RuleValues*, puede ser ‘AND’, si todas las métricas deben cumplirse para que el criterio sea válido, u ‘OR’, si solo se debe cumplir una métrica como mínimo. Por ejemplo, para el criterio *PVS1*, la regla podría ser ‘AND’, requiriendo que la variante observada sea nula y que se observe en una variante predicha como LOF (pérdida de función).

Para evaluar una variante, todos los datos necesarios se encuentran de acuerdo con un *DataModel*, que proporciona una definición compartida de los datos evaluados durante el proceso de clasificación. Cada dato se representa en la clase *DataElement*, y su valor proviene de *DataSource* externos, con sus propios esquemas de datos. Por ejemplo, el valor de la frecuencia alélica de una variante podría obtenerse de bases de datos como gnomAD o ExAC. Identificar un *DataElement* específico en el esquema *DataSource* original requiere un proceso de mapeo, cuyos resultados se representan en la clase *DataElementInDataSource*. Por ejemplo, el *DataElement* podría ser la frecuencia de la variante en una base de datos de control, y el *DataElementInDataSource* especificaría cómo se mapea este dato desde un repositorio de datos genómicos externo.

Finalmente, para obtener cada *MetricResult*, se evalúan todos los *DataElementInDataSource* que representan el *DataElement* evaluado por la métrica. La evaluación se especifica en el atributo “data_evaluation_condition” de la clase *Metric*, que se compone de la operación a realizar y un valor, como indica el tipo de datos personalizado *Condition*. El resultado de esta evaluación se refleja en el atributo “pass” de la clase *MetricDataEvaluationResult*, y se usa para determinar el *MetricResult* final evaluando el porcentaje de casos en los que se cumple la condición. Si este porcentaje supera el valor especificado en el atributo “min_percentage_data_fulfillment”, de la clase *Metric*, el *MetricResult* se considerará verdadero; de lo contrario, se considerará falso. Por ejemplo, en el criterio *PP3*, si

todas las métricas indican un efecto dañino, entonces el *MetricResult* sería verdadero, lo que apoyaría la clasificación de la variante como patogénica.

Para Costa et al., basar el diseño y desarrollo de un framework de clasificación en este metamodelo ofrece varias ventajas:

1. Proporciona un marco común para la clasificación de variantes, clarificando el proceso y resolviendo aspectos implícitos o ambiguos.
2. Permite identificar inconsistencias o conflictos dentro o entre las guías de clasificación.
3. Facilita la implementación práctica de la clasificación de variantes, permitiendo que las herramientas se basen en definiciones precisas y concretas en lugar de interpretaciones subjetivas.

Esta última ventaja es especialmente relevante teniendo en cuenta las limitaciones de las herramientas actuales que, como se ha dicho en la problemática, suelen ofrecer soluciones cerradas y no personalizables.

3. Diseño y desarrollo de un framework de clasificación de variantes

Con la sólida comprensión obtenida en la fase de investigación, esta fase de diseño se enfocará en la implementación práctica del framework para la clasificación de variantes genéticas.

Primero, se explicará el concepto de arquitectura de tres capas para el diseño y desarrollo del framework objetivo: la capa de presentación, representada por un prototipo inicial simple de interfaz gráfica, diseñado con Jupyter Notebook; la capa de aplicación, representada por el algoritmo de evaluación en Python; y la capa de datos, representada por la base de datos.

A continuación, se abordará el diseño de la base de datos, que admitirá cambios en tiempo real, asegurando la integridad y la coherencia de los datos durante todo el proceso de clasificación. Para ello, se ha implementado una base de datos relacional utilizando transacciones ACID, integridad referencial y mecanismos de bloqueo y concurrencia.

Seguidamente, será el momento de desarrollar el algoritmo de evaluación con el lenguaje Python. Este componente será el corazón del framework, encargado de procesar los datos genéticos, aplicar los criterios de clasificación correspondientes y generar los resultados del proceso.

Finalmente, se creará un Jupyter Notebook como acercamiento a una futura interfaz gráfica. Esta futura interfaz proporcionará una experiencia interactiva y fácil de usar para los usuarios, permitiéndoles interactuar con el framework y visualizar los resultados de la clasificación de manera clara y comprensible.

En resumen, esta fase de diseño permitirá transformar la investigación teórica en una herramienta práctica y funcional de tres capas para la clasificación de variantes genéticas, con una base de datos dinámica, un algoritmo de evaluación sólido y un prototipo de interfaz de usuario.

3.1. Arquitectura de tres capas

La arquitectura software se refiere a la organización del sistema, abarcando tanto los componentes, sus características externas y las interacciones que se establecen entre ellos [31].

Hay una gran variedad de estilos de arquitectura software, aunque este apartado se va a centrar en la arquitectura multicapa, concretamente en la arquitectura de tres capas, que es la que se ha utilizado para el diseño y desarrollo de este framework.

El blog IBM [32] define las tres capas que comprenden la arquitectura de la siguiente forma:

1) Capa de datos: En esta capa, también conocida como de base de datos, se encuentra el almacén y la gestión de información procesada por la aplicación. Aquí es donde se almacenan los datos de manera estructurada y segura, permitiendo su acceso y manipulación según sea necesario.
2) Capa de aplicación: Esta capa también es conocida como capa lógica, porque contiene toda la lógica y el procesamiento de datos de la aplicación. Aquí se hacen los cálculos, se aplican los algoritmos y se toman las decisiones basadas en los datos proporcionados por los usuarios.
3) Capa de presentación: En esta capa, también conocida como capa de interfaz de usuario, se maneja la interacción del usuario con la aplicación. Aquí los usuarios pueden introducir datos, ajustar los parámetros y visualizar los resultados de forma accesible y comprensible.

El objetivo inicial de este proyecto, diseñar y desarrollar un framework para automatizar la clasificación de variantes, implicaba implementar el algoritmo de evaluación en Python, correspondiente a la capa de aplicación, y diseñar la base de datos, correspondiente a la capa de datos. Aunque el enfoque principal estaba en las dos primeras partes, se decidió ampliar el trabajo para incluir un prototipo inicial de la interfaz gráfica en Jupyter Notebook, que corresponde a la capa de presentación. Este prototipo será la base para desarrollar la interfaz gráfica final en futuros trabajos.

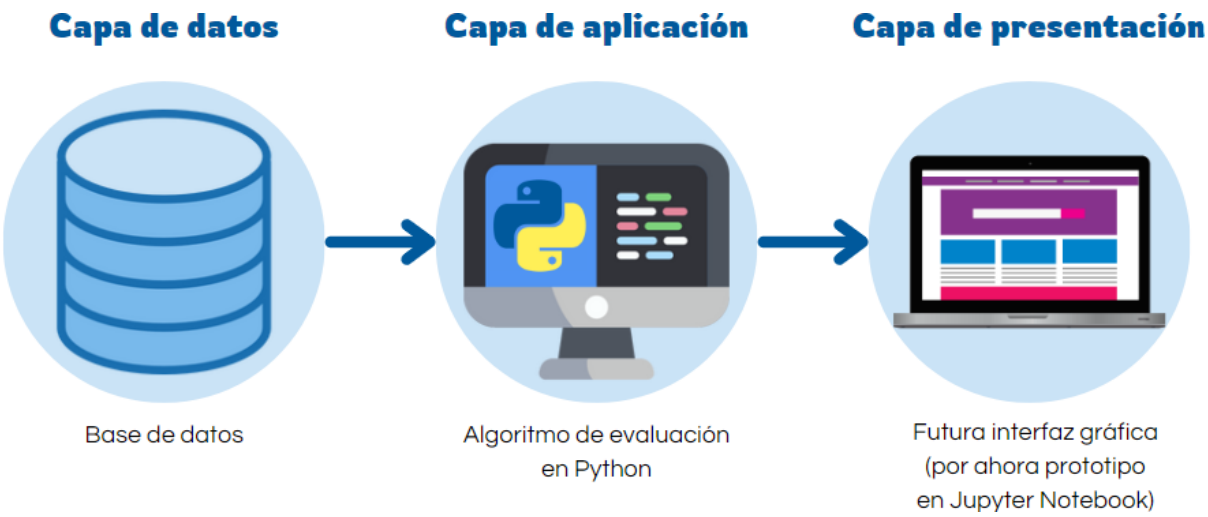

Figura 3 - Arquitectura de tres capas utilizada

En la Figura 3 se puede ver con claridad la arquitectura seguida para la realización de este trabajo.

3.2. Herramientas utilizadas

Para el diseño y desarrollo del framework se han utilizado varias herramientas especializadas, concretamente Visual Studio Code, MySQL Workbench y GitLab.

El desarrollo del código Python para este trabajo se llevó a cabo en Visual Studio Code (VS Code) [33]. Este entorno de desarrollo integrado (IDE) se ha convertido en una herramienta preferida por los desarrolladores gracias a su flexibilidad, extensibilidad y las numerosas extensiones disponibles que optimizan el flujo de trabajo. VS Code no solo soporta múltiples lenguajes de programación, sino que también facilita la depuración, la gestión de proyectos con integración de control de versiones y la instalación de extensiones específicas para Python, lo que fue fundamental para escribir y depurar el código de manera eficiente.

Para la gestión de la base de datos, se utilizó MySQL Workbench, una herramienta visual y un entorno integrado de desarrollo que facilita el diseño, modelado y administración de bases de datos MySQL de manera eficiente [34]. MySQL Workbench fue esencial para crear las tablas, ejecutar consultas SQL de manera gráfica e intuitiva, y asegurar la integridad y eficiencia del acceso a los datos durante el procesamiento. Esta herramienta es especialmente útil para trabajar con bases de datos complejas y optimizar su rendimiento.

El control de versiones del código se realizó mediante GitLab, una plataforma de desarrollo colaborativa basada en Git que permite almacenar y gestionar código fuente [35]. GitLab facilitó el seguimiento de los cambios en el código y la gestión de versiones, al igual que la colaboración con otros desarrolladores, la revisión de código y la integración continua. Estas herramientas fueron esenciales para mantener la integridad del proyecto, gestionar eficientemente el ciclo de desarrollo, y asegurar que todas las versiones del algoritmo estuvieran correctamente documentadas y respaldadas.

3.3. Base de datos

Como se mencionó anteriormente, en una arquitectura de tres capas, la capa de datos es crucial para almacenar y gestionar la información.

Una base de datos es una recopilación organizada de información o datos estructurados [36]. Las bases de datos son esenciales en el desarrollo de aplicaciones y sistemas informáticos, ya que permiten manejar grandes volúmenes de datos de manera coherente y segura. Existen bases de datos relacionales y no relacionales, cada una adecuada para diferentes tipos de aplicaciones y necesidades [37].

Las bases de datos relacionales, las cuales se han utilizado para el diseño y desarrollo del framework, organizan los datos en tablas estructuradas formalmente, conocidas como relaciones. Estas tablas están compuestas por filas y columnas, donde las columnas representan categorías de datos y las filas contienen instancias únicas de estos datos. Las bases de datos relacionales permiten aplicar dominios y restricciones para asegurar la integridad y consistencia de la información, ofreciendo capacidades transaccionales robustas con propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). El lenguaje SQL (Structured Query Language) se utiliza para acceder y modificar los datos, facilitando la realización de consultas y operaciones complejas de manera eficiente.

3.3.1. Estructura y definición de la base de datos

La base de datos utilizada para este proyecto, llamada “variant_insight_guidelines” se compone de varias tablas interrelacionadas que almacenan la información necesaria para la clasificación de variantes genéticas según las guías establecidas. Esta estructura se puede apreciar en la Figura 4.

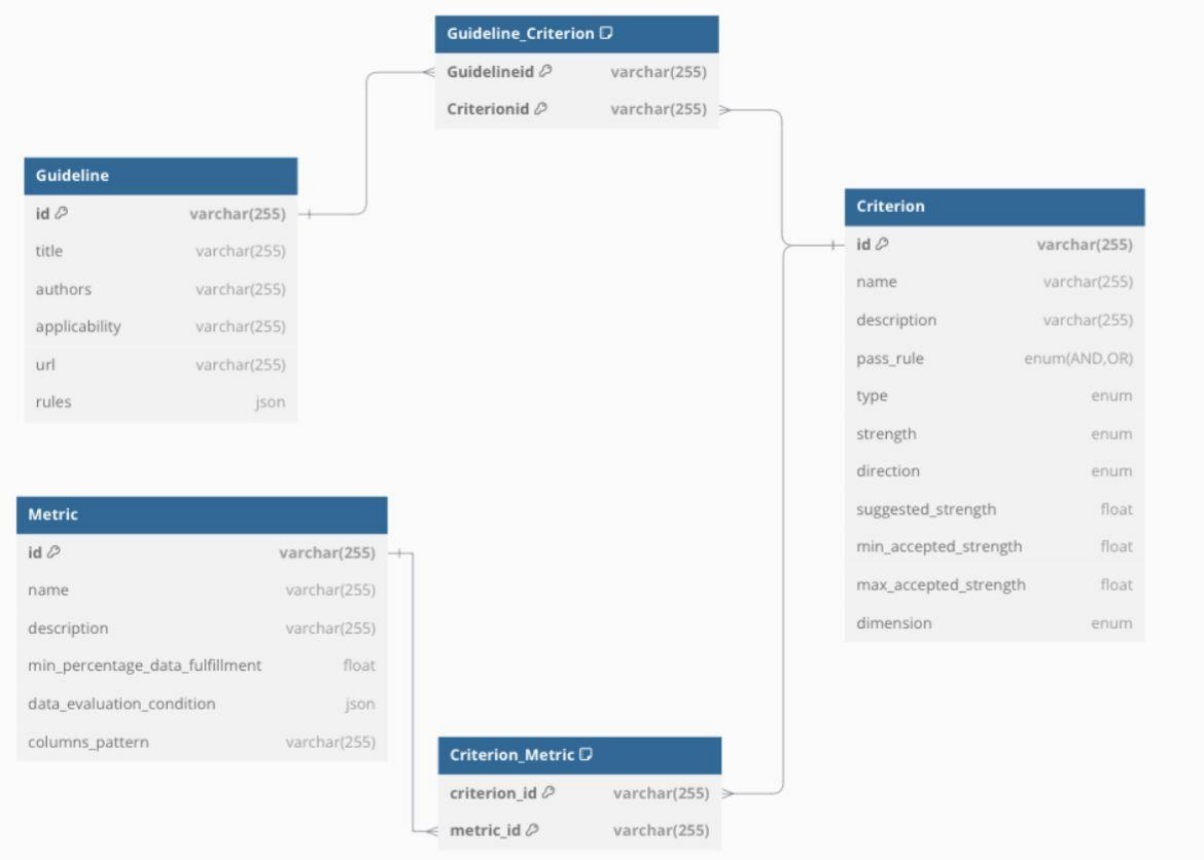

Figura 4 - Estructura de la base de datos "variant_insight_guidelines"

El núcleo de la base de datos se compone de tres tablas principales: *Guideline*, *Criterion* y *Metric*; que representan los componentes esenciales del proceso de clasificación de variantes.

La tabla *Guideline* almacena la información general sobre cada guía utilizada en la clasificación de variantes, incluyendo atributos como "id", "title", "author", "applicability", "url" y "rules". El atributo "rules", definido en formato JSON, contiene las reglas específicas que deben aplicarse en la clasificación, siguiendo la estructura de *ClassificationRule* mencionada en el metamodelo. Para asegurar la consistencia y validez de los datos, los atributos "id", "title", "author" y "rules" son obligatorios y no pueden quedar vacíos.

La tabla *Criterion* define los criterios específicos que se aplican a las variantes para determinar su clasificación, incorporando atributos clave como "id", "name", "dimension", "description", "pass_rule" y "type". Estos atributos capturan la información necesaria para identificar y contextualizar cada criterio dentro de su respectiva dimensión, permitiendo una organización lógica de los criterios según el tipo de información que evalúan, tal como se describe en el metamodelo. Por ejemplo,

un criterio dentro de la dimensión ‘Population Frequency’ analizará la frecuencia con la que una variante aparece en la población, mientras que un criterio en ‘Variant Type and Location’ se enfocará en la naturaleza y localización de la variante en el genoma.

El atributo “pass_rule” de la tabla *Criterion* se define como ‘AND’ u ‘OR’, reflejando el tipo de lógica aplicada para validar el cumplimiento de un criterio. Un criterio con la regla ‘AND’, por ejemplo, requeriría que todas las métricas relacionadas con ese criterio sean verdaderas para que el criterio sea validado. En la tabla *Criterion*, los atributos críticos como “id”, “name”, “description”, “pass_rule” y “type” son obligatorios, garantizando que cada criterio esté completamente definido y sea aplicable en el proceso de clasificación.

Dependiendo del tipo de criterio (‘boolean’ o ‘score’), la tabla *Criterion* puede incluir atributos adicionales específicos para cada tipo. Para los criterios de tipo ‘boolean’, se añaden los atributos “strength” y “direction”, que indican la fuerza del criterio y su dirección (‘pathogenic’ o ‘benign’), respectivamente. Por ejemplo, un criterio de tipo ‘boolean’ con una fuerza ‘strong’ y dirección ‘pathogenic’ sugiere que el criterio es fuerte y que la variante es patogénica. En el caso de los criterios de tipo ‘score’, se incluyen los atributos “suggested_strength”, “min_accepted_strength” y “max_accepted_strength”, que definen la fuerza sugerida y el rango de fuerza aceptada para ese criterio. Por ejemplo, un criterio ‘score’ podría evaluar la frecuencia alélica de una variante, sugiriendo una “suggested_strength” de ‘moderate’ con un rango aceptado de 0.1 a 0.5. Aunque estos atributos adicionales son importantes, se permite que queden vacíos si no son aplicables al tipo de criterio definido.

La tabla *Metric* almacena las métricas que se utilizan para evaluar si un criterio se cumple o no. Esta tabla incluye los atributos “id”, “name”, “description”, “columns_pattern”, “min_percentage_data_fulfillment” y “data_evaluation_condition”. El atributo “data_evaluation_condition”, definido en formato JSON, captura las condiciones específicas que deben cumplirse para que una métrica sea considerada válida, siguiendo la estructura de *Condition* establecida en el metamodelo. Como en las tablas anteriores, todos los atributos en la tabla *Metric* son obligatorios, asegurando que cada métrica esté claramente definida y lista para su aplicación en el proceso de evaluación.

Por ejemplo, el criterio *BA1* requiere que la frecuencia alélica de la variante sea mayor al 5%, especificando esto en la métrica *frequence_greater_than_0.05*. El

“columns_pattern” podría especificar que esta métrica se aplique a columnas que contengan datos de frecuencia de alelos y la “min_percentage_data_fulfillment” podría estipular que al menos el 90% de los datos deben cumplir con esa condición para que la métrica sea considerada válida.

Además de las tablas principales, la base de datos incluye dos tablas intermedias: *Guideline_Criterion* y *Criterion_Metric*; que establecen la relación entre las guías y criterios, y entre los criterios y las métricas, respectivamente. Por ejemplo, la tabla *Guideline_Criterion* podría asociar una guía específica con varios criterios, como por ejemplo los criterios *BP6 y BS2*. De manera similar, la tabla *Criterion_Metric* podría vincular el criterio *BA1* con la métrica *frequence_greater_than_0.05*. Estas tablas intermedias facilitan la gestión de las relaciones entre los componentes del sistema, permitiendo una organización flexible y escalable de la información necesaria para el proceso de clasificación.

Se puede apreciar que se han añadido algunos atributos no definidos en el metamodelo, como “columns_pattern” en la tabla *Metric*, o “dimension” y “type” en la tabla *Criterion*. Estos atributos se añadieron para adaptar mejor el concepto teórico a la implementación tecnológica, facilitando así la evaluación de los criterios en el algoritmo de evaluación.

Para insertar, eliminar, modificar o consultar los datos dentro de esta base de datos se utilizan sentencias SQL (*INSERT*, *DELETE*, *UPDATE* y *SELECT*) directamente desde el algoritmo de evaluación que se presentará a continuación.

El diseño de la base de datos incluye varias características clave para asegurar mantener la integridad y coherencia de los datos:

1. Transacciones ACID: Como se explica anteriormente, las bases de datos relacionales ofrecen transacciones ACID, garantizando que todas las operaciones de modificación de datos sean atómicas, consistentes, aisladas y duraderas.
2. Integridad referencial: Las tablas están interconectadas mediante claves foráneas. Por ejemplo, la tabla *Guideline_Criterion* tiene claves foráneas que referencian a las tablas *Guideline* y *Criterion* para asegurar una relación entre ellas, y la tabla *Criterion_Metric* tiene claves foráneas que referencian a las tablas *Criterion* y *Metric*. Estas relaciones aseguran que los datos se

mantengan consistentes y que las evaluaciones se realicen correctamente siguiendo la estructura del metamodelo VarClaMM.

3. Bloqueo y concurrencia: Las bases de datos relacionales también implementan mecanismos de bloqueo para gestionar el acceso concurrente a los datos, asegurando que los cambios realizados en tiempo real no generen conflictos ni inconsistencias.

3.4. Algoritmo de evaluación

Siguiendo el esquema de la arquitectura de tres capas utilizado, la capa de aplicación es crucial para el procesamiento de la información almacenada en la base de datos. Este nivel ejecuta el algoritmo de evaluación, transformando los datos en información significativa y útil, que posteriormente será mostrada por la capa de presentación.

Los principales lenguajes de programación utilizados en bioinformática son los siguientes [38]:

1. Python: Lenguaje de alto nivel y orientado a objetos, con semántica dinámica e interpretable, muy utilizado debido a su legibilidad y facilidad de uso.
2. Perl: Lenguaje de programación interpretado y versátil, conocido por su potente capacidad de manipulación de texto.
3. R: Lenguaje de programación especializado en análisis estadístico y visualización de datos, conocido por su amplia gama de paquetes y facilidad para realizar análisis complejos.
4. Java: Lenguaje orientado a objetos, conocido por su portabilidad y capacidad para manejar grandes volúmenes de datos.
5. C: Lenguaje de programación eficiente y rápido, que ofrece un alto control sobre los recursos del sistema.

A partir del blog Omics Tutorials [38], y respaldado por el estudio de Fourment y Gillings [39], se puede concluir que Python es ideal para crear scripts y herramientas de automatización, Perl, aunque ahora está en desuso, es especialmente adecuado para la manipulación y procesamiento de texto, R es muy adecuado para el análisis estadístico y la visualización de datos complejos, Java es excelente para

herramientas que necesitan ejecutarse en diversas plataformas con alta eficiencia y estabilidad, y C es el mejor para tareas intensivas del sistema.

La elección de Python para el diseño y desarrollo del framework de automatización es justificada por su sintaxis sencilla y legibilidad. Además, Python tiene muchos módulos y paquetes especializados en bioinformática [40], lo que permite implementar soluciones eficientes para el procesamiento y análisis de datos genéticos, reduciendo significativamente el tiempo y esfuerzo de desarrollo. A continuación, se detallarán los paquetes específicos de Python utilizados en el diseño y desarrollo del framework.

3.4.1. Paquetes utilizados

Uno de los paquetes más importantes de Python es “pandas” [41], que ofrece estructuras de datos avanzados y un conjunto de funciones que simplifican y aceleran el procesamiento de datos. Este paquete se distingue por su eficiencia gracias al uso de librerías internas especializadas y la implementación de operaciones vectorizadas. Estas características permiten manejar grandes volúmenes de datos de manera rápida y eficaz, facilitando tareas como la manipulación, el análisis y la visualización de datos.

Inicialmente, en una primera versión del algoritmo, se utilizó el método ‘read_csv’ para leer ficheros en formato CSV, ya que la información necesaria para la clasificación de variantes, es decir, los criterios pertenecientes a la guía utilizada para la evaluación y las métricas que los componen se extraían de varios ficheros CSV. Sin embargo, con la introducción de una base de datos, se optó por obtener la información en tiempo real, lo que hizo innecesaria esta solución inicial.

En el desarrollo del algoritmo, los DataFrames de “pandas” han sido esenciales para almacenar, evaluar y actualizar los resultados de varias métricas o criterios sobre un conjunto de datos. Los DataFrames son estructuras de datos bidimensionales, similares a una tabla, que organizan los datos en filas y columnas. Esta organización facilita la identificación y el análisis de las relaciones entre las diferentes variables en un conjunto de datos [42].

El módulo “enum” también ha sido muy importante a la hora de realizar este algoritmo. La clase ‘Enum’ de este módulo se utilizó para crear las enumeraciones necesarias que representan los tipos de datos personalizados definidos en el

metamodelo. Esto permitió implementar de manera efectiva las reglas y condiciones utilizadas en la clasificación de las variantes genéticas.

Para la conexión a la base de datos, se utilizó el módulo “mariadb”, que facilita la interacción con la base de datos MySQL utilizada en este proyecto. Este módulo permitió realizar las consultas necesarias para obtener, actualizar y almacenar la información utilizada para el algoritmo.

3.4.2. Estructura del algoritmo de evaluación

El algoritmo de evaluación ha sido implementado en Python, y su funcionalidad se organiza en varios módulos que se agrupan en diferentes archivos dentro del proyecto. Esta estructura modular está diseñada para facilitar la comprensión, mantenimiento y extensión del código.

En primer lugar, el proyecto incluye una carpeta “db” dedicada a la capa de datos, que contiene el archivo “create_database.sql”. Este archivo es fundamental para la creación de la base de datos, tal como se detalló en el apartado anterior. La separación de la capa de datos asegura que todos los elementos relacionados con la gestión de la base de datos estén centralizados y sean fácilmente accesibles.

Otra carpeta en el proyecto es la correspondiente a la capa de aplicación. La carpeta “src”, la cual es el núcleo del algoritmo de evaluación, incluye el archivo “main.py”, que coordina el proceso de clasificación, y el archivo “connection.py”, que gestiona la conexión con la base de datos. Además, se encuentran los archivos “guideline.py”, “criterion.py” y “metric.py”, que implementan los métodos esenciales para la clasificación de variantes genéticas.

Finalmente, el proyecto cuenta también con una carpeta correspondiente a la capa de presentación, la carpeta “notebook”. En esta carpeta se encuentran los archivos necesarios para el desarrollo de la interfaz gráfica. Sin embargo, por el momento, esta carpeta solo alberga los archivos que permiten la creación del notebook que funciona como primera aproximación a la interfaz gráfica final. En el apartado correspondiente al Jupyter Notebook se ofrecerá una explicación más detallada sobre esta carpeta y su contenido.

A continuación, se describe el propósito de cada uno de los archivos y su interrelación en el contexto del sistema. Primero, *connection.py* establece las conexiones esenciales para la gestión de datos. Luego, *main.py* actúa como el archivo inicial que integra y ejecuta todos los métodos del sistema. Después, los archivos

guideline.py, *criterion.py* y *metric.py* se encargan de aplicar las guías, criterios y métricas específicos, respectivamente, para realizar la clasificación de variantes genéticas. Este orden facilita la comprensión del flujo del programa, desde la conexión a la base de datos hasta la aplicación de guías y criterios específicos.

3.4.3. Definición del algoritmo de evaluación

connection.py

Este archivo se encarga de la conexión con la base de datos mediante el uso de un *ConnectionPool* de MariaDB. Los métodos definidos aquí, como *get_connection_pool*, *get_connection*, *close_connection_pool* y *query*, permiten establecer y cerrar conexiones, así como ejecutar consultas SQL de manera segura y eficiente para recuperar o insertar datos en la base de datos.

El método *query* es particularmente relevante, ya que centraliza la ejecución de todas las consultas SQL dentro del algoritmo, asegurando que todas las interacciones con la base de datos sean consistentes, seguras y manejadas de manera uniforme a lo largo del código. Este método se puede ver con claridad en la Figura 5.

```
def query(type, query, where=None):
        try:
                pconn = get_connection()
                cur = pconn.cursor()
                if where: cur.execute(query, where)
                else: cur.execute(query)
                if type.upper() == “INSERT”: pconn.commit()
                else: return cur
        except Exception as e: print(“Error: ”, e)
        finally: close_connection_pool()
```

Figura 5 - Método ‘query’

Este método gestiona tanto operaciones de lectura como de modificación de datos. Primero, establece una previa conexión con la base de datos y crea un cursor para ejecutar la consulta que previamente se especificará. El parámetro *type* es un atributo que indica el tipo de operación SQL que se va a realizar, como una inserción de datos (‘INSERT’) o una consulta de datos (‘SELECT’). El parámetro *where=None* es opcional y permite pasar condiciones específicas para la consulta, lo que hace posible filtrar los resultados según criterios.

Dependiendo de si la operación es un 'INSERT', realiza un *commit* para guardar los cambios; si es una consulta 'SELECT', devuelve los resultados mediante el cursor. El método maneja cualquier error que pueda ocurrir y asegura que la conexión se cierre adecuadamente al finalizar, optimizando así el uso de recursos.

main.py

El archivo "main.py" actúa como el punto de entrada principal del algoritmo de clasificación. Este código se estructura en torno a un método principal llamado *main*, el cual se encarga de coordinar la evaluación de variantes genéticas de acuerdo con una guía específica. Este método acepta dos parámetros: *variant_data*, que es un DataFrame que contiene los datos de las variantes genéticas a evluar; y *guideline_id*, que es el identificador de la guía que se utilizará durante la evaluación.

El primer paso en el método *main* es realizar la consulta a la base de datos que se muestra en la Figura 6, la cual sirve para recuperar la guía correspondiente usando el identificador *guideline_id*. Esta consulta se efectúa mediante el método *conn.query*, definido en el archivo "connection.py", el cual ejecuta una sentencia SQL que selecciona todos los campos de la tabla *Guideline* donde la *id* coincide con el valor de *guideline_id*.

```
cur = conn.query("select", "SELECT * FROM Guideline WHERE id = %s",
(guideline_id,))
```

Figura 6 - Obtener la guía deseada

El resultado de esta consulta se utiliza para instanciar un objeto de la clase *Guideline* como se ve en la Figura 7. La instancia se crea iterando sobre los resultados obtenidos en la base de datos y asignando los valores de los campos a los atributos de la clase. Una vez que se ha creado la instancia, se procede a cargar los criterios asociados a la guía utilizando el método *add_criteria*.

```
for (id, title, authors, applicability, url, rules) in cur:
        guidelines_instance = Guideline(...)
guidelines_instance.add_criteria()
```

Figura 7 - Instanciar una guía

Finalmente, el método *evaluate_guideline* de la instancia de *Guideline* se utiliza para evaluar las variantes genéticas presentes en *variant_data*. Este proceso de evaluación clasifica las variantes de acuerdo con los criterios y reglas definidos en la

guía, y el resultado es una lista de variantes clasificadas que se devuelven al finalizar la ejecución del método *main*. Esto se puede observar en la Figura 8.

```
variants_classified = guideline_instance.evaluate_guideline(variant_data)
return variant_classified
```

Figura 8 - Inicio del proceso de evaluación

guideline.py

El archivo “guideline.py” implementa la clase *Guideline*, encapsulando la lógica para evaluar una guía de clasificación. La clase *Guideline* incluye el método *add_criteria*, que carga los criterios desde la base de datos y los agrega a la guía. Los criterios se obtienen mediante una consulta SQL y se instancian como objetos de la clase *BooleanCriterion*, como se puede ver en la Figura 9.

```
cur = conn.query(“select”, “SELECT * FROM Criterion JOIN Guideline_Criterion ON Criterionid = id WHERE Guidelineid = %s”, (self.id,))
criteria_instance = []
if cur is not None:
        for row in cur:
                criterion = BooleanCriterion(...)
                criteria_instance.append(criterion)
```

Figura 9 - Instanciar un criterio

Una vez instanciados, los criterios se complementan con sus métricas correspondientes a través del método *add_metrics*. Estos criterios se almacenan en una lista dentro de la guía para su posterior evaluación. Este proceso se puede apreciar en la Figura 10.

```
for crit in criteria_instance: crit.add_metrics()
self.criteria = criteria_instance
```

Figura 10 - Almacenamiento de los criterios

Con los criterios cargados, el siguiente paso es la evaluación de las variantes genéticas, la cual se realiza en el método *evaluate_guidelines*. Este método organiza los criterios en dos grupos, *pathogenic_criteria* y *benign_criteria*, según la dirección del criterio (patogénico o benigno). Estos criterios, que previamente han sido añadidos a la guía mediante el método *add_criteria*, son evaluados contra un conjunto de datos de variantes genéticas, almacenado en un DataFrame de “pandas”.

El resultado de la evaluación de cada criterio se suma para determinar cuántos criterios se cumplen en cada categoría, y se agregan estas sumas al DataFrame original, como se ve en la Figura 11. Esta evaluación preliminar prepara los datos para la siguiente fase del proceso.

```
for key, value in pathogenic_criteria.items():
        result = record[value].sum(axis=1)
        record = record.join(result.to_frame(name = 'pathogenic_' + key))
for key, value in benign_criteria.items():
        result = record[value].sum(axis=1)
        record = record.join(result.to_frame(name = 'benign_' + key))
```

Figura 11 - Criterios por categoría

Finalmente, después de la evaluación preliminar de los criterios, se invoca el método *evaluate_classification_rules*, que aplica las reglas de clasificación definidas en la guía. Este método revisa las reglas de clasificación y asigna una categoría a cada variante según los resultados de estas reglas. Si una variante no cumple con ninguna de las reglas específicas, se clasifica como "VUS".

criterion.py

El archivo "criterion.py" define la estructura y funcionalidad de los criterios que se aplican durante la clasificación. En este archivo se encuentran las clases *Criterion* y *BooleanCriterion*, además de la enumeración *RuleValues*. Esta enumeración define si el resultado de todas las métricas tiene que ser verdadero o solo el de alguna de ellas. Este concepto es fundamental, ya que determina cómo se aplica la lógica de evaluación en los criterios. Aunque se planificó la implementación de la clase *ScoreCriterion*, sólo se implementó *BooleanCriterion,* ya que es la más utilizada, mientras que la otra se considera como trabajo futuro.

El método *add_metrics* de la clase *Criterion* asocia las métricas relevantes a un criterio particular a través de una consulta a la base de datos. Las métricas se obtienen mediante una consulta SQL, y se instancian como objetos de la clase *Metric*. Estos objetos luego son agregados a la lista de métricas del criterio para su posterior evaluación. En la Figura 12 se muestra el proceso de obtener e instanciar objetos de la clase *Metric*.

```
cur = conn.query("select", "SELECT * FROM Metric JOIN Criterion_Metric ON metric_id = id WHERE criterion_id = %s", (self.id,))

metrics_instance = []

if cur is not None:
        for row in cur: metrics_instance.append(metric)
```

Figura 12 - Instanciar una métrica

La clase *BooleanCriterion* extiende la funcionalidad de la clase *Criterion*. El método *evaluate_boolean_criterion* de esta clase es clave, ya que aplica la lógica para evaluar si el criterio se cumple o no, basándose en las métricas. Según el valor de *pass_rule*, el método evalúa, como se ve en la Figura 13, si todas las métricas ('AND') o alguna de ellas ('OR') cumplen con las condiciones establecidas, retornando un resultado booleano que indica el cumplimiento del criterio.

```
if self.pass_rule == RuleValues.AND.value:
        criterion_result = record[metrics_ids].all(axis=1)
        record = record.join(criterion_result.to_frame(name=self.id))

elif self.pass_rule == RuleValues.OR.value:
        criterion_result = record[metrics_ids].all(axis=1)
        record = record.join(criterion_result.to_frame(name=self.id))
```

Figura 13 - Evaluación según el pass_rule

metric.py

El archivo "metric.py" se centra en la implementación de las métricas utilizadas para evaluar los datos, a través de la clase *Metric* y las enumeraciones *EvaluationValues* y *ConditionValues*. Por un lado, *EvaluationValues* define las diferentes condiciones que se pueden aplicar a las columnas evaluadas. Estas condiciones incluyen si el valor de una columna debe ser igual ('EQUALS') o diferente ('DISTINCT') a una cadena especificada, si debe ser menor ('MINOR') o mayor ('MAJOR') a un número determinado, si está ('IN_RANGE') o no ('NOT_IN_RANGE') contenido en un rango de números, o si está ('IN_LIST') o no ('NOT_IN_LIST') incluido en una lista de caracteres. Por otro lado, *ConditionValues* establece si la métrica debe cumplirse en todas las filas, en una sola fila, o en un porcentaje específico de filas, utilizando las opciones 'ALL', 'ONE' o 'VALUE', respectivamente.

Este archivo comienza con el método *select_columns*, que se encarga de seleccionar las columnas correspondientes a los *DataElements* que se desean

evaluar. Estas columnas provienen de un archivo con la información sobre las variantes anotadas, incluyendo todos los *DataElements* y *DataSources* necesarios para el proceso de evaluación. Luego, el método *evaluate_data* aplica las reglas de evaluación a estas columnas seleccionadas, como se puede comprobar en la Figura 14.

```
def evaluate_data(self, record, columns_to_evaluate):
        data_eval_functions_dict = {
                EvaluationValues.DISTINCT.value: lambda row: row.astype(str) !=
                str(self.data_evaluation_value),
                EvaluationValues.EQUALS.value: lambda row: row.astype(str) ==
                str(self.data_evaluation_value),
                EvaluationValues.MINOR.value: lambda row: row.astype(float) <
                float(self.data_evaluation_value),
                EvaluationValues.MAJOR.value: lambda row: row.astype(float) >
                float(self.data_evaluation_value),
                EvaluationValues.IN_LIST.value: lambda row:
                row.astype(str).isin(self.data_evaluation_value.split(‘,’)),
                EvaluationValues.NOT_IN_LIST.value: lambda row:
                ~(row.astype(str).isin(self.data_evaluation_value.split(‘,’))),
                EvaluationValues.IN_RANGE.value: lambda row:
                row.astype(float).between(float(self.data_evaluation_value.split(‘,’)[0])
                , float(self.data_evaluation_value.split(‘,’)[1])),
                EvaluationValues.NOT_IN_RANGE.value: lambda row:
                ~(row.astype(float).between(float(self.data_evaluation_value.split(‘,’)[
                0]), float(self.data_evaluation_value.split(‘,’)[1])))
        }
        result =
        record[columns_to_evaluate].apply(data_eval_functions_dict[self.data_eval
        uation_rule], axis=1).sum(axis=1)
        return result
```

Figura 14 - Método ‘evaluate_data’

El resultado de *evaluate_data* es un conteo de las filas que cumplen con las condiciones especificadas que, posteriormente, el método *evaluate_metric* utiliza para determinar si la métrica se cumple en función del porcentaje mínimo definido por *ConditionValues* y el atributo *min_percentage_data_fulfillment*.

3.5. Jupyter Notebook

Para completar la arquitectura de tres capas, la capa de presentación de este proyecto ofrece un prototipo inicial de interfaz gráfica, mediante Jupyter Notebook,

que permite al usuario final interactuar con los datos procesados por la capa de aplicación.

Una interfaz gráfica de usuario (GUI) permite la interacción entre las personas y las computadoras [43]. Una buena GUI hace que una aplicación sea sencilla, práctica y eficaz de usar [44].

Los Notebooks son ampliamente preferidos porque permiten integrar texto en lenguaje natural, código fuente y sus resultados en un único documento interactivo [45], lo que facilita la presentación y el análisis de datos de manera más dinámica y comprensible.

El funcionamiento de los Notebooks se organiza en celdas [46]. Estas celdas son fragmentos de código que pueden modificarse y ejecutarse individualmente, con sus resultados apareciendo debajo de cada celda y almacenados como parte del documento. Esto permite la inclusión de salidas enriquecidas, como gráficos, ecuaciones matemáticas formateadas y elementos interactivos.

Jupyter es un proyecto de código abierto que puede trabajar con múltiples lenguajes de programación gracias a los diferentes motores de lenguaje, o kernels, que se comunican con Jupyter mediante un protocolo común y documentado [46].

El uso de Python en Jupyter Notebook permite una integración fluida de diversas bibliotecas y herramientas que son fundamentales para el análisis y la clasificación de variantes genéticas.

3.5.1. Paquetes utilizados

En la implementación del notebook se utilizaron varios paquetes de Python esenciales para facilitar el manejo de datos, la creación de identificadores únicos y la validación de entradas, entre otras funcionalidades.

El paquete “prettytable” es una herramienta que facilita la presentación de datos en un formato tabular, creando tablas ASCII que son visualmente atractivas y fáciles de leer [47]. La clase ‘PrettyTable’ de este paquete se utilizó para formatear y mostrar las tablas de la base de datos de manera organizada y fácil de leer.

Para la generación de identificadores únicos al insertar una guía, un criterio o una métrica en alguna tabla de la base de datos, se emplea el paquete ‘uuid’. Este paquete garantiza que cada entrada tenga un identificador único.

El paquete “re” proporciona soporte para expresiones regulares [48], las cuales son útiles para buscar, sustituir y manipular texto basado en patrones. En este caso, se utilizó para dividir la entrada del usuario en una lista de valores separados por comas, facilitando la manipulación de atributos que pueden tener múltiples valores.

Finalmente, como se utilizan algunos atributos que deben tener formato JSON, el paquete “json” se utilizó para verificar si el formato del atributo es correcto.

3.5.2. Estructura del notebook

Como se ha comentado en el punto anterior, los archivos utilizados para el desarrollo del notebook están contenidos en la carpeta con el mismo nombre, al igual que el notebook que funciona como aproximación a la interfaz gráfica. Estos archivos constituyen un sistema modular diseñado para la creación y gestión de guías, criterios y métricas en una base de datos. Cada archivo cumple una función específica dentro del proceso, como la validación y preparación de la entrada del usuario o la creación y almacenamiento de los elementos. Además, el archivo correspondiente al notebook integra todas estas funcionalidades para ofrecer la aproximación a la interfaz gráfica.

Es importante destacar que, para realizar la conexión a la base de datos, se requiere la importación del archivo “connection.py”, cuya funcionalidad ya ha sido explicada en la descripción del algoritmo de evaluación.

A continuación, se detalla el rol de cada archivo de este proceso, comenzando por aquellos que establecen las bases para la creación y validación de datos, hasta llegar al archivo principal que conecta y ejecuta todas estas funciones en el entorno del notebook. Este orden no solo refleja la lógica de la implementación, sino que facilita la comprensión del flujo de trabajo, desde las verificaciones iniciales hasta la integración completa de los elementos en el sistema.

Después de la definición del notebook, se proporcionará un ejemplo de uso para mostrar visualmente cómo funciona el prototipo de interfaz utilizado en el trabajo.

3.5.3. Definición del notebook

notebook.ipynb

El archivo “notebook.ipynb” es un Jupyter Notebook que proporciona un entorno interactivo de ejecución que integra todo el código y facilita la interacción del usuario. Este importa el archivo “create.py” para poder utilizar sus métodos.

Dentro del notebook, se utiliza el método *show*, que permite mostrar las relaciones almacenadas en la base de datos, tales como la relación entre guías y criterios, y la relación entre criterios y métricas. Además, el notebook también invoca el método *create_and_save* para permitir la creación i almacenamiento de guías, criterios y métricas de manera integrada.

En el siguiente apartado, se mostrará un ejemplo del uso de este notebook en el que se insertará una guía en la base de datos. A esta guía se le asignará un criterio con una métrica, insertando también en la base de datos tanto el criterio como la métrica, junto con las relaciones entre estos tres elementos.

notebook_checks.py

El archivo “notebook_checks.py” contiene una serie de funciones especiales para la validación de entradas de usuario en aplicaciones que requieren la creación y gestión de guías, criterios y métricas. Estos métodos aseguran que los datos ingresados cumplan con los formatos esperados, previniendo errores en el procesamiento posterior.

El método *check* se encarga de validar entradas simples del usuario. Una entrada simple es aquella que consiste en un único valor que no requieran un procesamiento complejo, como la validación de múltiples elementos o la conversión a un formato específico como JSON. Por ejemplo, una entrada simple podría ser un texto, un número o una selección de opciones predefinidas.

Inicialmente, el método *check* verifica que la entrada no esté vacía. Si se proporcionan opciones predefinidas, también comprueba que la entrada coincida con una de estas opciones. Además, en casos específicos, como cuando el mensaje está relacionado con valores numéricos, el método valida que la entrada sea un número flotante. Esta versatilidad permite usar este método en una variedad de contextos, adaptándose a diferentes tipos de validaciones. La funcionalidad de este método se puede apreciar en la Figura 15.

```
def check(mensaje, opciones=None):
        while True:
                entrada = input(mensaje).strip()
                if not entrada: print("Error: No se puede dejar en blanco")
                elif opciones and entrada.upper() not in opciones:
                        print("Error: Valor no válido. Opciones: {', '. join(opciones)}")
                elif mensaje in {"Suggested_strength: ", "min_accepted_strength: ",
                "max_accepted_strength: "}:
                        try:
                                float(entrada)
                                return entrada
                        except ValueError: print("Error: Valor no válido")
                else: return entrada
```

Figura 15 - Método 'check'

El método *check_combined* está diseñado para manejar entradas que puedan incluir múltiples valores, separados por comas. Convierte la cadena de entrada en una lista, permitiendo así que se gestionen múltiples opciones de manera eficiente, como se ve en la Figura 16.

```
def check_combined(mensaje):
        while True:
                entrada = input(mensaje).strip()
                if not entrada: print("Error: No se puede dejar en blanco")
                else: return [elem.strip() for elem in re.split(r',\s*|,', entrada)]
```

Figura 16 - Método 'check_combined'

El método *check_json* asegura que la entrada del usuario tenga un formato JSON válido, como se ve en la Figura 17, asegurando que los datos estructurados se almacenen correctamente.

```
def check_json(mensaje):
        while True:
                entrada = input(mensaje).strip()
                if not entrada: print("Error: No se puede dejar en blanco")
                try:
                        json.loads(entrada)
                        return entrada
                except ValueError: print("Error: Formato JSON no válido")
```

Figura 17 - Método 'check_json'

Por último, el método *check_value* es un poco más complejo y se encarga de validar los valores de evaluación de datos en función de la regla de evaluación seleccionada, de la misma forma que en la Figura 18.

```
def check_value(rule):
        while True:
                entrada = input(f"Data_evaluation_value ({rule} {'número' if rule in
                {'MAJOR', 'MINOR'} else 'valores'}): ").strip()
                if not entrada: print("Error: No se puede dejar en blanco")
                elif rule in {"MAJOR", "MINOR"}:
                        try:
                                float(entrada)
                                 return entrada
                        except ValueError: print("Error: Valor no válido")
                elif rule in {"IN RANGE", "NOT IN RANGE"}:
                        if " " not in entrada: print("Error: Deben ser dos números")
                else:
                        menor, mayor = entrada.split()
                        try:
                        me, ma = float(menor), float(mayor)
                        if me, ma = print("Error: El primer número debe ser menor que
                        el segundo")
                        else: return f"[{menor} - {mayor}]"
                        except ValueError: print("Error: Valor no válido")
                elif rule in {"IN LIST", "NOT INLIST"}: return [elem.strip() for elem in
                re.split(r',\s*|,', entrada)]
                else: return entrada
```

Figura 18 - Método 'check_value'

Dependiendo de cuál sea la regla (atributo rule, el método valida que la entrega sea del tipo y formato correcto.

- Si *rule* es 'MAJOR' o 'MINOR', se verifica que la entrada sea un número.
- Si *rule* es 'IN RANGE' o 'NOT IN RANGE', se verifica que se ingresen dos números separados por un espacio, y que el primer número no sea mayor que el segundo, formateando luego estos números en un rango.
- Si *rule* es 'IN LIST' o 'IN NOT LIST', se convierte la entrada en una lista de valores separados por comas.
- Si *rule* es 'EQUALS' o 'DISTINCT', se verifica que la entrada sea una cadena alfanumérica.

create.py

El archivo “create.py” actúa como el punto de entrada principal para la creación y gestión de guías, criterios y métricas. Este archivo organiza el proceso de creación mediante el método principal *create_and_save*, y contiene también el método *show*, con el que se realizan consultas a la base de datos para mostrar los registros actuales de una tabla específica, formateando la salida con ‘PrettyTable’.

El método show se utiliza para consultar y mostrar el contenido de una tabla en particular. Primero, ejecuta una consulta SQL el método *query*, importado del archivo “connection.py”, para seleccionar todos los registros de la tabla especificada (*tab*). Luego, utiliza *PrettyTable* para formatear y mostrar los datos. Las columnas de la tabla se definen según el tipo de elemento utilizando un diccionario *field_names*. Finalmente, se itera sobre los resultados de la consulta para agregar cada fila a la tabla formateada, que luego se imprime. Este método se ilustra en la Figura 19.

```
def show(tab):
        cur = conn.query(“select”, f“SELECT * FROM {tab}”)
        table = PrettyTable()
        field_names = { “Guideline”: [...], “Guideline_Criterion”: [...], “Criterion”: [...],
        “Criterion_Metric”: [...], “Metric”: [...] }
        table.field_names = field_names.get(tab)
        for row in cur: table.add_row(row)
        print(table)
```

Figura 19 - Método ‘show’

Por otro lado, el método *create_and_save* comienza solicitando al usuario que seleccione el tipo de elemento que desea crear. Luego, muestra el contenido actual de la tabla correspondiente utilizando el método *show*, ofreciendo al usuario la posibilidad de proceder o cancelar. Si el usuario decide continuar, se llama a la función específica del tipo de elemento que se esté creando, como *create_guideline* para una guía, *create_criterion* para un criterio, o *create_metric* para una métrica, según se muestra en la Figura 20.

```
while True:
        if tab == ‘Guideline’: create_guideline()
        elif tab == ‘Criterion’: create_criterion()
        else: create_metric()
```

Figura 20 - Empezar a crear una guía, criterio o métrica

create_guidelines.py

El archivo “create_guidelines.py” define el método *create_guideline*, que gestiona la creación de nuevas guías. Este método solicita al usuario que introduzca valores para los diferentes atributos de la guía, como el título, los autores, la aplicabilidad, la URL y las reglas de clasificación. Estos valores son validados mediante los métodos *check*, *check_combined* y *check_json* importados de “notebook_checks.py”, los cuales aseguran que los datos sean correctos antes de continuar. El método *check_combined* se utiliza en este caso sobre el atributo *authors*, ya que se puede aceptar tanto un autor como una lista de autores. En cambio, el método *check_json* se utiliza a la hora de insertar las reglas de clasificación, las cuales deben estar en formato JSON, como se ha indicado en el apartado de la base de datos.

Una vez que los datos han sido validados, el método genera un identificador único para la nueva guía utilizando el método *uuid4* del paquete “uuid”. Con los datos validados y el identificador generado, se instancia un objeto *Guideline* utilizando los valores proporcionados. Finalmente, la guía creada se inserta en la base de datos mediante una consulta SQL como la de la Figura 21, asegurando que todos los datos estén correctamente almacenados y listos para ser utilizados en evaluaciones futuras.

```
guideline = g.Guideline(id, title, authors, applicability, url, classification_rules, [])
conn. query(“insert”, f“INSERT INTO Guideline VALUES (‘{guideline.id}’,
‘{guideline.title}’, ‘{‘, ‘.join(guideline.authors)}’, ‘{guideline.applicability}’,
‘{guideline.url}’, ‘{guideline.classification_rules}’)”)
```

Figura 21 - Insertar una guía en la base de datos

create_criteria.py

El archivo “create_criteria.py” está diseñado para mejorar la creación de criterios y su vinculación con guías ya existentes. El método *show_guidelines* consulta la base de datos para recuperar todas las guías disponibles, mostrando su identificador y título en una tabla formateada mediante la clase ‘PrettyTable’.

El método *create_criterion*, que gestiona el proceso de creación de un nuevo criterio, comienza solicitando al usuario que proporcione valores para varios atributos del criterio, como el nombre, la descripción, la regla de evaluación (AND u OR), el tipo de criterio (BOOLEAN o SCORE) y la dimensión. Dependiendo del tipo seleccionado, se solicitan atributos adicionales específicos: para criterios booleanos, se solicitan la

fuerza, la dirección y la dimensión; mientras que, para los criterios de puntuación, se solicitan la fuerza sugerida, la fuerza mínima aceptada y la fuerza máxima aceptada. Estos valores son validados mediante el método *check*, importado desde "notebook_checks.py", para asegurar que los datos sean correctos antes de continuar.

Una vez que los datos han sido validados, el método genera un identificador único para el nuevo criterio utilizando el método *uuid4* del paquete "uuid". Con los datos validados y el identificador generado, se instancia un objeto *BooleanCriterion* utilizando los valores proporcionados. A continuación, el criterio creado se inserta en la base de datos mediante una consulta SQL como la de la Figura 22, asegurando que todos los datos estén correctamente almacenados y listos para ser utilizados en evaluaciones futuras.

```
criterion = c.BooleanCriterion(id, name, description, strength, direction, pass_rule, [])
conn.query("insert", f"INSERT INTO Criterion VALUES ('{criterion.id}', '{criterion.name}', '{criterion.description}', '{criterion.pass_rule}', 'boolean', '{criterion.strength}', '{criterion.direction}', 'NULL', 'NULL', 'NULL', '{dimension}')")
```

Figura 22 - Insertar un criterio en la base de datos

Finalmente, se muestran las guías disponibles y se solicita al usuario que introduzca el identificador de la guía con la que se debe asociar el criterio, insertando esta relación en la base de datos con la consulta de la Figura 23.

```
conn.query("insert", f"INSERT INTO Guideline_Criterion VALUES ('{guide_id}', '{criterion.id}'")
```

Figura 23 - Insertar la relación entre un criterio y una guía en la base de datos

create_metrics.py

El archivo "create_metrics.py" se encarga de la creación de métricas y su asociación con criterios ya existentes. El método *show_criteria* consulta la base de datos y muestra todos los criterios disponibles, listando su identificador y descripción en una tabla formateada mediante la clase 'PrettyTable'.

El método *create_metric*, que gestiona el proceso de creación de una nueva métrica, comienza solicitando al usuario que proporcione valores para los atributos de la nueva métrica, como el nombre, la descripción, el patrón de columnas y la regla de evaluación de datos. Estos valores son validados mediante los métodos *check*,

check_combined y *check_value*, importados de “notebook_checks.py”, los cuales aseguran que los datos sean correctos antes de continuar. El método *check_combined* se utiliza en este caso sobre el atributo *columns_pattern*, ya que se puede aceptar tanto una columna como una lista de columnas sobre las que se debe evaluar. También se utilizar para admitir estar asociada con varios criterios a la vez.

Una vez que los datos han sido validados, el método genera un identificador único para la nueva métrica utilizando el método *uuid4* del paquete “uuid”. Una vez los datos están validados y el identificador generado, se instancia un objeto *Metric* utilizando los valores proporcionados. A continuación, la métrica creada se inserta en la base de datos mediante una consulta SQL como la de la Figura 24, asegurando que todos los datos estén correctamente almacenados y listos para ser utilizadas en evaluaciones futuras.

```
metric = m.Metric(id, name, description, columns_pattern, data_evaluation_rule, data_evaluation_value, min_percentage_data_fulfillment, [])
data_evaluation_condition = '[{\"' + data_evaluation_rule.upper() + '\":\"' + data_evaluation_vaule + '\"}]'
conn.query("insert", f"INSERT INTO Metric VALUES ('{metric.id}', '{metric.name}', '{metric.description}', '{metric.min_percentage_data_fulfillment}', '{data_evaluation_condition}', '{}'")
```

Figura 24 - Insertar una métrica en la base de datos

A la hora de insertar la condición de evaluación en la base de datos, como se ha explicado, se debe introducir una operación y un valor en formato JSON. En este caso, se ha pedido por pantalla al usuario que introduzcan los valores por separados, porque a la hora de instanciar el objeto se introducen en dos atributos diferentes, para luego unirse consiguiendo el formato JSON requerido.

Posteriormente, se muestran los criterios disponibles y se solicita al usuario que introduzca el identificador del criterio o de los criterios con los que la métrica debe estar asociada, insertando estas relaciones en la base de datos como en la Figura 25.

```
for crit_id in crits_id:
        conn.query("insert", f"INSERT INTO Criterion_Metric VALUES ('{crit_id}', '{metric.id}'")
```

Figura 25 - Insertar la relación entre una métrica y un criterio en la base de datos

3.5.4. Ejemplo de uso

Para comenzar con todo el proceso, el usuario debe abrir y ejecutar el notebook proporcionado. Para abrir el notebook, primero se debe tener instalado Jupyter Notebook y algún entorno compatible como VS Code, JupyterLab u otro similar. Otra opción para iniciar el notebook, una vez instalado, es ejecutar *jupyter notebook* directamente en la terminal.

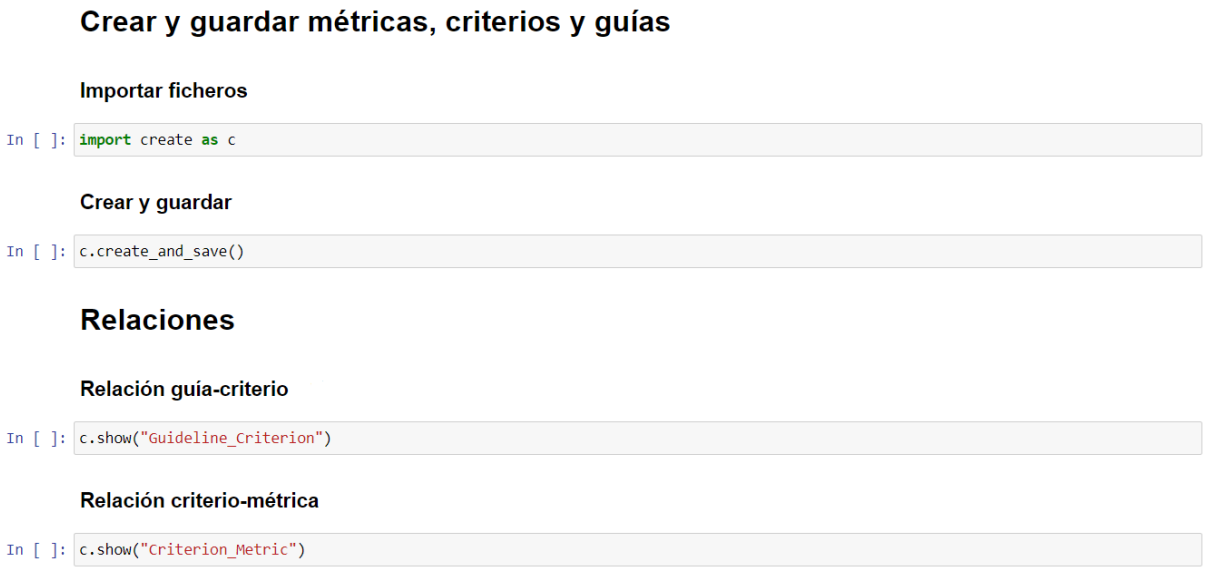

Figura 26 - Inicio del notebook

Una vez abierto el notebook, el usuario se encuentra con una pantalla como la que se muestra en la Figura 26, con una serie de celdas que contienen texto y código. Las celdas con texto funcionan como instrucciones que guían al usuario en el proceso de crear y guardar nuevas guías, criterios y métricas.

Para ejecutar una celda de código, se debe seleccionar la celda y hacer clic en el botón “Run” de la barra de herramientas, o pulsar las teclas “Shift” y “Enter”. Es importante ejecutar las celdas en el orden en que aparecen, comenzando desde la primera, para asegurarse de que todas las variables y funciones necesarias estén disponibles.

El primer paso para realizar el ejemplo de uso sería ejecutar la primera celda, con la que se importa el archivo “create.py”, explicado en el apartado de la descripción del notebook, para poder utilizar sus métodos *create_and_save*, para insertar guías, criterios o métricas en la base de datos, y *show*, para ver la relación entre las guías y los criterios, y entre los criterios y las métricas.

Una vez importado el archivo necesario, ya se puede empezar con el procedimiento de crear, ejecutando la celda con la función *c.create_and_save*. Esto abrirá un diálogo en el que se pregunta al usuario qué tipo de elemento desea crear: una guía, un criterio o una métrica.

Como en este ejemplo se parte desde cero, donde no hay nada insertado aún, se añadirá primero una guía. En este caso, se creará una guía de ejemplo para la que se insertarán valores inventados. El proceso para crear la guía se ilustra en la Figura 27.

```
c.create_and_save()
¿Qué quieres crear? (GUIDELINE/CRITERION/METRIC): GUIDELINE

Tabla original:
+----+-------+--------+---------------+-----+-------+
| id | title | author | applicability | url | rules |
+----+-------+--------+---------------+-----+-------+
+----+-------+--------+---------------+-----+-------+

¿Desea crear una guía? (s/n): s
Title: Guide_example
Authors: Laia
Applicability: -
URL: -
Classification_rules (introdúcelas en formato JSON): {"classification": "pattern"}
```

Figura 27 - Creación de una guía

Primero, se especifica que se quiere crear una guía y se muestran las guías disponibles, por si la que se quiere utilizar está ya creada o no. Si finalmente se decide crearla, se insertan los atributos correspondientes para las guías. Por último, cuando se terminan de insertar todas las guías necesarias, se muestra el resultado de la tabla 'Guideline' de la base de datos al realizar dicha actualización.

```
c.create_and_save()
¿Qué quieres crear? (GUIDELINE/CRITERION/METRIC): CRITERION

Tabla original:
+----+------+-------------+-----------+------+----------+-----------+--------------------+----------------------+----------------------+-----------+
| id | name | description | pass_rule | type | strength | direction | suggested_strength | min_accepted_strength | max_accepted_strength | dimension |
+----+------+-------------+-----------+------+----------+-----------+--------------------+----------------------+----------------------+-----------+
+----+------+-------------+-----------+------+----------+-----------+--------------------+----------------------+----------------------+-----------+
¿Desea crear un criterio? (s/n): s
Name: Crit1_example
Description: ...
Pass_rule (AND / OR): AND
Type (BOOLEAN / SCORE): BOOLEAN
Strength: strong
Direction (PATHOGENIC / BENIGN): PATHOGENIC
Dimension (POPULATION FREQUENCY / VARIANT TYPE AND LOCATION / CASE-LEVEL INFORMATION / FUNCTIONAL AND COMPUTATIONAL INFORMATION
/ REPUTABLE SOURCE): REPUTABLE SOURCE
Guías disponibles:
+---------------------------------------+---------------+
|                  id                   |     title     |
+---------------------------------------+---------------+
| guide_03c4a71e-8151-426d-a380-0243f4e13f9e | Guide_example |
+---------------------------------------+---------------+
Guideline_id: guide_03c4a71e-8151-426d-a380-0243f4e13f9e
```

Figura 28 - Creación de un criterio y asociación a la guía

Después de crear la guía, el usuario procedería a crear los criterios que la componen. En este caso, se va a añadir un criterio y se asociará a la guía creada en el paso anterior. Esto se puede apreciar en la Figura 28.

Primero, se especifica que se quiere crear un criterio y se muestran los criterios disponibles, por si el que se quiere utilizar está ya creado o no. Si finalmente se decide crearlo, se insertan los atributos correspondientes para los criterios, dependiendo del tipo de criterio que se haya especificado. Cuando se terminan de insertar todos los criterios necesarios, se muestra el resultado de la tabla 'Criterion' de la base de datos al realizar dicha actualización y se pide al usuario que introduzca la guía a la que se debe asignar. Por último, se muestra la tabla 'Guideline_Criterion', donde se podrá observar a esta relación.

Finalmente, el usuario creará las métricas necesarias para poder evaluar cada criterio introducido anteriormente, y las asociará a dicho criterio, como se puede observar en la Figura 29.

```
c.create_and_save()
¿Qué quieres crear? (GUIDELINE/CRITERION/METRIC): METRIC

Tabla original:
+----+------+-------------+---------------------------------+---------------------------+-----------------+
| id | name | description | min_percentage_data_fulfillment | data_evaluation_condition | columns_pattern |
+----+------+-------------+---------------------------------+---------------------------+-----------------+
+----+------+-------------+---------------------------------+---------------------------+-----------------+

¿Desea crear una métrica? (s/n): s
Name: Met1_example
Description: ...
Columns_pattern: variant
Data_evaluation_rule (MINOR / MAJOR / EQUALS / DISTINCT / IN_LIST / NOT_IN_LIST / IN_RANGE / NOT_IN_RANGE): MINOR
Data_evaluation_value (MINOR número): 3.1
Min_percentage_data_fulfillment (escribe el porcentaje que se deben cumplir): 25
Criterios disponibles:
+-----------------------------------------+-------------+
|                    id                   | description |
+-----------------------------------------+-------------+
| crit_fd556fec-f829-4359-84c6-f60df11e7386 |     ...     |
+-----------------------------------------+-------------+
Criteria_id (si hay varios, sepáralos por comas): crit_fd556fec-f829-4359-84c6-f60df11e7386
```

Figura 29 - Creación de una métrica y asociación al criterio

Primero, se especifica que se quiere crear una métrica y se muestran las métricas disponibles, por si la que se quiere utilizar está ya creada o no. Si finalmente se decide crearla, se insertan los atributos correspondientes para las métricas. Cuando se terminan de insertar todas las métricas necesarias, se muestra el resultado de la tabla 'Metric' de la base de datos al realizar dicha actualización y se pide al usuario que introduzca el criterio o los criterios a los que se debe asignar. Por último, se muestra la tabla 'Criterion_Metric', donde se podrá observar a esta relación.

Para comprobar que los criterios se han asignado a las guías correspondientes y las métricas a los criterios, se pueden ejecutar los métodos *c.show* para visualizar estas relaciones tras insertar los elementos en la base de datos (Figura 30).

Relación guía-criterio

```
c.show("Guideline_Criterion")
```

```
+----------------------------------------+----------------------------------------+
|               Guidelineid              |               Criterionid              |
+----------------------------------------+----------------------------------------+
| guide_03c4a71e-8151-426d-a380-0243f4e13f9e | crit_fd556fec-f829-4359-84c6-f60df11e7386 |
+----------------------------------------+----------------------------------------+
```

Relación criterio-métrica

```
c.show("Criterion_Metric")
```

```
+----------------------------------------+----------------------------------------+
|               Criterion_id             |               Metric_id                |
+----------------------------------------+----------------------------------------+
| crit_fd556fec-f829-4359-84c6-f60df11e7386 | met_d86d0255-7716-485f-97ad-1db7d21b4262 |
+----------------------------------------+----------------------------------------+
```

Figura 30 - Relaciones guía-criterio y criterio-métrica

Se puede comprobar en las Figuras 31, 32 y 33 que, al ejecutar una consulta directamente en MySQL Workbench, se obtienen las guías, los criterios y las métricas insertadas mediante el método *c.create_and_save* del notebook. En la Figura 34 también se puede observar que las relaciones también se han insertado correctamente.

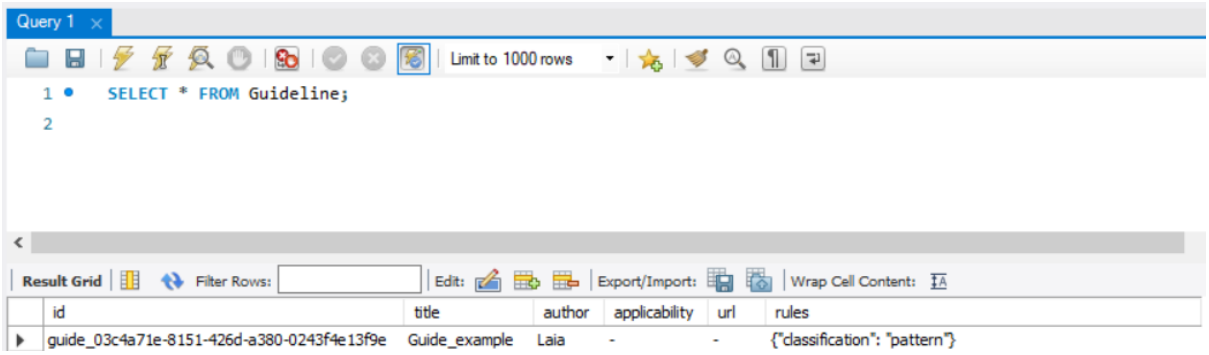

Figura 31 - Consulta de la tabla 'Guideline'

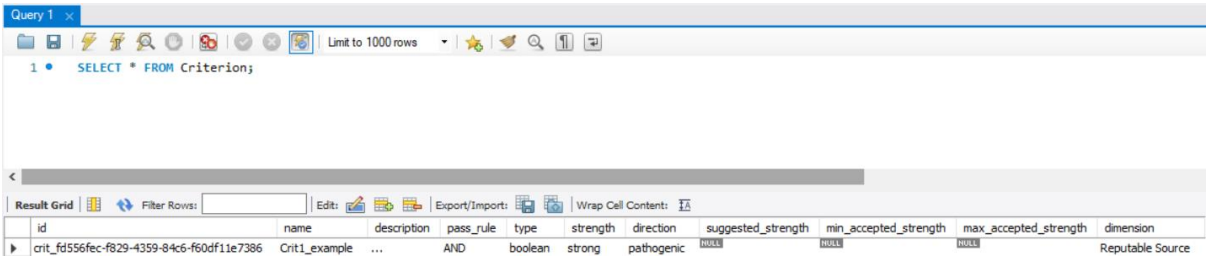

Figura 32 - Consulta de la tabla 'Criterion'

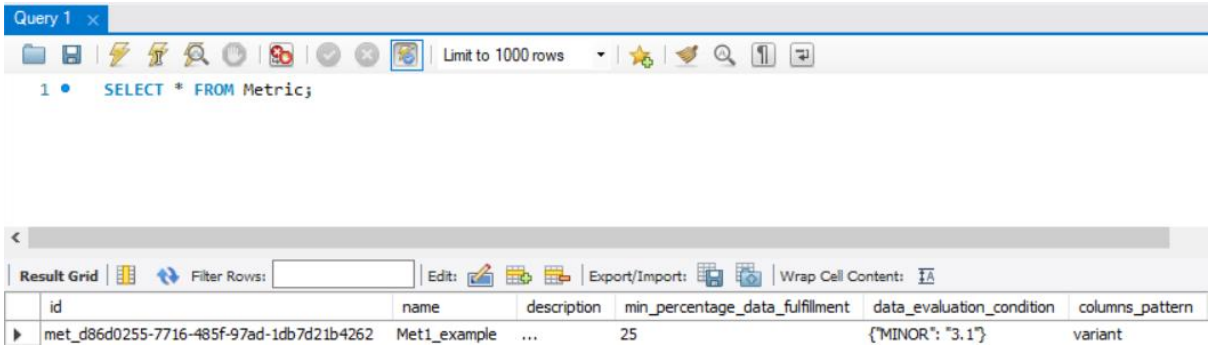

Figura 33 - Consulta de la tabla 'Metric'

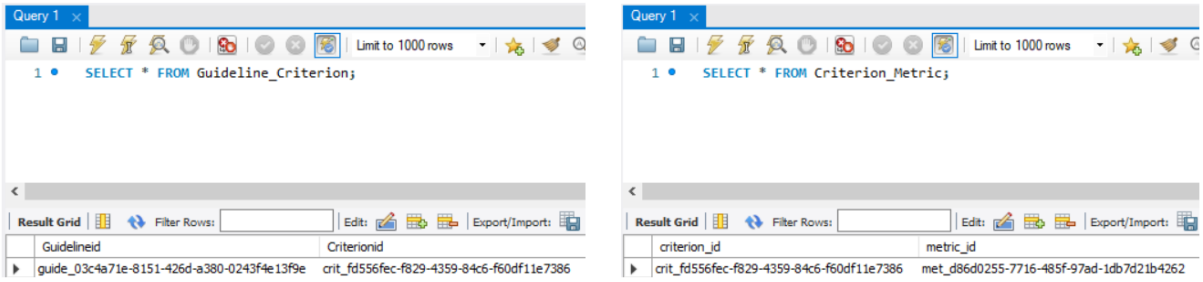

Figura 34 - Consulta de las tablas 'Guideline_Criterion' y 'Criterion_Metric'

4. Validación de la solución

Una vez completada la investigación y el diseño de la solución, la siguiente fase es la validación de la solución, que constituye la última etapa del ciclo de diseño de la metodología Design Science. Esta fase se centrará en la implementación práctica de los componentes esenciales del framework, utilizando como caso de uso a pacientes reales con cáncer hereditario de Chile.

Estos datos son fundamentales para probar la eficacia y aplicabilidad del framework en un contexto clínico real, permitiendo evaluar cómo la solución desarrollada puede contribuir a la clasificación de variantes genéticas.

4.1. ACMG/AMP con VCFs de test

Este caso de prueba tiene como objetivo principal evaluar la efectividad y precisión del framework desarrollado para la clasificación de variantes genéticas, utilizando como referencia las guías ACMG/AMP. Para ello, se utilizarán archivos VCF correspondientes a cinco pacientes reales con cáncer hereditario de Chile.

Cada archivo VCF contiene información detallada sobre las variantes genéticas detectadas en los pacientes. La validación se enfocará en verificar la capacidad del framework para procesar y clasificar correctamente estas variantes genéticas, de acuerdo con los criterios y métricas establecidos por las guías ACMG/AMP.

Además, se realizará una comparación exhaustiva entre los resultados generados por el framework y los informes emitidos por expertos en el campo. Esta comparación permitirá determinar la concordancia entre framework y la evaluación experta, proporcionando una medida adicional de precisión y confiabilidad de la herramienta en un entorno clínico real.

4.1.1. Preparación del entorno

Antes de ejecutar el caso de prueba, se ha preparado el entorno conforme se va a comentar. Se han recibido cinco DataFrames, cada uno correspondiente a un paciente real de cáncer hereditario de Chile, proporcionados por el grupo PROS. Estos DataFrames resultan de la anotación de los cinco archivos VCF individuales con datos provenientes de bases de datos, que incluyen la información necesaria para

evaluar las guías ACMG/AMP, así como los criterios y métricas necesarios para la clasificación.

Este entorno ha sido configurado específicamente para garantizar que el framework pueda recibir y procesar adecuadamente los datos de entrada, facilitando así una evaluación precisa de la clasificación de variantes genéticas en el contexto clínico real de los pacientes.

4.1.2. Ejecución del caso de prueba

Con el entorno preparado y configurado, se procede a la ejecución del caso de prueba, donde se procesan y evalúan las variantes genéticas utilizando el framework diseñado y desarrollado para clasificarlas según los criterios establecidos.

Primero, se carga en el framework el DataFrame que contiene la información de los cinco VCF junto con los datos de la base de datos. Durante este proceso, se extraen las variantes genéticas se preparan para su posterior evaluación. Esta preparación garantiza que la información esté lista para ser analizada bajo los estándares de la guía ACMG/AMP.

Una vez que las variantes han sido preparadas, el framework aplica los criterios y las métricas que se han definido previamente en la base de datos. Estos criterios incluyen tanto aspectos patogénicos como benignos, los cuales se evalúan para determinar si una variante cumple o no con las condiciones especificadas por la guía. Este paso es crucial, ya que determina la dirección en la que se clasificará cada variante.

Tras la evaluación, el framework clasifica automáticamente las variantes en una de las siguientes categorías: patogénicas, probablemente patogénicas, probablemente benignas, benignas o de significado incierto (VUS). Esta clasificación se basa en el número de criterios cumplidos y en la dirección de estos (patogénicos o benignos). El proceso garantiza que la clasificación sea precisa y conforme a los estándares establecidos.

Finalmente, los resultados de la clasificación se almacenan en un DataFrame. Este registro proporciona un resumen claro y estructurado de cómo cada variante ha sido clasificada, lo que facilita la revisión y el análisis posterior.

4.1.3. Datos utilizados

Como se ha comentado, en este caso de prueba se ha utilizado la guía ACMG/AMP, explicada en el apartado de investigación del problema. Para asegurar una evaluación precisa y alineada con las mejores prácticas, se ha consultado expertos en el área para definir la instanciación de estas guías. Concretamente, los criterios y métricas específicos utilizados en el framework son los que se pueden ver en la Tabla 1.

Criterio	Métrica	Descripción de la métrica
PM2	not_in_pop_db	La variante está ausente en todas las bases de datos de población disponibles. Los umbrales siguen las recomendaciones originales de ACMG/AMP.
PM4.2	stop_loss_variant	La variante es stop_loss.
BP6	variant_clinical_actionability_is_benign	La variante tiene una acción clínica benigna o probablemente benigna.
	variant_is_benign	La variante es benigna o probablemente benigna.
PM1.1	variant_in_hotspot	La variante se encuentra en un hotspot.
BA1	freq_greater_than_0.05	La frecuencia alélica de la variante es mayor a 0.05.
BP7	variant_is_synonymous	La variante es sinónima.
	variant_not_conserved	La variante no está conservada durante la evolución.
	variant_no_affects_splicing	La variante no afecta el splicing.
PM4.1	not_in_repeat_region	La variante no se encuentra en una región repetitiva.
	in_frame_variant	La variante es una inserción o deleción.
PM1.2	variant_in_domain	La variante se encuentra en un dominio funcional.
BS2	observed_in_controls	La variante ha sido observada en al menos un individuo adulto sano.
BP1	lof_mechanism_gene_loeuf	El gen tiene un mecanismo de enfermedad por pérdida de función según el puntaje LOEUF de gnomAD. Se utilizan los umbrales recomendados.
	missense_variant	La variante es missense.
PS4	relevant_or_value	El estudio estadístico reporta un odd ratio estadísticamente significativo.
	relevant_p_value	El estudio estadístico reporta un p value estadísticamente significativo.
PP5	variant_is_pathogenic	La variante es patogénica o probablemente patogénica.

	variant_clinical_actionability_is_pathogenic	La variante tiene una acción clínica patogénica o probablemente patogénica.
BP4	variant_is_predicted_tolerated	La variante es predicha como tolerada por al menos el 75% de los predictores considerados.
BS1	freq_greater_than_0.01	La frecuencia alélica de la variante es mayor al 1%
PP3	variant_is_predicted_deleterious	La variante es predicha como deletérea por al menos el 75% de los predictores.
PVS1	lof_nmd_variant	La variante es predicha como LOF o NMD por SnpEff.
	null_variant	La variante es nula, lo que incluye nonsense, frameshift, canonincal splice site, single o multiexon deletion.
BP3	in_repeat_region	La variante se encuentra en una región repetitiva.
	in_frame_variant	La variante es una inserción o deleción.

Tabla 1 - Criterios y métricas utilizados

4.1.4. Resultados

Una vez finalizada la ejecución del framework, se procede a la recopilación y análisis de los resultados generados. Los resultados obtenidos incluyen la clasificación detallada de las variantes genéticas, categorizadas según los criterios preestablecidos. Además, se realiza un análisis comparativo entre las clasificaciones automáticas y las evaluaciones realizadas por expertos en genética, lo que permite identificar posibles discrepancias y fortalezas del framework en un entorno clínico real. La discusión de sensibilidad y especificidad para cada paciente también proporciona una visión más profunda sobre la eficacia del framework.

Analizados los pacientes, se presentará una tabla como resumen, que facilitará la comparación de los resultados de clasificación de variantes para cada paciente.

Primer paciente (F340027697_21657):

La variante *c.8987T>A* fue clasificada como patogénica tanto en la clasificación manual como en la automática. Esta concordancia resalta la capacidad del algoritmo para identificar variantes de alta relevancia clínica.

Sin embargo, las variantes *c.1833C>G* y *c.2996delC*, clasificadas como patogénicas en la clasificación automática, se identificaron como falsos positivos en

la clasificación manual. Específicamente, la variante *c.1833C>G* se encuentra en una región no relacionada con el cáncer hereditario, lo que invalida su clasificación como probablemente patogénica en este contexto. En cuanto a la variante *c.2996delC*, aunque cumple con los criterios de ser probablemente patogénica, ha sido interpretada como benigna en ClinVar para otras enfermedades. Además, se ha observado en varios pacientes, lo que sugiere que no tiene un vínculo directo con la patología que se está evaluando.

Por último, la variante *c.1278+1delG* se clasificó como una variante de significado incierto (VUS) en la clasificación automática, lo que resalta la diferencia en la interpretación de las variantes más ambiguas.

En cuanto a los criterios utilizados en la variante *c.8987T>A*, ambos métodos coincidieron en los criterios PM2 y PVS1, demostrando nuevamente una alineación de variantes de alta relevancia. No obstante, se observó una discrepancia a la hora de aplicar los criterios para la clasificación, ya que la clasificación manual aplicó el criterio PS4, mientras que en la clasificación automático se utilizó el criterio PP3. Además, la clasificación manual incluyó los criterios PM5, PS1, PP2 y PP4, que no están automatizados porque requieren el uso de información clínica específica de los pacientes, que no se puede integrar en el proceso automático de clasificación.

Por último, en la clasificación manual, el criterio PP5 no se aplicó con la fuerza establecida en las guías ACMG/AMP, lo que implica una diferencia en la interpretación de la evidencia entre ambas evaluaciones.

Aunque el algoritmo demostró una coincidencia del 100% en algunas variantes clave, su sensibilidad y especificidad variarían según la capacidad para detectar variantes patogénicas y diferenciar entre benignas y patogénicas. La identificación de falsos positivos como en las variantes *c.1833C>G* y *c.2996delC* subraya un área donde la especificidad podría mejorarse. Estos aspectos pueden ser relevantes al evaluar la precisión del algoritmo para su uso clínico.

Segundo paciente (F340027697_21808):

La variante *c.3260G>A* fue clasificada como patogénica en la clasificación automática, pero fue identificada como falso positivo en la clasificación manual. La razón principal es que esta variante está relacionada con la contracción muscular, lo cual es relevante en otro contexto clínico, pero no en el cáncer hereditario.

En cuanto a la variante *c.85C>T*, fue clasificada como patogénica tanto en la clasificación manual como en la automática, lo que indica una coincidencia en su interpretación y valida la capacidad del algoritmo para identificar variantes de alta relevancia clínica.

En el caso de la variante *c.2996delC*, que también apareció en el primer paciente, se clasificó como patogénica en la clasificación automática. Sin embargo, aunque cumple con los criterios de patogenicidad, ha sido interpretada como benigna en otras condiciones clínicas según datos de ClinVar. Esto sugiere que podría no estar directamente relacionada con la patología que se está evaluando en este contexto.

Finalmente, la variante *c.1278+1delG* se clasificó como VUS en la clasificación automática, lo que resalta la dificultad de interpretación de esta variante en ambos casos.

En cuanto a los criterios aplicados para la variante *c.85C>T*, ambos métodos coincidieron en el criterio PVS1. Sin embargo, hubo discrepancias aplicando criterios, puesto que en la clasificación manual se optó por aplicar el criterio PM2, pero en la clasificación automática se optó por el criterio PM1. Además, los criterios PS3 y PP4 se utilizaron en la clasificación manual, pero no pudieron ser automatizados porque requieren el uso de información clínica específica de los pacientes que no se puede integrar en el proceso automático de clasificación.

Al igual que en el primer paciente, se puede observar una coincidencia del 100% en algunas variantes clave, como la *c.85C>T*, lo que valida la sensibilidad del algoritmo en ciertos casos. Sin embargo, la identificación de falsos positivos, como la variante *c.3250G>A*, pone de relieve la necesidad de mejorar la especificidad del algoritmo para evitar clasificaciones incorrectas en contextos clínicos no relacionados directamente con la patología evaluada.

Tercer paciente (F340027697_21809):

La variante *c.995C>T* fue inicialmente clasificada como patogénica en la clasificación automática, pero se identificó como un falso positivo en la clasificación manual. Esto se debe a que, aunque la variante es relevante para la hiperplasia suprarrenal congénita, no tiene relación directa con el cáncer hereditario, lo que la excluye del contexto patológico evaluado.

La variante *c.1039G>T* fue clasificada como patogénica tanto en la clasificación manual como en la automática, mostrando una coincidencia total en su interpretación y confirmando la fiabilidad del algoritmo en este caso.

En cuanto a la variante *c.3521C>T*, fue clasificada como patogénica en la evaluación automática, pero la clasificación manual la consideró un falso positivo. Esto es porque la interpretación manual sugiere que la variante podría ser tanto probablemente benigna, mostrando una ambigüedad en su clasificación y subrayando las limitaciones del algoritmo en algunos casos.

En este paciente también se puede ver la variante *c.2996delC*, que, como en los otros, se clasificó como patogénica en la clasificación automática, pero ha sido interpretada como benigna en otras condiciones clínicas según datos de ClinVar.

Finalmente, se tiene también la variante *c.1278+1delG* clasificada como VUS en la clasificación automática, lo que sigue siendo un patrón consistente con los otros pacientes.

En cuanto a los criterios aplicados para la variante *c.1039G>T*, ambos métodos coincidieron en los criterios PVS1 y PP5. No obstante, mientras que la clasificación manual aplicó el criterio PM2, en la clasificación automática se aplicó el criterio PP3.

Como en los pacientes anteriores, se observa una coincidencia del 100% en variantes clave como las *c.1039G>T*, lo que destaca la capacidad del algoritmo para identificar variantes patogénicas relevantes. No obstante, la identificación de falsos positivos, como las variantes *c.9955C>T* y *c.3521C>T*, subraya la importancia de mejorar la especificidad del algoritmo, ya que algunas variantes relevantes en otros contextos clínicos fueron clasificadas incorrectamente.

Cuarto paciente (F340027697_21778):

La variante *c.328G>T*, clasificada como probablemente patogénica en la clasificación automática, se identificó como falso positivo en la clasificación manual. Esta discrepancia se debe a que la variante no está directamente relacionada con el cáncer hereditario, lo que invalida su clasificación como patogénica en este contexto específico.

En este paciente también se clasificó la variante *c.2996delC*, que ya se ha mencionado en los otros pacientes, y que fue clasificada como patogénica en la clasificación automática, pero ha sido interpretada como benigna en otras condiciones

clínicas según los datos de ClinVar. Esto sugiere que la variante podría no estar vinculada directamente con la patología evaluada en este paciente.

Por último, la variante *c.1A>G* fue clasificada como VUS tanto en la clasificación manual como en la automática, lo que demuestra una coincidencia en su interpretación. Esta concordancia es principalmente porque en la clasificación automática no se activó el criterio PM2, lo que permitió alinearse con la evaluación manual.

En cuanto a los criterios utilizados en la variante *c.1A>G*, ambos métodos coincidieron en el criterio PVS1, aunque hubo discrepancias en la aplicación de algunos criterios, ya que en la clasificación manual se escogió aplicar los criterios PM2 y PP5, pero en la clasificación automática se aplicó el criterio PP3. Además, la clasificación manual incluyó los criterios PS3, PM6, PS2 y PM3, que no están automatizados porque requieren el uso de información clínica específica de los pacientes que no se puede integrar en el proceso automático de clasificación.

En este paciente, la coincidencia en la clasificación de la variante *c.1A>G* como VUS destaca la consistencia al 100% del algoritmo en algunas interpretaciones. No obstante, la identificación de la variante *c.238G>T* como un falso positivo resalta la necesidad de mejorar la especificidad del algoritmo, especialmente en la clasificación de variantes que no tienen relevancia directa en el cáncer hereditario.

Quinto paciente (F340027697_21787):

La clasificación de las variantes en este paciente se consideraría un fallo, ya que no se encontró ninguna variante cuya clasificación coincidiera con la clasificación manual realizada por los expertos.

La variante *c.178C>T* fue clasificada como patogénica en la clasificación automática, pero fue considerada como falso positivo en la clasificación manual. Esto es debido a que esta variante es relevante para la ictiosis cognitiva autosómica recesiva, pero no para el cáncer hereditario, lo que invalida su relevancia en este contexto clínico.

Las variantes *c.43A>T*, *c.2996delC*, *c.802-3dupT*, *c.1236-3dupT* y *c.331C>T*, clasificadas como probablemente patogénicas en la clasificación automática, también se identificaron como falsos positivos en la clasificación manual. Específicamente, las variantes *c.43A>T* y *c.331C>T* se encuentran en variantes no relacionadas con el

cáncer hereditario, lo que hace incorrecta su clasificación como probablemente patogénica en este contexto. Por otra parte, las variantes *c.802-3dupT*, *c.1236-3dupT* y *c.2996delC*, la última ya mencionada en los otros pacientes, cumplen con los criterios de ser probablemente patogénicas, pero han sido interpretadas como benignas en ClinVar.

Por último, la variante *c.638G>A* fue clasificada como VUS en la clasificación automática, lo que resalta la dificultad de interpretación de esta variante en ambos casos.

En cuanto a los criterios aplicados a la variante *c.638G>A*, ambos métodos coincidieron en el criterio PP3. No obstante, hubo discrepancias al aplicar el criterio PM2 en la clasificación manual y no activarlo en la clasificación automática. Además, la clasificación manual incluyó también los criterios PM5 y PS3, que no están automatizados porque requieren el uso de información clínica específica de los pacientes que no se puede integrar en el proceso automático de clasificación. Por último, en la clasificación manual, los criterios PM1 y PP5 no se aplicaron con la fuerza establecida en las guías ACMG/AMP, lo que implica una diferencia en la interpretación de la evidencia entre ambas evaluaciones.

El caso de este paciente pone en evidencia limitaciones importantes en el algoritmo, ya que no logró coincidir con la clasificación manual en ninguna variante. Esto resalta la necesidad de mejorar la sensibilidad y la especificidad del algoritmo, dado que tanto las variantes patogénicas como las probablemente patogénicas fueron consideradas falsos positivos, subrayando que el algoritmo no pudo discriminar adecuadamente las variantes no relacionadas con el cáncer hereditario, lo que compromete su eficacia en este contexto.

Resumen de resultados

En la Tabla 2 se presenta un resumen de los resultados de la clasificación de las variantes para los cinco pacientes ya descritos. La tabla destaca en color verde las variantes que coinciden en la clasificación automática y manual, y en color amarillo aquellas se consideran falsos positivos. En las observaciones se indican los criterios utilizados para la clasificación de las variantes que concordaron en ambas clasificaciones, o se ofrece una breve explicación sobre por qué se ha considerado falsos positivos.

Paciente	Variante	Clasificación	Observaciones
1	c.8987T>A	Patogénica	Concordancia en PM2 y PVS1. Diferencias en PS4 (manual) y PP3 (automática). Manual incluye PM5, PS1, PP2 y PP4. No se aplicó PP5 con la misma fuerza en ambas clasificaciones.
	c.1833C>G	Falso positivo	No relacionada con cáncer hereditario.
	c.2996delC	Falso positivo	Clasificada como benigna en ClinVar para otras enfermedades.
	c.1278+1delG	VUS	
2	c.3260G>A	Falso positivo	No relacionada con cáncer hereditario.
	c.85C>T	Probablemente patogénica	Concordancia en PVS1. Diferencias en PM2 y PS4 (manual) y PM1 (automática). Manual incluye PS3 y PP4.
	c.2996delC	Falso positivo	Clasificada como benigna en ClinVar para otras enfermedades.
	c.1278+1delG	VUS	
3	c.955C>T	Falso positivo	No relacionada con cáncer hereditario.
	c.1039G>T	Patogénica	Concordancia en PVS1 y PP5. Diferencias en PM2 (manual) y PP3 (automática).
	c.3421C>T	Falso positivo	Cuadra la interpretación de probablemente patogénica y probablemente benigna.
	c.2996delC	Falso positivo	Clasificada como benigna en ClinVar para otras enfermedades.
	c.1278+1delG	VUS	
4	c.328G>T	Falso positivo	No relacionada con cáncer hereditario.
	c.2996delC	Falso positivo	Clasificada como benigna en ClinVar para otras enfermedades.
	c.1A>G	VUS	Concordancia en PVS1. Diferencias en PM2 y PP5 (manual) y PP3 (automática). Manual incluye PS3, PM6, PS2 y PM3.
5	c.178C>T	Falso positivo	No relacionada con cáncer hereditario.
	c.43A>T	Falso positivo	
	c.2996delC	Falso positivo	Clasificada como benigna en ClinVar para otras enfermedades.
	c.802-3dupT	Falso positivo	
	c.1236-3dupT	Falso positivo	
	c.331C>T	Falso positivo	No relacionada con cáncer hereditario.

	c.638G>A	VUS	Concordancia en PP3. Diferencias en PM2 (manual, en automática no se activó). Manual incluye PM5 y PS4. No se aplicaron PM1 y PP5 con la misma fuerza en ambas clasificaciones.

Tabla 2 - Resumen de resultados por paciente

4.1.5. Precisión de los resultados

El análisis de los resultados obtenidos al aplicar el algoritmo de clasificación de variantes genéticas muestra que, de un total de 23 variantes analizadas en los cinco pacientes, 9 coincidieron en su clasificación entre la evaluación automática y la manual, lo que representa una precisión de poco más del 39%. Estas coincidencias incluyen variantes relevantes clínicamente, con una correcta identificación de las variantes patogénicas en todos los casos excepto en el quinto paciente, donde la divergencia entre los expertos dificultó la clasificación final.

En los casos en que el algoritmo coincidió con la clasificación manual, como en las variantes *c.8987T>A*, *c.85C>T* y *c.1039G>T*, el algoritmo demostró su capacidad para identificar correctamente variantes de alta relevancia clínica en contextos específicos. Esto subraya su efectividad en ciertos escenarios y resalta su valor como una herramienta de apoyo en la identificación de variantes patogénicas.

Sin embargo, en el casi 61% de los casos se observaron discrepancias entre la clasificación automática y la clasificación manual, aunque muchas variantes se repitieron entre diferentes pacientes. En estas discrepancias, el algoritmo clasificó algunas variantes como patogénicas o probablemente patogénicas que fueron identificadas como falsos positivos en la clasificación manual. Aun así, estas diferencias no restan valor al algoritmo, sino que destacan áreas donde se puede mejorar, particularmente en la especificidad y en la adaptación a contextos clínicos más complejos.

A pesar a las discrepancias observadas, el algoritmo muestra un potencial significativo, especialmente como herramienta preliminar de clasificación variantes genéticas. Con ajustes adicionales en trabajos futuros, como la integración de criterios clínicos más detallados y la mejora en la identificación de variantes específicas de

determinadas patologías, es probable que la precisión y efectividad del algoritmo se incrementen, fortaleciendo su utilidad en entornos clínicos reales.

5. Conclusiones

Este Trabajo de Fin de Grado ha logrado cumplir los objetivos planteados al inicio del trabajo, confirmando la efectividad del framework diseñado y desarrollado para automatizar la clasificación de variantes genéticas.

En primer lugar, se llevó a cabo una investigación profunda sobre la clasificación de variantes genéticas, explorando la problemática actual y las metodologías existentes, incluyendo enfoques de aprendizaje automático y guías ACMG/AMP. Esta revisión permitió, junto con un metamodelo conceptual, guiar la implementación técnica del framework.

En segundo lugar, se diseñó y desarrolló un algoritmo de clasificación de variantes basado en un modelo conceptual, utilizando Python y Jupyter Notebook. Este algoritmo demostró ser flexible y eficaz, facilitando una evaluación uniforme y objetiva de las variantes genéticas. Además, se integró con una base de datos relacional para gestionar la información necesaria y se desarrolló un prototipo inicial de interfaz gráfica en Jupyter Notebook.

Por último, el framework fue instanciado y validado mediante un caso de uso de prueba basado en las guías ACMG/AMP, lo que permitió confirmar su funcionalidad y adecuación en un contexto real de clasificación de variantes genéticas.

El proceso de desarrollo de este proyecto permitió aplicar y consolidar conocimientos adquiridos durante la carrera, integrando disciplinas como la bioinformática, la programación y el diseño de bases de datos. El uso de tecnologías como Python, junto con herramientas específicas para el manejo de datos genéticos, ha sido crucial para el éxito del proyecto. Este trabajo no solo ha fortalecido las competencias técnicas, sino que también ha subrayado la importancia de la automatización en áreas críticas como la medicina de precisión.

A pesar de que el framework ha sido validado satisfactoriamente, es importante reconocer algunas limitaciones que afectan su precisión y aplicabilidad en entornos clínicos más amplios. Una de las principales es la alta tasa de falsos positivos en la clasificación de variantes. Esto se debe en parte a la dificultad de automatizar criterios

clínicos subjetivos y a la falta de incorporación completa de los criterios ACMG/AMP, lo que limita la exhaustividad de sus clasificaciones.

Además, como trabajo futuro, sería relevante desarrollar una interfaz de usuario más accesible y funcional. En este proyecto, solo se ha implementado un prototipo preliminar a través de Jupyter Notebook, lo que limita su usabilidad a usuarios con conocimientos técnicos. Una interfaz más amigable permitiría a investigadores sin conocimientos avanzados de programación utilizar el framework de manera eficiente, facilitando así su adopción en diversos contextos de investigación.

El diseño y desarrollo de este framework representa una contribución significativa al campo de la bioinformática y la medicina de precisión, al facilitar un proceso más rápido y eficiente en la clasificación de variantes genéticas. Este avance no solo tiene el potencial de mejorar la investigación genética, sino también de acelerar la adopción de soluciones automatizadas en entornos clínicos, mejorando la calidad de la atención médica y optimizando los recursos disponibles.

Bibliografía

[1] SHIN, C. et al. Precision medicine for psychopharmacology: A general introduction. En: *Expert Review of Neurotherapeutics*. 2016, 16(7), 831-839. Disponible en: https://doi.org/10.1080/14737175.2016.1182022.

[2] RICHARDS, C. et al. Standards and guidelines for the interpretation of sequence variants: A joint consensus recommendation of the American College of Medical Genetics and Genomics and the Association for Molecular Pathology. En: *Genetics in Medicine*. 2015, 17(5), 405-424. Disponible en: https://doi.org/10.1038/gim.2015.30.

[3] *Variante genética: ¿benigna o patogénica? - Blog Mendelics.* Blog Mendelics. Disponible en: https://blog.mendelics.com.br/es/variante-genetica-benigna-patogenica-acmg/.

[4] COSTA, M. et al. *A reference meta-model to understand DNA variant interpretation guidelines.* En: *Conceptual modeling* (375-393). Springer, Cham, 2023. Disponible en: https://doi.org/10.1007/978-3-031-47262-6_20.

[5] AGAOGLU, N. B. et al. Consistency of variant interpretations among bioinformaticians and clinical geneticists in hereditary cancer panels. En: *European Journal of Human Genetics*. 2022, 30, 378-382. Disponible en: https://doi.org/10.1038/s41431-022-01060-7.

[6] WAHBEH, A. H. et al. A comparison study between data mining tools over some classification methods. En: *International Journal of Advanced Computer Science and Applications*. 2011, 8(2), 18-26. Disponible en: https://doi.org/10.14569/specialissue.2011.010304.

[7] WIERINGA, R. J. *Design science methodology for information systems and software engineering*. Springer Berlin Heidelberg, 2014. Disponible en: https://doi.org/10.1007/978-3-662-43839-8.

[8] PEFFERS, K. et al. A design science research methodology for information systems research. En: *Journal of Management Information Systems*. 2007, 24(3), 45-77. Disponible en: https://doi.org/10.2753/mis0742-1222240302.

[9] HEVNER, A. et al. Design science in information systems research. En: *MIS Quarterly*. 2004, 28(1), 75-105. Disponible en: https://doi.org/10.2307/25148625.

[10] REHM, H. L. et al. ACMG clinical laboratory standards for next-generation sequencing. En: *Genetics in Medicine*. 2013, 15(9), 722-747. Disponible en: https://doi.org/10.1038/gim.2013.92.

[11] LANDRUM, M. J. et al. ClinVar: Public archive of interpretations of clinically relevant variants. En: *Nucleic Acids Research*. 2016, 44(D1), D862-D868. Disponible en: https://doi.org/10.1093/nar/gkv1222.

[12] PLON, S. E. et al. Sequence variant classification and reporting: recommendations for improving the interpretation of cancer susceptibility genetic test results. En: *Human Mutation*. 2008, 29(11), 1282-1291. Disponible en: https://doi.org/10.1002/humu.20880.

[13] AMENDOLA, L. M. et al. Performance of ACMG-AMP variant-interpretation guidelines among nine laboratories in the Clinical Sequencing Exploratory Research

consortium. En: *The American Journal of Human Genetics*. 2016, 98(6), 1067-1076. Disponible en: https://doi.org/10.1016/j.ajhg.2016.03.024.

[14] REHM, H. L. et al. ClinGen - The clinical genome resource. En: *The New England Journal of Medicine*. 2015, 372(23), 2235-2242. Disponible en: https://doi.org/10.1056/NEJMsr1406261.

[15] TAVTIGIAN, S. V. et al. Modeling the ACMG/AMP variant classification guidelines as a Bayesian classification framework. En: *Genetics in Medicine*. 2018, 20(9), 1054-1060. Disponible en: https://doi.org/10.1038/gim.2017.210.

[16] *¿Qué es un framework de automatización de pruebas?* Software testing & QA - QAlified. Disponible en: https://www.qalified.com/es/blog/framework-automatizacion-pruebas/.

[17] HARRISON, S. M. et al. Clinical laboratories collaborate to resolve differences in variant interpretations submitted to ClinVar. En: *Genetics in Medicine*. 2017, 19(10), 1096-1104. Disponible en: https://doi.org/10.1038/gim.2017.14.

[18] LIBBRECHT, M. W.; NOBLE, W. S Machine learning applications in genetics and genomics. En: *Nature Reviews Genetics*. 2015, 16(6), 321-332. Disponible en: https://doi.org/10.1038/nrg3920.

[19] LEUNG, M. K. K. et al. Machine learning in genomic medicine: A review of computational problems and data sets. En: *Proceedings of the IEEE*. 2016, 104(1), 176-197. Disponible en: https://doi.org/10.1109/JPROC.2015.2494198.

[20] HUANG, S. et al. Applications of Support Vector Machine (SVM) learning in cancer genomics. En: *Cancer genomics & proteomics*. 2018, 15(1), 41-51. Disponible en: https://doi.org/10.21873/cgp.20063.

[21] BREIMAN, L. Random forests. En: *Machine Learning*. 2001, 45, 5-32. Disponible en: https://doi.org/10.1023/A:1010933404324.

[22] KOPANOS, C. et al. VarSome: The human genomic variant search engine. En: *Bioinformatics*. 2018, 35(11), 1978-1980. Disponible en: https://doi.org/10.1093/bioinformatics/bty897.

[23] LI, Q.; WANG, K. InterVar: Clinical interpretation of genetic variants by the 2015 ACMG-AMP guidelines. En: *The American Journal of Human Genetics*. 2017, 100(2), 267-280. Disponible en: https://doi.org/10.1016/j.ajhg.2017.01.004.

[24] NICORA, G. et al. CardioVAI: An automatic implementation of ACMG-AMP variant interpretation guidelines in the diagnosis of cardiovascular diseases. En: *Human Mutation*. 2018, 39(12), 1835-1846. Disponible en: https://doi.org/10.1002/humu.23665.

[25] WHIFFIN, N. et al. CardioClassifier: Disease- and gene-specific computational decision support for clinical genome interpretation. En: *Genetics in Medicine*. 2018, 20(10), 1246-1254. Disponible en: https://doi.org/10.1038/gim.2017.258.

[26] NG, P.; HENIKOFF, S. SIFT: Predicting amino acid changes that affect protein function. En: *Nucleic Acids Research*. 2003, 31(13), 3812-3814. Disponible en: https://doi.org/10.1093/nar/gkg509.

[27] ADZHUBIE, I. A. et al. A method and server for predicting damaging missense mutations. En: *Nature Methods*. 2010, 7(4), 248-249. Disponible en: https://doi.org/10.1038/nmeth0410-248.

[28] KIRCHER, M. et al. A general framework for estimating the relative pathogenicity of human genetic variants. En: *Nature Genetics*. 2014, 46(3), 310-315. Disponible en: https://doi.org/10.1038/ng.2892.

[29] BORNBERG-BAUER, E.; PATON, N. W. Conceptual data modelling for bioinformatics. En: *Briefings in Bioinformatics*. 2002, 3(2), 166-180. Disponible en: https://doi.org/10.1093/bib/3.2.166.

[30] DURÁN, F.; TROYA, J.; VALLECILLO, A. *Desarrollo de software dirigido por modelos*. Universitat Oberta de Catalunya, 2013.

[31] BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software architecture in practice*. 2ª ed. Boston: Addison-Wesley, 2003.

[32] *¿Qué es la arquitectura de tres niveles?* | *IBM.* IBM - United States. Disponible en: https://www.ibm.com/es-es/topics/three-tier-architecture.

[33] *Qué es Visual Studio Code y qué ventajas ofrece*. OpenWebinars. Disponible en: https://www.openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece.

[34] *¿Qué es MySQL Workbench?* Keepcoding. Disponible en: https://www.keepcoding.io/blog/que-es-mysql-workbench.

[35] *Qué es GitLab Concepto y definición. Glosario.* GAMCO. Disponible en: https://www.gamco.es/glosario/gitlab/#:~:text=GitLab%20es%20una%20plataforma%20de,código%20fuente%20de%20sus%20proyectos.

[36] *¿Qué es una base de datos?* Oracle | Cloud Applications and Cloud Platform. Disponible en: https://www.oracle.com/es/database/what-is-database/.

[37] JATANA, N. et al. A survey and comparison of relational and non-relational database. En: *International Journal of Engineering Research & Technology (IJERT)*. 2012, 1(6), 1-5.

[38] *Top ten programming languages for bioinformatics in 2023 - Omics tutorials.* Omics tutorials - Bioinformatics, Genomics, Proteomics and Transcriptomics. Disponible en: https://omicstutorials.com/top-ten-programming-languages-for-bioinformatics-in-2023/.

[39] FOURMENT, M; GILLINGS, M. R. A comparison of common programming languages used in bioinformatics. En: *BMC Bioinformatics*. 2008, 9(82). Disponible en: https://doi.org/10.1186/1471-2105-9-82.

[40] SCHUERER, K.; LETONDAL, C. *Python course in bioinformatics.* Pasteur Institute, 2002.

[41] NELLI, F. *The pandas library – an introduction.* En: *Python data analytics: With Pandas, NumPy and Matplotlib* (73-114). Berkeley, CA: Apress, 2023. Disponible en: https://doi.org/10.1007/978-1-4842-3913-1_4.

[42] *¿Qué es un DataFrame?* Formación en ciencia de datos | DataScientest.com. Disponible en: https://datascientest.com/es/que-es-un-dataframe.

[43] NORMAN, D. *The psychology of everyday things*. Basic Books, 1988.

[44] JANSEN, B. J. The graphical user interface. En: *ACM SIGCHI Bulletin*. 1998, 30(2), 22-26. Disponible en: https://doi.org/10.1145/279044.279051.

[45] QUARANTA, L; CALEFATO, F.; LANUBILE, F. KGTorrent: A dataset of Python Jupyter Notebooks from Kaggle. En: *2021 IEEE/ACM 18th international conference*

on mining software repositories (MSR). IEEE, 2021. Disponible en: https://doi.org/10.1109/msr52588.2021.00072.

[46] KLUYVER, T. et al. Jupyter Notebooks – a publishing format for reproducible computational workflows. En: *Positionning and Power in Academic Publishing: Players, Agents and Agendas*. 2016, 87-90. Disponible en: https://doi.org/10.3233/978-1-61499-649-1-87.

[47] *prettytable*. PyPl. Disponible en: https://www.pypi.org/project/prettytable.

[48] *Python RegEx*. W3Schools Online Web Tutorials. Disponible en: https://www.w3schools.com/python/python_regex.asp.

[49] *Portada - Desarrollo Sostenible*. Naciones Unidas. Disponible en: https://www.un.org/sustainabledevelopment/es.

Anexos

Anexo 1. Objetivos de Desarrollo Sostenible

En 2015, la Organización de las Naciones Unidas (ONU) aprobó la Agenda 2030 sobre el Desarrollo Sostenible, la cual cuenta con 17 Objetivos de Desarrollo Sostenible (ODS). Cada uno de los 17 objetivos tiene metas específicas para erradicar la pobreza, proteger el planea y asegurar la prosperidad para todos [49].

La Tabla 2 muestra el grado de relación con los ODS.

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Tabla 3 - Tabla ODS

En los siguientes puntos, se explorará cómo el diseño y desarrollo de un framework para la automatización de la clasificación de variantes genéticas contribuye particularmente a los ODS 3 (salud y bienestar) y ODS 9 (industria, innovación e infraestructuras).

ODS 3: Salud y bienestar

El ODS 3 busca garantizar una vida sana y promover el bienestar para todos en todas las edades.

La automatización de la clasificación de variantes genéticas tiene un impacto directo en la medicina de precisión, permitiendo un diagnóstico más rápido y preciso de enfermedades genéticas, y facilitando tratamientos más efectivos y personalizados, mejorando la calidad de vida de los pacientes. Este avance ha llegado a un grado de cumplimiento alto del ODS 3, ya que mejora considerablemente los resultados de salud y bienestar, acercando más a la sociedad a las metas establecidas.

Además, puede contribuir a una mejor vigilancia de la salud pública, permitiendo la identificación temprana de predisposiciones genéticas en poblaciones y facilitando intervenciones preventivas más eficaces.

De esta manera, se fomenta un sistema de salud más equitativo y eficiente, alineado con las metas del ODS 3 de mejorar la salud y el bienestar global.

ODS 9: Industria, innovación e infraestructuras

El ODS 9 busca construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación.

La creación de un sistema automatizado para la clasificación de variantes genéticas representa un avance tecnológico significativo que impulsa la investigación y el desarrollo en genética y genómica. Sin embargo, el grado de cumplimiento del ODS 9 es medio, ya que, aunque se han logrado avances en innovación e infraestructura, aún se requiere más trabajo para asegurar que estas mejoras sean sostenibles y ampliamente adoptadas en diversos sectores.

La implementación de este framework puede servir como un modelo para futuras innovaciones en otros campos de la bioinformática y la biotecnología, estimulando el desarrollo de nuevas tecnologías y metodologías que pueden ser aplicadas en una variedad de contextos clínicos y de investigación.

Así, este proyecto contribuye a la construcción de infraestructuras tecnológicas avanzadas y al fomento de la innovación, en plena consonancia con los objetivos del ODS 9 de impulsar una industrialización sostenible y resiliente.

Anexo 2. División de los criterios según la fuerza de la evidencia

Este anexo muestra la división de Richards et al. [2] de los criterios según la fuerza de la evidencia. En la Tabla 3 se aprecian los criterios para clasificar las variantes benignas, y en la Tabla 4 los criterios para clasificar las patogénicas.

Evidencia muy fuerte	**BA1**	Variantes con una frecuencia alélica superior al 5% en el Exome Sequencing Project, 1000 Genomes o ExAC.
Evidencia fuerte	**BS1**	Variantes con una frecuencia alélica mayor de lo esperado para el trastorno.
	BS2	Variantes observadas en adultos sanos para trastornos recesivos, dominantes o ligados al X con pertinencia completa esperada a una edad temprana.
	BS3	Variantes respaldadas por estudios funcionales bien establecidos in vitro o in vivo que muestran que no hay efecto dañino en la función de la proteína o el empalme.
	BS4	Variantes que no segregan con la enfermedad en familias afectadas.
Evidencia de soporte	**BP1**	Variantes missense en genes para los que se sabe que las variables truncantes causan la enfermedad.
	BP2	Variantes observadas en trans con una variante patogénica para un gen o trastorno dominante con penetrancia completa; u observada en cis con una variante patogénica en cualquier patrón de herencia.
	BP3	Variantes con deleciones/inserciones en regiones repetitivas sin función conocida.
	BP4	Variantes respaldadas por múltiples líneas de evidencia computacional que no sugieren ningún impacto en el gen.
	BP5	Variantes encontradas en casos con una base molecular alternativa para la enfermedad.
	BP6	Variantes reportadas benignas para fuentes reputadas.
	BP7	Variantes sinónimas para las que los algoritmos de predicción de empalme no predicen un impacto en la secuencia de consenso de empalme ni la creación de un nuevo sitio de empalme y el nucleótido no está altamente conservado.

Tabla 4 - Criterios para clasificar variantes benignas

Evidencia muy fuerte	**PVS1**	Variante nula en un gen donde la pérdida de función es un mecanismo conocido de la enfermedad.
Evidencia fuerte	**PS1**	Variantes con cambio en el aminoácido idéntico al de una variante patogénica previamente establecida.
	PS2	Variantes de novo en pacientes con la enfermedad y sin antecedentes familiares.
	PS3	Variantes in vitro o in vivo que apoyan un efecto dañino en el gen.
	PS4	Variantes significativamente más frecuentes en individuos afectados que con la prevalencia en controles.
Evidencia moderada	**PM1**	Variantes localizadas en puntos críticos mutacionales y/o en dominios funcionales críticos y bien establecidos.
	PM2	Variantes ausentes en controles.
	PM3	Variantes trans con variantes patogénicas en trastornos recesivos.
	PM4	Variantes que causan cambios en la longitud de proteínas.
	PM5	Variantes missense novedosas en un residuo de aminoácido donde ya se ha observado previamente un cambio missense diferente clasificado como patogénico.
	PM6	Variantes asumidas como de novo sin la confirmación de paternidad y maternidad.
Evidencia de soporte	**PP1**	Variantes co-segregadas con la enfermedad en múltiples miembros de la familia afectados en un gen definitivamente conocido por causar la enfermedad.
	PP2	Variantes missense en genes con baja variación benigna.
	PP3	Variantes respaldadas por múltiples líneas de evidencia computacional.
	PP4	Variantes con fenotipo o historial familiar específico.
	PP5	Variantes reportadas como patogénicas por fuentes reputadas.

Tabla 5 - Criterios para clasificar variantes patogénicas